UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

Departamento de Electrónica e Computación

PhD Thesis

DYNAMICALLY RECONFIGURABLE ARCHITECTURE FOR EMBEDDED COMPUTER VISION SYSTEMS

Author:

Alejandro Manuel Nieto Lareo

PhD supervisors:

David López Vilariño

Víctor Manuel Brea Sánchez

Santiago de Compostela, September 2012

Dr. David López Vilariño, Profesor Titular de Universidade da Área de Electrónica da Universidade de Santiago de Compostela

Dr. Víctor Manuel Brea Sánchez, Profesor Contratado Doutor da Área de Electrónica da Universidade de Santiago de Compostela

FAN CONSTAR:

Que a memoria titulada **DYNAMICALLY RECONFIGURABLE ARCHITECTURE FOR EMBEDDED COMPUTER VISION SYSTEMS** foi realizada por **D. Alejandro Manuel Nieto Lareo** baixo a nosa dirección no Departamento de Electrónica e Computación e no Centro Singular de Investigación en Tecnoloxías da Información (CITIUS) da Universidade de Santiago de Compostela, e constitúe a Tese que se presenta para optar ao grado de Doutor.

Santiago de Compostela, setembro de 2012

David López Vilariño
Codirector da tese

Víctor Manuel Brea Sáncez
Codirector da tese

Doutorando
Alejandro Manuel Nieto Lareo

Aos meus pais e ao meu irmán,
pero sobre todo, aos meus avós.

Don't worry head, the computer will do our thinking now!

Homer J. Simpson (after buying one)

People who are really serious about software should make their own hardware.

Alan Kay

Acknowledgements

It is a pleasure to thank the many people who made this thesis possible.

It is difficult to overstate my gratitude to my PhD. supervisors, Dr. David López Vilariño and Dr. Víctor Manuel Brea Sánchez. With their enthusiasm, their inspiration, and their great efforts to explain things clearly and simply, they helped to make research fun for me. Throughout all my thesis period, they provided encouragement, sound advice, good teaching, good company, and lots of good ideas. I would have been lost without them.

I wish to express my warm and sincere thanks to Associate Professor Javier Díaz Alonso from the Universidad de Granada, and Professor Nigel Topham from the University of Edinburgh, who not only allowed me to improve the results of this work and expand the initial objectives, but also they have given me the opportunity to work with leading research groups and have allowed me to enjoy new experiences.

My sincere thanks to Professor Diego Cabello Ferrer, Associate Professor Paula López Martínez and Dr. Fernando Rafael Pardo Seco from the Universidade de Santiago de Compostela, Associate Professor Roberto Rodríguez Osorio from the Universidade da Coruña and Dr. Carmen Alonso Montes, for all the help they rendered me during my research period. I also wish to thank Dr. Jordi Albó Canals from the Universitat Ramon Llull, Dr. Freddie Qu and Christopher Thompson from the University of Edinburgh, and all members of the Departamento de Arquitectura y Tecnología de Computadores of the Universidad de Granada, for they support during my research stays. My gratitude is also extended to all members of the Departamento de Electrónica e Computación of the University of Santiago de Compostela, and in particular to the Grupo de Visión Artificial.

I am indebted to my many colleagues for providing a stimulating and fun environment in which to learn and grow. I am especially grateful to David, Bea B., Bea P., Roi, Fernando, Levo, Natalia, Manuel, Cris, Yago, Pablo, María and Juan at the Departamento de Electrónica

e Computación of the Universidade de Santiago de Compostela, for all the emotional support, comraderie, entertainment, and caring they provided. I also wish to thank Xabier, Lorena, Mar, Isa, Raquel, Paloma, Ana, Yolanda and many others who have been there all these years. I also want make special mention to the people of the Banda de Música de Arca, because work is not everything. And in general, to all those who in one way or another have been there all these years.

Tamén quero agradecer á Xunta de Galicia a creación do Programa María Barbeito, así como aos proxectos PGIDT06TIC10502PR, 10PXIB206168PR e 10PXIB206037PR, que me permitiron financiar esta tese e gozar da experiencia de colaborar con outros grupos de investigación, algo moi importante para a consecución dos meus obxectivos.

Finalmente, e o máis importante, quero darlle as gracias aos meus pais, Mª Obdulia e Manuel, ao meu irmán Adrián, e aos meus avós Secundino e Obdulia. Eles déronme todo o apoio que precisei estes anos non so para acabar este traballo senón para medrar como persoa. A eles lles dedico esta tese.

Santiago de Compostela, September 2012

Contents

Resumo da tese

Seguindo o regulamento dos estudios de terceiro ciclo da Universidade de Santiago de Compostela, aprobado na Xunta de Goberno do día 7 de abril de 2000 (DOG de 6 de marzo de 2001) e modificado pola Xunta de Goberno de 14 de novembro de 2000, o Consello de Goberno de 22 de novembro de 2003, de 18 de xullo de 2005 (artigos 30 a 45), de 11 de novembro de 2008 e de 14 de maio de 2009; e, concretamente, cumprindo coas especificacións indicadas no capítulo 4, artigo 30, apartado 3 de dito regulamento, amósase a continuación un resumo en galego da presente tese.

Motivación e obxectivos

Os algoritmos e técnicas de Visión por Computador son cada vez máis robustos e precisos, ofrecendo novas posibilidades para o tratamento da información visual. Sen embargo, isto implica un aumento dos requisitos de cómputo polo que a súa implementación en dispositivos máis aló do alcance dos prototipos está a voltarse máis complexo. Son precisos novos enfoques que non afecten ás características dos algoritmos para acadar solucións de compromiso en termos de velocidade, latencia, consumo de potencia ou custo. Isto inclúe tanto ao apartado do software como ao da arquitectura hardware para poder aproveitar eficientemente as características do dispositivo e explotar plenamente as capacidades dos algoritmos.

Para solventar un problema no ámbito da Visión por Computador existen moitas técnicas e aproximacións dispoñibles na literatura. A variedade de algoritmos é moi ampla e presentan características moi diferentes, incluíndo non so a representación de datos ou as operacións aritméticas, se non tamén os paradigmas de computación ou a complexidade dos programas. Isto fai que o deseño das arquitecturas hardware sexa un desafío se é a mesma arquitectura a que ten que afrontar algoritmos diferentes cumprindo unhas restricións estritas.

Os ordenadores de escritorio son a miúdo empregados para desenvolver os algoritmos. Ofrecen unha gran flexibilidade para executar calquera algoritmo e avaliar a súa viabilidade e precisión. Con todo, o dispositivo obxectivo depende da aplicación e adoitan empregarse módulos adicionais para acelerar o proceso. Para cumprir os requisitos da aplicación pode ser incluso necesario deseñar unha arquitectura específica. Neste caso é posible optimizar todas as figuras de mérito implicadas. Os diferentes campos de aplicación fixan diferentes requisitos polo que as distintas arquitecturas deben adaptarse aos mesmos de forma eficaz.

Os avances na industria do semicondutor permiten incluír cada vez máis recursos na mesma unidade de área, mantendo un consumo de potencia contido. Isto fixo posible desenvolver novos sistemas embebidos cun rendemento similar aos sistemas convencionais do pasado recente. Deste xeito, as novas aplicacións son tecnicamente factibles e o mercado de aplicacións de visión embebidos está aumentando significativamente. Os sistemas nun chip posibilitan embeber no mesmo chip todos os módulos necesarios para un sistema dado, incluíndo computación, control e comunicacións, proporcionando unha vantaxe significativa sobre o seu equivalente sistema multi-chip. Sen embargo, os sistemas nun chip requiren un longo período de deseño e moitos recursos, tanto humanos como materiais. Ademais, as arquitecturas a medida tamén requiren un período de adestramento polos enxeñeiros de software para explotar plenamente as súas capacidades, reducíndose así a vantaxe competitiva destes sistemas. Isto é crítico cando medra o número de unidades de propósito específico que executan os diferentes algoritmos empregados.

O obxectivo deste traballo é proporcionar unha arquitectura capaz de executar a maioría dos algoritmos de Visión por Computador cun rendemento adaptable ás distintas etapas de cada algoritmo. Ademais, ten que ser capaz de executar as etapas restantes de xeito que o rendemento global non se vexa comprometido. A arquitectura tamén está pensada para reducir o tempo empregado en migrar os algoritmos, reducindo a etapa de migración facilitando a paralelización e a sincronización entre os diferentes bloques concurrentes do algoritmo. Desde o punto de vista da flexibilidade, o sistema ten que ser modular e escalable para afrontar diferentes áreas de mercado con diferentes requisitos, como rendemento, consumo de área ou custo. As melloras na arquitectura ou as modificacións baixo demanda deben ser posible sen mudar dramaticamente a forma en que o hardware traballa. Todos estes elementos combinados proporcionarán suficiente flexibilidade e rendemento para executar eficientemente a maior parte dos algoritmos de Visión de Computador, co beneficio de empregar un so dispositivo.

Esta tese está dividida en tres capítulos. O Capítulo 1 presenta os desafíos da Visión por Computador, incluíndo os diferentes paradigmas de computación e as arquitecturas hardware e dispositivos empregados para afrontar os requisitos dos algoritmos. No Capítulo 2 preséntase unha nova arquitectura hardware para executar algoritmos de visión temperá. Despois de examinar diferentes aproximacións, estudiaranse as capacidades dun procesador masivamente paralelo para o procesamento de imaxes binarias. As conclusións deste estudio son esenciais para o deseño dunha arquitectura máis eficiente e capaz para tarefas de visión temperá. Testearase empregando un algoritmo usado en aplicacións médicas, altamente custoso en termos computacionais. No Capítulo 3, os resultados obtidos da comparativa do procesador de propósito xeral descrito no capítulo anterior son empregados para definir unha nova arquitectura. Esta arquitectura está deseñada para aplicacións embebidas e enfocada na reconfiguración dinámica do seu camiño de datos para reproducir o fluxo de execución do algoritmo en execución. Deste xeito, a arquitectura aumenta o seu rango de operación de modo que todas as etapas dun algoritmo de Visión de Computador xenérico poden ser implementadas de xeito eficiente nun único dispositivo embebido. Os resultados da execución de diversos algoritmos de procesado de imaxe e a comparativa con outras solucións similares completan este capítulo. Finalmente, extráense as principais conclusións deste traballo.

Introdución

A gran variedade de aplicacións de Visión por Computador fai difícil clasificalas en categorías claramente diferenciadas. Como resultado, o deseño dunha única arquitectura hardware que execute eficientemente todas as etapas de procesado que inclúe calquera algoritmo de Visión por Computador convértese nunha tarefa moi complexa. Na literatura están dispoñibles diversos estudios onde diferentes plataformas son testeadas baixo as mesmas condicións. Estes estudios amosan que o axuste fino dos parámetros é crítico en termos de rendemento e que o deseño de novas técnicas de computación paralela é un requisito para explorar as capacidades dos dispositivos que inclúen máis dunha unidade de cómputo. Ademais, o aumento das vendas e a aparición de mercados emerxentes fai que os investimentos en novo hardware sexa unha necesidade.

A plataforma máis accesible é un computador persoal equipado cunha GPU con soporte para cómputo de propósito xeral. Tanto como plataforma de deseño como dispositivo final, o computador persoal reduce enormemente o tempo de desenvolvemento e os custos asociados.

As GPUs ofrecen un alto rendemento nas tarefas máis intensivas mentres que as CPUs ofrecen extensións multimedia que permiten acelerar o cómputo nas etapas nas que non é posible empregar ás GPUs. Ademais, inclúen todos os elementos necesarios para comunicacións, almacenamento e interacción co usuario. A dispoñibilidade de modelos é moi grande polo que é posible seleccionar a plataforma máis adecuada segundo para cada aplicación, cumprindo así os seus requisitos. Cando o rendemento da CPU non é adecuado, os DSPs son unha seria alternativa. Ademais, o seu uso é case obrigatorio cando estamos a falar de sistemas embebidos e non queremos comprometer o rendemento, e onde o consumo de potencia ou o factor de forma son moi restritivos. Son amplamente utilizados para prototipar chips de aplicación específica aínda que as FPGAs reduciron o seu nicho de mercado. Os FPGAs ofrecen unha alta integración e flexibilidade ademais dun grande número de unidades xa dispoñibles que permiten reducir os custos asociados ao deseño, desenvolvemento e testeo. A pesar de que todos os dispositivos descritos son de aplicación específica, non foron concibidos para unha aplicación única. Para reducir os custos, os fabricantes expanden o seu rango de operación, aínda que é posible atopar familias de produtos específicas para certos nichos. Sen embargo, tamén hai dispositivos específicos para tarefas moi concretas, onde os requisitos son tan estritos que ningún outro dispositivo os cumpre. A flexibilidade é completa e non hai restricións para empregar as tecnoloxías máis avanzadas que so estarán dispoñibles en dispositivos comerciais nun futuro próximo.

Case todas as aplicacións de Visión por Computador necesitan afrontar todas etapas de procesamento nun menor ou maior grao. A miúdo, isto conleva aplicar mecanismos que inclúen o manexo do paralelismo espacial (dato), temporal (instrución) e secuencial, ou incluso unha combinación deles. Cada etapa corresponde aproximadamente cun tipo de paralelismo, co que todos estes mecanismos adoitan implementarse na maior parte das aplicacións. As etapas de baixo nivel benefícianse do paralelismo masivo con sistemas de distribución de datos e operacións aritméticas sinxelos. Cando a abstracción aumenta, durante as tarefas de medio nivel, os algoritmos requiren maior información acerca do problema a resolver, polo que se incrementa a súa complexidade. Isto conleva estruturas de datos e fluxos de execución máis complexos, onde a distribución da información fai difícil explotar o paralelismo espacial, aínda que normalmente está presente. As arquitecturas paralelas a nivel de tarefa son capaces de explotar de forma máis eficiente e sinxela esta etapa. As etapas de baixo e medio nivel poden implementarse en solucións hardware puras porque con frecuencia so executan tarefas de cómputo. Sen embargo, a etapa de alto nivel está máis próxima ao software e os enxeñeiros

poden explotar este feito para construír sistemas máis complexos usando un procesador de propósito xeral. Ademais, permitirá actuar como sistema de control das unidades de cómputo ou dos sistemas de comunicación. Isto non está directamente relacionado cos problemas da Visión por Computador pero sen embargo é un requerimento para un prototipo funcional. Neste caso, o uso dun procesador de propósito xeral é un claro beneficio pois facilita enormemente o control e incrementa a flexibilidade do sistema.

A pesar de que é case imposible desenvolver un sistema capaz de executar todas as operacións dun xeito óptimo dada a rica natureza das aplicacións de Visión por Computador, é desexable que proporcione a capacidade para executar calquera operación. O deseño ten que ser escalable para adaptalo ás necesidades específicas de cada aplicación. Deste xeito é posible proporcionar unha familia de dispositivos con diferentes capacidades de cómputo de xeito sinxelo. A arquitectura interna debe ser tamén modular, de modo que desde un esbozo básico se poidan engadir máis características sen cambios dramáticos. En xeral, un microprocesador de alto nivel é un requisito para xestionar comunicacións e operacións complexas entre o sistema e os compoñentes externos do sistema completo. Un certo número de unidades auxiliares capaces de manexar os paradigmas de computación paralelos a nivel de dato e de tarefa executarían a parte máis custosa do algoritmo. En particular, unha unidade de SIMD grande aumentará o rendemento nas primeiras etapas. Un controlador integrado para memorias de alta velocidade é tamén crítico para reducir os problemas de acceso aos datos na memoria externa. Todos estes elementos combinados son capaces de afrontar eficientemente a maioría das situacións que se atopan nos algoritmos máis comunmente empregados.

Abordando a etapa de baixo nivel

As tarefas da etapa de baixo nivel caracterízanse por ser simples, repetitivas e aplicadas sobre unha gran cantidade de datos, e non requiren unha alta precisión. Sen embargo, son moi custosas computacionalmente, especialmente en sistemas embebidos, o ámbito que se aborda neste traballo.

No segundo capítulo preséntase un procesador de propósito xeral para o procesado de imaxes a baixo nivel. En primeiro lugar, preséntase un estudio dunha arquitectura para imaxes binarias (branco e negro), onde se conclúe que o paralelismo espacial masivo non representa a solución máis eficiente. A pesares que a distribución matricial do conxunto de unidades de cómputo con comunicación local entre elas pode explotar moi eficientemente as caracte-

rísticas das tarefas que executan nesta etapa, a correspondencia dun procesador por píxel da imaxe ten varias serias desvantaxes. Por un lado, o número de recursos hardware necesarios é elevado, resultando en procesadores que traballan con imaxes con pouca resolución e con frecuencias de reloxo baixas. A pesares de tratarse de problemas técnicos, o rendemento real e a conveniencia deste tipo de procesadores é menor do esperable teoricamente. Deste xeito, solucións a priori menos óptimas poden superar estas limitacións e ofrecer mellores figuras de mérito.

Como resultado, este procesador preliminar foi ampliado para procesar imaxes non binarias, estendendo as capacidades de cada unidade de cómputo e reducindo o paralelismo. Esta arquitectura, unha matriz de procesadores de gran groso, proporciona mellores figuras de mérito. En particular, é máis sinxelo escalar a matriz sen comprometer a frecuencia de reloxo e permite manexar imaxes de maior resolución. Cando se compara con outras propostas, unha implementación en FPGA desta arquitectura mellora os resultados dun procesador SIMD masivamente paralelo que conta con 8500 % máis unidades de cómputo. Esta densidade de integración é acadada empregando arquitecturas de sinal mixta que limitan o tamaño da matriz e a frecuencia de reloxo, limitando a potencia que se pode extraer del. Sen embargo, o consumo de potencia acadado é moi difícil de bater. As matrices masivamente paralelas de procesadores tamén amosan un alto rendemento explotando o paralelismo temporal, aínda que as operacións recursivas reducen en gran medida o rendemento, sendo difícil atopar unha solución de compromiso. O algoritmo empregado para comparar o rendemento e posterior discusión é representativo da súa clase, proporcionando información adicional acerca de como os datos interactúan cando se concatenan os diferentes operadores. Isto danos unha vantaxe durante a etapa de deseño dunha arquitectura especializada en acelerar este tipo de operacións. Os resultados tamén amosan que CPUs convencionais non son eficientes para tarefas de visión temperá, polo que outras aproximacións están claramente xustificadas.

As principais conclusións refírense a como as unidades aritméticas están organizadas e como se distribúen os datos entre elas. Como se víu, unha organización matricial permite afrontar de xeito eficiente operacións de baixo nivel, que inclúen operacións píxel a píxel e interaccións locais. Polo contrario, cando se procesan datos que non son puramente imaxes, unha matriz bidimensional non proporciona a necesaria flexibilidade. Isto acentúase segundo o nivel de abstracción aumenta. A matriz masivamente paralela de procesadores, que explota en gran medida o paralelismo temporal é unha boa proba deste feito. Un problema non resolto polo momento é a entrada e saída de datos. As tarefas de procesado de imaxe normalmente

requiren un gran ancho de banda e unidades de almacenamento de alta capacitade, o que pode limitar o rendemento se as unidades de cómputo están consumindo tempo esperando polos datos a procesar. Superpoñer a transferencia de datos e o seu procesado é esencial.

As conclusións expostas neste capítulo lévannos a propoñer unha versión mellorada da arquitectura inicial, enfocándose en estender o rango de aplicación sen comprometer o rendemento, ampliar os requisitos hardware ou o consumo de potencia.

Expandindo o rango de operación

O terceiro capítulo amosa unha extensiva optimización da arquitectura proposta no segundo capítulo. A primeira proposta foi deseñada para abordar a etapa de baixo nivel que inclúen a maioría das aplicacións de Visión por Computador, que consumen a maior parte da carga de traballo da CPU. A matriz de procesadores de gran groso probou ser moi eficaz nestas tarefas. Sen embargo, a súa flexibilidade está limitada e non manexa de forma eficiente as posteriores etapas de procesado. A distribución das unidades aritméticas, a representación dos datos e as operacións de entrada/saída non son as adecuadas para as etapas de medio e alto nivel.

Para deseñar a arquitectura finalmente proposta seleccionáronse as mellores características dos paradigmas de computación SIMD (espacial) e MIMD (temporal), e combináronse para reducir os recursos hardware empregados. Deste xeito, a nova arquitectura pode explotar tanto os paradigmas de computación paralela a nivel espacial e temporal, executando operacións masivamente paralelas en modo SIMD, a nivel de tarefa en modo MIMD e de alto nivel ou de xeito serie nun procesador secuencial. O paradigma de computación pode cambiarse en tempo de execución segundo os requisitos que marque cada un dos algoritmos implementados. O manexo da entrada/saída de datos foi mellorado e permite superpoñer computación e transferencia de datos, reducindo o colo de botella e facilitando o acceso á memoria externa onde se almacenan os datos. Un dos obxectivos desta arquitectura é prover da suficiente flexibilidade como para manexar situacións moi diversas, incluíndo diferentes tipos de datos, tamaños de imaxe ou representacións abstractas. Para este propósito empréganse dúas redes diferentes para intercambiar datos entre as unidades de cómputo. As súas características facilitan a distribución dos datos e reducen o tempo empregado no proceso, aumentando así o rendemento. Ademais da flexibilidade, outro obxectivo perseguido é o de facer a arquitectura altamente configurable, de forma que esta non dependa do número de unidades de cómputo, o tamaño das memorias ou o tipo de operacións aritméticas empregadas. Isto permite crear unha

familia de procesadores que aborden diferentes aplicacións en diferentes campos, axustando así non so o rendemento ou o consumo de potencia, se non tamén o custo.

O procesador SIMD/MIMD híbrido foi testeado nun sistema embebido baseado en FPGA. Un conxunto de operadores e algoritmos foi implementado para avaliar o seu rendemento e a súa factibilidade. Os resultados proban que se pode alcanzar un alto rendemento cando os diferentes modos de computación se empregan adecuadamente. A arquitectura foi compara con outras propostas similares, incluíndo a matriz de procesadores de gran groso proposta no segundo capítulo. Os resultados amosan que o prezo a pagar debido ao incremento de hardware e á inclusión de características de cómputo de propósito xeral, que poderían penalizar o rendemento en tarefas específicas, é menor que a ganancia en termos de flexibilidade xa que o rendemento non se ve comprometido.

Conclusións

Neste traballo, presentouse unha nova arquitectura hardware para acelerar aplicacións de Visión por Computador en sistemas embebidos. Esta arquitectura hardware proporciona un dispositivo dun único chip capaz de executar a maioría dos algoritmos de procesado de imaxe e tarefas relacionadas.

En primeiro lugar introduciuse o problema da Visión por Computador e analizáronse as familias de algoritmos relacionados. Despois de reseñar as súas características, tipo de operacións e complexidade do fluxo de programa, avaliáronse os diferentes paradigmas de computación. Partindo deste coñecemento, propúxose a arquitectura dun procesador masivamente paralelo avaliando a súa viabilidade con tecnoloxía actual. Este procesador incluía un gran número de unidades de cómputo, moi sinxelas, dispostas nunha matriz bidimensional con conexións locais entre eles para intercambiar datos. A pesar de que soamente procesaba imaxes binarias, os resultados amosan que as implementacións prácticas están limitadas a aplicacións que procesan imaxes de baixa resolución e onde o ancho de banda é o maior limitante do sistema. Unha versión posterior deste procesador, estendido para manexar imaxes en escala de grises e a cor, mellora os resultados reducindo o paralelismo pero mellorando as características das unidades de cómputo. Esta arquitectura, como a anterior, prototipouse nunha FPGA para validar as melloras engadidas. Para isto utilizouse un algoritmo que extrae de xeito automático a árbore de veas a partir de imaxes retinianas. Trátase dunha aplicación moi esixente e representativa dos algoritmos de visión temperá, etapa de baixo nivel que o procesador debe

ser capaz de abordar. A arquitectura proposta, unha matriz de procesadores de gran groso, explota o paralelismo de dato dos algoritmos, e foi comparada cunha CPU de propósito xeral, un procesador de plano focal cun esquema dun procesador por píxel, e unha matriz de procesadores masivamente paralela, que explota o paralelismo temporal.

Os resultados obtidos da comparativa das diferentes arquitecturas foi moi valiosa para espandir o rango de operación. Unha das conclusións obtidas é que unha distribución bidimensional das unidades de cómputo non proporciona a suficiente flexibilidade para abordar as etapas posteriores ao baixo nivel. Ademais, explotar o paralelismo a nivel de tarefa non é posible, algo moi importante en gran cantidade de algoritmos incluso aínda que a súa presencia sexa reducida, tal e como amosa a comparativa anterior. Estas conclusións leváronnos a deseñar unha nova arquitectura, baseada na anterior, onde os diferentes módulos permiten reconfigurarse baixo demanda e en tempo de execución.

As novas melloras introducidas aumentan a flexibilidade e o rendemento da arquitectura. Deste xeito, é posible configurar dous modos de execución, SIMD e MIMD. No primeiro modo, todos os procesadores executan a mesma instrución. As unidades de cómputo dispóñense nunha matriz unidimensional con conexións locais entre unidades adxacentes. Neste modo, o paralelismo de dato pode ser plenamente explotado empregando a memoria interna que cada unidade aritmética inclúe. Os operadores baseados en ventá, moi importantes nos primeiros pasos de case todos os algoritmos de Visión por Computador, poden implementarse facilmente almacenando varias filas da imaxe de forma adxacente. No modo MIMD, cada unidade de cómputo executa un pequeno programa, unha parte do algoritmo completo. Os diferentes módulos interconéctanse empregando unha rede local programable para reconstruír o fluxo de execución do algoritmo. Este modo explota o paralelismo temporal de forma nativo xa que todas as unidades están traballando de xeito concurrente. A rede autoxestiónase polo que non é preciso incluír control adicional, facilitando a migración de depuración do algoritmo. En ambos modos, dous procesadores independentes controlan a saída e entrada de datos entre as unidades de cómputo e a memoria externa. Isto permite superpoñer a transferencia de datos e o cómputo, incrementando o rendemento global do sistema.

A arquitectura final foi prototipada nunha FPGA e diversos algoritmos foron avaliados para determinar os beneficios que supoñen estas melloras. A pesar de que presenta capacidades de propósito xeral, os resultados mostran que é posible de conseguir un rendemento similar á arquitecturas máis específicas, como a presentada no capítulo segundo, sendo posible aumentar de xeito considerable a flexibilidade sen comprometer o rendemento.

Traballo futuro

A arquitectura proposta neste traballo proporciona unha base sólida sobre a cal é posible engadir novas características para aumentar a súa flexibilidade e rendemento, entre outras figuras de mérito. En particular e baseado nos resultados obtidos, unha arquitectura multi-núcleo proporcionará melloras importantes. Para executar simultaneamente ambos paradigmas de computación, SIMD e MIMD, unha interconexión interna para transferir os datos entre os diversos núcleos é máis interesante que aumentar o número de unidades de cómputo mantendo un so chip.

Máis aló de modificacións estruturais e implementacións baseadas en FPGAs, a arquitectura foi concibida para implementarse en ASICs. A arquitectura pode ser optimizada a un nivel máis profundo, especialmente o apartado referente ás unidades aritméticas, restrinxidas en capacidade debido á limitación dos recursos dispoñibles na FPGA. Isto abre novas direccións de investigación e permite avaliar de xeito máis preciso parámetros como o consumo de potencia ou os requisitos de área.

Introduction

Motivation and objectives

Computer Vision algorithms and techniques are increasingly robust and accurate, offering new possibilities for the treatment of visual information. However, this entails an increasing in the computing requirements, and their implementation on devices beyond the scope of the prototypes is becoming more complex. Novel approaches are required in order not to trim the characteristics of the algorithms and to meet the different trade-offs such as speed, latency, power consumption or cost. This includes both hardware architectures and software techniques to fully exploit the computing capabilities of the target device and to take advantage of the characteristics of the algorithms.

For a given Computer Vision problem there are many different techniques and approaches proposed in the literature. The variety of algorithms is very broad and they present very different characteristics, including not only data representation or arithmetic operations but also different computing paradigms and program complexity. This makes the design of the hardware devices very challenging if the same architecture has to face different algorithms with tight constraints.

Desktop computers are often employed to develop the algorithms. They offer large flexibility to run any algorithm and evaluate its feasibility and accuracy. However, the target device is application-dependent and supplementary hardware modules are frequently used to speed-up the computation. In order to meet the requirements of the application, a custom architecture may be a need. In this case, all figures of merit are likely to be optimized. There are many different fields where the different trade-offs are more or less important so the different architectures have to deal with different restrictions.

The advances in the semiconductor industry enable embedding more and more resources in the same unit of area while maintaining a restrained power consumption. This have made possible to develop new embedded processors with similar performance to recent-past conventional systems. This way, new applications are now technically feasible and the importance of the embedded vision market is increasing significantly. Custom systems-on-chip enable to embed on the same chip all the necessary modules for a given system, including computing, control and communication, providing a significant advantage over equivalent multi-chip systems. However, custom systems-on-chip require a long period of design and many resources, human and materials. In addition, custom hardware also requires a training period by the software engineers to fully exploit the capabilities of the hardware, reducing the competitive advantage of these systems. This becomes critical as the number of specific units grows to face the different algorithms employed to solve the Computer Vision problem.

The objective of this work is to provide a single-chip architecture able to run most of the Computer Vision algorithms with adaptive performance on the critical steps of the algorithms. In addition, it must be able to run all the other steps so that the overall performance is not compromised. The architecture also aims to reduce the time spent in migrating the algorithms, reducing the mapping stage by easing algorithm parallelization and synchronization between the different concurrent kernels of the algorithm. From the flexibility point of view, the system has to be modular and scalable in order to address different targets on the market and meet different trade-offs such as performance, area or cost. Architectural improvements and custom modifications must be done easily without dramatically change the way the hardware works. All these elements combined will provide enough flexibility and performance in order to run efficiently many different Computer Vision algorithms, with the benefit of employing a single device.

Contributions

The primary contributions of this dissertation are:

- An analysis of how representative and conceptually different hardware architectures implement diverse Computer Vision algorithms by taking advantage of their mathematical operations and dataflow.

- An architecture for low-level and early vision image processing which solve a number of technical problems found on existing devices, expanding its range of operation, flexibility, accuracy, cost and development times.
- A modular and scalable architecture for embedded Computer Vision systems, which focus on easing algorithm migration by natively parallelize the processing threads and data transfer. This architecture features general-purpose capabilities and dynamically reconfiguration at runtime to adapt the internal datapath according to the operations of the algorithm it is running.
- A demonstration and evaluation of the potential value of reconfigurable datapaths to face complex operations, beyond the Computer Vision field.

These contributions can be found in the publications listed below:

- **SIMD Array on FPGA for B/W Image Processing.** Alejandro Manuel Nieto Lareo, Víctor Manuel Brea Sánchez and David López Vilariño, in *11th International workshop on Cellular Neural Networks and their Applications (CNNA 2008).*
- **SIMD and Cellular Neural Networks as Fine-Grained Parallel Solutions for Early vision on FPGAs.** Alejandro Manuel Nieto Lareo, Natalia F., Jordi A.C., J. Riera, Víctor Manuel Brea Sánchez and David López Vilariño, in *23th Conference on Design of Circuits and Integrated Systems (DCIS 2008).*
- **A Digital Cellular-Based System for Retinal Vessel-Tree Extraction.** César R., Alejandro Manuel Nieto Lareo, Roberto O., Víctor Manuel Brea Sánchez and David López Vilariño, in *19th European Conference on Circuit Theory and Design (ECCTD 2009).*
- **On-Chip Retinal Image Processing: Performance Analysis on Different Approaches.** Alejandro Manuel Nieto Lareo, Roberto O., Víctor Manuel Brea Sánchez and David López Vilariño, in *24th Conference on Design of Circuits and Integrated Systems (DCIS 2009).*
- **An FPGA-based Topographic Computer for Binary Image Processing.** Alejandro Manuel Nieto Lareo, Víctor Manuel Brea Sánchez and David López Vilariño, in *Image Processing (In-Tech Education and Publishing, 2009).*

- **FPGA-Accelerated Retinal Vessel-Tree Extraction.** Alejandro Manuel Nieto Lareo, Víctor Manuel Brea Sánchez and David López Vilariño, in *19th International Conference on Field Programmable Logic and Applications (FPL 2009).*

- **Performance analysis of massively parallel hardware architectures for medical image processing.** Alejandro Manuel Nieto Lareo, Víctor Manuel Brea Sánchez and David López Vilariño, in *Eurasip Journal on Image and Video Processing - Special Issue on Real-Time Image Processing on Multi-Cores, Many-Cores and High-level FPGA-based Platforms (2011).*

- **Towards the optimal hardware architecture for Computer Vision.** Alejandro Manuel Nieto Lareo, Víctor Manuel Brea Sánchez and David López Vilariño, in *Machine Vision (In-Tech Education and Publishing, 2011).*

- **Feature detection and matching on an SIMD/MIMD hybrid embedded processor.** Alejandro Manuel Nieto Lareo, David López Vilariño and Víctor Manuel Brea Sánchez, in *8th IEEE Workshop on Embedded Vision (EVW 2012).*

- **SIMD/MIMD dynamically-reconfigurable architecture for high-performance embedded vision systems.** Alejandro Manuel Nieto Lareo, Víctor Manuel Brea Sánchez and David López Vilariño, in *23rd IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP 2012).*

The results of this work are also reflected in the following patent:

- **Arquitectura híbrida SIMD/MIMD dinámicamente reconfigurable de un coprocesador para sistemas de visión.** Alejandro Manuel Nieto Lareo, David López Vilariño and Víctor Manuel Brea Sánchez. *Patent Application Number P201101381, Spain (December 30th, 2011).*

Outline

This thesis is divided in three chapters. Chapter 1 introduces the challenges of Computer Vision. The different computing paradigms and hardware architectures and devices to face the algorithms requirements are also reviewed. In Chapter 2, a new hardware architecture for early vision processing is presented. After examining different approaches, a study of a

massively parallel processor for binary image processing is carried out. The conclusions of the study are essential for the design of a more efficient and capable architecture for general-purpose early vision processing, which is extensively tested employing a highly computationally consuming algorithm employed in medical applications. In Chapter 3, the results of the comparison of the general-purpose early vision processor are employed to define a new and enhanced architecture. This architecture is intended for embedded devices and focuses on dynamic reconfiguration to reproduce the algorithm's dataflow. This way, the architecture increases its range of operation so that all steps of a general Computer Vision algorithm can be performed efficiently by the embedded device. The results of implementing several common image processing algorithms and a comparison with similar approaches complete this chapter. Finally, the main conclusions of this thesis are conveyed.

Chapter 1

Background and related work

Computer Vision systems are experiencing a large increase in both range of applications and market sales [1]. From industry to entertainment, Computer Vision systems are becoming more and more important and the research community is making a big effort to make them able to handle complex scenes focusing on the accuracy and the robustness of the results. The new algorithms enable more advanced and comprehensive analysis of the images, expanding the set of tools to implement these applications [2].

Although there are new algorithms available to approach sophisticated new applications with a high degree of accuracy, not all the algorithms are adequate to be deployed in industrial systems. Parameters like power consumption, integration with other system modules, cost and performance limit the range of suitable platforms. In most cases, the algorithms must be adapted to achieve a trade-off solution and to take advantage of the selected platform.

Conventional systems like PCs or GPUs are increasingly improving performance and including more features but their use is limited to areas where portability, power consumption and integration are not critical. When the algorithm is highly complex, with an irregular execution flow, complex data representation and elaborated patterns to access to data, a significant gain is not achieved when moving the algorithm to an *ad hoc* hardware design. In this case, a high-end CPU and a GPU with support for GPGPU is a flexible and very powerful combination that will outperform other choices [3].

However, when a conventional system does not meet the requirements of the application, a more ambitious planning is needed. For instance, migrating the algorithm to a dedicated device such as a DSP, an FPGA or a custom chip [4]. At this point, the designers have to

consider alternatives as to reduce operating range, accuracy and robustness of the results, or to remove expensive operations in order to simplify the hardware that will be implemented [5]. Otherwise, a significant improvement will not be achieved. PC-based systems enable a great flexibility at cost of performance, so pure software-based algorithms hardly match pure hardware implementations. This is a serious limitation because it can compromise the efficiency of the application. This is the reason why the industry is making great efforts to develop novel architectures that enable greater flexibility to adapt the algorithms without compromising the quality of the results.

Computer Vision applications are often divided into several stages depending on the abstraction level and thus in the complexity of the operations. Initially, operations are quite simple and repetitive but applied over a large set of data with a very reduced data dependent program flow, so massive parallelism is essential for performance. Then, the level of abstraction increases, resulting in more complex algorithms. Temporal and task parallelism is key as the data set is smaller and the program flow is often data-dependent. Finally, the system output has to be determined, performing high level operations over complex data representations but reduced data sets and with higher precision as usual requirement. An efficient Computer Vision system must deal with all these stages. The selection of the computation model will determine the performance of the device in each one of the stages. According to Flynn's Taxonomy [6], paradigms can be classified into four categories based on the number of instructions and data streams processed simultaneously. So, which is the optimal paradigm for such applications? Increase the number of concurrent instructions? Increase the number of data elements processed simultaneously? A combination of both?

Taking into account the application requirements, a suitable platform to build the system must be selected. Besides general parameters (performance, cost and integration), there is a set of factors that restrict the computing paradigms and the devices that can be selected. For instance, a critical parameter is the way the data are transferred to the device because I/O operations are one of the bottlenecks in high-performance systems. Data type (integer, fixed-point or floating-point) and representation will affect the computational units. The program flow will constrain the inner connections between these units and the storage elements. To tackle these and other parameters, a careful analysis of the state-of-the-art of the algorithms has to be made.

This chapter addresses a review of different computing paradigms and platforms oriented to image processing. In addition, a representative set of Computer Vision algorithms covering

the three levels of processing is evaluated. This study will allow us to observe the algorithms based on a set of common characteristics: operations, data type, program flow, etc. This is critical to design new hardware architectures in order to maximize performance. The analysis from the hardware point of view will highlight the best features of the most used computing paradigms in order to establish a relationship between the type of operation, data, programming model and hardware architecture. An efficient architecture for Computer Vision must combine all the selected features. The analysis of the characteristics of the different algorithms will lead us to an optimized general-purpose hardware architecture for Computer Vision.

1.1 The challenges of Computer Vision

Traditionally, Computer Vision (CV) applications include building blocks from three computation levels: low-, mid- and high-level vision computing. The type of operations, the data representation and the flow execution of programs depend deeply on the considered level of this hierarchy. Nevertheless, current CV-algorithms are composed of many different processing steps regarding the type of data and the way these are computed, which makes difficult to classify them only in one subgroup. Following, a rather rough classification of widely used CV-algorithms is made, keeping in mind the data domain and the complexity of the involved operations.

1.1.1 Low-level vision

After image acquisition, some preprocessing steps are often required. These are intended to provide reliable input data for subsequent computing stages. Some typical operations are noise reduction, color balancing, geometrical transformation, etc. Most of these operations are based on point or near-neighborhood operations. Point operations are performed at pixel-level in such a way that the output only depends on the value of any individual pixels from one or several input images. With this type of operation it is possible to modify the pixel intensity to enhance parts of the image, by increasing contrast or brightness. Equally, simple pixel-to-pixel arithmetic and Boolean operations also enable the construction of operators as *alpha blending*, for image combination or color space conversion. Neighborhood operations take also into account the value of adjacent pixels. This operation type is the basis of filtering, binary morphology or geometric transformation. They are characterized by simple operations, typically combining weighted sums, Boolean and thresholding processing steps.

After preprocessing stages, useful information has to be extracted from the resulting images. Common operations are edge detection, feature extraction or image segmentation. Edges are usually defined as step discontinuities in the image signal so finding local maxima in the derivative of the image or zero-crossings in the second derivative are suitable to detect boundaries. Both tasks are usually performed by the convolution of the input image with spatial filtering masks that approximate a first or second derivative operator.

Feature points are widely used for subsequent computing steps in multiple CV-applications. Basically, a feature represents a point in the image which differs from its neighborhood. One of the benefits of local features is the robustness against occlusion and the ability to manage geometric deformations between images when dealing with viewpoint changes. In addition, they improve accuracy when, in the same scene, objects are at different planes, (i.e. at different scales). One of the most popular techniques is that proposed by Harris [7] to detect corners. It is widely used due to its strong invariance to rotation, image noise and no large illumination changes. It uses the local auto-correlation function, which describes the gradient distribution in a local neighborhood of each image point to detect the location of the corners. Using the locally averaged moment matrix from the image gradients, corners will be located at the maximum values. Another frequently used technique is the Scale-Invariant Feature Transform (SIFT) [8]. SIFT localizes extrema both in space and scale. Using the *Difference of Gaussians* as scale-space function, the images are filtered with Gaussian kernels of different sizes (scales). This is performed for different image sizes (octaves). The response of each filter is subtracted from the immediately following in the same octave. The interest points are scale-space extrema so local maxima and minima are extracted by comparing the neighborhood points in the same, the previous and the subsequent scales. To improve accuracy, a sub-pixel approximation step is done, interpolating the location of the feature inside the scale-space structure. The number of octaves and scales can be tuned to meet the system requirements. SIFT provides invariance against scale, orientation and affine distortion, as well as partial occlusion and illumination changes. Other algorithms were proposed to improve accuracy or performance like, the *Speeded Up Robust Features* (SURF) [9] or the *Gradient Location and Orientation Histogram* (GLOH) [10]. This kind of detectors are quite complex and their performance can be low even using custom hardware. For this reason, less reliable algorithms are still in use, as Harris Corner Detector, FAST [11] or the Smallest Univalue Segment Assimilating Nucleus (SUSAN) [12] corner detectors. In this sense, techniques such as [13] provide a framework for very fast feature detection and matching.

Segmentation refers to the process of separating the data into several sets according to certain characteristics. There are several techniques to carry out this task, either based on boundaries or regions [14] [15]. Nevertheless, most of them rely on near-neighborhood operations. Particular attention deserves the clustering methods like the popular k-means which partitions the data set into several clusters according to a proximity criterion defined by a distance function. These methods are not restricted to image data. N-dimensional sets of abstract data can also be partitioned. Furthermore, information about the scene or domain can be introduced (number and characteristics of the target clusters). Therefore, they might be classified either as a low or mid-level vision stage.

1.1.2 Mid-level vision

Mid-level CV stages usually operate on images from previous processing steps, often binary images, and produce a lower amount of data but with a higher concentration of information. Some common operations are object classification and scene reconstruction.

One of the goals of Computer Vision is to recognize objects in a scene. Based on object location, pose or 2D/3D spatial relations between the objects, the algorithms have to be able to analyze the scene and its content. This involves issues such as dealing with object models, classifiers and the ability to integrate new information in the models. In the literature, a large amount of techniques can be found, usually classified as global methods, more intended for object detection and local feature-based methods for object recognition. In all of them good image registration is essential for both accuracy and performance [16]. As for the global methods, common techniques are based on *template-based matching*, which employs a convolution mask or template to measure the similarity between an object patch and the template. In this sense, *normalized cross-correlation* (NCC), *sum of squared differences* (SSD) or *sum of absolute differences* (SAD) are widely used. As for local methods, local feature descriptors play an important role. Roughly speaking, a descriptor is an abstract characterization of a feature point based on its environment. One of the most popular techniques is the proposed in second part of the SIFT algorithm based on stacked orientation histograms which associate a high dimension vector to each keypoint. In order to reduce the amount of false positives and negatives during the matching stage the search area is limited by using strategies like the *nearest neighbor search* (NNS) which attempts to find the nearest points of a given one in a vector space. An *indexing structure* allows to search for features near a given feature rapidly.

This is the case of the *K-dimensional trees*, which organize points in a k-dimensional space in such a way that each node has at most two child nodes.

Scene reconstruction consists of the generation of scene models starting from their components. There are different techniques to reconstruct one or several objects in a scene. To build a 3D model, coordinates of scene points have to be calculated from the objects. If the location of the camera is known, 3D coordinates of a scene point can be determined from its projection on image planes of different viewpoints. The whole process starts with feature extraction and matching. Using geometric consistency tests it is possible to eliminate wrong matches. There are different solutions to estimate the fundamental matrix, as the *RANdom SAmple Consensus* (RANSAC). Once the matches between images are consistent, camera pose and scene geometry is reconstructed using *Structure from Motion* methods and refined with *Bundle adjustment* techniques [17].

1.1.3 High-level vision

The high-level stage often starts from an abstract representation of the information. This stage is highly application-dependent, but due to the variety of operations, data structures, memory access patterns and program flow characteristics are often only compatible with a general purpose processor. High-level processing is characterized by the use of a small set of data to represent knowledge about the application domain. More complex data structures are needed to store and process this information efficiently, making the operations and the memory access patterns more elaborated. This, together with the inherent complexity of decision making, makes the program flow very variable.

Robust pattern recognition, object identification, complex decision making or system adaptation are some of the benefits of integrating Artificial Intelligence methods with Computer Vision. Otherwise, the system would be limited to a predetermined set of actions. Machine learning makes computers capable of improving automatically with experience. This way it is possible to generalize the behavior from unstructured information with techniques such as neural networks, decision trees, genetic algorithms, regression models or support vector machines. Machine learning has emerged as a key component of intelligent computer vision systems, contributing to a better understanding of complex images [18] [19].

Data mining is the process of analyzing data using a set of statistical techniques in order to summarize into segments of useful information. This makes possible analyze data from different dimensions or angles, categorizing and summarizing the identified relationships, making

evident hidden relationships or patterns between events. Contrary to Machine learning, data mining focuses on discovering hidden patterns instead of generalizing known patterns using the new data.

It is very difficult to establish a classification of tasks and operations for high-level processing. Some of the performed tasks fall within the scope of the measurement of application specific parameters such as size and pose of objects, fault detection and monitoring specific events such as traffic situations, for example. The algorithms and technologies are very diverse and most of them lie in statistical analysis and artificial intelligence domains.

As it was previously mentioned some tasks that initially fit into the low or medium level stages due to its context actually are more like high-level operations by the type of operations performed. This is related with the commonly used bottom-up image analysis, which starts from raw data to extend the knowledge of the scene. However, new approaches include feedback to perform top-down analysis. This way, low- and mid-level stages can be controlled with general knowledge of the image, improving the results.

1.2 Computing platforms

Given the wide range of algorithms and applications of Computer Vision, it is clear that it does not exist a unique computing paradigm or an optimal hardware platform. The type of operations, the complexity of data structures and especially the data access patterns greatly determine parameters such as the range of application, performance, power consumption or cost. This section presents some of the most prominent platforms in image processing, focusing on their strengths and weaknesses.

1.2.1 Computing paradigms

The Flynn's taxonomy [6] classifies the computer architectures in four big groups according to the number of concurrent instructions and data sets processed. Image processing tasks perform more or less efficiently depending on the selected paradigm. In order to develop a Computer Vision application it is crucial to exploit the spatial (data) or temporal (task) parallelism to meet trade-offs among performance, power consumption or cost.

SISD

Single Instruction Single Data (SISD) refers to the conventional computing model. A single processing unit executes a sequence of instructions on a unique data stream. Most modern computers are placed under this category and, although only one processor and one memory element are present, those which are able to pipeline their data-path are generally classified under the SISD category as they are still *serial computers*.

This paradigm performs better when spatial and temporal parallelism are hard to exploit. As seen previously, high-level image processing fits the SISD paradigm because most tasks are sequential, with a complex program flow and strong dependences between data. As processing is done sequentially, most optimizations aims to enhance the access between the memory and the arithmetic unit. Memory and processor speed are the main constraints. Furthermore, some kind of parallelism can be exploited when pipelining the data-path. Data allocation, pre-fetching and reducing stalls in the pipeline are some of the possible optimizations.

SIMD

SIMD (*Single Instruction Multiple Data*) computers have an unique control unit and multiple processing units. This control unit sends the same instruction to all processing units, which operates over different data streams. This paradigm focuses on exploiting the spatial parallelism. It is also possible to pipeline the data-path or to employ several memories to store the data in order to increase the bandwidth. SIMD computers are commonly specific-purpose, intended to speed-up certain critical tasks.

One of the drawbacks of SIMD machines is data transference. A network is required to both supply data to each processing unit and share data among them. Its size grows with the number of connected nodes so SIMD architectures have a practical limitation. Another restriction is data alignment when gathering and scattering data into SIMD units. This results in a reduction of flexibility in practical implementations. It is needed to determine the correct memory addresses and reordering data adequately, affecting performance. In addition, as this paradigm exploits spatial parallelism, program flow is heavily limited because all units execute the same instruction. Additional operations are needed to enable at least simple flow control tasks.

Low-level image processing benefits greatly of SIMD units. As it was described previously, most operations are quite simple but repetitive over the whole set of data. In addition,

certain tasks of mid and high-level can also take advantage of SIMD units when using in conjunction with others paradigms. The simplicity of the arithmetic units and the memory access patterns make feasible to design efficient units, which is crucial to increase the parallelism. Memory bandwidth and data distribution among the processors is also key for performance.

Two types of SIMD accelerators can be distinguished based on the number of processing units: fine-grain and coarse-grain processors, where the major difference is the number of processing units. While the first includes a large amount of very simple processors with a rigid network, the second features major flexibility although with a much lower parallelism. When using in low-level image processing, fine-grain processor arrays match with massively parallel operations such as filters or morphological operations. Using a processor-per-pixel scheme and local communications, neighborhood operations are completed in just a few instructions. On the contrary, when the parallelism level is lower a configuration as *vector processor* is usually preferable. By reducing communications and increasing core complexity, they are much more flexible and efficient not only for low-level operations but also for other processing stages.

MIMD

Multiple Instruction Multiple Data (MIMD) refers to architectures where several data streams are processed using multiple instruction streams. MIMD architectures have several processing units executing different instructions to exploit task parallelism. Processors perform independently and asynchronously. MIMD systems are classified depending on the memory architecture.

In *Shared Memory Systems*, all processors have access to an unique memory. Connection hierarchy and latencies are the same for all processors. This scheme eases data transference among processors although simultaneous access must be taken into account to avoid data hazards. Scalability is also reduced because it is hard to increase the memory bandwidth at the same rate as the number of processors.

If each processor has its own and private memory it is possible to upscale more easily as memory and processors are regarded as a unit. This scheme is known as *Distributed Memory System*. In addition, local memory access is usually faster. The major disadvantage is the access to data which are located outside the private memory because dedicated buses and a *message passing system* to communicate with the processors are needed. This can result in high access times and an increase of hardware requirements.

In a *Distributed Shared Memory System*, the processors have access to a common shared memory but without a shared channel. Each processor is provided with local memory which is interconnected with other processors through a high-speed channel. All processors can access to different banks a global address space. Access to memory is done under the schemes as *Non-Uniform Memory Access* (NUMA), which takes less time to access the local memory than to access the remote memory of other processor. This way scalability is not compromised.

Very long Instruction Word (VLIW) and *superscalar* architectures are also classified within MIMD paradigm because they exploit instruction-level parallelism, executing multiple instructions in parallel. *Pipelining* also executes multiple instructions but splitting them in independent steps to keep all the units of the processor working at a time.

Mid-level image processing and some operations of the other processing levels of Computer Vision can exploit MIMD processors. Operations are relatively simple, with data-dependent program flow. Temporal and task parallelism are easier to exploit than spatial parallelism although a reduced degree is usually present in this type of algorithms. Each MIMD processing element can include SIMD units. This way it is possible to process complex tasks more efficiently, from kernel operations as when pre-processing images concurrently in multi-view vision systems to high level tasks as multiple object recognition and tracking. In general, any set of tasks with weak dependences between them to reduce internal communications can take advantage of this paradigm.

MISD

There is one more paradigm, *Multiple Instruction Single Data* (MISD), which achieves higher parallelism than SISD executing different instructions over the same data set employing several computing units.

Systolic arrays are regular n-dimensional arrays of simple cores with nearest-neighbors interconnections. Each core operates on the input data and shares the result to its neighbor, flowing the data synchronously usually with different flow in different directions. They are employed for tasks such as image filtering or matrix multiplication. Pipelined architectures belong to this type, as they are considered one-dimensional systolic arrays, but they are commonly considered an improved version of the other aforementioned paradigms.

This paradigm is rarely used for Computer Vision as the others paradigms match better and offer higher performance and flexibility when dealing with real Computer Vision problems.

Summary

Low-level Computer Vision entails the largest processing times in most applications. Data sets are usually very large and the kind of operations simple and repetitive. Operations are inherently massively parallel and the data access patterns are regular. It is feasible to exploit these features to design very optimized SIMD custom hardware accelerators or to migrate the algorithms to existing hardware.

It is harder to extract parallelism in the mid-level stage because operations involve more complex data-flow. In addition, the data set is smaller so the benefits of including dedicated units to speed-up the computation are lower than expected. Despite this, hybrid processors (SIMD-MIMD) able to exploit both spatial and temporal parallelism can overcome this limitation.

Finally, the amount of data involved in the high-level stage is usually small so it is rarely necessary to sacrifice precision in order to get better performance. Moreover, unlike in previous stages, the disparity in the type of data makes the use of floating-point often a requirement. Another characteristic of this stage is the program flow, far more complex, which may even consume more computation time than the arithmetic operations. The kind of computation performed at this stage is so varied that the best option is often a general purpose SISD processor.

1.2.2 Current devices

There are different possibilities to implement the aforementioned computation paradigms. There is not an unique and direct correspondence between a paradigm and its hardware implementation. On the one hand, it can be designed a dedicated hardware which follows the original conception. On the other hand, the paradigm can be emulated both in hardware or software.

Microprocessors

Microprocessors, SISD machines, are the most straightforward devices to develop a Computer Vision application. Their main advantage is their versatility, the ability to perform very different tasks for a low cost. They can perform any type of data processing, although its efficiency, measured in parameters such as cost, power consumption or integration capabilities, is not always optimal because of their general-purpose condition. The large variety of available technologies, libraries, support and programs cut down the *cold start*, enabling to

get the system ready for development in a short time. Developers can focus on the problem itself instead of technical issues [20].

Basically, they are composed of a main memory and a processing unit which includes the arithmetic and the control modules. From this basic structure more optimized microprocessors can be designed. From caches to cut down the memory access times, tightly coupled high-speed memory controllers or specialized units for critical tasks, the variability of architectures is as large as the amount of fields in the market [21]. However, despite the evolution of the industry pure SISD microprocessors do not offer adequate performance for a large set of tasks. That is why a wide range of accelerator modules have been included, as specific-purpose arithmetic units and sets of instructions or co-processors. The inclusion of SIMD units is decisive for tasks such as video encoding and decoding, but any data-intensive algorithm can take advantage of them [22].

As it will be discussed later, the advances in the semiconductor industry allows to increase the integration density so it is possible to include more processing power on the same Silicon area. This has led to abandon the race for speed (to increase the working frequency) to more efficient systems where energy consumption is vital and parallelism is the way to overcome the limitations of Moore's Law [23]. Most of modern processors are multi-core and the number of cores is expected to grow in the near future. Programming languages and techniques as well as image processing algorithms have to be adapted to this new reality [24].

Microprocessors are employed in a wide range of applications, from developing and testing algorithms such as autonomous driving [25] to final platforms as medical image reconstruction [26]. Even its use in restrictive stand-alone devices is also viable such as autonomous flight [27]. Microprocessors stand out in high-level tasks, as the latest stages of image retrieval [28] and scene understanding [29], where handling image databases, storing and communicating data are fairly complex to implement them on specific purpose devices. Video surveillance tasks can take advantage of these features for image processing [30] and event control in complex distributed systems [31].

Mobile processors have become a benchmark in innovation and development after the explosion of the mobile market. As discussed below, they integrate several general purpose cores, graphics processing units and other co-processors on a single chip keeping power consumption very low. The applications they can address are increasingly complex [32] [33] [34].

Graphics Processing Units

A *Graphics Processing Unit* (GPU) is a specialized co-processor for graphic processing to reduce the workload of the main microprocessor in PCs. They implement highly optimized graphic operations or *primitives*. Current GPUs provide a high processing power and exploit the massively spatial parallelism of these operations. Because of their specialization, they can perform operations faster than a modern microprocessor even at lower clock rates. GPUs have hundreds of independent processing units working on floating point data. Memory access is critical to avoid processing downtimes, both in bandwidth and speed.

Their high processing power makes GPUs an attractive device for non-related graphic tasks. *General Purpose GPU* (GPGPU) is a technique to perform general computation not related to graphics on these devices [35] [36]. This makes possible to use its specialized and limited pipeline to perform complex operations over complex data types. In addition, it eases memory management and data access. Flow control, as looping or branching, is restricted as in other SIMD processors. Modern GPUs architectures as [37] or [38] have added support for these operations, although slightly penalizing throughput. Word size is also a limitation in GPGPU techniques. It was reduced to increase the integration density as graphic operations do not usually require high precision. However, since this was a serious limitation for scientific applications, large word-sizes support was added later [39].

GPUs are effective when using *stream processing*, a paradigm related to SIMD [40]. A set of operations (*kernel*) is applied to each element of a set of data (*stream*). The flexibility is reduced to increase the parallelism and to lower the communication requirements when involving hundreds of processing elements. Otherwise, providing data to hundreds of processors would be a bottleneck. Processors are usually pipelined in a way that results pass from one arithmetic unit to the next one. This way, the locality and concurrency are better exploited, reducing communication requirements because most of the data are stored on-chip.

The use of GPUs to speed-up the computing has greatly increased, specially after the optimization of libraries and functions that mask low-level technical difficulties. Most image processing kernels and algorithms were adapted to work in GPGPUs, obtaining significant improvements. Basic image processing [41], FFT transforms [42], feature extraction [43] or stereo-vision [44], all of them computationally expensive, benefit greatly of the massively parallelism of GPU. They can be used also to emulate other computing paradigms frequently used in low level vision [45]. However, some authors argue that the gap between GPUs and CPUs is not as large as it seems if key optimizations are carried out [46].

Digital Signal Processors

A *Digital Signal Processor* (DSP) is a microprocessor-based system with a set of instructions and hardware optimized for intensive data applications. They are specially useful for real-time processing of analog signals but they offer a high throughput in any data intensive application. DSP market is well established and offers a large range of devices, optimized for each particular task [47]. Apart from attached processors, to assist a general purpose host microprocessor, DSPs are often used in embedded systems, including all necessary elements and software.

They are able to exploit parallelism both in instruction execution and data processing. In a von Neumann architecture, instruction and data share the same memory space. However, DSP applications usually require several memory accesses to read and write data per instruction. To exploit concurrency, many DSPs are based on a Harvard architecture, with separate memories for data and instructions. Many modern devices are based on *Very Long Instruction Word* (VLIW) architectures, so they are able to execute several instructions simultaneously [48]. Compilers are fundamental to find the parallelism in the instructions, and a large improvement can be obtained after an efficient placement of data and programs in memory. Superscalar processors and pipelined data-paths also improve the overall performance, although this is done by hardware. They include specialized hardware for intense calculation, such as *multiply-accumulate* operation, which is able to produce a result in one clock cycle. Although many DSPs have floating-point arithmetic units, fixed-point units fit better in battery-power devices. Formerly, floating-point units were slower and more expensive but this gap is getting smaller and smaller. They also include zero-overhead looping, rounding and saturated arithmetic or dedicated units for address management [49, 50, 51].

DSPs are usually designed for intensive data processing so performance can be penalized in mixed tasks. However, current all-in-one devices are able to handle complete applications efficiently. In addition, DSPs are not only available as independent devices, but also as part of integrated circuits such as FPGAs or SoCs.

DSPs have a large tradition on image processing tasks. As optimized versions of conventional processors, they were used to accelerate the most expensive operations. These operations were related mainly with low-level image processing [52], where parallelism and data access enable a large performance increase. Stereo vision [53], Fourier transform [54] or video matching and tracking [55] are some samples. However, higher level algorithms also

suit for DSPs, specially in industrial tasks [56] [57]. Nowadays, DSPs are able to handle large sets of operations efficiently both for co-processing [58] or standalone [59].

Field Programmable Gate Arrays

A *Field Programmable Gate Array* (FPGA) is a device with user-programmable hardware logic. It is made of a large set of *logic cells* connected together through a network. Both elements are programmable in such a way that the logic cells emulate combinational functions and the network permits to join them to build more complex functions. In addition, the *program* can be rewritten as many times as needed.

The main advantage of FPGAs is their high density of interconnections between cells, which provides a very high flexibility. This network has a complex hierarchy with optimizations for specific functions. It provides specialized lines to propagate clock or reset signals across all the FPGA or to build buses with high *fan-out* within acceptable time delays. With these very basic elements it is possible to build highly complex modules as arithmetic units, controllers or even embedded microprocessors. Large memory elements, DSP arithmetic units, networking and memory controllers or even embedded microprocessors are available on modern FPGAs [60].

These devices are an excellent mechanism to build proof-of-concept prototypes. On the one hand due to their flexibility any computing paradigm can be implemented, restricted mainly for the number of available cells. On the other hand, thanks to the re-programmability it is possible to debug and test on real hardware. Even more, nowadays some final products are implemented exclusively on FPGAs, instead of migrating the design to custom integrated circuits. The achieved performance can be tens of times higher with lower power consumption than standard PC-based approaches [61]. FPGAs are widely employed as co-processors in personal computers such as GPUs or as accelerators in specific purpose devices as high-capacity network systems [62] or high-performance computing [63]. Nowadays, it is possible to embed full systems on a single FPGA.

One of the major disadvantages of FPGAs is the set-up time, still higher than in pure-software approaches. Traditional FPGA programming is done with HDL languages, forcing to design at a very low level. High-level languages, such as C extensions, are more friendly for software engineers, although the control over the design is much lower [64]. The FPGA-based designs can be exported and distributed as *IP Cores* because these languages are platform-independent, unless very specific features of a given FPGA are employed. This way, *non-*

recurring engineering costs (NRE) are cut-down. Therefore FPGAs are in an intermediate stage between software and hardware. Algorithms are programmed by software and *compiled* to a hardware architecture, so a careful hardware/software codesign is fundamental [65]. They are able to exploit both spatial and temporal parallelism very efficiently. Since logic cells are independent many arithmetic units can process concurrently, with custom routing between them. In addition, the memory subsystem can be tuned to exploit the on-chip memory banks, reducing the access to the external memories.

Regarding Computer Vision, FPGAs are widely used both in industry and research. They offer a high degree of flexibility and performance to handle many different applications. Most compute-intensive algorithms were migrated to FPGAs: stereo vision [66], geometric algebra [67], optical flow [68], object recognition [69] or video surveillance [70] [71] to name some examples. Low and mid-level image processing stages, based on SIMD/MIMD paradigms can be efficiently implemented. However, high-level processing, although it could be possible to implement on an FPGA, fits better on conventional processors. Nevertheless, FPGAs can use external processors connected through high-speed off-chip communications or include general-purpose processors. There are available soft-core (emulated) [72] and hard-core [73] embedded processors. While the first offers a large degree of configuration, adapting all parameters of the design to the particular needs of the applications, the last features better performance. This way FPGAs are able to implement all the stages of a complete application [74].

Application-specific integrated circuits

An *Application-Specific Integrated Circuit* (ASIC) is a device designed for a particular task instead of for general purpose functionality. In this case, designers have to work at the very bottom level of design, so this process is long and error-prone. As there are elements present in almost all ICs, a set of *libraries* is usually provided making easier system-design. This way it is possible to work at different levels of abstraction. *Full Custom* designs require a greater effort because it is necessary to design both functionality and physical layout. However, it allows better optimizations and performance. *Standard Cells* allow to focus on the logic operation instead on the physical design, splitting the process into two parts. Physical design is usually done by the manufacturer, who provides from simple logic gates to more complex units such as *Flip-Flops* or adders. Apart from simple units, third party manufacturers provide more complex modules for specific functions, known as *IP Cores*. This way, they can be used

as subcomponents in a large design. There is a large variety of cores to tackle all needs, as it happens with the FPGAs, and there is available from IO controllers (RAM, PCI-Express, ethernet) to arithmetic cores (signal processing, video and audio decoding) or even complete microprocessors.

The IC-level design allows to build embedded systems, *System-on-Chip*, efficiently. It is possible to include all the elements of the system on a single chip, even if there are digital, analog and mixed-signal modules. Nowadays, custom ASICs are the unique alternative for complex SoCs although FPGAs size grows with each generation and are becoming a viable alternative. IC design leads to high costs, both in design and manufacturing when production volume is low. However, some applications still need ICs because the alternatives do not match with performance, power consumption or size [75].

It is true that some of the devices previously mentioned are in some way ASICs. However, the design of a custom architecture for a concrete algorithm provides the best possible results. Many custom chips were designed and built instead of mapping them in a programmable device, for specific algorithms [76], domain applications [77] or complete general purpose SoCs [78].

Because of this flexibility a large set of *exotic* devices can be found in the market or in the specialized literature. While some aim to address very specific tasks or novel computing paradigms, others are taking their first steps on the market after that technology has allowed its viability. *Pixel-parallel Processor Arrays* are the natural platform for low-level vision. They are massively parallel SIMD processors laid down on a 2D grid with a processor-per-pixel correspondence and local connections among neighbors. Each processor, very simple, can also include an image sensor to eliminate the IO bottleneck. Some representative examples are [79, 80, 81]. There are also approaches closer to the biological vision, as [82] or [83]. More information is available in [84]. *Massively Parallel Processor Arrays* (MPPAs) provide hundreds to thousands of processors. They are encapsulated and have their own program and memories. They work in MIMD mode but also include internal improvements as pipelines, superscalar capabilities or SIMD units. Some examples are [85, 86, 87].

1.2.3 Discussion

As described previously in this section, there is a wide range of hardware devices suitable for Computer Vision. Depending on the application requirements, a compromise between performance, cost, power consumption and development time is needed. Commercial applications

are heavily constrained by the time-to-market, so suboptimal solutions are preferable if the development cycle is shorter. This way, software-based solutions are usually better from the commercial point of view.

As discussed before, the large amount of highly optimized libraries make PCs (conventional microprocessors) the first choice both as development and production platform. Multicore and SIMD programming are key for performance, although this can significantly increase the development time. One of the benefits of choosing a PC as platform is that a GPU is included *"at no cost"*. This is, most applications require some kind of graphical display so including a GP-capable GPU provides a much greater benefit with a very low cost/performance ratio, even using a low-cost card. The combined use of CPU-GPU has proved to be very effective, although very restricted in terms of form factor and specially in power consumption. Only if these parameters are very constrained DSP-based solutions are preferable. This is the case of mobile applications, although low-power microprocessors are taking advantage in this field. As these are extensively used, development kits, compilers and libraries are very optimized, helping to cut down time-to-market and related costs.

However, if the aforementioned devices do not provide acceptable results, it will be required to go into the hardware. As application developers, we need to search for *exotic* devices such as MPPAs or dedicated image processors. In contrast to the previous devices, these are not normally industry standards, therefore a greater effort during development is needed. However, we are still under the software coverage. On the contrary, if the requirements are very strict or if the production volume is very high, a custom chip is the unique alternative to reach the desired performance or to lower the cost per unit. FPGAs are an excellent platform for testing before manufacturing the final design on a custom chip. As they are reconfigurable, different architectures can be evaluated before sending to the foundry. On the other hand, some authors claim that software development is the bottleneck in the current ASIC development. The *software stage* can not start until a device where to do tests has been built. Although emulators (both functional and cycle-accurate) are used, their performance is very low, resulting in very large test cycles and poor feedback for those programmers at the hardware control layer. This is why FPGAs are widely used as proof-of-concept devices, as they enable software development cycles many months before a test chip is built.

Table 1.1 shows performance comparison of a computationally intensive algorithm as SURF [9] for different platforms. PCs offer uneven performance if heavily use multithreading programming [88] or a straightforward implementation [89]. GPUs feature very large

performance at the expense of high power consumption. However, FPGA-based implementation delivers the best performance in terms of speed and power consumption. Table 1.2 shows a comparison between a low-power CPU and GPU, compared to a standard laptop CPU. Authors conclude that optimization is critical and a carefully analysis of low-level operations must be performed, but the achieved performance is quite close to standard conventional processors. Complex algorithms which include a higher abstraction level as Viola-Jones detector [18] are also candidate for mobile platforms. In [90], a Beagleboard xM board with a Texas Instruments DM3730 SoC (ARM Cortex-A8 and TMS320C64X DSP) achieves around $0.5x$ speed-up compared with a conventional Intel 2.2 GHz processor, both using openCV [20] library. In [91], a DSP-based embedded system for object recognition achieves up to 4 fps, including SIFT-based feature detection and description and object recognition. Although with lower performance than other approaches, the major advantage is that the whole application fits in a single device, reducing also power consumption.

	Device	**Performance**	**Power (W)**
[89]	Intel Core 2 Duo 2.4 GHz	$<$ 7 fps	N/S
[88]	Intel Core 2 Duo P8600 2.4 GHz	33 fps	25
[92]	nVidia GeForce 880 GTX	56 fps	200
[89]	Xilinx Virtex 5XC5VFX130T	70 fps	$<$ 20

Table 1.1: Summary of different SURF [9] implementations on different platforms for images of 640×480 px.

Device	**Un-optimized**	**Optimized**
Intel Core 2 Duo Merom 2.4GHz	9.03 fps	16.37 fps
Intel Atom 1.6GHz	2.59 fps	5.48 fps
GMA X3100 GPU 500MHz	1.04 fps	5.75 fps

Table 1.2: Performance of SIFT [8] implementations in low-power devices for images of 640×480 px. See [93] for details.

Some operations, as the Fourier transform, are computationally very expensive and yet required in many applications, including low-power, low-cost or high performance devices. Optimized libraries for both CPU [94] and GPU [95] aim to exploit SIMD units and multitasking, achieving high performance with regard to straightforward implementations. In particular, Fourier transform operation is very suitable for FPGAs and ASICs if performance and

power consumption is critical. In [96], an FFT core design for FPGAs is proposed, consuming less than 1 W in the worst case, and lowering manufacturing costs more than $x15$ compared with a DSP implementation. In [97], a more aggressive approach is done, developing an *application-specific instruction set processor* (ASIP). With very little hardware overhead and a consumption of few tenths of a watt, outperforms standard software and DSP implementations more than $x800$ and $x5$ times respectively. However, these designs have larger development cycles, as [98] depicts. In this work, some low-level operations (phase-based optical flow, stereo and local image features) are compared both on FPGA and GPU. Table 1.3 summarizes some of the results of this work. As authors conclude, high-performance or low-cost implementations should be done on CPUs with GPU co-processing. GPUs overcome FPGAs in terms of absolute performance due to their memory throughput. However, if a standalone platform is needed, an FPGA board should meet the requirements or establish the basis for testing and validating an ASIC design.

Device	**Power (W)**	**Cost ($)**	**Time-to-Market (months)**
nVidia GeForce GTX 280	236	N/S	2 (1 persons)
nVidia GeForce GTX 580	244	499	2 (1 persons)
Xilinx Virtex4 xc4vfx100	7.2	2084	15 (2 persons)
Xilinx Virtex5 xc5vlx330t	5.5	12651	12 (2 persons)

Table 1.3: Main GPU and FPGA costs for optical flow, stereo and local image features implementation. See [98] for complete details and performance results.

Finally, general-purpose custom designs converge form factor, power consumption and performance. For instance, SCAMP processor [99] exploits the massively spatial parallelism of low-level operations, integrating processing units and sensors in a processor-per-pixel fashion. As it is an analog design, the integration density and the performance is very high, keeping power consumption under 240 mW. Current digital solutions also offer similar performance and many advantages as faster development and array scalability. ASPA processor [100] includes novel techniques to increase performance, specially on global operations, without sacrificing other parameters. Hardware-oriented algorithms are also key to take advantage of custom hardware. Nevertheless, this kind of solutions, as other custom designs depicted in this chapter, are not suitable to handle a whole Computer Vision application as they are intended to reduce the workload of the main processor in the more computational expensive

tasks, such as the low-level image processing. Approaches as [101] or [102] can completely embed highly complex applications without compromising its efficiency.

Looking ahead

The progress of new technologies, marked by Moore's Law allows increasingly integration density. More hardware resources, with higher clock frequencies, are available for the designer. However, although the ultimate goal is to increase the performance, other parameters come into play. Nowadays, one of the critical trade-offs is power consumption, directly related with energy efficiency and power dissipation, some of the most decisive limitation design constraints [103].

Power consumption is driven by two sources, *dynamic* and *static*. Static consumption is a result of the leakage current and it refers when all inputs are held, so the circuit is not changing state. On the contrary, the dynamic term refers to the circuit switching at a given frequency. This power is dominated in today CMOS circuits, being directly proportional to frequency. This is one of the capital reasons why the semiconductor industry moves from a *race for frequency* to a *race for parallelism*. In recent years, the industry is making a big effort to increase the parallelism of most devices to keep the performance increase rate. Apart from more arithmetic units, leading architectures integrate more systems previously contained on separated circuits, as microcontrollers or GPUs. To achieve these results, it is still necessary to scale down the transistors. In this sense, the advent of emerging technologies like CMOS-3D [104] will permit to integrate heterogeneous functions on the same monolithic solution more easily. A vision-oriented ASIC could integrate the image acquisition stage to an eventual processor. At the same time, more conventional solutions as PCs or FPGAs would yield large parallelism using this and other advances such as the Tri-Gate technology [105]. However, this involves problems as the increment of leakage currents, thereby increasing static power consumption, not negligible at all nowadays [106] [103]. In addition, new manufacturing methods are more expensive because the yield is lower and more time is needed to compensate for the investments. Or equivalently, it is necessary to sell more devices to continue growing at the rate set by Moore's Law.

Conventional microprocessors are in the leading edge of evolution. There is a large market which justifies large investments in R&D to meet the growing needs of consumers, especially by large increase in media consumption. This way, it is now possible to find low-cost multicore microprocessors. It is expected that the current evolution towards a greater number of

cores will be maintained but increasingly including more elements previously located on external chips, reducing the bottleneck when communicating with off-chip elements [21]. New parallel computing techniques need to be developed to take advantage of the available multi-threading capabilities.

PCs also benefit of GPU capabilities. GPU performance grows at a higher rate than microprocessors. As they are very specialized devices, although featuring general purpose computing, the technical improvements in the semiconductor industry are clearly more beneficial. As discussed previously, there are available hardware resources to increase the parallelism and enhance the datapath pipeline. Leading GPUs have more than 1000 processing units and high speed and bandwidth memories. It is also possible to combine multi-core GPUs to work together, achieving a very large throughput. Still, their major disadvantage is being the power consumption. New architectures are taking advantage of the fixed-function hardware to improve area usage and power efficiency. GPU design will focus entirely on improving GPGPU computing [107].

DSPs are also moving to multicore architectures. As specialized microprocessors, they can take advantage of all the improvements in the consumer market, both in hardware and software improvements such as compilers or other optimization techniques. Although competitors are strong, DSP will continue to be used because they lead to compact circuit boards, lower power consumption and cost, if the appropiate device is selected based on the application requirements. In addition, they benefit of the extensive experience in DSP development, with shorter time-to-market thanks to the very optimized compilers and libraries. This is specially relevant in embedded applications, to take advantage of the multi-core capabilities of modern DSPs. This way it is possible to integrate several DSP cores, each one optimized for a specific task, on a single chip [108]. Low-power devices which still keep reasonable performance are fundamental in hand-held and portable devices, where traditional microprocessors are not suitable.

Microprocessors and GPUs tend to converge on a single chip. Apart from the obvious benefits of integration, reducing cost, size, power consumption, the performance will increase because the reduction of off-chip communications. In addition, architectures as AMD Fusion integrate in the same units 3D acceleration, parallel processing and other functions of GPUs [109]. On the other hand, mobile microprocessors are becoming more important. These microprocessors embed very low-power GPUs and auxiliary DSP units for co-processing on the same chip [110].

Programmable systems, not only FPGAs, are able to get the same performance as recent past ASICs, keeping time-to-market and non-recurring engineering costs lower compared to custom ICs. As discussed previously, FPGAs are between software and hardware solutions. Modern FPGAs experienced a large increase in hardware resources, both in dedicated units and logic cells. High-level programming languages are another major reason why FPGAs are becoming increasingly competitive, specially when dealing with complex FPGAs and to maintain and keep the designs portable [111]. Nowadays these devices can address complete SoCs, integrating memory and IO controller natively. Manufacturer roadmaps show their inclusion in a very near future and it is expected a big leap in performance and flexibility [112].

1.3 Related work

To accurately solve the problems they face, Computer Vision algorithms become more complex, leading to tight requirements in order to efficiently address the computation they involve. Application-specific processors become a requirement when standard processors are not able to meet a trade-off solution which involves performance, power consumption, form factor or cost. Although this is also applicable in many other fields, Computer Vision applications are particularly constrained to the processing capabilities of the hardware platforms. The literature is plenty of new hardware architectures which, besides taking advantage of the improvements of the semiconductor industry, aim to increasing the processing power and the flexibility without penalizing other figures of merit.

Low-level image processing usually represents most of the workload of a Computer Vision application. Improving this stage will significantly increase the overall performance. Based on this assumption, the scientific community has proposed a wide range of general-purpose and application-specific processors, auxiliary units, computing paradigms and many other approaches to efficiently address this low-level stage.

Traditionally, conventional microprocessors are employed for algorithm design and validation. The performance is constrained due to the general-purpose capabilities, being hard to reach tight requirements of certain Computer Vision applications. The inclusion of SIMD units for intensive operations such as video compression significantly improves the performance. Several SIMD instructions sets were proposed [113] to take advantage of the inherent spatial parallelism of the operations of the low-level stage. Besides parallelizing the compu-

tation, application-specific operands can also reduce the computational time. In this sense, DSPs usually outperform conventional CPUs by including specific instruction sets and architectures which take advantage of data parallelism, including a number of computation units [114], executing several instructions at a time [48] or including specific units for address calculations [115]. Although DSPs are widely used, the improvements in the semiconductor industry has made possible to greatly increase the number of computation units, leading to more specific architectures. Processor arrays can fully take advantage of the nature of the low-level operations. These processors implement the SIMD paradigm and offer impressive performance when executing native operations [84]. However, when the requirements are very tight, specially in terms of computational power and power consumption, it is necessary to use specialized processors. Below are reviewed some relevant application-specific processors.

The **Xetal** processor [116] implements a massively-parallel SIMD unit to exploit data parallelism. It includes a set of *line memories* which store several lines of the input image. This on-chip storage supplies data to a linear processor array greatly reducing the memory bottleneck and enabling single-cycle operand readout and result storage. It features general-purpose capabilities. The **Xetal-II** architecture [117] improves not only parallelism and power consumption but also data I/O by including dedicated input and output processors to manage data transfers and perform certain post-processing tasks. The Xetal-II prototype chip achieves up to 140 GOPS at 110 MHz, consuming 785 mW. The **IMAP-CE** [118] is an embedded processor for video recognition. As the Xetal architecture, it implements a linear memory array to supply data to a highly parallel SIMD linear processor, resulting in 128 units of 8 bit 4-way VLIW each, clocked at 100 MHz. An additional RISC processor controls the whole system. In addition, it enables more flexible memory access patterns than the Xetal architecture, permitting to address more image processing techniques by applying parallel and systolic algorithmic techniques to the linear processor array.

The **MIPA4k** processor [119] is a focal plane processor array. It includes a 2-dimensional grid of SIMD mixed-signal processing units which share data employing local connections. It also integrates the sensing stage on each processing unit, permitting to effectively integrate sensing and processing by eliminating the bottleneck between these two stages. It consists of an 8×8 array, implemented with $0.13 \mu m$ CMOS technology. The **Xenon V3** processor [81] is a similar processor which employs 3D manufacturing techniques for a more efficiently integration between sensors and processors. The test ASIC implementation is able to process 64×64 px images using UMC $0.18 \mu m$ technology. The **SCAMP-3** processor, described later

in Section 2.5 also features similar capabilities. To achieve a high integration and very low power consumption, mixed-signal architectures are employed in focal plane processor arrays. This yields lower accuracy and large development times compared to digital designs. The **ASPA** processor [120] is a focal plane digital processor array which features synchronous and asynchronous processing capabilities. The 19×22 test device provides a peak performance of 9.6 MOPS per processing unit when operating at 150 MHz. **FLIP-Q** [121] is a smart image sensor composed by a processor-per-pixel SIMD array. It is intended for scale space and Gaussian pyramid generation and multiresolution scene representation. The analog processing units are arranged into a QCIF-sized array and requires very low power (17.6 mW).

MorphoSys [122] is a reconfigurable system for computation-intensive applications. Besides a RISC processor for control tasks, it includes reconfigurable cell array, a coarse-grain SIMD coprocessor, of 8×8 elements. The elements of the array are arranged through a 2-dimensional mesh with an enhanced connectivity between blocks of 4×4 elements. Therefore, it is an intermediate approach between linear processor such as Xetal and focal plane processor as ASPA. Other approaches such as [123] provide a set of arithmetic units and the ability to reconfigure them into 1D and 2D arrays to match the low-level image processing operations. This multi-SIMD architecture also includes broadcast and summation functions and other different data access patterns to perform additional computation on-chip and reduce bandwidth requirements.

Many early vision algorithms are addressed employing other computing paradigms such as Cellular Neural Networks (CNNs) [124] or bio-inspired models [125]. The CNN computation paradigm is widely used in autonomous embedded and fast response systems. The **ACE16k** [126] processor is a focal-plane mixed-signal processor capable of operating at frame rates higher than 1000 FPS on 128×128px images, featuring up to 330 GOPS and very low power consumption. It implements CNN computation as well as standard SIMD arithmetics. Although CNNs are able to fully exploit analog computing, their implementation is not straightforward. Digital CNN emulator such as **Falcon** [127] aims to address technical issues and to provide more flexibility. This is a highly flexible FPGA-based architecture which enables high performance on different CNN configurations. Bio-inspired processors as [128] achieve very high performance on tasks such as object recognition. This implementation is able to detect and recognize several objects at a time employing biologically inspired neural networks and fuzzy logic circuits.

All the above architectures and devices take advantage of the massively parallelism of the low-level operations and some mid-level tasks to achieve this goal. However, to provide a more abstract representation of the scene content is considerably expensive in terms of computation and power consumption. Subsequent steps of Computer Vision algorithms require more flexible architectures, so these devices are not able to handle them. In spite of that, this shows that tackling the very first steps of most algorithms is essential for a successful algorithm implementation. Low-level image processing and certain mid-level tasks perform efficiently when employing the SIMD paradigm. However, subsequent tasks do not always fit this approach. MIMD paradigm enables a more flexible solution when the program flow becomes more complex and eases to exploit the task-level parallelism.

Stream processing has proved to be very effective for the sake of exploiting a limited form of parallel processing. In this mode, a series of operations are applied to a *stream of data* in a predetermined order, taking advantage of the on-chip memories and restricting the parallelism. There are a variety of devices that implement this concept, some of which are described below.

The **Imagine** stream processor [129] employs this paradigm for multimedia applications. It includes a specific programing model to directly operate with streams. This architecture permits to scale the number of arithmetic units and internal registers to meet the computation requirements. The prototype consists of 48 floating-point arithmetic units and it is able to achieve 18.3 GOPS while consuming 2.2 W. The arithmetic units are grouped into 8 clusters, each one connected to a register file which provides a high bandwidth and interfaces four external SDRAM memories. A microcontroller manages data input and output between the register file and each arithmetic cluster. During the execution of each kernel, all clusters execute the same VLIW instruction (Very Large Instruction Word). This scheme permits to take advantage of the on-chip memories to store intermediate results.

The **Merrimac** processor [130] is a stream processor which focuses on scientific applications. It includes a scalar core to perform control and to issue instructions to the stream unit. This module contains 64 64-bit floating point units, arranged in 16 clusters of 4 units each, and a register hierarchy which exploits the parallelism of the algorithm under execution. The registers of each cluster interfaces a larger register, which at the same time interfaces a set of cache banks. The bandwidth of this network is higher as it gets closer to the computing units, and permits to store intermediate results without accessing the external memory. At 1 GHz, the peak performance is 128 GFLOPS. Each cluster executes the same VLIW instruction,

which configures the internal network according to build the desired datapath for the kernel under execution.

The **CRISP** stream processor (Coarse-grained Reconfigurable Image Stream Processor) [131] aims to fill the gap between DSPs and ASICs to meet the processing requirements without compromising the cost. In particular, it addresses the image processing pipeline of digital cameras, acting as a specific processor in image preview mode and as a DSP for picture taking. Employing 0.18 μm technology, it outperforms state-of-the-art DSPs by a factor larger than 80. Its architecture is made up of a set of specific-purpose processing units including ALUs, MACs, line buffers, and window-based registers and specific-purpose units for image downsampling, color interpolation or pixel-based operations. An internal reconfigurable network permits to combine them in order to build the desired datapath for the current operation, adapting them to the algorithm requirements.

Other approaches are closer to the MIMD paradigm, allowing to execute different tasks at a time instead of focusing on data parallelism. This is the case of the **MORA** processor (Multimedia Oriented Reconfigurable Array) [132]. Intended for multimedia processing, it includes a large set of simple processing elements, which exchange data employing a limited network. As a difference with other approaches, it does not include a global storage but a distributed storage as each processing element includes a small dedicated RAM. Processing unit programming and routing control is also eliminated by configuring them adequately during the program loading. This way, the architecture can be configured according to the algorithm dataflow.

Network-on-chip processors implement on the same chip several processing units connected employing a communication subsystems. It permits to efficiently interconnect general-purpose and special-purpose units to accomplish certain tasks. Architectures such as [133] for SIFT feature extraction have proven its efficiency. This processor integrates 10 SIMD processing units and 8 image-specialized memories. The advantages of the low overhead interconnection between the processing units eases task level parallelism. The specialized memories are optimized for window-based processing while the processing units perform the most basic operations for SIFT feature extraction. The processor has a peak performance of 81.6 GOPS running at 200 MHz.

The previous approaches are able to exploit a reduced form of parallelism when the program dataflow becomes more complex. The increased level of abstraction of mid-level and high-level steps limits the efficiency of large SIMD units as it was mentioned at the begin-

ning of this section. However, combining both kinds of processing, SIMD and MIMD, in the same unit permits to address both stages employing the same processor, greatly reducing the hardware requirements [134].

The **HERA** processor (HEterogeneous Reconfigurable Architecture) [135] is an FPGA-based reconfigurable architecture which implements SIMD and MIMD computing paradigms simultaneously. It consists of a set of processing elements arranged into a 2-dimensional mesh, which fits matrix-based computation. A NEWS (north-east-west-south) network enables data sharing between these units. A 32-bit floating-point unit is the main module of the processing element, which also includes two dual-port local memories for data and program storage. A global unit issues instructions when operating in SIMD mode. Dataflow control is limited as this architecture focuses on data-intensive computation.

The **Ter@Code** FPGA architecture [136] implements a 128-unit processor for multi-processing computation. It consists of several general-purpose processors for sequential processing assisted by a number of processing units which exchange data through two different networks, latency- and throughput-optimized respectively. As the general-purpose processors are independent, they feature MIMD computation. The processing units, which depend on a given general-purpose processor, enable SIMD computation. The number of units each general-purpose processor have depends on the particular implementation and can be configured according to the target application. In fact, they do not have to be identical. The 128-unit implementation only include a general-purpose processor, which runs at 150 MHz. The processing units include an ALU and two RAM slices for private storage and data exchanging through the network. The peak performance is 19.2 GOPS.

The **IMAPCAR-XC** processor [137] is a reconfigurable SIMD/MIMD architecture for embedded vision systems. It consists of 128 SIMD units which can be configured to 32-MIMD independent processing units, each one containing independent data and instruction storage besides a floating-point unit. A central RISC core executes the sequential part of the algorithm, performs address computation and executes control tasks. The processing elements share data employing three different networks. One of them interfaces the external memory and permits data I/O. Another one connects adjacent processing elements for fast data exchange. The last one provides an extensive network along the 8 processing elements which form a cluster. The architecture is based on a linear processor array but permits flexible data access patterns not to compromise performance.

Low-level image processing usually consumes much of the computation time. However, subsequent tasks are also time-consuming and custom accelerators are often a requirement. In this sense, hybrid architectures permit to face both processing stages, reducing hardware requirements and taking advantage of the interaction between these stages to improve performance, instead of considering them independently.

1.4 Summary

The large variety of Computer Vision applications makes difficult to classify them into tight categories. As a result, it is extremely challenging to design a unique hardware architecture which handles efficiently all processing stages of any Computer Vision algorithm. In the literature there are available several studies where different platforms are tested under the same conditions [138, 139, 140, 52]. They show that to tune-up is key for performance and that new parallel computing techniques are a requirement to exploit parallel devices. In addition, the increase of the sales and the appearance of emerging markets make investment in new hardware platforms a necessity.

The most accessible platform is a Personal Computer equipped with a GP-capable GPU. Either as test or final platform, it cuts down development time and costs. GPUs give enough performance for most intensive tasks, while by using the CPU multimedia extensions it is possible to meet the requirements in the other stages. In addition, they include all necessary elements for user IO, communication, storage and information display. The availability of models is large enough to select the adequate platform according to the application trade-offs. When CPU performance is not adequate, DSPs are a serious alternative. In addition, it becomes almost mandatory when dealing with embedded devices without compromising performance, where power consumption and form factor are very restrictive. They are widely used for prototyping custom ICs but FPGA-based applications have their own niche. Integration and high flexibility besides a large number of available IP Cores allow to drop NRE costs. Although all devices described in this chapter are ASICs, they were not conceived for an unique application. To lower costs, the manufacturer expands their range of application although it is possible to find families specialized in specific tasks. But there are available devices very specific for critical tasks, where the requirements are very tight and any other device does not comply with them. Flexibility is complete and there is not restriction to employ cutting-edge technologies which are not available in commercial devices until a near future.

Almost all Computer Vision applications need to face all processing stages in a lesser or a greater degree. Often, this leads to implement efficient mechanisms to tackle massively spatial parallelism, mixed spatial and temporal parallelism and sequential processing. Each stage matches with a level of processing so all mechanisms have to be implemented in most applications. Low-level stages benefit of massively parallelism with simple data distribution systems as operations. When the data abstraction level grows, during mid-level tasks, more information about the problem is required by the algorithms, increasing their complexity. This leads to complex architectures, where information distribution and sharing makes it difficult to exploit spatial parallelism, although it is usually present. Task-parallel architectures are able to exploit better their characteristics. Low and mid-level processing stages can be implemented in pure hardware solutions because they often implement kernel operations. However, high-level is closer to software and designers can take advantage of this to build complex systems easier by using general purpose processors. In addition, the device which performs the image processing related tasks needs to communicate or to control other devices. This is not strictly related with the Computer Vision domain but it is clearly a requirement in the final solution. In this case, the use of a general purpose processor is beneficial because it allows easier control and it increases the flexibility of the whole system.

Although it is almost impossible to develop a system able to run all operations in an optimal way due to the rich nature of the Computer Vision applications, it is desirable to provide the capability to perform any operation. The design must be scalable to adapt it to the specific needs of each application. This way, a product ranging from low to high-end devices can be easily built. The internal architecture should be also modular, so that from a basic outline more features could be added without dramatic changes. In general, a high-end microprocessor is a requirement to manage complex operations and communications between the system and the external components of the complete system. A number of auxiliary units able to handle both SIMD and MIMD computing paradigms would tackle the most expensive computation. In particular, a large SIMD unit will greatly increase the performance by addressing the first steps of most algorithms, specially in the low-level stage. Embedded high-speed memory controllers are also key to reduce the data-access bottleneck. All these elements, together, are able to face efficiently most of the situations described throughout this chapter.

CHAPTER 2

ADDRESSING THE LOW-LEVEL STAGE

As discussed previously in Chapter 1, low-level and certain mid-level Computer Vision operations are characterized for being repetitive and simple and applied over a large set of data. Therefore, these operations usually represent the highest percentage of workload for a processor running image processing tasks. In particular, the overall performance of the application will increase significantly only optimizing the low-level steps.

An efficient Computer Vision processor has to deal efficiently with a wide range of operations, data dependencies and program flows. Its hardware must be flexible enough to face the different algorithms with an adequate performance without wasting hardware resources. Regarding to early vision or low-level, the operations do not usually require large word sizes, so we can take advantage of this to increase the density of integration and thus to enhance the parallelism as it is key for performance. However, a large amount of processing units is not effective without a flexible interconnection to exchange data between them. The design must also be modular, scalable and easy to adapt to the needs of the different applications. This way, memory or word sizes as well as the mathematical operations available on the core computing units have to be adjustable to obtain a flexible and capable processor for the most computational expensive tasks.

This chapter introduces an architecture for low-level image processing, focusing on the inherent massively spatial-parallelism of the operations to efficiently address this computation. The low-level image processor, conceived as a co-processor of a conventional general-purpose microprocessor, was designed keeping in mind the conclusions drawn in Chapter 1. Finally, a comparison between this accelerator and other related approaches is accomplished to deter-

mine its advantages and weaknesses. For this purpose, an algorithm which includes the most representative operations of the low-level stage was selected. It will permit us to determine the advantages and weaknesses of the architecture in order to improve them in a subsequent design.

2.1 Evaluating fine-grain processor arrays

During the analysis of the Computer Vision algorithms and applications made in Section 1.2 it has been concluded that low-level operations are computationally too costly: a co-processor to reduce the workload of the main microprocessor is a requirement. During the exploration of the computing paradigms and hardware devices in Section 1.2, it has been extracted that low-level operations are characterized for being quite simple, with fixed patterns for data access and repetitive over the whole set of data. The algorithm selected to benchmark the architecture proposed in this chapter, which will be detailed later in Section 2.3, is a clear example which satisfies these characteristics.

For all seen before, a processor array is the natural platform for low-level image processing. It is able to exploit the inherent massively parallelism, matching at the same time the communication relationships between the processing cores. In this section, an analysis of a fine-grain binary processor array is issued. This analysis is made for two main reasons: to study the characteristics, limitations and advantages of processor arrays and as an initial step towards a general-purpose processor to handle gray or color images. In addition, an only-binary processor has its own niche of applications. The image quality may be a not strict requirement, reformulating the problem to be handled uniquely with binary images, or the most expensive part of the algorithm relies on binary data, or even to reduce the cost of the equipment as a binary processor is expected to have much lower hardware requirements.

2.1.1 Processor architecture

The architecture approached in this section will exploit the inherent spatial parallelism of the low-level stage. It will be a Massively-Parallel (MP) SIMD architecture, with a correspondence of a processor-per-pixel, where all computing units execute the same operation. A fixed, local and reduced network permit to exchange data locally between the processors or Processing Elements. As large arrays have large area requirements, the design of the Processing Element will be heavily constrained with the purpose of reducing area consumption

and thus increasing integration density. In order to achieve this goal, the instruction set must be very simple and condensed, so that the buses which propagate the control flags would be as narrow as possible to reduce the fan-out. This parameter will be the major limitation to achieve a high clock frequency. It should be noted that data-buses are 1-bit wide, as we are considered binary (B/W) images.

The Processing Element

As it has been mentioned before, the design is focused on area, aiming at the smallest possible Processing Element, and thus the largest possible array. With this, the resultant Processing Element is quite simple. Figure 2.1 displays its schematic view. It has three main components: a Logic Unit, a Memory Bank, and a classical NEWS (North, East, West and South) system for local connectivity among neighbors within the array.

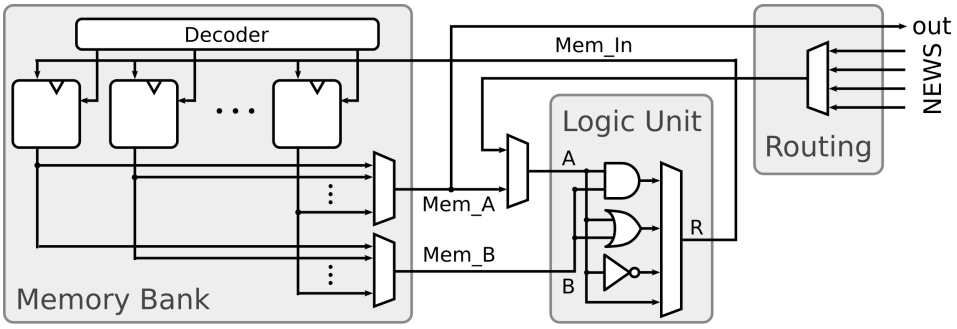


Figure 2.1: The Processing Element for the fine-grain B/W processor array.

The Logic Unit performs just the most basic Boolean functions: AND, OR, NOT and Identity. This set of operators makes up a functionally complete logic, so it is feasible to implement any algorithm for B/W image processing. The Logic Unit takes two inputs, namely A and B, and gives one output, R. The operand B drives only the AND and OR gates. The operand A feeds the four Boolean operators. The operand B comes from memory. The operand A comes from either memory or the neighborhood through the NEWS system. This provides an adequate solution to operate with values from both memory and the neighborhood. The identity operator can be used to transfer data among memory elements within the cell under study, or to save a neighbor variable into the local memory. Also, the identity operator can make synchronous shifts through the NEWS system of either columns or rows in the array.

The function to be done by the Logic Unit is selected through a 4:1 multiplexer. In addition, it would be easy to include new operators in the Logic Unit to meet the time needs of specific applications or algorithms, although at the expense of area.

The Memory Bank comprises a configurable set of 1-bit registers. The Memory Bank takes a 1-bit word as input, labeled Mem_In, and provides two simultaneous outputs, labeled Mem_A and Mem_B. This configuration permits to handle both internal operations and data exchange among neighbors. The identity operator in the Logic Unit is needed for the latter. In addition, it is doable to read and write simultaneously at the same memory address. This reduces the number of storage elements needed for a generic algorithm, as it is feasible to do operations of the type $Reg(0) = Reg(0) \otimes Reg(1)$. An enable/disable flag prevents for undesirable writes in the register selected by the address decoder. It should be noted that the Memory Bank is one of the most critical modules in terms of area consumption, so the number of storage elements should be reduced as much as possible.

The connectivity among Processing Elements is set through the NEWS system. Every Processing Element counts on four inputs and one output. A 4:1 multiplexer decides which one of the four neighbors is used for processing at the Processing Element under study. Inner connections draw this value to the appropriate module. The instruction determines what to do with the output from every Processing Element.

The processing array

Figure 2.2 displays the schematic view of the global configuration of the proposed architecture. The emphasis has been put on the array, so the global system architecture addressed here is not optimized, as it is only intended as an operation tester of the processing array. Data can be uploaded/downloaded to/from the array by means of iterative shifts from right to left (using the Identity operator) of the whole array. In addition, a ring of registers close to the input and output pins are required for a complete synchronous uploading/downloading. With this approach, an array of $N \times N$ would require N cycles to upload/download the whole array. This avoids the design of row and column decoders, leading to an important area reduction, although at the expense of not having random access. The array is completed with a ring of dummy Processing Elements that can be set to either high or low. Also, to save time the uploading/downloading of the image at the same time as it is being processed would be possible, leading to higher processing rates. The global configuration is made up of the following elements:

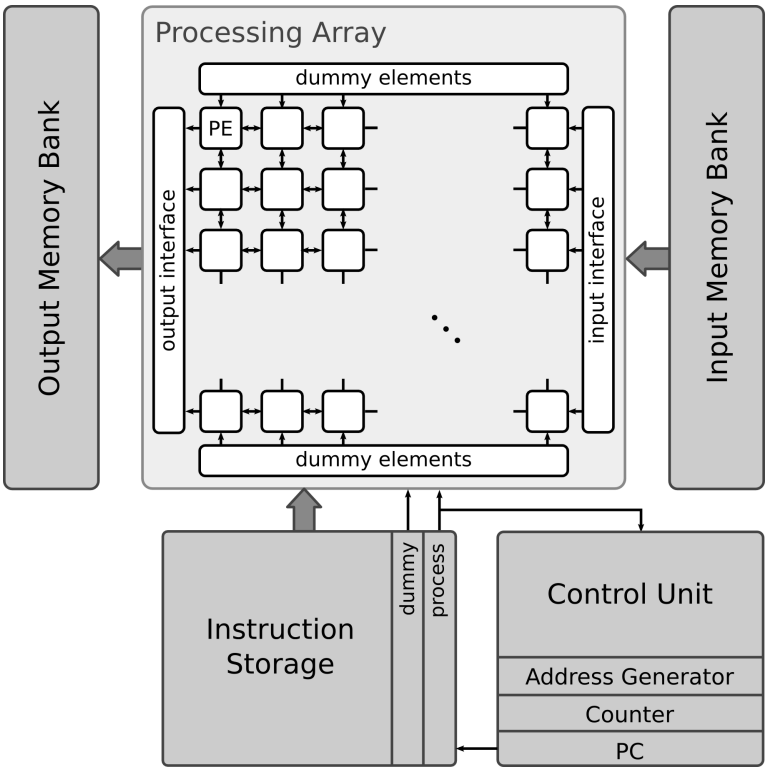


Figure 2.2: Schematic of the fine-grain binary processor array.

- Input and Output Memory Banks: to store images to be processed and the processed images. Both of them provide a column per clock cycle.
- Instruction Storage: it stores the instructions, the value of the dummy ring for each instruction and an interrupt signal, *process*.
- Control Unit: it synchronizes all existing modules. It includes the instruction decoder and the Address Generator unit, which calculates the address of the next instruction to be executed. It also allows loops in order to execute blocks of instructions iteratively, reducing memory requirements for storing the instructions.
- The Processing Array

opcode	A		B	R		out
--	from	address	address	wr_en	address	address

Figure 2.3: Instruction format of the fine-grain binary processor array.

Instruction set

The instruction set has to face the following functionality of the Processing Element:

- Select neighbor
- Select two memory values
- Select between two operation modes:
  - with two operands, either with two memory values or with a memory value and a neighbor (AND/OR)
  - with one operand, either from the memory or from a neighbor (NOT/Identity)
- Write the operation result into the Memory Bank
- Provide the Processing Element output

The control for the Processing Element can be implemented with a few global logic gates if an adequate instruction set is designed. The instruction format is shown in Figure 2.3. The instructions are split into five segments. The first segment, *opcode*, decides the operation to be done by the Logic Unit (AND, OR, NOT or Identity). The second segment, *A*, encodes the first operand to the Logic Unit. It has two sub-segments. The first sub-segment is a 1-bit flag to indicate the source of the operand (neighbors or inner memory) and the second gives its address. The third segment in the instruction, *B*, means the second operand address. The fourth segment, *R*, yields the memory address in which the output from the Logic Unit is saved. This segment comprises two parts: a flag to enable or disable the writing, and the address itself. The last segment, *OUT*, contains the memory address from which the output value is sent to the neighbors. The selected format permits a very easy decodification as most fields directly encode the different flags used by the Processing Element, leading to a very low number of global lines between the Control Unit and the Processing Elements.

2.1.2 Hardware implementation

FPGA evaluation

The fine-grain processor array was evaluated in a reconfigurable hardware, an FPGA. The device chosen for the implementation is an RC2000 card from Celoxica [141], which is made up of an RC2000 PMC card and a PCI-PMC carrier card, which allows the connection of the card to the PCI bus of a personal computer. The RC2000 PMC card includes a Xilinx Virtex-II xc2v6000-4 FPGA, six banks of Zero Bus Turnaround (ZBT) RAM of 2 MB each and two banks of ZBT RAM 4MB each accessible only from the FPGA. The architecture was completely developed with VHDL and synthesized using ISE 9.2i from Xilinx tools [142]. RAM and PCI access is done through proprietary modules provided by Celoxica.

The array synthesized on the Virtex-II FPGA has a resolution of 48×48 (2304) Processing Elements. The Memory Bank in every Processing Element contains 8×1-bit registers. In addition to the external communication modules, a 32-instruction storage was included for testing purposes. The resources employed are indicated in Table 2.1. As it can be seen, there are still resources available on the FPGA chip that can be used to include more Processing Elements. If we opt for keeping the same memory, the maximum possible array size would amount to 56×56. Concerning speed, the highest frequency attained by the implementation was set to 67.3 MHz. The same design, with minor optimizations to take advantage of the new organization of the Virtex-5, which employs 6-input LUTs and larger CLBs, was synthesized on a Virtex-5 xc5vlx110-3 FPGA, achieving lower resource usage and nearly doubling the frequency, amounting to 123.6 MHz.

FPGA	**LUTs**	**Flip-Flops**	**Max. Frequency**
Virtex-II xc2v6000-4	50649/67584 (74%)	20165/67584 (29%)	66.7 MHz
Virtex-5 xc5vlx110-3	32289/69120 (47%)	18597/69120 (26%)	123.6 MHz

Table 2.1: Implementation results of a 48×48 array on Xilinx FPGAs. *Note: Virtex-II employs 4-input LUTs while Virtex-5 are 6-input.*

The impressive advance of new technologies, marked by Moore's Law, allows to increase more and more the density of integration. This enables higher clock frequencies, considerably increasing the performance. Thus, programmable systems, such as FPGAs, are able to get the same performance as recent-past application specific integrated circuits, reducing the engineering costs and time-to-market (TTM).

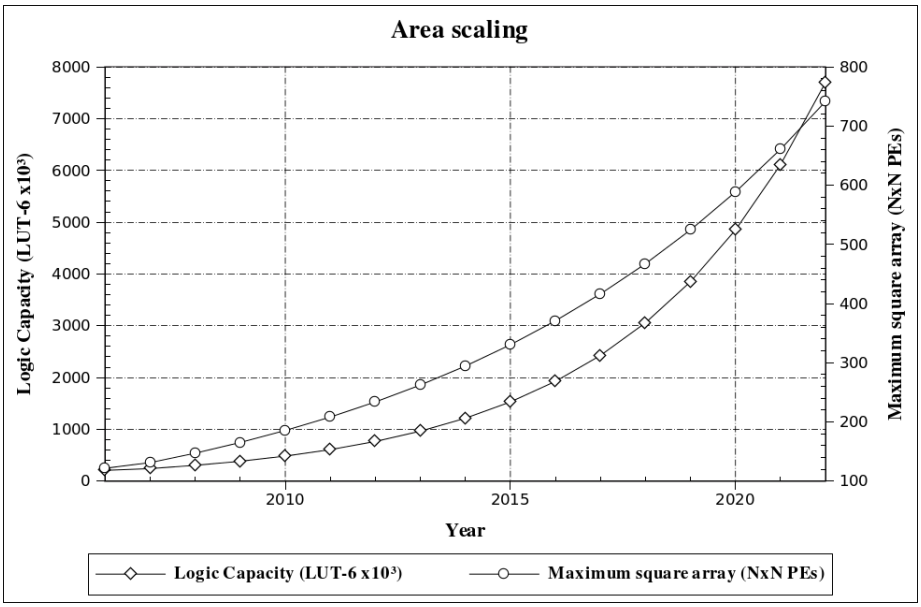


Figure 2.4: Scaling of the fine-grain processor array on FPGAs according to the ITRS roadmap.

The International Technology Roadmap for Semiconductors (ITRS) [143] provides a work plan for the semiconductor industry. This document, created in collaboration with organizations and major worldwide companies, sets the guidelines to follow for the semiconductor industry to achieve the technological progress necessary to continue Moore's Law predictions. The roadmap provides a forecast for the next 15 years, indicating the major milestones.

The increasing amount of resources available in newer devices make the FPGAs more versatile and powerful, so they can be used not only as proof-of-concept devices but also as target devices. The TTM is another reason why FPGAs are becoming more competitive. Below, a study of the logic capacity and clock frequency of the commercial FPGAs is shown. This study is based on the predictions of the ITRS roadmap. This way, the parallelism achieved by the fine-grain processor array will grow considerably, achieving enough resolution to employ the FPGA as final device. These results are shown in Figure 2.4.

The method employed for this estimation is the following. Given the logic capacity of the FPGA in number of logic cells for a given year and the ITRS prediction, the logic capacity in terms of logic cells for the target year is

$$Capacity_B = Capacity_A \cdot \frac{Density_B}{Density_A} \cdot \frac{Die\_Size_B}{Die\_Size_A}$$

where A and B subscripts refer to the known and the target years. The parameter *Die_Size* refers to the chip size, although the ITRS prediction indicates that the chip size will not change in the next years. In the *Density* parameter, the measure of the number of million of transistors per square centimeter also includes the effects of the routing and the different size of N and P MOS transistors, as well as other parameters which limit the density of integration. This equation is only valid if the internal architecture of the FPGA does not change. This study is based on [144], updating its results.

The FPGA employed as a reference is a Xilinx Virtex-5 xc5vlx330. It includes more than 50000 logic cells (207360 6-input LUTs) and employs 65nm technology. This family replaced the 4-input LUT internal architecture for a new 6-input LUT, and a different packaging of the logic cells, enabling more density of integration. Figure 2.4 shows how this family scales according to the ITRS predictions. These results have to be considered as an upper boundary of the capacity as other elements of the FPGA are not considered. Embedded units such as multipliers of Block RAMs are becoming more important and new designs should take advantage of their inclusions. However, the accuracy of the study is enough for our purpose as the fine-grain processor does not make use of none of the embedded slices of the FPGA.

Under this new architecture, each Processing Element of the fine-grain processor array employs 8 Flip-Flops and 14 6-LUTs, with the same configuration of the array indicated in this section. Therefore, the limiting factor is the number of LUTs available in the FPGA. Results show that employing the technology of 2009, a QCIF (144×176px) is feasible, while around 2013 and 2019, it would be possible to manipulate images of 256×256px and 512×512px respectively, without splitting them into sub-windows.

2.1.3 Algorithm evaluation

By the nature of the instruction set, a Boolean equation that means how each PE changes its value according to a given neighborhood must be found. This equation can be obtained in various ways: from the original image and the expected image, observing the differences between them; translating the effects of a mask or a filter; directly designing the Boolean equation, etc. Next, a selection of algorithms to test and show as the proposed system works is listed. The examples were extracted from the *Cellular Wave Computing Library* (CWCL) [145], which collects the most used templates in image processing on Cellular Neural Net-

works (CNNs). This selection is not a limitation because it is a widely used platform for these tasks, being a representative set of low-level image processing operators.

A Cellular Neural Network (CNN) [124] is a non-linear processing system made of an n-dimensional matrix of identical and dynamic processing elements. These elements interact through local connections on a limited neighborhood. This local and reduced interconnectivity, besides recursive operations, permit global processing. This is a parallel computing paradigm similar to neural networks which features general-purpose computing and enables very high performance although there are certain technical limitations in practical applications such as image resolution or computation accuracy [126]. Their characteristics make it suitable for low-level image processing and bio-inspired computer vision.

In terms of notation, the following variables are used to refer to the neighborhood of the central pixel, *c*: *n*, *nw*, *se*, refer to *north*, *northwest*, *southwest*, and so on. We will refer to instructions on the type *neighbor(address)*, for example *north(R2)*, meaning *access to the #2 memory address of the northern neighbor*. If a neighbor is not specified, it is understood that it is the local memory of the Processing Element under study.

Binary edge detector

The edge detector is a good model to show how the architecture works. This operator corresponds to the binarized version of the edge detector present in the CWCL, which can be interpreted as follows: when one or more neighbors *n*, *s*, *e* or *w*, are active, the central pixel is activated. This is equivalent to an OR between these four values, the mask shown in Eq. 2.1.

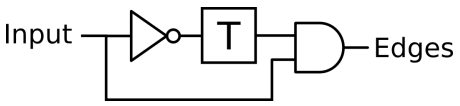


Figure 2.5: Edge detection algorithm. Template T is shown in Eq. 2.1.

$$T = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \rightarrow T = n + e + w + s \tag{2.1}$$

As the Logic Unit is not capable of operating simultaneously on the four neighbors, it is necessary to divide it into four sub-operations. Thus, in the first clock cycle the image is inverted. Then, the mask is applied, requiring 4 clock cycles, to finally perform the AND

between the previous results. Altogether, it takes 6 clock cycles. The pseudo-code is listed below:

Listing 2.1: Binary edge detector

```
Loading: R0 = input image
Invert: R1 = NOT R0
Apply mask:
   R2 = north(R1)
   R2 = R2 OR east(R1)
   R2 = R2 OR west(R1)
   R1 = R2 OR south(R1)
And: R1 = R0 AND R1
```

In this case, a four input OR which handles the four input signals of the PE may be implemented to increase the speed in applications that require intensive use of edge detection. It clearly illustrates the possibilities of expanding the proposed Logic Unit for this architecture.

Pattern matching finder

As its name suggests, this operation finds certain patterns on an image. As an example, we consider the pattern shown in Figure 2.6. The symbol '-' means does not matter if this pixel is present or not. This operator is applied in every pixel of the image and the output consists of a binary image representing the locations of the 3×3 pattern. It can be translated easily onto the SIMD architecture, the current pixel will be active if the neighborhood matches with the pattern, i.e., the condition $OUT = nw \cdot \bar{n} \cdot ne \cdot c \cdot sw \cdot \bar{s} \cdot e$ is true.

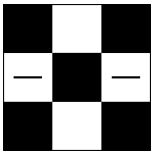


Figure 2.6: Pattern matching template example.

Listing 2.2: Pattern matching finder

```
Loading: R0 = input image
Apply equation:
  R1 = NOT north(R0)
  R1 = R0 AND R1
  R2 = AND south(R0)
  R1 = R1 AND R2
  R2 = shift(right)
  R1 = R1 AND north(R2)
  R1 = R1 AND south(R2)
  R2 = shift(left)
  R1 = R1 AND north(R2)
  R1 = R1 AND south(R2)
```

It should be noted that to access a pixel that is not directly connected to the pixel of interest, it is only necessary to perform shifts until the value of the pixel shifted reaches one of the four neighbors (and not the position of the pixel under consideration). This allows us to increase largely the performance.

Hole filling

The hole filling is an iterative operation. It is used to fill the holes in all the objects that are part of an image. In a synchronous architecture, as the proposed here, it is executed iteratively a number of times that can be fixed beforehand or determined during the execution. The first case is the most common and it is the considered here. The algorithm used is described in [146] and shown in Figure 2.7, where T is the same template described above in Eq. 2.1.

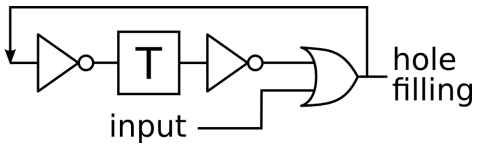


Figure 2.7: Edge detection algorithm. Template T is shown in Eq. 2.1.

Listing 2.3: Hole filling

```
Loading: R0 = input image
Invert: R1 = NOT R0
Apply mask:
   R2 = north(R1)
   R2 = R2 OR east(R1)
   R2 = R2 OR west(R1)
   R1 = R2 OR south(R1)
Invert: R1 = NOT R1
Or: R1 = R0 OR R1
```

A complete iteration of the algorithm requires 7 clock cycles. The number of iterations needed depends on the shape and the size of the objects of the image.

Skeletonization

Skeletonization is an operation which finds the skeleton of a black and white object. Figure 2.8 displays it flow diagram. In this case we use the masks listed in the CWCL. Eqs. 2.2 and 2.3 are the first two CNN templates of the algorithm. The rest of the templates are rotated versions of Eq. 2.2 (*SkelBW3, SkelBW5, SkelBW7*) and Eq. 2.3 (*SkelBW4, SkelBW6, SkelBW8*). In this case, black pixels have been assigned to +1 and white pixels to -1, as usual in the CNN terminology.

$$SkelBW1 : A = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}, I = -1 \tag{2.2}$$

$$SkelBW2 : A = \begin{bmatrix} 2 & 2 & 2 \\ 0 & 9 & 0 \\ -1 & -2 & -1 \end{bmatrix}, I = -2 \tag{2.3}$$

The approach of the B/W skeletonization is not straightforward as it is necessary to perform an analysis of the cases in which the active pixel changes state. The previous templates can be rewritten as follows:

$$SkelBW1 = \overline{\overline{nw} \cdot \bar{n} \cdot \bar{w} \cdot e \cdot s} \cdot c = \overline{\overline{(nw + n + w)} \cdot e \cdot s} \cdot c \tag{2.4}$$

$$SkelBW2 = \overline{\overline{nw} \cdot \bar{n} \cdot \overline{ne} \cdot s \cdot (sw + se)} \cdot c = \overline{\overline{(nw + n + ne)} \cdot s \cdot (sw + se)} \cdot c \tag{2.5}$$

Listing 2.4: Skeletonization: SkelBW1 template.

```
Loading: R0 = input image
R2 = west(R0)
R1 = north(R2)
R1 = R1 OR north(R0)
R1 = R1 OR west(R0)
R1 = NOT R1
R1 = R1 AND east(R0)
R1 = R1 AND south(R0)
R1 = NOT R1
R1 = R0 AND R1
```

Listing 2.5: Skeletonization: SkelBW2 template.

```
Loading: R0 = input image
R2 = west(R0)
R3 = east(R2)
R1 = north(R2)
R1 = R1 OR north(R0)
R1 = R1 OR west(R3)
R1 = NOT R1
R1 = R1 AND south(R0)
R4 = south(R2)
R4 = R4 OR south(R3)
R1 = R1 AND R4
R1 = NOT R1
R1 = R0 AND R1
```

In this case, the number of cycles required to compute these two templates is 9 for *SkelBW1* and 12 for *SkelBW2*. After loading the image, the number of registers used are 2 and 3 respectively. One of them stores the result. For the whole algorithm, we need 84 cycles and only 3 registers.

Large-neighborhood access

As an example of large-neighborhood template we have realized a 5×5 line detector similar to the *LE3pixelLineDetector* found in the CWCL. The template addressed here deletes lines with more than three pixels in a row along the horizontal, vertical and the two diagonal directions,

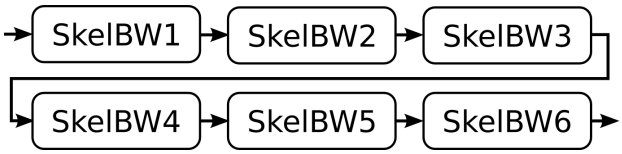


Figure 2.8: Flow diagram of the skeletonization operation.

keeping only the lines with less than or equal to three pixels. Figure 2.9 depicts the algorithm flow for these tasks, where the employed templates are shown in Eqs. 2.6-2.8. The different variations of templates T1 and T2 applied in the first stage are rotated versions of those shown in the previous equations. As it was described previously in the pattern matching finder, it is necessary to shift the image several times to access to elements located farther than the closest Processing Elements. This is done using the identity operator and, as described previously, it is only necessary to perform shifts until the value of the pixel shifted reaches one of the four neighbors and not the position of the Processing Element under consideration. Despite this, there are required 59 cycles and just 3 registers.

$$T1_h : A = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 \end{bmatrix}, I = -3.5 \tag{2.6}$$

$$T2_h : A = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 \end{bmatrix}, I = -3.5 \tag{2.7}$$

$$T3 : A = \begin{bmatrix} -1 & -1 & -1 \\ -1 & +1 & -1 \\ -1 & -1 & -1 \end{bmatrix}, I = -8.5 \tag{2.8}$$

Listing 2.6: Binary line detector. T1 and T2 in diagonal direction.

```
Loading: R0 = input image
R1 = east(R0)
R1 = R0 AND north(R1)
R2 = east(R1)
R1 = R0 AND north(R2)
R2 = west(R0)
R1 = R1 AND south(R2)
```

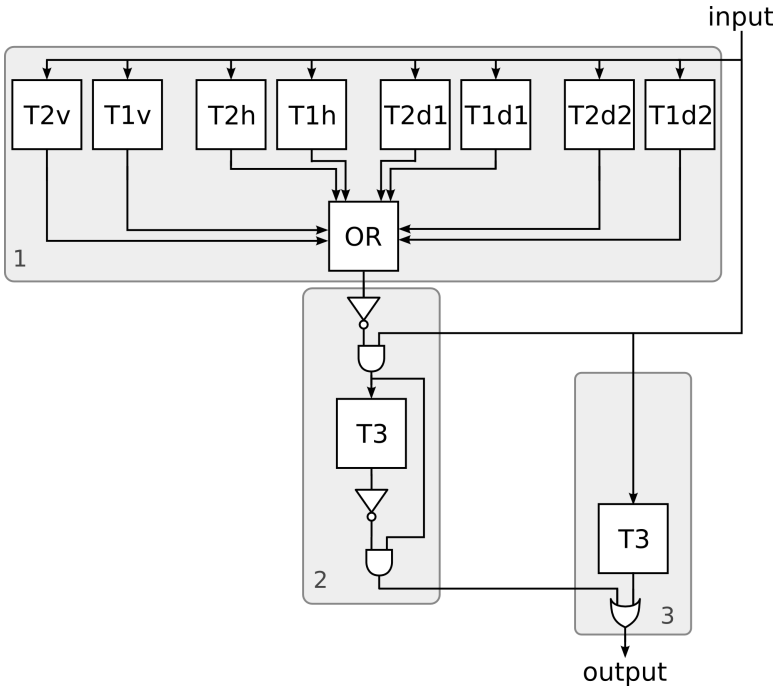


Figure 2.9: Flow diagram of the line detector algorithm, which employs large-neighborhood access.

Listing 2.7: Binary line detector. T1 and T2 in vertical direction.

```
Loading: R0 = input image
R1 = R0 AND north(R0)
R2 = R0 AND north(R1)
R2 = R2 AND south(R0)
```

Listing 2.8: Binary line detector. T3 template.

```
Loading: R0 = input image
R1 = west(R0)
R2 = east(R0)
R3 = north(R0)
R3 = R3 AND north(R1)
R3 = R3 AND north(R2)
```

```
R3 = R3 AND south(R0)
R3 = R3 AND south(R1)
R3 = R3 AND south(R2)
R3 = R3 AND east(R0)
R3 = R3 AND west(R0)
R3 = NOT R3
R3 = R0 AND R3
```

Shortest path problem

Finally, an implementation of the algorithm that solves the problem of the minimum path was done. The application is significantly more complex than the other examples outlined previously and it illustrates the capability of the binary processing array. The aim is to determine the shortest path between two points, avoiding a series of obstacles. It is based on the implementation discussed in [147], which proposes a new approach to solve this problem by using CNN computing. In line with this strategy, a wave front with constant speed explores the labyrinth from the starting point. At each branching of the labyrinth, the wave front is divided. When two wave fronts are at an intersection, the first to reach will continue evolving while the rest remains static, avoiding the collision. Then, a prune of all paths is done, maintaining fixed the start and end points, which are external parameters of the system, so only the shortest path between those points remains. The algorithm has two stages, both to carry out iteratively. The templates *T1* and *T2* are defined in Eqs. 2.9 and 2.10 along with its translation into Boolean equations.

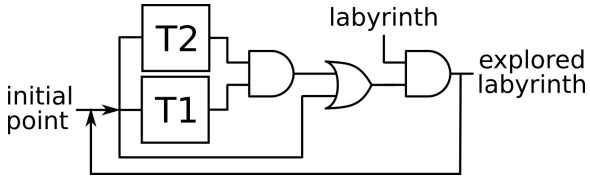


Figure 2.10: Flow diagram of exploration phase of the shortest path problem.

$$T1 : A = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}, I = -0.5 \rightarrow T1 = n + e + w + s \tag{2.9}$$

$$T2:A=\begin{bmatrix}0&-1&0\\-1&0&-1\\0&-1&0\end{bmatrix},I=-1\rightarrow T2=\overline{n\cdot(w+s+e)+e\cdot(w+s)+w\cdot s} \tag{2.10}$$

The second stage, the pruning, is done executing iteratively *T3*, defined in Eq. 2.11. This template is equivalent to an AND between the labyrinth and explored the result of invert the application of *T2* on the explored labyrinth, so the above equations will be used again.

$$T3:A=\begin{bmatrix}0&1&0\\1&3&1\\0&1&0\end{bmatrix},I=-2 \tag{2.11}$$

During the exploration phase, *T1* requires 4 clock cycles, *T2* 9 cycles and the additional logic operations, 3 cycles. All in all, each iteration requires 16 cycles. The pruning phase is executed in 9 cycles, 8 for *T2* and one for the additional operations. The number of necessary iterations for each stage depends on the labyrinth. Figure 2.11 shows the different stages of each phase on a test labyrinth.

Listing 2.9: Shortest Path Problem. Exploration phase.

```
Loading: R0 = input image
Loading: R1 = start point
Apply T1
   R2 = north(R1)
   R2 = R2 OR east(R1)
   R2 = R2 OR west(R1)
   R1 = R2 OR south(R1)
Apply T2
   R3 = west(R1)
   R4 = R3 AND south(R1)
   R5 = R3 OR south(R1)
   R3 = R5 AND east(R1)
   R3 = R3 OR R4
   R4 = R5 OR east(R1)
   R4 = R4 AND north(R1)
   R3 = R3 OR R4
   R3 = NOT R3
Others
   R2 = R2 AND R3
   R2 = R2 OR R1
```

```
   R1 = R0 AND R2
```

Listing 2.10: Shortest Path Problem. Pruning phase.

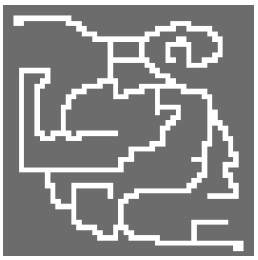
```
Loading: R7 = Destination point
Loading: R1 = R1 OR R7
Apply T2
   R3 = west(R1)
   R4 = R3 AND south(R1)
   R5 = R3 OR south(R1)
   R3 = R5 AND east(R1)
   R3 = R3 OR R4
   R4 = R5 OR east(R1)
   R4 = R4 AND north(R1)
   R3 = R3 OR R4
   R3 = NOT R3
Others
   R3 = NOT R3
   R1 = R1 AND R3
```

Summary

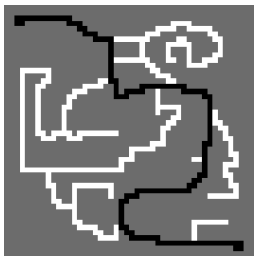
Table 2.2 gives a summary of processing times for each algorithm, considering only one iteration. The maximum working frequency for the Virtex-II FPGA xc2v6000 is 67.3 MHz. It also includes the number of required registers per Processing Element for its execution, counting the one used to store the input image, which does not change during processing, although this would not be necessary in all cases. For the iterative algorithms, Table 2.3 shows the total execution times. The test images have the same size as the matrix, i.e., 48×48 pixels. We have to remark that each instruction only takes one clock cycle to be executed.

2.1.4 Discussion

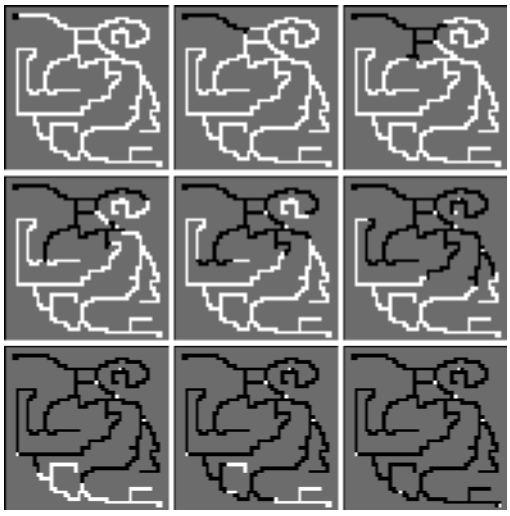
After the study of fine-grain processor arrays, we can extract some major conclusions. As the processor matches the operations of the low-level stage and as data exchange between computing cores is done at the same time as computation, overlapping both processes, the throughput is very high and the peak performance is easily achieved. In addition, the performance is independent of the size of the image. However, fine-grain processor arrays feature



(a) Labyrinth



(b) Shortest path



(c) Exploration phase

(d) Prune phase

Figure 2.11: Example of the shortest path problem and intermediate steps during algorithm execution.

Algorithm	**# cycles**	**Time μs**	**# registers**
Edge detector	6	0.089	3
Hit and Miss	10	0.149	3
Hole Filling	7	0.104	3
5×5 neighborhood access	59	0.885	3
Skeletonization	84	1.248	5
Shortest Path: exploration	16	0.238	6
Shortest Path: prune	9	0.138	4

Table 2.2: Processing times for the tested operations with a frequency of 67.3 MHz.

Algorithm	**# cycles**	**# iterations**	**Time μs**
Array load/download	1	48	0.71
Hole Filling	7	45	4.68
Skeletonization	84	40	49.93
Shortest Path: exploration	16	125	29.72
Shortest Path: prune	9	75	10.03

Table 2.3: Processing times for the iterative algorithms with a frequency of 67.3 MHz.

some important limitations. First of all, they are not able to handle large images at a reasonable cost with the current technologies. Not only due to the required area, but also due to the complexity of the design to drive the global signals to every Processing Element. This results in low clock frequency. This is even more critical if employing large neighborhood operations as continuous data shifts slows down the computation. Although the processor described here focuses on binary image processing, the conclusions can be expanded to more capable processors which handle gray or color images. The major benefits of this kind of processors are achieved when integrating the sensors with the Processing Element [148].

As a conclusion, the strategy to increase the processor capabilities is to move to a lower degree of parallelism, this is, with a lower number of Processing Elements, but enhancing their capabilities. In addition, the internal network must be improved in order to make available a larger neighborhood, reducing the network usage. As a result, it is expected to lower the area requirements and to achieve higher clock frequencies.

2.2 General-purpose coarse-grain processor array

The next step in our goal of designing an architecture for accelerating low-level image processing tasks is to extend the architecture proposed in Section 2.1 towards non-binary images. This will permit to handle images in grayscale or color, which requires greater precision and greater number of bits in its representation. As concluded after the analysis of the strengths and weaknesses of the fine-grain processor array, we will move to a lower degree of parallelism. The Processing Element will be enhanced in order to reduce the global area requirements and achieve higher clock frequencies. In addition, neighborhood access must be improved in order to handle larger neighborhood operations more efficiently. As a result, the new processing array will feature a coarse-grain spatial parallelism but will be able to handle larger images, permitting to upscale easily.

2.2.1 Instruction Set

As it was detailed previously, the instruction set of the fine-grain processor array manages efficiently both computation and data exchange between the computing units. The instruction set of the coarse-grain processor array pursues a similar goal. In order to execute operations which require large neighborhood access such as convolution, image-shifting is included natively in every instruction. The instruction set has the following format:

image[R] = image[A] [operation] shift(image[B], amount)

or

image[R] = constant [operation] shift(image[B], amount)

Most algorithms can be approached by a reduced set of mathematical operators, so the operation field includes additions, subtractions or multiplication with image data, besides Boolean functions for binary image processing. On the other hand, flow control is a need, so branches and integer (non-vector) operations must be added. In both cases, the capability of working with constant operators or immediates will improve the performance and the flexibility of the system. The system architecture will consist of two processing units, one specialized in image data and another one, much simpler, to handle the program flow control. Likewise, two types of instructions are needed. These instructions are summarized in Table 2.4. Their format is outlined in Figure 2.12. A fixed-width instruction set allow to implement a simple control with higher performance than in the case of variant-width instructions. Every instruc-

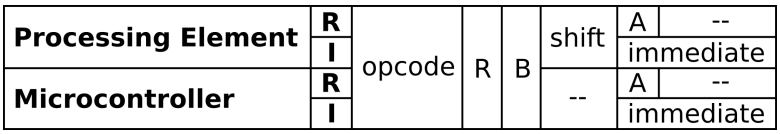


Figure 2.12: Fixed-width instruction format of the coarse-grain processor array.

Unit	Type	Operation
PE	R	add, sub, mult, mac, thr, identity, and, or, not
	I	addi, subi, multi, maci, thri, andi, ori
Microcontroller	R	add, sub, slt
	I	addi, subi, beq, bne, j

Table 2.4: Implemented instructions on the Coarse-Grain Processor Array.

tion can take either register contents (R-type) or an immediate (I-type) as operands. Some of the operations included are addition (*add*), subtraction (*sub*), threshold (*thr*), set less than (*slt*), or multiply and accumulate (*mac*). Arithmetic instructions also permit signed/unsigned operation and data saturation. This set of instructions makes up a functionally complete logic although custom instructions for time-sensitive applications can be easily added. The threshold operation is a good example of this.

2.2.2 Processor Architecture

Figure 2.13 shows the top-level view of the system architecture. The microcontroller stores the program, issues the instructions and controls the program flow, handling branches and related operations like the increment of a variable, e.g. loops. The Processing Array performs the image processing. It is composed of a set of Processing Elements (PEs) interconnected through the classical NEWS (North-East-West-South) network. The PEs work synchronously in SIMD mode. Every PE stores and processes a sub-window of the whole image. Larger images require larger sub-windows in every PE and thus more memory space, or more PEs. The microcontroller decodes the instruction and sets the adequate flags for all PEs, except the calculation of the address where the data are located in the internal memory of each PE. This is done by the Address Generator. Finally, the I/O Controller allows the communication with the computer and the external RAM. The control module is split into two blocks. The

microcontroller, besides instruction decodification, also controls the execution flow of the program. Therefore, this architecture includes two types of instructions, one set which controls the PEs, and an auxiliary set which runs in the microcontroller to manage the program flow and synchronize I/O.

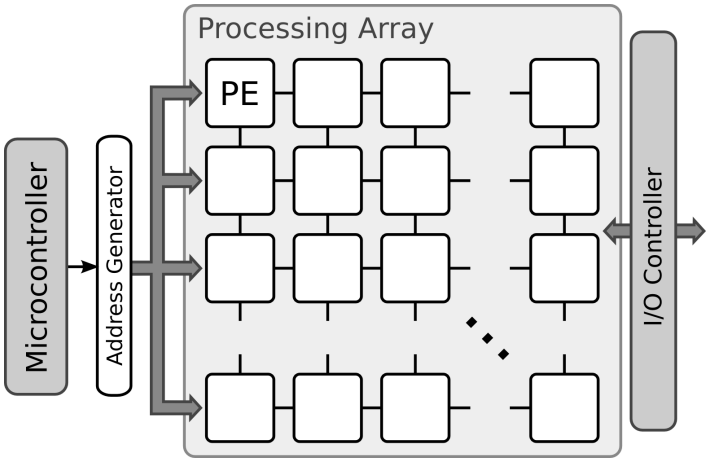


Figure 2.13: Top level view of the Coarse-Grain Processor Array architecture.

Concerning algorithm execution, once the image to be processed and the program are stored in the respective memories, the microcontroller issues the first instruction. This can be executed either by the microcontroller or by the processing array. The instruction is executed in the microcontroller either when it encodes a variable update, like a loop increment, or when it handles a branch. In both cases the processing array is idle. On the contrary, if the instruction is run on the processing array, all PEs start processing while the microcontroller remains idle until the array finished. It should be noted that in SIMD mode, every instruction is executed serially on each pixel of the sub-window managed by each PE. The microcontroller does not issue any other instruction until the whole sub-window is processed. Thus the image is completely updated for the next operation.

The Processing Element

The design of the PE is critical to meet the goals of the algorithm. It must be as simple as possible in order to reduce hardware requeriments, but it also must be powerful enough

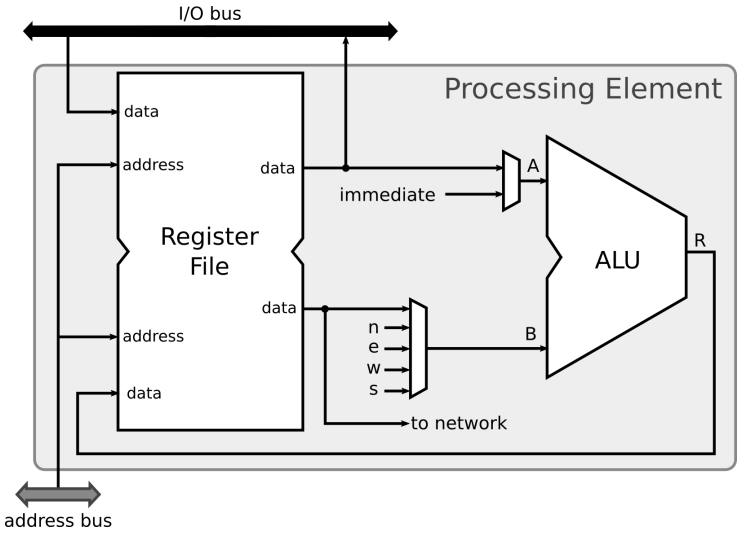


Figure 2.14: Internal architecture of the Processing Element of the Coarse-Grain Processor Array.

in order not to compromise performance. Figure 2.14 shows its schematic view. The main elements of each PE are a local memory, the Register File, and an Arithmetic Logic Unit (ALU).

The Register File of each PE is a Dual Port RAM that acts as a local memory. The Dual Port RAM provides two simultaneous outputs which supply data to the ALU. As the Dual Port RAM only has two input addresses, time multiplexing is needed if the instruction involves three operands. The execution of each instruction is done in two stages. First, data are read from the Register File using the addresses of operands A and B. In the second stage the result of the ALU is stored in the destination address, R, on port B. The write flag must be set adequately in both cases on port B. As it is shown in Figure 2.14, one of the operands (B) is always a pixel value and can come from the local memory of from the network. The other operand (A) is a local pixel value or an immediate value encoded in the instruction. This matches the instruction format shown in Figure 2.12.

Port A is used in a similar way to load the sub-window of the processing image into every PE. The data come from a global bus and is stored employing port A. The address and write

flags are controlled by the I/O Controller module (the latter displayed on Figure 2.13). The downloading of each sub-window of the processed image proceeds similarly.

The ALU provides operators to add, subtract, multiply with accumulation and the most basic Boolean operators. It is able to add and subtract two operands using the same hardware. As some operators such as Gaussian filtering employ fractional values, the multiplier is able to deal with fixed-point values. In order to simplify the hardware, word width is fixed, as it will be discussed later. The accumulator adds the result of the multiplication with the accumulated value, enabling MAC operations and guaranteeing the precision of the operation. The output of the MAC register is rounded when writing back. The thresholder uses the output of the subtracter to set the MSB bit of the pixel high, if $A > B$, or low when the opposite. Although it is possible to implement a threshold operation using a combination of other operators, it was included to increase performance. The thresholder and the logic gates AND, OR and NOT take the MSB bit of the operands as inputs and, as they make up a functionally complete logic, any binary algorithm can be implemented. The Identity operator, which simply puts on the output of operand B, allows data transfers between different PEs (e.g. to perform shifting). There is also possible to employ saturated arithmetic to ensure the result is in range, handling automatically underflow and overflow cases. Multiplication scales up/down the image using usually an immediate value (in fixed-point representation), as pixel-to-pixel multiplications are not present in the low-level image processing tasks we are considering.

The microcontroller

The microcontroller interacts with the Processing Array providing the information needed to process the image. It has to 1) provide the adequate instruction, handling the flow of the program, 2) decode the instruction and 3) set the correct flags to perform the computation. Figure 2.15 shows a schematic view of the microcontroller.

The microcontroller architecture is a simplified version of a PE but it has some substantial differences. First of all, it is not connected to the network. Secondly, the memory size is much smaller, having only small set of registers which store independent variables. In particular, the microcontroller stores the same number of variables as the PEs, allowing for a fixed-width instruction set with the same fields both for the microcontroller and the Processing Element and thus delivering lower hardware resources. As a final difference with the PE circuitry, the number of operations of the microcontroller is reduced, removing the multiplier and the binary operands as they are not required for flow control in this version of the architecture.

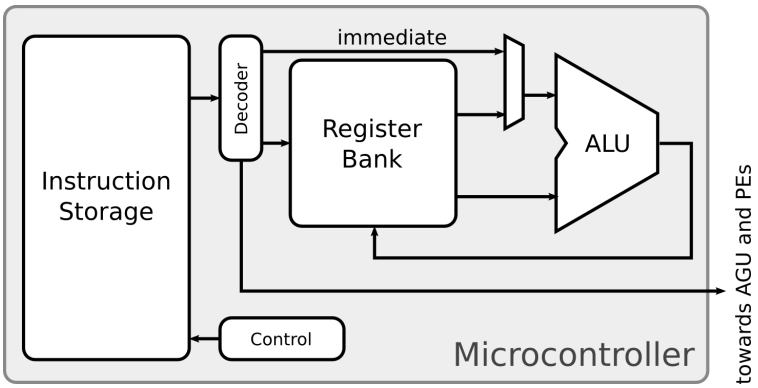


Figure 2.15: Microcontroller of the Coarse-Grain Processor Array.

To handle branches, a *zero* flag and the *set on less than* operator were added. They are used do decide if a branch must be performed or not, determining the address of the next instruction. A branch can be performed with regards to the current memory address, i.e. a relative branch, e.g. (*beq*, branch-if-equal, *bne* branch-if-not-equal instructions), or it could be an absolute branch (*j*, jump-unconditionally), i.e. a pointer to a predefined memory address. Thus it is possible to perform the basic flow control operations, as loops or the selective execution of code blocks.

A RAM block is used to store the program. A dedicated register, PC, stores the program counter, the address of the next instruction. The Register Bank stores integer values of temporal variables for loop and branch control, mainly. The ALU just includes an adder/subtracter and simple extensions for branch control. Auxiliary adders take care of PC updates.

A control unit manages and synchronizes all the modules of the system. This module can be summarized in the state machine shown in Figure 2.16. On each state, the write flags are set to their right values. In addition, it decodes the non-dependent state signals from the current instruction, as the ALU operation of the array or the source of operand B. This module also takes into account if a branch must be done and it sets the adequate flags in the microcontroller to load the correct instruction.

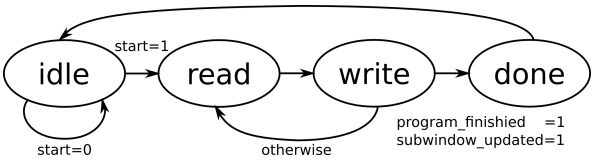


Figure 2.16: State machine of the Coarse-Grain Processor Array Microcontroller.

The Address Generator

Another goal of our design is to ease algorithm mapping onto the architecture. In addition to straightforward operators, which make it easier to translate an algorithm using the previously described assembly instructions, the platform must be easily scalable, i.e. the interaction with the platform should not depend on the number of PEs or the memory size. Bearing this in mind, the Address Generator is proposed.

Once a instruction is issued, it is executed over all the pixels of the sub-window. If the required data are outside the Register File, the source of the operand B must be set. In both cases, the programmer has to deal with low-level details of the architecture. To avoid this, the Address Generator self-manages the pointers to the data in the local memory of every PE, handling both cases.

The schematic view of the Address Generator is depicted in Figure 2.17. It has a counter which encodes the row and column of the current pixel. The instruction encodes the base address of each image in the memory of every PE (used to store the sub-windows of a whole image). For instance, if the sub-window has a 16×16px size, two successive images are separated by 256 memory positions so the counter is 8-bit width. As a result, the memory address of the pixel (x,y) is determined by the simple concatenation of the appropriate base address and the current row and column. The base address indicate which one of the sub-windows is selected.

A special case is the address of the operand B, whose source can be the local memory or an external value transferred through the network. In order to account for neighboring pixels, an additional instruction field is used, *shift* (see Figure 2.12). It encodes the direction, vertical or horizontal, and the amount of the shift in two's complement, so a 4-neigborhood is directly accessible due to the representation range. Two adders give the position of the shifted operand B. No overflow control is needed to calculate the B address, i.e. the new address is always

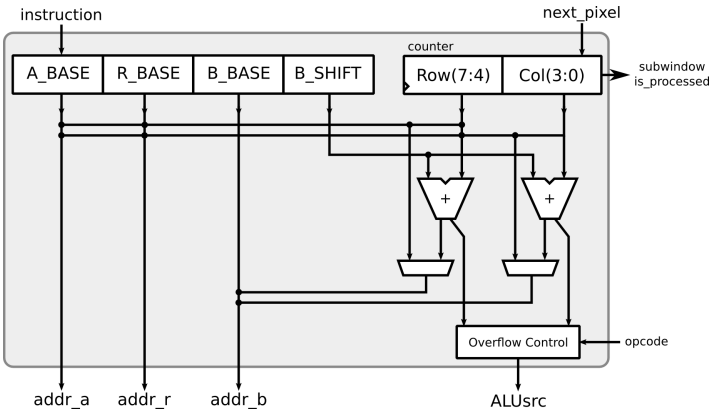


Figure 2.17: Internal datapath of the Address Generator of the Coarse-Grain Processor Array.

correct. However, it is needed to select the source of the operand because if an overflow or an underflow occurs, the data would be outside the local memory and the network should be used. In this way the system becomes self-governing, being able to handle any array and memory size.

In order to reduce the fan-out of the global signals, the calculated addresses are distributed through a bus, as shown in Figure 2.13, which is pipelined. This results in an increment up to a 60% on the clock frequency.

2.3 Case of study: retinal vessel-tree extraction

The image processor under study is conceived to handle low-level image processing tasks. Instead of designing a test-bench with different operators, a more practical approach is done. It is important to study not only the operators, but also the relationships between them when integrated in real-world applications. This is related with internal and external data storage. For instance, a processor can offer a poor performance executing a single operation and a high performance when linking together several of them, taking advantage of on-chip memory to avoid an intensive use of the external RAM, considerably slower. For this purpose a representative low-level algorithm was selected, a retinal vessel-tree extractor, designed focusing on parallelism and performance. It includes not only common tasks such as point-to-point, neighborhood or morphological operators but also recursive and data-dependent program flow. As

it can be drawn from this section, the algorithm satisfies all our requirements. The algorithm will permit to determine the advantages of the proposed architecture and the key aspects to be improved for a successful design. The algorithm is highly representative of the operations present in the early stages of a Computer Vision application. Therefore, it will also permits to verify if the minimum requirements in terms of flexibility and performance are met. The conclusions of this study will lead to a new architecture capable of dealing with more complex algorithms.

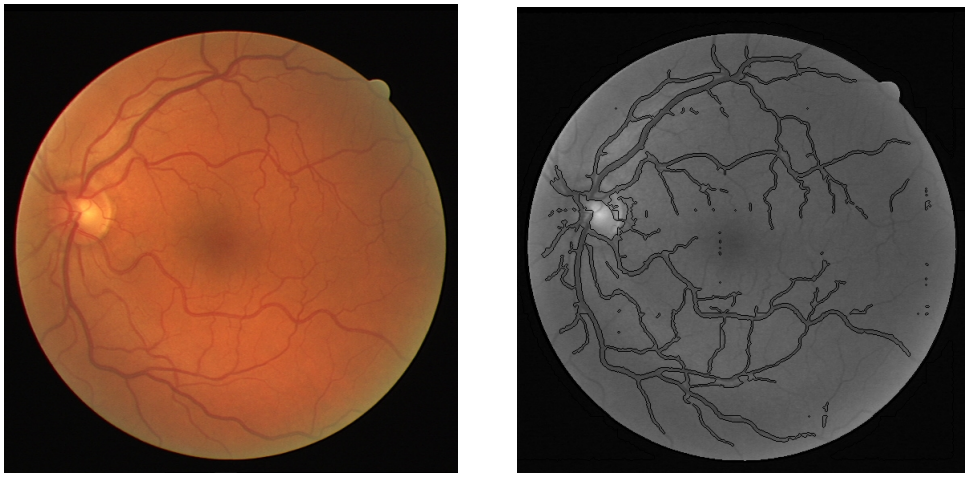
Retinal vessel-tree extraction is a very demanding computational task. It can be used in applications as early diagnoses of diseases like diabetes [149] or in person authentication [150]. In these practical operations, complex algorithms have to be processed fast. Usually, retinal images feature high resolution, so providing the vessel-tree requires thousands of operations at pixel-level. For instance, this processing step consumes more than 90% of the overall time performance in the person authentication application addressed in [151].

This retinal vessel tree extraction algorithm was proposed by Alonso-Montes et al. [152]. This technique uses a set of active contours that fit the external boundaries of the vessels. This is an advantage against other active contour-based techniques which start the contour evolution from inside the vessels. This way, narrow vessels are segmented without breakpoints and the central reflection in the widest ones is sorted out, providing better results. In addition, automatic initialization is more reliable, avoiding human interaction in the whole process. Figure 2.18 shows the result of applying the algorithm to a retinal image.

An active contour (or snake) is defined by a set of connected curves which delimit the outline of an object [153]. It may be visualized as a rubber band that will be deformed by the influence of constraints and forces trying to get the contour as close as possible to the object boundaries. The contour model attempts to minimize the energy associated to the snake. This energy is the sum of different terms:

- The internal energy, which controls the shape and the curvature of the snake.
- The external energy, which controls the snake movement to fit the object position.
- Other energies with the aim of increasing the robustness, derived from potentials (as the so-called inflated potential) or momenta (as the moment of inertia) [154].

The snake will reach the final position and shape when the sum of all these terms reaches a minimum. Several iterations are normally required to find this minimum. Each step is computationally expensive, so the global computational effort is quite high. Also, the placement of



(a) Input retinal image (b) Extracted vessel-tree overlaid to the input image

Figure 2.18: Retinal vessel-tree extraction algorithm applied over a test image.

the initial contour is very important in order to reduce the number of intermediate steps (lower computational load) and to increase the accuracy (less likely to fall into a local minimum). Although they can fit to local minima of energy positions instead of the real contour location and an accurate convergence criteria requires longer computation times, such techniques are widely used in image-processing tasks. Snakes or active contours offer advantages as easy manipulation with external forces, autonomous and self-adapting and tracking of several objects at a time.

There are several active contour models. Among the plethora of different proposals, the so-called Pixel-Level Snakes (PLS) [155] was selected. This model represents the contour as a set of connected pixels instead of a higher-level representation. In addition, the energy minimization rules are defined taking into account local data. This way, it will perform well in massively parallel processors because of its inherent parallelism. The algorithm operation is divided into two main steps: (1) initialize the active contours from an initial estimation of the position of vessels and (2) evolve the contour to fit the vessels.

The algorithm proved to be very efficient from the point of view of the operation. It was designed to perform pixel-parallel computing. Furthermore, the retinal images can be split in several sub-images which can be processed independently from each other. As it can be drawn

from this section, a Massively-Parallel (MP) SIMD architecture comes up as a natural choice to execute the low-level processing stages in this kind of applications. Its result is the input to the subsequent higher-level stages. In the same way, this algorithm summarizes most of the low-level image processing operations so it will be a method of evaluation of the proposed architecture.

2.3.1 Algorithm execution flow

One of the most important steps in active contours is initialization. Two input images are needed: the initial contour from which the algorithm will evolve and the guiding information, i.e., the external potential. Figure 2.19 summarizes this process.

The first task is intended to reduce noise and pre-estimate the vessels boundaries, from which the initial contours will be calculated. In so-doing, adaptive segmentation is performed, subtracting a heavily diffused version of the retinal image itself followed by a threshold by a fixed value, obtaining a binary map. To ensure that we are outside of the vessels location, some erosions are applied. The final image contains the initial contours.

The second task is to determine the guiding information, i.e., the external potential. It is estimated from the original and the pre-estimation vessels location images (calculated in the previous task). An edge-map is obtained by combining the boundaries extracted from those images. Dilating several times this map, diffusing the result and combining it with the original boundaries estimation will produce the external potential. It actually represents a distance map to the actual vessels position.

These two tasks are done only once. External potential is a constant during all the process. Once the active contours image is obtained, it is updated during the evolution steps.

As Figure 2.19 shows, PLS is executed twice for this concrete application. During the *fast PLS*, topological transformations are enabled so the active contours can be merged or split. This operation is needed to improve accuracy to remove isolated regions generated by the erosions required for the initial contour estimation. In this stage, the inflated potential is the main responsible of the evolution because the contour is far from the real vessels location and the rest of potentials are too weak to carry out this task. The aim of this stage is to evolve the contour to get it close to the vessels. It is called *fast* because a small number of iterations is needed. During the second PLS iteration, the *slow PLS*, topological transformations are disabled. The external potential is now in charge of the guidance of the contour evolution and the internal potential prevents the evolution through small cavities or discontinuities in the

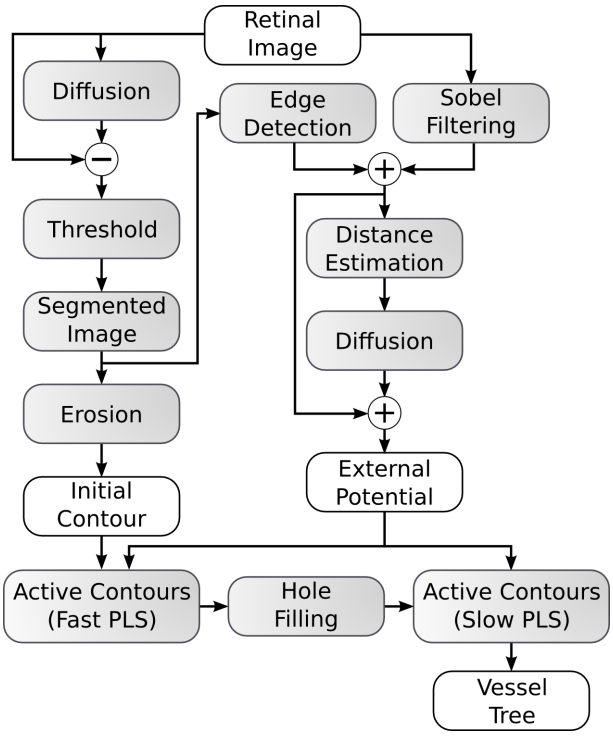


Figure 2.19: Block diagram of the retinal vessel-tree extraction algorithm.

vessels topology. The accuracy of the result depends deeply on this stage, so a higher number of iterations are needed (*slow evolution*). Between both stages, a hole-filling operation is included in order to meet greater accuracy, removing isolated holes inside the active contours.

2.3.2 Pixel-Level Snakes

It was commonly said that an active contour is represented as a spline. However, the approach selected here, the PLS, is a different technique. Instead of a high-level representation of the contour, this model uses a connected set of black pixels inside a binary image to represent the snake. We must note that a *black pixel* means a pixel *activated*, i.e., an active pixel of the contour. The contours evolve through an activation and deactivation of the contour pixels through the guidance of potential fields. This evolution is controlled by simple local rules. Its natural parallelism eases hardware implementations and it is one of its main advantages.

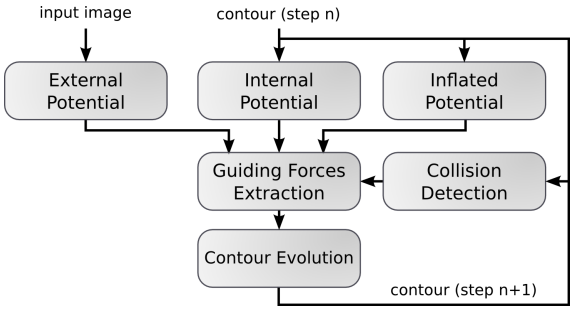


Figure 2.20: Overview of the Pixel-Level Snakes algorithm.

Figure 2.20 shows the main blocks of the PLS. First of all, the different potential fields must be computed:

- The external potential is application-dependent, so it must be an external input. This was discussed previously in this section. It is constant during all the evolution.
- The internal potential is calculated from the current state of the contour. Then it is diffused several times to obtain a topographic map that helps avoid abrupt changes in the shape of the contour.
- The inflated potential simply uses the current active contour, without any change. It produces inflating forces to guide the contour when the other potentials are too weak as is the case when the boundaries are trapped in local minima.

The involved potentials are weighed, each one by an application-dependent parameter, and added to build the global potential field. Active contours evolve in four directions: north, east, west and south (NEWS). Next algorithm steps are dependent on the considered direction, so four iterations are needed to complete a single evolution step.

The next step is to calculate a collision mask. The collision detection module enables topographic changes when two or more active contours collide. Topographic changes imply contour merging and splitting. This module uses a combination of morphological hit-and-miss operations, so only local access to neighbors is needed. The obtained image that contains pixels are forbidden to be accessed in the current evolution.

Listing 2.11: Pixel-Level Snakes pseudocode.

```
% Input:  Initial contour (C), External potential (EP)
% Output: Resulting contour (C)
% All variables are images (2DArray). All operations are performed
% over all pixels of the image before the execution continues.

void main() {
  C = HoleFilling(C);
  for (i = 0; i < iterations; i++) {
    IP = InternalPotential(contour);
    foreach (dir in (N, E, W, S)) {
      IF = InflatedPotential(C);
      CD = CollisionDetection(C, dir);
      GF = GuidingForce(EP, IP, IF, CD, dir);
      C  = ContourEvolution(GF, C, dir);
    }
  }
  C = BinaryEdges(C);
}

2DArray InternalPotential(C) {
  aux = BinaryEdges(C);
  IP = smooth(aux, times);
  return IP;
}

2DArray InflatedPotential(C) {
  IF = C;
  return IF;
}

2DArray CollisionDetection(C, dir) {
  if (enable) {
    if (dir == N) {
      aux1 = shift(C, S) & (~C);
      aux2 = shift(aux1, E);
      aux3 = shift(aux1, W);
      CD   = aux1 | aux2 | aux3
    } else {
      % Other directions are equivalent
```

```
    }
  } else {
    CD = zeros();
  }
  return CD;
}

2DArray GuidingForce(EP, IP, IF, CD, dir) {
  aux1 = EP + IP + IF;
  aux2 = aux1 - shift(aux1, dir);
  aux3 = threshold(aux2, 0);
  GF = aux3 & (~CD);
  return GF;
}

2DArray ContourEvolution(GF, C, dir) {
  aux = shift(C, dir) & GF;
  C = C | aux;
  return C;
}
```

During the guiding forces extraction step, a directional gradient is calculated from the global potential field. As this is a non-binary image, a thresholding operation is needed to obtain the pixels to which the contour will evolve. At this point, the mask obtained from the collision detection module is applied.

The final step is to perform the evolution itself (contour evolution module). The active contour is dilated in the desired direction using the information from the guiding forces extraction module.

Except when the potentials are involved, all the operations imply only binary images, so computation uses only Boolean operations. The pseudocode in Listing 2.11 shows all the steps. We have to remark that all the variables (except the iterators) are images, two-dimensional arrays. Each operation has to applied over all the pixels of the image before continuing with the next operation.

This is an adapted version of the PLS for the retinal vessel tree extraction algorithm. There are only stages of expansion. The way in which the contour is initialized ensures that alternating phases of expansion/contraction required in any other active contours method is

	Initialization	Fast PLS	Hole filling	Slow PLS
Pixel-to-pixel				
Arithmetic	7	5	–	7
Boolean	1	4	–	9
Pixel-to-neighborhood				
2D filters	11	8	–	8
Binary masks	10	2	1	5

Table 2.5: Type and number of operations per pixel per step of each task.

	# iterations	Operations per px
Initialization	1	189
Fast PLS	6	697
Hole filling	18	199
Slow PLS	40	5761
Total		6846

Table 2.6: Number of operations per pixel. Pixel-to-neighborhood operations shown in Table 2.5 are transformed to pixel-to-pixel operations. This includes program flow operations.

not necessary here, which simplifies the code and increases performance. Further details of this active contours-based technique can be found in [155, 156, 157].

2.3.3 Performance remarks

As it can be extracted from the above, the algorithm has an inherent massively spatial parallelism which can be exploited to improve the throughput. Additionally, the image can be split into multiple sub-windows and can be processed independently without decreasing the accuracy [158]. In addition, the required precision for the data representation is low (see [152]), so the accuracy will not be seriously affected by this parameter. All these advantages will be exploited during the algorithm migration to the hardware platform. One of the drawbacks of this extraction method is that it is hard to exploit temporal parallelism as the heaviest computational effort comes from the PLS evolution, where each iteration directly depends on the previous one. This forces to execute all the steps serially.

Table 2.5 summarizes the type of operations present in the algorithm per pixel of the image and iteration of the given task. Table 2.6 sums up the total number of operations including

the number of iterations per task and program flow-related tasks. The number of iterations was determined experimentally and agrees with the worst case of those studied to ensure the convergence of the contours. To evaluate the efficiency of the implementation, the DRIVE database was used [159]. The retinal images were captured with a Canon CR5 non-mydriatric 3CCD. They are 8-bit three channel color images with a size of 768×584px. Considering this image size and that by each pixel 6846 operations must be performed, around 3 giga-operations are required to process the entire image. The operations of this algorithm are very representative of image-processing operations which are part of the low- and mid-level stages. They comprise operations as filtering, basic arithmetics, logic operations, mask applications or basic program flow data dependencies. Any image-processing-oriented hardware must deal properly with all these tasks.

The retinal vessel tree extraction algorithm was tested employing a PC-based solution. It was developed using OpenCV and C++ on a computer equipped with an Intel Core i7 940 working at 2.93 GHz (4 physical cores running 8 threads) and 6 GB of DDR3 working at 1.6 GHz and in triple-channel configuration. Using this computer, each image requires more than 13 s to be processed. This implementation makes use of the native SSE support which OpenCV offers. To take advantage of the multi-core CPU, OpenMP was used to parallelize loops and some critical blocks of the algorithm which are implemented outside the OpenCV framework. This allows around a 15% higher performance. Previous implementations using MatLab [160] require more than 40 s to extract the retinal vessel-tree. Although it would be possible to do certain optimizations to further improve performance, it would be very difficult to achieve times below 10 s. This is because the algorithm is not designed to run on such architectures, not because of the algorithmic complexity, but due to the large number of memory accesses required, not present in fine-grain processors.

Even with a high-end computer, the result is not satisfactory in terms of speed. Candidate systems using this algorithm require a faster response. To address this and other problems associated with a conventional PC, such as size or power consumption, a dedicated processor or an embedded device attached to the camera is a requirement. It is expected that MP-SIMD architectures will perform better than other approaches. This point will be discussed later in Section 2.5.

2.4 Performance evaluation

2.4.1 FPGA prototyping and validation

The system chosen for the validation is an XEM3050 card from Opal Kelly [161] with a High-speed USB 2.0 board and a Xilinx Spartan-3 FPGA. The most remarkable features of this FGPA are 6912 CLBs or 62208 equivalent logic cells (1 Logic Cell = 4-input LUT and a D-flip-flop), 96×18 Kb embedded RAM blocks (where 2 Kb are parity bits) and 520 Kb of Distributed RAM, 96 dedicated 18-bit multipliers and a Speed Grade of -5. This FPGA uses 90 nm process technology. The board also includes two independent 32MB SDRAM units. The implementation has been developed with VHDL and synthesized with the Xilinx ISE 10.1 [142]. This board was chosen aiming to lower the cost of the prototype at the expense of lower performance, as the Xilinx Spartan-3 family was designed focusing on cost-sensitive and high volume consumer electronic applications.

The synthesized array on this FPGA has a resolution of 9×10 (90) Processing Elements and is able to handle a window of 144×160px with 8-bits of resolution per pixel. The original implementation of the algorithm demonstrated that this word width provides sufficient precision for practical implementations. This will be discussed later in Section 2.5.3. Fixed-point representation handles only 4-bits for the decimal part, providing adequate accuracy for low-level image processing arithmetic. The Register File of each PE handles 8 different sub-windows of 16×16px each (16 Kbit per PE). The ALU contains an embedded multiplier. The microcontroller stores 8 independent integer variables too, and up to 512 instructions. The resources consumed when implementing the architecture in the Spartan-3 FPGA are indicated in Table 2.7. As we can see, there are still resources available on the FPGA chip, although the usage of RAM Blocks and Multipliers limits the number of PEs we can embed. Concerning speed, the highest clock frequency attained by the implementation was set to 53 MHz.

Besides to the proposed architecture, an additional module was included to handle I/O. Opal Kelly provides a proprietary high-speed USB interface (a Cypress FX2LP - CY68013A [162]) to communicate the board with the computer. The amount of resources needed for synchronization and data buffering (previous to its storage in the PEs local memory) is quite reduced, and it only consumes LUTs. An additional 18 Kbit RAM Block is needed to pack the input and output buffers. Image data are transferred to the PEs sequentially, employing a dedicated pipelined bus, as discussed previously. External RAM is not used, as we are only evaluating the performance of the Processing Array.

	Spartan-3 xc3s4000-5
Array size	9×10 (90)
LUTs	10195/62208 (18%)
RAM Blocks	91/96
Multipliers/DSP Blocks	90/96
Max. Frequency (MHz)	53.0

Table 2.7: Implementation results on the Xilinx Spartan-3 FPGA.

	Spartan-3 xc3s4000-5
Window size (px)	144×160
Image size (px)	768×584
Required windows	20
Window process time (ms)	66.1
Total processing time (s)	1.323
Total time with I/O (s)	1.349

Table 2.8: Summary of the overall time execution on the Xilinx Spartan-3 FPGA.

2.4.2 Algorithm evaluation

The inherent parallelism of the retinal vessel tree algorithm makes its hardware implementation simple with high performance. This algorithm matches with a processor-per-pixel scheme, where the processors employ local communications for neighborhood access. The implementation of the algorithm in a fine-grain processor array is straightforward. The same applies to the coarse-grain processor array, although in this case each processor handles a subwindow of the input image. Thanks to the Address Generator unit, programming the processor array is independent of its size, and the network access is self-managed. However, there is a limitation in terms of resolution. As the size of the array is lower than the size of the image, it must be split into several sub-windows that are processed independently.

The algorithm was implemented completely trustworthy to the original algorithm proposal, without any changes that might result from the particularities of the architecture. Instruction set and processors topology match algorithm operations. However, it is still needed to split the input images into several sub-windows. Table 2.8 shows the overall time execution on the Spartan-3 FPGA. A comparison with other devices and a discussion of the results is done in Section 2.5.

2.4.3 Architectural improvements

In the light of the obtained results, an enhanced version of the architecture is proposed. This new version does not change the original concept, although the internal datapath of the Processing Element includes some minor features. One of the major drawbacks of the current implementation is the limited use of the Register File, which only permits two simultaneous accesses. Instructions which employ three operands take two clock cycles, as Figure 2.16 shows. This considerably increases the computation time. Apart from other minor and technical enhancements, the datapath was modified by adding an additional stage to the pipeline. This is done for two reasons: to increase the clock frequency, highly limited due to the NEWS routing and the ALU's internal datapath length, and to provide more flexibility to overcome the limitations of the Register File implementation. The control unit of the microcontroller is now able to handle data-hazards when the instructions include three operands, so instructions only take additional cycles when necessary.

The Xilinx Spartan-3 FPGA was selected to lower the cost of the final device. However, this also leads to less available hardware resources and lower performance. The target platform for the new implementation of the architecture is a Xilinx Virtex-6 xcv6vlx240t-1 FPGA [163], included on the Xilinx ML605 Base Board. The selected FPGA has an intermediate size within the Virtex-6 family but provides enough resources to evaluate the architecture without sacrificing features when adapting the design to the limitations of the device. The FPGA provides 37680 slices (each one containing four 6-input LUTs and eight flip-flops), 416 Block RAMs (36Kb each) and 768 DSP48E1 slices (25×18 two's complement multiplier/accumulator with pre-adder, Boolean functions and barrel shifting capabilities) as the most relevant resources.

The synthesized array on the Virtex-6 FPGA has a resolution of 16×12 (192) Processing Elements, handling a window of 384×256px with 8-bits of resolution per pixel. The large amount of resources of this FPGA permits to double the array size and to quadruple the window resolution. As in the Spartan-3 implementation, fixed-point representation with 4-bits for the decimal part is employed. The Register File of each PE also handles 8 different sub-windows of 32×16px each (32Kbit per PE), and it is implemented using a dedicated 36Kbit Dual-Port Block RAM. A DSP48E1 slice is used instead of the simple multiplier of the Spartan-3 devices, so the internal adder/subtracter can be employed, saving LUTs for other purposes. With these parameters and the included improvements, the maximum clock

	Spartan-3 xc3s4000-5	**Virtex-6 xcv6vlx240t-1**
Array size	9×10 (90)	12×16 (192)
LUTs	10195/62208 (18%)	31651/150720 (21%)
RAM Blocks	91/96	193/416
Multipliers/DSP Blocks	90/96	192/768
Max. Frequency (MHz)	53.0	150.0

Table 2.9: Implementation results on Xilinx FPGAs. *Note: Spartan-3 uses 4-input LUTs. Virtex-6 has 6-input LUTs and the enhanced datapath described in Section 2.4.3.*

	Spartan-3 xc3s4000-5	**Virtex-6 xcv6vlx240t-1**
Window size (px)	144×160	384×256
Image size (px)	768×584	768×584
Required windows	20	4
Window process time (ms)	66.1	30.8
Total processing time (s)	1.323	0.123
Total time with I/O (s)	1.349	0.126

Table 2.10: Summary of the overall time execution on both Spartan-3 and Virtex-6 FPGAs.

frequency is 150 MHz. Table 2.9 summarizes the implementation results on both Spartan-3 and Virtex-6 FPGAs.

Table 2.10 shows the overall execution times of both implementations. One of the advantages of the architecture is that the program does not depend on the array size thanks to the inclusion of the Address Generator unit and that the microcontroller is able to handle automatically data-hazards, so the same code is executed on both devices without modifications. Besides the increase of the array resolution, thus requiring to split the input image in less windows, the enhanced datapath allows to triple the clock frequency. This results in a larger performance increase, up to 10 times higher than the original approach.

2.5 Comparison with other approaches

In many applications, accuracy is a requirement. However, in some of them, the computational effort is also the main issue. The retinal vessel-tree extraction algorithm was designed specifically for its utilization in fine-grained SIMD architectures with the purpose of improving the computation time. The algorithm has been tested on a massively parallel processor,

which features a correspondence of a processor-per-pixel. This solution provides the highest theoretic performance. However, when using off-the-shelf devices, we have to face certain limitations imposed by the technology (i.e. integration density, noise or accuracy), so the results might be worse than expected. At this point, other *a priori* less suitable solutions can provide similar or even better performance.

The algorithm can process the image quickly and efficiently, making it possible to operate online. This speeds up the work of the medical experts because it allows not only to have immediate results, but also they can change parameters in real-time observation, improving the diagnosis. It also reduces the cost of the infrastructure, as it is not necessary to use workstations for processing. The algorithm can be integrated into a device with low cost, low form factor and low power consumption. This opens the possibility of using the algorithm outside the medical field, for example, in biometric systems [151].

Although the algorithm was designed for massively parallel SIMD (MP-SIMD) processors, it can be also migrated to other devices. DSPs or GPUs provide good results in common image-processing tasks. However, reconfigurable hardware or custom ASICs solutions permit to improve the matching between architecture and image-processing algorithms, exploiting the features of vision computing, and thus potentially leading to better performance solutions. In this section, we want to analyze devices that allow us to fully integrate the entire system on an embedded low power device. The algorithm under study was designed to operate online, immediately after the stage of image capture and integrated into the system, and not for off-line processing, so we select devices that allow this kind of integration. DSPs are a viable solution, but we cannot take advantage of the massively parallelism of the algorithm. On the other hand, the high power consumption of GPUs discards these for standalone systems [164].

Among the plethora of different platforms that today offer hardware reconfigurability, this section focuses on the suitability of FPGAs and massively parallel processor arrays (MPPA) for computer vision. FPGAs are widely used as prototyping devices and even final solutions for image-processing tasks [165, 166]. Their degree of parallelism is much lower than what an MP-SIMD provides, but they feature higher clock frequencies and flexible data representations, so comparable results are expected. Advances in the miniaturization of the transistors allow higher integration densities, so designers can include more and more features on their chips [167]. MPPAs are a clear example of this because until a few years ago, it was not possible to integrate several hundred microprocessors, even if they were very simple. These

devices are characterized by a different computation paradigm, focusing on exploiting the task parallelism of the algorithms [168].

This section shows the results of the implementation of the automatic method to extract the vessel-tree from retinal images we are using as benchmark. The architecture presented in Section 2.2 on an FPGA is compared with the native platform of the algorithm, an MP-SIMD processor, and a completely opposed computing paradigm, an MPPA. This way we cover massively parallelism, coarse-grain parallelism and temporal (task) parallelism respectively.

2.5.1 Pixel-Parallel Processor Arrays

Conventional image-processing systems (which integrate a camera and a digital processor) have many issues for application in general-purpose consumer electronic products: cost, power consumption, size and complexity. One of the main disadvantages is the data transmission bottlenecks between the camera, the processor and the memory. In addition, low-level image-processing operations have a high and inherent parallelism which only can be exploited if the access to data is not heavily restricted. Computer Vision is one of the most intensive data processing fields, and conventional systems do not provide any mechanism to address adequately this task, so this issue comes up as an important drawback.

Pixel-parallel processor arrays aim to be the natural platform for low-level image-processing and pixel-parallel algorithms. They are MP-SIMD processors laid down in a 2D grid with a processor-per-pixel correspondence and local connections among neighbors. Each processor includes an image sensor, so the I/O bottleneck between the sensor and the processor is eliminated, and the performance and power consumption are highly improved. This and their massively parallelism are the main benefits of these devices.

Pixel-parallel processor arrays operate in SIMD mode, where all the processors execute simultaneously the same instruction on their local set of data. To exchange information, they use a local interconnection, normally present only between the nearest processors to save silicon area. Concerning each processor, although with local memories, data I/O and sensing control to be self-contained, they are as simple as possible in order to reduce area requirements, but still powerful enough to be general purpose. The idea behind these devices is that the entire computing is done on-chip, so that input data are logged in through the sensors and the output data are a reduced and symbolic representation of the information, with low-bandwidth requirements.

One of the drawbacks of this approach is the reduced integration density. The size of the processors must be as small as possible because: for a 256×256 px image, more than 65k processors plus interconnections must be embedded in a reduced area. This is the reason why many approaches utilize analog or mixed-signal implementations, where the area can be heavily optimized. Nevertheless, accuracy is its main drawback because it is hard to achieve large data-word sizes. In addition, a careful design must be done, implying larger design periods and higher economic costs. The scalability with the technology is not straightforward because of the human intervention in all the process, which does not allow automation. The size of the arrays is also limited by capability of distributing the signals across the array. The effective size of the arrays forces us to use low-resolution images. Examples of mixed-mode focal-plane processors are the Eye-Ris vision system [169] or the programmable artificial retina [170].

Other approaches use digital implementations with the aim to solve the lack of functionality, programmability, precision and noise robustness. The ASPA processor [171] and the design proposed by Komuro et al. [172] are examples of this kind of implementations.

As each processor includes a photo-sensor, it should occupy a large proportion of the area to receive as much light as possible. However, this will reduce the integration density. New improvements in the semiconductor industry enables three-dimensional integration technology [173]. This introduces a new way to build visions system adding new degrees of freedom to the design process. For instance, [174] proposes a 3D analog processor with a structure similar to the eye retina (sensor, bipolar cells and ganglion cells layers, with vertical connections between them) and [175] presents a mixed-signal focal-plane processor array with digital processors, also segmented in layers.

As a representative device of this category, the SCAMP-3 Vision Chip was selected to map the retinal vessel tree extraction algorithm described in Section 2.3.

The SCAMP-3 processor

The SCAMP-3 Vision Chip prototype [176] is a 128×128px cellular processor array. It includes a processor-per-pixel in a mixed-mode architecture. Each processor, an Analog Processing Element (APE), operates in the same manner as a common digital processor but working with analog data. It also includes a photo-sensor and the capability to communicate with others APEs across a fixed network. This network enables data sharing between the nearest neighbors of each APE: NEWS. All processors work simultaneously in SIMD manner.

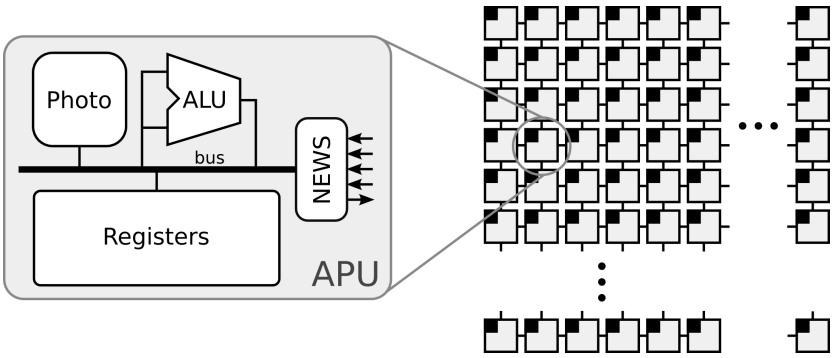


Figure 2.21: Overview of the SCAMP-3 main elements.

Figure 2.21 shows its basics elements. Each APE includes a photo-sensor (Photo), an 8 analog register bank, an arithmetic and logic unit (ALU) and a network register (NEWS). A global bus connects all the modules. All APEs are connected through a NEWS network, but the array also includes row and column address decoders to access to the processors and extract the results. The data output is stored in a dedicated register (not shown).

Operations are done using switched-current memories, allowing arithmetic operation and enabling general-purpose computing. As current mode is used, many arithmetic operations can be done without extra hardware [177]. For example, to add two values, a simple node between two wires is needed (Kirchhoff's law).

The SCAMP-3 was manufactured using $0.5\mu m$ CMOS technology. It works at 1.25 MHz consuming 240 mW with a maximum computational power of 20 GOPS. Higher performance can be achieved by increasing the frequency, at the expense of a higher power consumption. Using this technology, a density of 410 APEs / mm^2 is reached (less than $50\ \mu m \times 50\ \mu m$ per APE).

Implementation

The implementation of the retinal vessel tree extraction algorithm is straightforward. The selected algorithm, as well as other operations present in the low- and mid-level image-processing stages, matches well with this kind of architectures. The SCAMP features a specific programming language and a simulator to test the programs which speeds up the process. For instance, the simulator allows to select the accuracy level of the operations, allowing fo-

cusing first on the program functionality and then on the precision of the implementation. This is a necessary step to solve the problem caused by not so accurate memories. Specific details, specially those referred to current-mode arithmetic can be found in [177].

However, some modifications had to be added to the algorithm because of the particularities of the device. Some operations were added to increase the accuracy of the algorithm. The volatility and the errors due to mismatch effects during the manufacture of the memories must be taken into account and the SCAMP has methods to improve the results. The distance estimation during the external potential estimation and accumulated adding are operations that need carefully revision due to the short retention time of the switched-current memories.

Other point to take into account is the data input. While for many applications the optical input is the best option, for other applications, where the images are high resolution or the photo-detectors are not adequate to sense the images, a mechanism to upload the image is needed. However, one of the greatest benefits of these devices is lost, the elimination of the bottleneck between the sensing and processing steps. For instance, to integrate the APEs with the sensors of the camera used for the retinal image capture (a Canon CR5 non-mydriatric 3CCD [159]) will provide better results.

It has to be noted that in the SCAMP-3, the size of the array is much lower than the size of the utilized images. The input images can be resized, but the result will be seriously affected. This forces us to split the image into several sub-images and process it independently. As it was mentioned in Section 2.3, this algorithm allows to consider the sub-images as independent without affecting the quality of the results. However, it affects the performance and it can be not applicable on other algorithms, highlighting the problems of these devices when their size is not easily scalable. More details of the implementation can be found in [178].

2.5.2 Massively Parallel Processor Arrays

MPPA provides hundreds or even thousands of processors. Each unit is encapsulated and works independently, so they have their own program and data memories. All units are connected to a programmable interconnection, enabling data exchange between them. These are usually point-to-point channels controlled by a message passing system which allows its synchronization. MPAAs also include distributed memories which are connected to the network using the same channels. This independent memories will help during the development process to store data when the local memory of each processor is not enough or to build FIFO queues, for instance.

The main differences between MPPA and multicore or manycore architectures are the number of processing units (which traditionally was much higher, though latest GPUs have increasingly computational units, making them comparable), their originally conceived general-purpose character and that they do not include a shared memory (as is the case of symmetric multiprocessing) [179].

They are focused on exploiting the functional and temporal parallelism of the algorithms instead of the spatial parallelism (as happened with the Pixel-parallel processor array). The idea is to split the original algorithm into several subtasks and match each with one or several processors of the array. Each processor executes sequential code, a module of the algorithm or the application. The different modules are connected together using the channels in a similar way of a flow diagram of an algorithm. This way they attempt to solve the bottleneck between processors and the external memory and avoid the need to load all data at a time while the functional units are stopped. Stream computing has been proved to be very efficient [168]. The parallelism is obtained running different modules in parallel. However, processors can include internal SIMD units making them even more powerful and flexible.

As they are encapsulated, higher working frequencies can be achieved, making them competitive despite the lower parallelism level if we compare them with a cellular processor, for example. The use of multiple computational units without explicitly managing allocation, synchronization or communication among those units are also one of its major advantages. This is one of the goals of MPPAs, to ease the development process. As all these processes can be done (commonly) through a friendly high-level programming language, some authors say that MPPAs are the next evolution of FPGAs [180]. Designers are increasingly demanding high-performance units to address parts of the application which are difficult to map on a pure-hardware implementation. This is one of the reasons why future FPGAs will include high-end embedded microprocessors [181]. MPPAs already provide this capability including a standard interface with the rest of modules of the system. Dedicated hardware as this will cut down power consumption and hardware resources while performance will be heavily increased. However, FPGAs are still faster for intensive computing applications [182].

As remarkable examples of MPPAs devices, we should cite the picoChip [85], the Tilera Processor [183] or the PARO-design system [184].

The Ambric Am2045 processor

The selected platform where to migrate the retinal vessel tree extraction algorithm is the parallel processor Am2045 from Ambric [180]. It is made up of a large set of fully encapsulated 360 32-bit RISC processors and 360 distributed memories. Each RISC processor runs its own code, a subtask of the complete algorithm. A set of internal interconnections allows data exchange between them. Synchronization between processors is done automatically through a flexible channel hierarchy. A simple handshake and local protocol between registers enables synchronization without intermediate logic, as it happens when using FIFOs. A chain of those registers is known as a *channel*. Figure 2.22 shows the main blocks of this architecture.

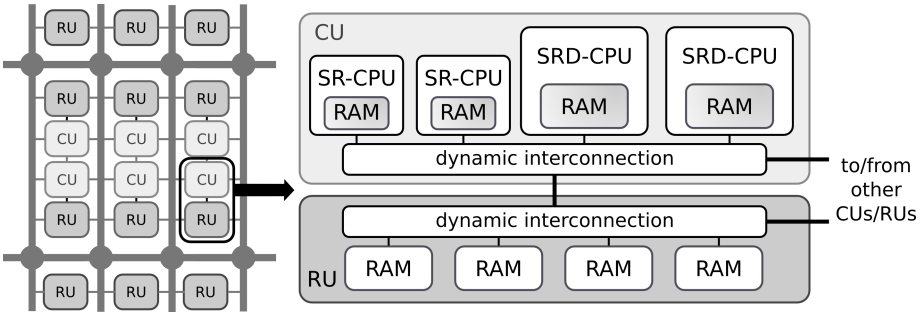


Figure 2.22: Ambric architecture overview: Computational Units (CU) and RAM Units (RU).

Ambric uses two kinds of processors, SR and SRD. Both are 32-bit RISC processors specially designed for streaming operations. SRD CPU enables instruction and data-level parallelism. Sub-word logical and integer operations as well as fixed point operations are also possible. SRD processors also include 3 ALUs, two of which work in parallel, and a 256-word local memory. SR CPUs are a simpler version of SRDs, specially designed to deal with simple tasks where DSP extensions are not needed. They only have an ALU and a 64-word local memory. A group of 2 SRD and 2 SR CPUs is known as a Compute Unit (CU) and includes a local interconnection to let direct access between them.

Distributed memory is organized in modules known as RAM Units (RU). Each RU has 4 banks of 256 words connected together through a dynamic configured interconnection. There are also direct links between the RU and the SRD CPUs (not shown in Figure 2.22) which provide random access or FIFO queues both for instructions and for data to increase performance and flexibility.

A group of 2 CU and 2 RU is called Brick. Bricks are connected using a distant reconfigurable channel network, which works at a fixed frequency.

The Am2045 chip uses 130-nm standard-cell technology and provides 360 32-bit processing elements and a total of 4.6 Mb of distributed RAM, working at a maximum frequency of 333 MHz and featuring a power consumption about 10 W. It also includes two DDR2-400 SDRAM interfaces and a 4-lane PCI-Express, to enable fast internal and external I/O transactions. This device does not feature shared memory, but it has an external memory that can only access certain elements that control the I/O to the chain of processors which map the algorithm. A more detailed description of the hardware can be found in [180].

Implementation

The computational paradigm in Ambric's device is completely opposed to the two former implementations addressed in this paper. While using cellular or coarse-grain processors, we were focusing on the characteristics of the algorithm that allow to exploit its massive spatial parallelism, now this is not suitable. Although with stream processors it is possible to implement applications in a pure SIMD fashion, it is more appropriate to modify the algorithm and make certain concessions in order not to compromise performance. For instance, the iterative nature of operations as the hole-filling are very expensive in terms of both time and hardware resources (number of processors). This is one of the drawbacks of the computational paradigm used by this platform. Recursive operations are quite resource consuming because each iteration requires to replicate the set of processors which implements the operation. This is not a strict rule, and there are other approaches that can address this problem differently through a reorganization of operations and modules. However, most low-level and much of the mid-level operations have data dependencies that oblige to complete the previous operation over the whole or most part of the data set before applying the next operation.

Figure 2.23 summarizes the algorithm mapped on the Am2045 device. 16-bit instead of 8-bit words are used to guarantee accuracy when computing partial results. The SIMD capabilities of the SRD processors are also used. This is specially advantageous for binary images because 32 pixels can be processed at a time, increasing greatly the performance during the PLS evolution. Many operations can be seen as 2D convolutions, split into vertical and horizontal filters. While for horizontal filtering the implementation is straightforward, for vertical filtering the local memories must be used to store the previous rows. FIFO queues balance the paths and avoid internal blockages during the processor communication. Although

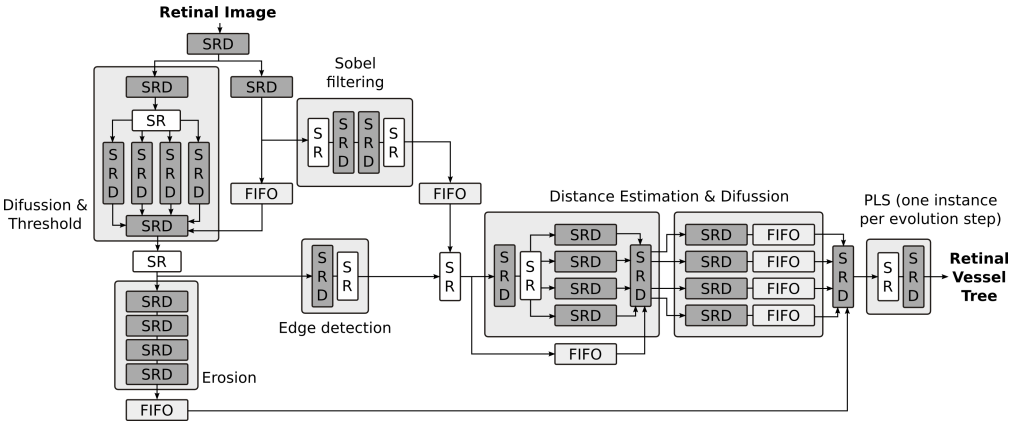


Figure 2.23: Retinal vessel tree extraction algorithm mapped in Ambric Am2045 device.

synchronization is automatic, it may occur that some paths were much faster than others. For example, consider a processor which supplies data to two processing paths, one much more slower than the other and a second processor where both paths are combined. When a processor tries to read from an empty channel, a stall occurs. FIFO queues avoid those stalls, but a bottleneck analysis is necessary.

Some operations of the algorithm were not implemented. The hole-filling step, located between slow and fast PLS evolutions and introduced to improve results (see Figure 2.19), was removed because of its heavy resource consumption. In the same way, some internal improvements in the PLS were eliminated, as the internal potential. This greatly simplifies the implementation of the PLS, leading to binary operations only. Each PLS step requires just one processor, and there is no need to store partial results, leading to a large increase in performance. Otherwise, PLS step will require more processors to be implemented. Although normally this is not a problem, when performing a large number of iterations, it will be something to consider. This leads to an accuracy reduction in the results that, depending on the application, may make the results invalid (see Table 2.12). This point is discussed in depth in Section 2.5.3.

On the other hand, to tune up this kind of operations is tough because the chain structure (see Figure 2.23) must be modified. Recursive operations over the same set of data make

the implementation in stream processors much more complex than in cellular processors, and these operations are common during the first stages of image processing [185].

2.5.3 Results and comparison

The main results of the algorithm implementation on the different platforms are summarized in Table 2.11. We can see that even with a high-end CPU, the results are not satisfactory. Systems running this kind of algorithms usually are required for a rapid response, something that a PC hardly can provide.

As it was detailed in Section 2.3, it was developed using OpenCV/OpenMP and C++ on a computer equipped with a 4-core Intel Core i7 940. This way, compared with the initial straightforward MATLAB implementation (for test purposes) time execution drops from 41 to 13.7 s. As explained above, this algorithm was designed to work on-line following the capture stage, so a PC is not a good choice. However, we will use it as a reference system for comparison.

When executing the retinal vessel tree extraction algorithm, we were not only seeking ways to reduce the processing time but also we wanted to test different platforms and determine their strengths. This way, SCAMP-3 implementation adds operations to improve accuracy (necessary when using analog memories), FPGA gets the most accurate results because it is possible to implement the original method trustworthy (at the cost of reducing performance) and Ambric's device provides the fastest results but reducing the accuracy (required hardware resources would be very high otherwise).

It can be seen in Table 2.11 that Ambric's device gives the highest performance, lowering the total execution time and achieving a high speed-up. This is in part due to the simplifications we made in the algorithm as it was explained previously in Section 2.5.2. This implementation enables a performance not achievable by computers or FPGAs (at least using a low-cost device). We must emphasize that the simplification of the algorithm provides accurate and valid results, but they may not be suitable for certain applications. For instance, they are still valid to obtain the skeleton of the retinal tree but not to measure the vascular caliber. When migrating the algorithm to this platform, we have faced a trade-off between speed and validity of the results for any application and in this case and for comparison purposes, we give priority to the processing speed.

	Intel Core i7 940	**SCAMP-3**	**Ambric Am2045**	**Spartan-3 FPGA**	**Virtex-6 FPGA**
Window size (px)	–	128×128	–	144×160	384×256
Processors (used/available)	4/4	16,384/16,384	125/360	90/90	192/192
Working frequency (MHz)	2930	1.25	333	53	150
Window execution (ms)	–	6.55	–	66.1	30.8
Required windows	1	30	1	20	4
Computation time (s)	13.6	0.193	0.008	1.323	0.123
Execution time (w/ IO (s))	13.7	0.230	0.0087	1.349	0.126
Speed-up	1x	59.6x	1574.7x*	10.3x	110.6x
Cycles-per-pixel (w/o IO)	357,993	8,950	742*	14,070	7898

Table 2.11: Most relevant results of the retinal vessel tree extraction algorithm implementation on the different devices. **It should be noted that the Ambric Am2045 runs a simplified version of the algorithm. See Section 2.5.2 for details.*

The other main reason for the high performance of the Ambric's device is the high working frequency of the Am2045 device (333 MHz), which is considerably faster than those achieved by the SCAMP or the FPGA. Digital solutions provide higher clock frequencies but they are commonly limited by the interconnection between the processing units, as is the case of the FPGA. MPPAs architectures implement point to point connections with minor reconfigurable options than FPGAs, so clock frequency does not depend on the algorithm which is being running. However, recent FPGAs families increase its computing capacity considerably. This was observed during the implementation of the coarse-grain processor array. While the Spartan-3 offers good performance compared to a desktop PC, it can not compete against an ASIC. However, a newer FPGA as the Virtex-6 largely improves the throughput without compromissing accuracy in the original algorithm as the Ambric device does. It should be taken into account that the Virtex-6 implements a larger array and an enhanced datapath, being this possible thanks to that the selected FPGA family focuses on performance rather than cost.

The cycles-per-pixel (CPP) metric measures the number of clock cycles required to execute all operations that result on each one of the pixels in the resulting image (see Table 2.6), normalizing the differences in frequency and number of processing cores. The above discussion is summarized using this value. It should be noted that the Ambric Am2045 runs a simplified version of the algorithm. When the number of operations is corrected, this leads us to an obvious reduction in performance, approximately multiplying by 3 the CPP value. Although the theoretical performance would remain higher than the other approaches, the problem we face is different: there is not enough processors and interconnections to fit the algorithm in the device. This was the main reason why it was decided to simplify it.

Analog computing increases greatly the density of integration. In the SCAMP-3 with a relatively old $0.5\mu m$ CMOS technology, each processing element occupies an area lower than $50\mu\text{m} \times 50\mu\text{m}$ and it remains flexible enough to implement any kind of computation. However, accuracy must be considered due to the nature of the data representation (current mode) and the technology issues (mismatch, memory volatility, noise effects...). Analog computing allows to integrate a processor per pixel of the sensor, eliminating one of the most important bottlenecks in traditional architectures. Given the number of bits of the data representation, digital platforms guarantee accuracy independently of the technology process. While 7–8 bits are hard to reach using mixed-signal architectures [158], 32-bit or 64-bit architectures are common in digital devices.

	Manual	**Intel Core i7 940**	**SCAMP-3**	**Spartan-3**	**Am2045**
MAA	0.9473	0.9202	0.9180	0.9192	0.8132

Table 2.12: Maximum Average Accuracy (MAA) for each implementation, including the manual segmentation by an expert

The discussed algorithm is robust enough to run in platforms with short data representations as SCAMP-3. However, this is not only the unique factor which affects the final result. As it was discussed above, each device requires us to make certain concessions to guarantee its viability. Table 2.12 summarizes the maximum average accuracy (MAA) [186] of each implementation compared with the manual segmentation available in the DRIVE database [159]. We can see that using the Ambric device, the accuracy drops around a 10% when removing the mentioned operations. The main reason is the appearance of many false positives that now are not eliminated using the hole-filling operation. However, once skeletonized the vascular tree, they can be easy removed using hit-and-miss masks if the application requires it. Concerning algorithm issues, a comparison with other approaches can be found in [152].

Although digital designs consume more silicon area, the improvements in the semiconductor industry are lowering the silicon area requirements, providing higher rates of integration density. The benefits are an easily scaling and migration of the architectures, which it is possible to carry out with straightforward designs as a difference with analog computing. FPGAs and MPPAs are clear examples. Analog Cellular Processors have a large network that connects the processing elements. To upscale this network, keeping a high yield with different array sizes and manufacturing processes is extremely difficult. This is one of the main reasons why they are not able to process high-resolution images and why customers have more availability of digital devices. This way, MPPA devices are the most suitable platform to deal with large images because they have not restrictions in this sense (stream processing modules can also be implemented on FPGAs). Pure SIMD-matrix approaches offer good performance because they match the most common early vision operations, but the image size is limited. In those cases, a sliding window system is a need to process larger images.

Image size constrains the amount of RAM needed in the system, especially when working with high-resolution images. One advantage of this algorithm is that it needs a small amount of off-chip memory and that it can take advantage of the embedded RAM to perform all computation, reducing I/O. External RAM is mainly used to store the input image and the results

so 4–8 MB are enough. The SCAMP-3 Vision Chip can store on-chip up to 8 128×128px images (equivalent to 128 Kb, taking into account that the computation in done in analog mode), the architecture proposed for the Spartan-3 up to 8 144×160px images (176Kb) and the Ambric Am2045 can store up to 4.6 Mb of data. This is the main reason why PC performance is so low: the selected devices are capable of doing all the processing on-chip, accessing to the external memory just for loading the input image. On the contrary, the PC should make an intensive use of external memory to store partial results, making memory access a bottleneck. Explicit load/store operations are needed and this is why CPP is much higher than the other approaches.

Analog processing allows to integrate the processors with the image sensors. But this kind of processor distribution, although adequate for low and some steps of mid-level image processing, is not suitable for complex algorithms with more complex data dependencies. SCAMP-3 is very powerful for early vision tasks, but it lacks the flexibility to address higher-level operations. Using FPGAs, different architectures and computing paradigms can be easily emulated. Its dense network, although it consumes a large silicon area, provides this flexibility. MPPAs attempt to build more powerful computing elements by reducing interconnection between processors. The fixed network of Cellular Processors limits its range of application. The programmable and dense network of FPGAs enables any kind of architecture emulation, but reducing integration density and the working frequency. MPPAs are located in an intermediate position. Its programmable network provides enough flexibility to cover a wide range of applications, freeing up space to build more powerful processors, but limiting the kind of computations that can address. This is why recursive operations are hard to implement and the resource usage is so high. PLS had to be simplified to deal with this trade-off.

With respect to the algorithm development process, Time-To-Market (TTM) is key in industry. Ambric's platform offers the system with the lowest TTM. A complete software development kit that provides a high-level language and tools for rapid profiling make the development much faster than in other platforms. This is one of the purposes of the platform, to offer a high-performance device keeping prototyping rapid and closer to software. Using HDL language to develop complex architectures or software-to-hardware implementations is much more expensive because they are closer to hardware than to software. This is specially true in the second case, where a high-level language (as SystemC or ImpulseC) is recommended.

Regarding portability, it is clear that a computer-based solution (even a mobile version) is not a valid solution because of size, power consumption or lack of integration and compact-

ness between its components. For early vision tasks, a focal-plane processor (as SCAMP-3) is the best choice. The processors are integrated together with the sensors and their power consumption is very reduced. To accomplish complex operations, FPGAs offer reduced size and power consumption in its low-end products, allowing to build complex Systems-on-Chip and compacting all the processing on the same chip. For better performance, an MPPA or a larger FPGA is needed. Power consumption will be higher, specially for the high-end FPGAs, but the amount of processing units we can include is considerably higher.

Although the conclusions we have drawn in this section come from a specific algorithm, this has features common to most algorithms for low- and many medium-level image-processing tasks, as discussed in Section 2.3. We have seen that visual processors as SCAMP are excellent in early vision operations, but specific algorithms are needed to address the subsequent processing steps. MPPAs provide an environment closer to the programmer, with a great performance, limited by the low-level operations. FPGAs enable us to replicate any application with very acceptable results, although the development time is higher.

In summary, this algorithm was specifically designed focusing on performance and to operate online in a device with reduced size, cost and power consumption, opening new possibilities in other areas apart from the medical imaging. It was specifically designed focusing on performance, so an MP-SIMD processor array offers good results. Due to the algorithm inherent nature, an MP-SIMD processor array offers good results. However, the technology limitations, especially array size and a limited accuracy, make us consider other approaches. FPGAs enable us to speed up most applications with acceptable results. They take advantage of its highly reconfigurable network to improve the matching between architecture and algorithm, exploiting its characteristics and potentially leading to better performance solutions. Advances in semiconductor industry enable to integrate more and more functional units in a reduced silicon area. MPPAs take advantage of this integrating hundred of processors on the same chip. They focus on exploiting the task parallelism of the algorithms, and results prove that this approach provides remarkable performance. However, certain trade-offs must be done when dealing with low-level image processing not to compromise efficiency.

Results show that even using a high-end CPU, a significant gain can be achieved using hardware accelerators. A cost-sensitive FPGA outperforms the Intel Core i7 940 by a factor of 10x. With a focal plane-processor, this factor reaches 60x. Using the selected MPPA, a factor of more than 1500x was reached, but we have to take into account that the algorithm was simplified in order to sort the limitations of the platform when dealing with low-level

image processing. The accuracy drops about 10% which might compromise its suitability for some applications. The coarse-grain processor array has proven to be very effective on FPGA, leading to high throughput and straightforward array scaling, as Spartan-3 and Virtex-6 FPGAs implementations show. This permits to outperform the focal-plane processor, which is the natural platform for this kind of algorithms.

The retinal vessel-tree extraction algorithm presents common features to most of the low- and mid-level algorithms available in the literature. Except for high-level operations over complex data sets, where high precision is needed, the presented architectures perform adequately for low- and mid-level stages, where operations are simple and have to be applied over a large set of data. They are able to exploit the massive spatial parallelism of low-level vision, featuring general-purpose computation. Ambric's processor requires a special mention because, although it can exploit spatial parallelism of low-level vision, its throughput is very high when dealing with the mid-level stage, where task parallelism is clearly advantageous. However, SCAMP-3 is mainly restricted to the low-level stage where its low power consumption and form factor fit well. FPGAs are flexible enough to cover the complete application. Their major drawback is the time required to get the system ready.

2.6 Summary

In this chapter, a dedicated processor for general-purpose low-level image processing is presented. The tasks of the low-level stage are characterized for being simple, repetitive and applied over a very large set of data, and do not require high accuracy. Therefore, they are very expensive computationally, specially on embedded systems, the field where this processor is focused.

After a preliminary study where a binary (B/W) processor was designed, it has been concluded that a massively parallel architecture does not face adequately low-level operations. Although a 2D arrangement of arithmetic cores can take advantage of the nature and interrelationships between image pixels, a processor-per-pixel correspondence is heavy resource consumption, dealing with low resolution processors and low clock frequencies. Even though these are technical issues, the real performance and suitability of these processors are much lower than theoretically expected, so other solutions *a-priori* less suitable might overcome these limitations.

As a result, the preliminary processor was enhanced to process non-binary data, extending the capabilities of each processing core and reducing the parallelism. This architecture, a coarse-grain processor array, provides better figures of merit, as results show. In particular, it is easier to upscale the array without compromising the clock frequency, leading to processor able to handle images of larger resolution. When comparing it with other approaches, an FPGA-based implementation of the coarse-grain processor array outperforms a MP-SIMD processor which has an 8500% more computing units. This integration density is achieved using mixed-signal architectures which limit the array size and the clock frequency, leading to poorer results although the power consumption requirements achieved are very difficult to beat. MPPA devices also show a very high performance exploiting task parallelism, although recursive operations drops performance or accuracy, been hard to reach a trade-off solution. The algorithm employed for this comparison and posterior discussion is representative of its class, providing additional information about how data interacts when concatenating different operators. This gives us an advantage during the design stage of a dedicated architecture to speed-up this kind of operations. Results show that conventional CPUs are not efficient for early vision tasks, so other different approaches are clearly justified.

The major conclusions of this chapter refer to arithmetic cores organization and data distribution between them. As seen before, a 2D arrangement faces adequately low-level pixel-to-pixel and neighborhood operations, especially when processing image data. On the contrary, when processing non-image (abstract) data, a 2D array does not provide the necessary flexibility for its processing. This is more acute with a higher level of abstraction. The MPPA processor, which takes advantage of task parallelism, is a proof. One unresolved issue is data I/O. Image processing tasks require a high bandwidth and large storage elements, limiting the performance if the arithmetic units remain waiting for data to process. Overlap data processing and transferring is essential.

The conclusions drawn in this chapter lead us to propose an improved architecture, focusing on an extended range of operation without compromising performance, resource and power consumption.

CHAPTER 3

EXPANDING THE RANGE OF OPERATION

Chapter 2 summarizes the efforts made to speed-up the low-level stages on Computer Vision applications. These stages represent most of the workload of a typical Computer Vision application. A 2-dimensional massively parallel processor array to tackle higher level stages was discarded as its flexibility is reduced. This is also true for the coarse-grain processor array, although it solves issues regarding operation accuracy and image size capability. However, the inclusion of large on-chip memories helps to provide large throughput. On the other hand, task parallelism has proven to be very effective even facing natively spatial parallel operations. However, some operations have to be discarded as the number of processing units is limited to efficiently implement them.

SIMD processors are widely used to speed-up repetitive and massively parallel operations. However, programs with more complex dataflow do not fit well in this kind of processors, being needed a different scheme. In this case, MIMD computers can address more efficiently streaming processing because they are focused on exploiting the task parallelism. Our goal is to combine both paradigms on a single architecture in a way that it could be configured according to the type of parallelism of the algorithm. This way, spatial-parallel operations are executed in SIMD mode, task-parallel operations are carried out in MIMD mode and the high-level steps are performed using the SISD paradigm, with a sequential processor.

This chapter introduces a hybrid SIMD/MIMD architecture for embedded devices which reconfigures its internal connections to face low-, mid- and some operations of the high-level stages. It focuses on flexibility and easing programming without compromising performance, permitting to adapt internal parameters to address different key applications. The architec-

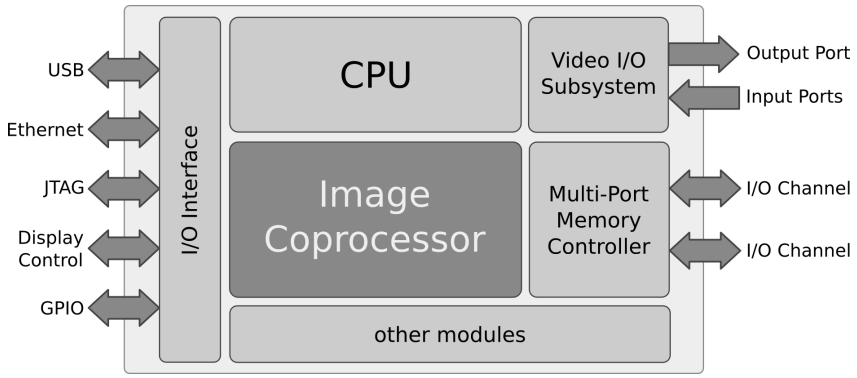


Figure 3.1: Main modules of the System-on-Chip.

ture should be able to reconfigure the internal connections at runtime to adapt the datapath to the particular characteristics of the algorithm under execution, avoiding the necessity to implement separate units by sharing the functional modules. As we are focusing on a general-purpose accelerator, the architecture has to be able to execute efficiently most image processing tasks. This is important in order to guarantee a suitable performance without compromising algorithm accuracy avoiding to cut down its capabilities. Ease algorithm migration by hiding to the programmer low level details (e.g. data coherence, synchronism between computing units, etc) is key for rapid development to meet time-to-market requirements. The architecture must be modular and scalable, so that it is possible to build different processor families targeting different application scopes, as smart cameras, autonomous devices or desktop computer accelerators.

3.1 Processor architecture

3.1.1 Processor datapath

The processor is intended but not limited to embedded devices. Aiming at higher performance, better integration and lower power consumption a System-on-Chip will be considered. Figure 3.1 illustrates the main modules of an SoC which includes the architecture proposed in this chapter.

The main unit is a low-power high-end CPU, an SISD machine to face the high-level stages of the Computer Vision applications. In order not to compromise performance, a

signal-processing optimized CPU is essential to manage complex operations, irregular data access patterns and intricate program flows. The remaining image-specific processing tasks are carried out by the Image Coprocessor, which is able to run in SIMD or MIMD modes depending on its configuration. A Multi-Port Memory Controller provides a high bandwidth to enable simultaneous communication between the external RAM and the different modules of the SoC, in particular, the Image Coprocessor. As Computer Vision applications make intensive use of both on-chip and off-chip memories, this aspect is essential not to compromise the performance. The CPU also controls the whole system, managing other modules such as the video subsystem or the Ethernet interface. High-level operations do usually represent an small fraction of the whole computation so control tasks are feasible to be executed in the same processor without compromising performance.

Although the main CPU is able to fully run Computer Vision algorithms, the performance does not usually meet the requirements. The coprocessor is intended to reduce the workload during the most computational expensive tasks. As discussed previously, Computer Vision applications have a large disparity in operations, data representation and memory access patterns, so a general-purpose architecture for embedded image processing has to offer enough flexibility without compromising performance. It has to exploit massively spatial-parallel operations, keeping a high throughput on data-dependent and complex program flows. In addition, the design must also be modular, scalable and easy to adapt to the needs of a different range of applications.

Figure 3.2 sketches the main modules of the coprocessor. It is composed of three major modules: a Programmable Input Processor (PIP), a Programmable Output Processor (POP) and a set of Processing Elements (PEs). The two former modules are responsible for data retrieving while the array of PEs performs the computation. The array supports two working modes to accomplish efficient algorithm processing. In SIMD mode, all PEs execute the same instruction over their own data set, located in a private memory space. The *side-to-side* network, which connects uniquely adjacent PEs in a 1-dimensional array, is employed to collaboratively exchange data. In MIMD mode, each PE stores its own program and data set in a private storage, working in an isolated manner. However, they are able to exchange information using a *local network*, a 2-dimensional auto-synchronized and distributed network. The operation modes are explained in detail in Section 3.1.2. Below, the different modules of the coprocessor are described.

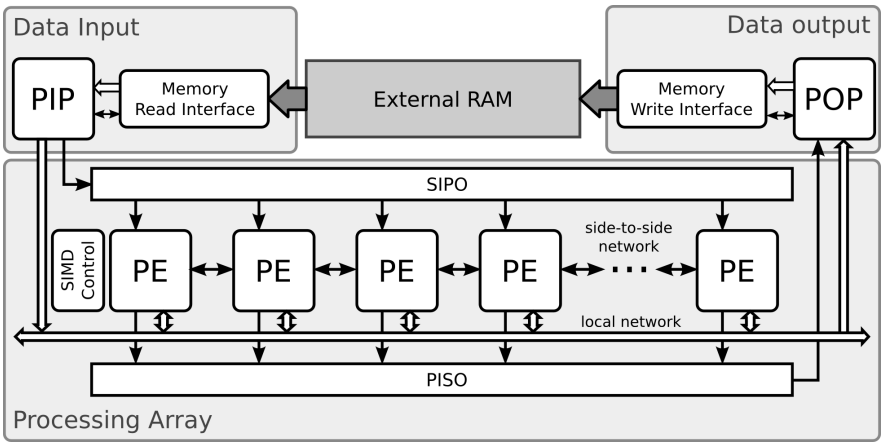


Figure 3.2: Schematic view of the proposed Hybrid Image Coprocessor. The interface with other modules of the SoC is not shown.

I/O Processors

The I/O processors, called Programmable Input Processor (PIP) and Programmable Output Processor (POP), are responsible for transferring data between the external memory and the PEs. The PIP supplies data to the processing array, while the POP extracts the results and stores them in the off-chip memory. Both PIP and POP have their own program space and work in parallel, overlapping in/out operations with the computation operations when possible. In addition, data transfers do not need to be synchronous, improving performance when the size of the input data stream is different from the output data stream.

Figure 3.3 shows the outline of an I/O Processor. It comprises a memory bank for instruction storage, a set of registers, an address generation unit (AGU) and a data cache. Both PIP and POP have the same internal architecture. The major difference lies in the Memory Interface. External RAM memory is interfaced with PIP using a read-only port, while POP employs a write-only port. In addition, the SIMD and MIMD interfaces depend on the mode the coprocessor is configured, as detailed later.

As seen before, Computer Vision algorithms employ a wide range of data access patterns. I/O processors include a dedicated address generation unit to ease data transfers and memory management. This unit automatically calculates the source and destination addresses of the data streams enabling linear, modulo and reverse-carry arithmetic addressing modes. This is

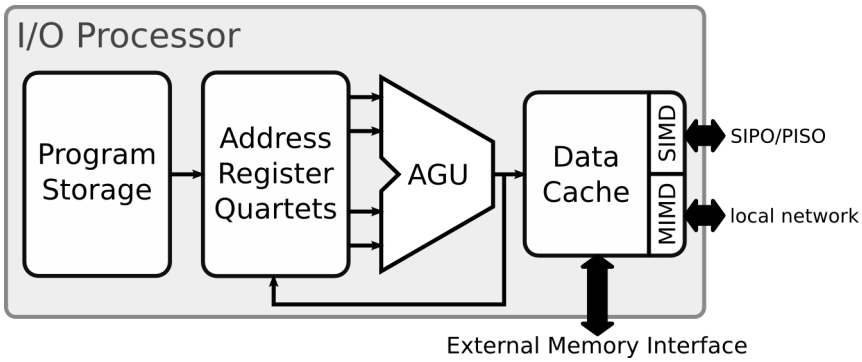


Figure 3.3: Detail of the I/O Processor. PIP and POP have the same internal architecture.

done with a set of quad-registers which configure each pattern and enable to manage several data streams at a time. Each quartet comprises a base register (base address of the data set), an index register (relative displacement inside the data set), an increment register (increment of the index value after each read/write operation) and a modifier register (type of address arithmetic).

The instruction bank stores the program that PIP or POP runs. Each pattern is defined with the four parameters aforementioned. It is required a single instruction to perform the transfer, which sets which one of the quad-registers are employed to calculate source and destination addresses. In order to increase flexibility, the quad-registers can be managed as independent registers. This permits to modify their value at runtime and reusing without needing to load a new program. In addition, zero-delay loops are available to increase throughput when transferring large data sets. In this case, the registers are employed as standard registers to store variables when checking loop termination. Simple addition/subtraction and jump operations are enough for this purpose. This scheme reduces the complexity of the memory management. All calculations occur in parallel so each processor is able to provide a valid address and update the quartet values or execute an auxiliary operation in a single clock cycle.

To reduce the latency of the off-chip memory, PIP and POP units include a direct-mapped data cache connected to a dedicated port of the Multi-Port Memory Controller, as detailed previously in this section. Using cache blocks larger than one word takes advantage of spatial-locality. The block size changes according to the mode, being larger for SIMD than for MIMD as the first is able to make greater use of spatial parallelism. The latter would suffer greater penalties if a miss happens. Employing a RAM block to store data and two different structures

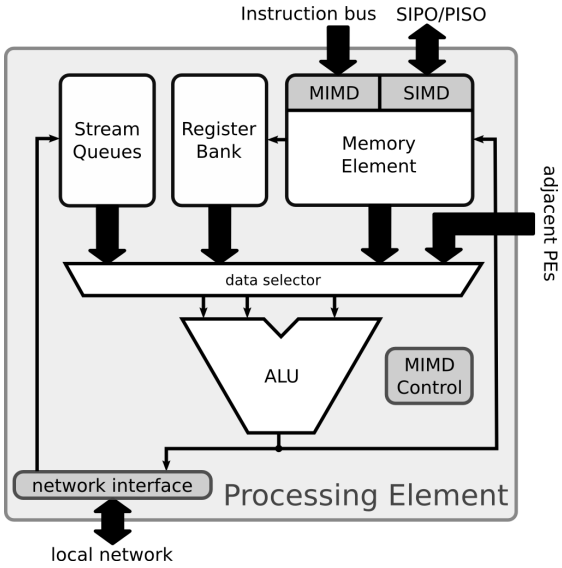


Figure 3.4: Processing Element of the Hybrid Image Coprocessor for both SIMD/MIMD modes.

to store the *label*, *index* and *valid* fields, as usual on cache architectures, enables this feature. The performance increase compensates for the additional resources.

Processing Elements Array

The Processing Element (PE) is key for a successful design. It has to be as small as possible to reduce hardware resources and to build the largest possible array, but flexible enough to face most of the arithmetic operations efficiently. The PEs have a Reduced Instruction Set Computer (RISC) architecture. The purpose is to make the segmentation and the parallelism of the instruction execution easier, integrating a small and common set of regular instructions for both modes. This requires less resources, specially important in MIMD mode where each PE includes its own controller as each one is running an independent program. The processing array is made of a set of independent and encapsulated processors connected through a reduced and programmable network, as Figure 3.2 shows.

Figure 3.4 depicts the Processing Element. Each PE comprises a Memory Element (ME), an Arithmetic and Logic Unit (ALU), a Register Bank and a set of *Stream Queues*. The in-

struction set contemplates the standard signal processing and logic operations up to three operands, as well as result saturation: basic arithmetic (addition, subtraction, multiplication,...), DSP (multiply-add, add-multiply, abs, abs-subtraction,...), helpers (max, min) and Boolean (bitwise and shifts) operations. To save hardware resources, the ALU only supports signed/unsigned and fixed-point data representation. Data-hazards are handled automatically by bypass to speed-up the computation and to avoid halts in the pipeline. Section 3.1.3 describes the available operations and the instruction format.

The *data selector* drives the operands to the ALU. The operands come from the different modules which store the input data and the partial results. The Memory Element, a dual-port RAM, stores both data and instructions depending on the operation mode. The Register Bank is employed to store partial results as the Memory Element is only able to provide up to two simultaneous operands. While the Memory Element is able to store hundreds of words, the Register Bank has a smaller size, storing just a few words, but provides three independent ports (two read-only, one write-only). Additional operands reach the ALU through the network interface depending on the operation mode. This network is dual, i.e. there are side-to-side connections between adjacent PEs and a point-to-point interconnection with automatic synchronization between each PE and some of its neighbors. The Stream Queues are employed in MIMD mode to provide buffering and synchronization while the direct connections between adjacent PEs are directly driven to the ALU in SIMD mode. The combination of the Memory Element with the programmable network greatly expands the flexibility of the array by changing radically the way the PE works. This point will be explained in depth in Section 3.1.2.

3.1.2 Operation modes

The architecture of the Image Coprocessor is intended to execute the different sub-tasks of a given algorithm according to the most fundamental type of parallelism present in the mathematical operations. This way, massive operations are executed in SIMD mode, task-parallel operations are carried out in MIMD mode and higher level operations are completed in the CPU. In addition, the CPU is responsible to program, initialize and perform post-processing over the coprocessor outputs.

SIMD mode

As seen before, the algorithms are split into three main blocks. One, controlled by the PIP, supplies data to the computing units, the PEs. Other, controlled by the POP, extracts the results from the PEs and store them in the external memory. The last block controls the PEs. When working in SIMD mode all PEs execute the same instruction but retrieve data from their own data storage, the Memory Element. Therefore, it is only needed a single module to control all the PEs, the SIMD Control unit.

Figure 3.5 shows how the PE is configured when working in SIMD mode. The *data selector* drives the operands to the ALU, selecting up to three, which come from the Register Bank, the internal or the neighbor Memory Element (left or right) or an immediate value encoded in the instruction. The Stream Queues and the *local network* are not employed in this mode. The result is always stored in the Register Bank, which is used as temporal storage. The Memory Element only has two independent ports for reading and writing, so employing the Register Bank to store partial results (two read and one write ports, all simultaneously accessible) greatly increases the throughput.

The PEs are laid down as a 1-dimensional array, exchanging data with the *side-to-side* network. As all PEs execute the same instruction, a block of data with the same size of the number of PEs must be transferred between PIP or POP and the processors. For this purpose, the Serial-In-Parallel-Out (SIPO) and Parallel-In-Serial-Out (PISO) queues were added (see Figure 3.6) to introduce or retrieve data from the Memory Element of each PE, which be used as data cache. When loading data, the PIP fills the SIPO queue serially, and then the whole queue is transferred in parallel to the processors, one word to the Memory Element of each PE. The POP performs the opposite task. First, the PISO is filled in parallel retrieving a word from the Memory Element of each processor. Then, the queue is emptied serially by the POP. This way, PIP and POP take control of the Memory Element for only one clock cycle, greatly reducing the time in which each PE is stalled due to data transfer operations as the PEs can continue processing while the queues are being emptied or filled.

The aim of this mode is to make an intensive use of the Memory Element to store the input data, the intermediate results and the desired output. Therefore, it is only needed to extract the final results and write them back, reducing the number of accesses to the external memory. For instance, for an image of 640px width it is needed to store at least 1920px to perform a 3×3 image convolution, i.e. three rows of the image. If the array has 64 PEs, each Memory Element must store 30px. The larger the Memory Element, the higher the throughput. The

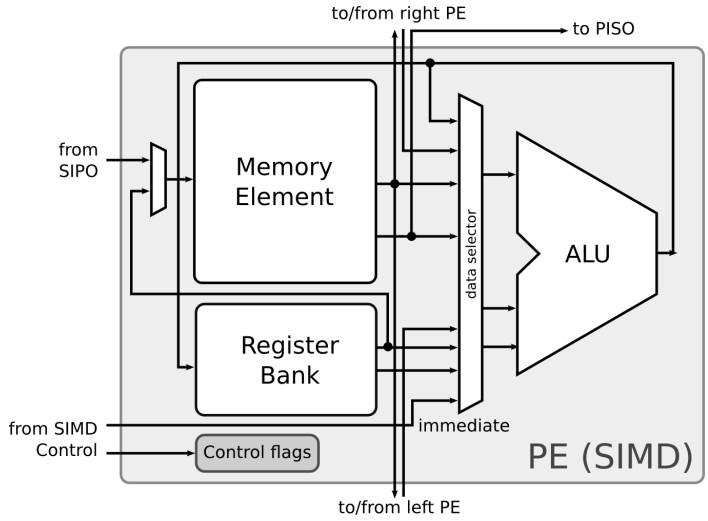


Figure 3.5: Simplified sketch of a PE in SIMD mode.

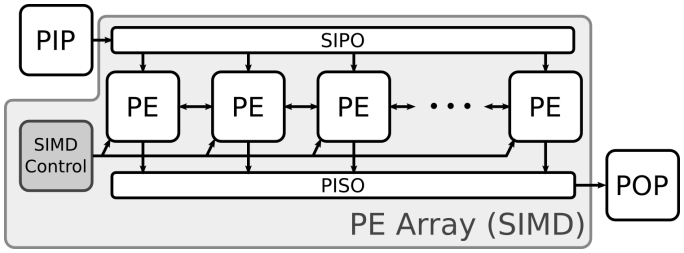


Figure 3.6: SIMD mode processor array layout. Only the *side-to-side* network is used.

optimum size of the Memory Element depends on the number of PEs, in such a way that the total amount of storage permits to completely store small images or tens of rows of high resolution images, considerably reducing the accesses to the external memory. Considering the previous example, with 64 PEs, a size in the order of hundreds of words (500-2000) will permit to address many applications involving medium-large images.

Figure 3.6 shows how the PEs are arranged in SIMD mode. They are laid down as a 1-dimensional array, exchanging data with the *side-to-side* network, allowing to shift data one position left and right. Boundary PEs are also connected, enabling circular shifts, although this is not shown for sake of clarity. To access to PEs at a further distance, several shift

operations must be performed. However, setting adequately the quad-registers of the PIP, it is possible to interlace the input data stream and therefore reduce the communication overhead, as discussed later in Section 3.2.2. This network is made of synchronous direct links and has zero-latency, so additional operations or clock cycles are not necessary to share data as this operation is available in each instruction.

A single controller for the array is needed, the SIMD Control unit, which stores program and performs automatic addresses calculation. Internally, it is similar to the PIP and POP structure, providing quad-registers and the same address generation capabilities. It permits to manage flexible access patterns to retrieve data from the Memory Element and to control program flow. The reason to include an address generator is that the Register Bank contains a few registers which can be directly managed setting their addresses explicitly on each instruction. This is not applicable to the Memory Element as its size is too large. The size of the instructions and data management would be impractical. However, a reduced form of direct addressing is included to face irregular patterns, although the performance is lower as only an operand can be managed at a time. Unlike PIP and POP, the SIMD Control unit provides up to two simultaneous addresses to read two operands from the Memory Element. The ALU output is always stored in the Register Bank, so to write the result back to the Memory Element, only an additional instruction and an address is needed. Besides address generation, the SIMD Control unit also decodes the instruction, controls the side-to-side network and sets all the control signals, which are driven to the array of PEs using a pipelined bus, as all of them execute the same instructions.

MIMD mode

As in SIMD mode, the algorithm is divided into three parts, leaving the input, output and processing operations to the PIP, POP and PEs respectively. However, there is a major difference which dramatically affects the programming of the processing array. The instructions for data processing are not longer stored into a single unit: every PE handles its own code, a small program which includes computation, flow control and network access. As in SIMD mode, all PEs also work in parallel, but each one runs its own program.

Each PE works as an independent and encapsulated processor connected to the network, as Figure 3.7 depicts. It uses the Memory Element to store data and the program. In this mode, the ALU operands come from the Register Bank, the Stream Queues or can be immediate values directly encoded in the instruction. Unlike in SIMD mode, the Memory Element is not

directly connected to the data selector. Data can be moved between the Register Bank and the Memory Element if larger storage is needed, although load and store operations take several clock cycles as the processor is heavily segmented. As detailed previously, data-hazards are handled by bypass to speed-up processing. However, load/store hazards and branch-hazards are not handled to save hardware resources. Independent instructions or *bubbles* must fill the pipeline to avoid it. The ALU output can be stored both in the Register Bank or directly in other PE using the network.

The Stream Queues are employed during network access to buffer and synchronize communication. As seen before, MIMD mode aims to handle irregular program executions with data-dependent execution to take advantage of the task-parallelism. Stream processing has proved to be very efficient on these tasks. This mode configures the processing array as an *enhanced pipeline*, where each stage does not execute a single operation but a micro-kernel. In particular, it works as a Kahn Process Network (KPN) [187]. A KPN is a distributed computing model where several deterministic processors, communicated between them through unbounded FIFO channels, perform a specific task that is part of a more complex computation. A suitable algorithm partitioning and the design of an efficient communication pattern between each of the parts is essential to map each of the tasks to run on the available processor. Non used processors can be turned-off to reduce power-consumption.

Based on previous studies as [188] [189], and keeping in mind the existent trade-offs, a 2D-Torus network was selected. As a difference with other network models such as hypercube or a 2D-mesh, 2D-Torus uses a reduced amount of resources without sacrificing the communications between processors. Figure 3.8 shows the local network and how the PIP and POP interact with the PEs. Each PE includes four independent Stream Queues (one per direction) of several words each.

The local network is built of point-to-point stream connections synchronized by FIFO queues. No routing or any other kind of control is needed because the data flow through the network is totally determined by the program each PE runs. Network access is transparent to the programmer as the source and destination Stream Queues are treated as standard registers. This way, all modules can work concurrently, improving performance by overlapping communication. The access to the stream queues is done without latency and synchronization is automatic, easing programming: network access blocks the processor until the data is available to be read or there is a memory position available to write the result, ensuring data coherence. Although the KPN model considers that the FIFO queues are unbounded, this can

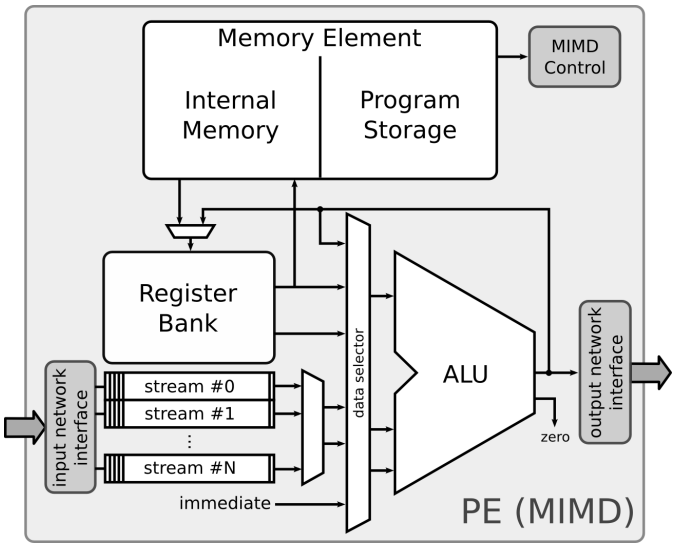


Figure 3.7: Simplified sketch of a PE in MIMD mode.

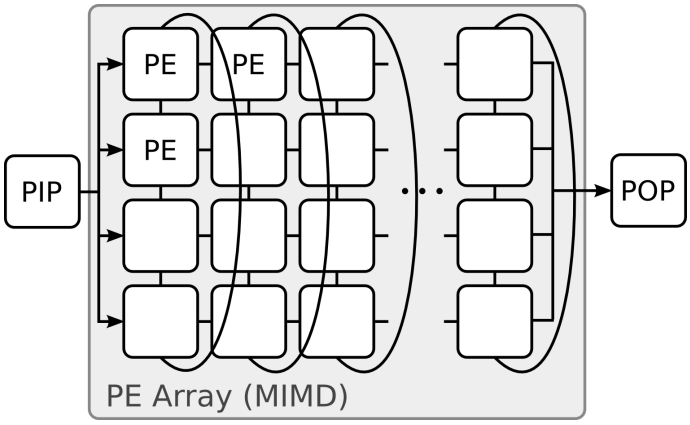


Figure 3.8: MIMD mode processor array layout. Only the *local network* is used. Connections between PEs represent two independent channels, one per direction.

not be carried out in practical implementations. This parameter is configured according to the final application. However, a careful algorithm partitioning must be done in order to avoid

very unbalanced paths, as *fast paths* can overload the queues and drop the performance or even block the computation.

In MIMD mode, the PE uses an expanded instruction set which includes flow control operations (unconditional-jump, branch-if-equal, branch-if-not-equal...). Each PE includes its own control unit, the MIMD Control, as each one runs a different program.

The data access patterns in MIMD mode are the same as in SIMD mode. PIP and POP functionality (flexible access patterns, data alignment or data caches) and architecture remain the same. However, PIP and POP do not use the SIPO and PISO queues to preload data. Instead of that, they access directly to the Stream Queues which form a local network using a simple handshake (FIFO) interface. This is the reason why the I/O processor have two interfaces, one per each mode (see Figure 3.3).

In SIMD mode, PIP and POP run until the SIPO/PISO queues are full or empty, transferring as many data as PEs has the array, one word per processor. This permits to exploit massively spatial parallelism. On the other hand, when taking advantage of the temporal parallelism, it is expected that the typical patterns consist of sparse memory accesses of a number of consecutive words. In addition, MIMD mode resembles a systolic array with a *left to right* flow of data. As shown in Figure 3.8, PIP and POP are only connected to the first and last columns of PEs. In MIMD mode, a configurable amount of data is transferred to a single PE, indicated in each transfer. Despite this, data transferring is done in the same way as described previously. A single instruction sets which one of the quad-registers is employed to calculate the source addresses in the external memory and the destination PE, encoded explicitly in the instruction. Additionally, a third field encodes the size of the transfer. This is done by PIP to supply data, but POP works in the same manner.

Selecting the mode

The selection of the operation mode depends on multiple factors. From the operational point of view, the following steps are involved. The coprocessor is fully guided by instructions, the algorithm which the user runs. First of all, the CPU selects the mode by setting a global flag which directly indicates if the processor is in SIMD (0) or MIMD (1) mode. Then the CPU has to load the programs of all involved modules, this is, the PIP, POP, and the SIMD Control unit (if SIMD mode) or the used PEs (in MIMD mode).

As detailed before, the instruction set is common for both modes to ease decoding, although separate control units handle each mode. Depending on the value of the global flag,

they take the control or inhibit its functionality. This way, switching the flag changes mode with zero-delay. However, results might be undefined if we do not ensure that the previous operations are completed. This is met when the *end of the program* is reached in all the units.

PIP, POP and SIMD Control modules act during the program transferring. All of them are directly accessed through the bus of the system. However, to load the program into all PEs when running in MIMD mode, a pipelined shared bus is employed to avoid large resource consumption and not to drop the clock frequency due to a high fan-out. Program storage is done serially but with random access, greatly reducing loading time if just a few processors are employed or a compact program is run.

3.1.3 Instruction set

The selection of the instruction set is important to meet some of the trade-offs of the architecture. Regular and compact instructions permit to decrease hardware requirements and ease instruction decoding. A RISC architecture was selected to achieve this goal. However, even more important is to select a set of operands and complementary operations that permit to efficiently implement most of the Computer Vision algorithms. Inasmuch as the architecture is focused on a general-purpose solution, the instruction set has to contain the basic building blocks that enable to execute more complex operands. However, this may result in a performance decrease if for every major operation the number of sub-operations is too large. It is not possible to upscale the number of processors at the same ratio to compensate for this limitation. Taking into account the proposed architecture, the following trade-off solution is considered, aiming to balance both general and CV-oriented operations.

I/O Processors

PIP and POP are responsible for data transferring. Figure 3.9 shows the instruction format for the I/O processors. The different operations are grouped according to their format. *R*-type refers to regular, *I*-type to immediate operation, *J*-type to unconditional jumps and *S*-type to other non-standard (special) operations. The *move* instruction transfers a set of data to the desired location. Considering the PIP in SIMD mode, the source and destination addresses are calculated using the Address Generation Unit and the configuration stored in the quad-registers. In MIMD mode, the source is also encoded in a quad-register. However, the destination address is a mask which sets the destination PEs of the first row of the array. Data transferring is equivalent for POP, although inverting *source* and destination fields.

<table>
<tr><th>Type</th><th colspan="6">Fields</th><th>Mode</th><th>Unit</th></tr>
<tr><td>R</td><td rowspan="3">opcode</td><td rowspan="2">dst</td><td rowspan="2">src[0]</td><td>src[1]</td><td>--</td><td rowspan="5">all</td><td rowspan="5">PIP
POP</td></tr>
<tr><td>I</td><td colspan="2">#immediate</td></tr>
<tr><td>J</td><td colspan="4">#address</td></tr>
<tr><td rowspan="4">S</td><td>loop</td><td>--</td><td>src</td><td colspan="2">--</td></tr>
<tr><td>rep</td><td colspan="4">#times</td></tr>
<tr><td rowspan="2">move</td><td rowspan="2">q-src</td><td>q-dst</td><td colspan="2">--</td><td>SIMD</td><td rowspan="2">PIP</td></tr>
<tr><td colspan="2">dst-mask</td><td>size</td><td>MIMD</td></tr>
<tr><td rowspan="2">q-dst</td><td>q-src</td><td colspan="2">--</td><td>SIMD</td><td rowspan="2">POP</td></tr>
<tr><td colspan="2">src-mask</td><td>size</td><td>MIMD</td></tr>
</table>

Figure 3.9: Instruction format for PIP and POP processors.

As discussed previously, I/O processors are also able to perform basic arithmetic and control operations. They only include signed integer arithmetics and support for immediate values. As a difference with the *move* operation, the registers are considered as independent. For example, quartet *Q0* includes individual registers *R0-R3*. This permits to set the values of the quartets at runtime during program execution. A zero-flag is employed for conditional branch checking. Additionally, two fast-loops are available. The first (*loop*) automatically decreases the register value and jumps when it reaches the zero value. The second (*rep*) repeats the immediately following instruction the indicated number of times with zero-delay, employing an additional dedicated register without modifying the program counter until the loop ends. While the *loop* instruction permits nested loops, the *rep* does not. The last is intended for block transfers, e.g. to transfer 10 chunks of a 640px width image to a 64-unit SIMD array.

Processing Element

The instruction set for the PEs is the same for both operation modes. However, some operations are not available in SIMD mode; those related with flow control. It should be taken into account that in SIMD mode, the SIMD Control unit stores and decodes the program, acting the PEs as just computation units, while in MIMD mode each PE stores and decodes their own program. Figure 3.10 shows the instruction format for the PEs. The instruction type has the same meaning detailed for the I/O Processors.

In SIMD mode, the SIMD Control unit provides the addresses for the operands, being in the Memory Element or the Register Bank. While the last are directly encoded in the instruc-

<table>
<tr><th>Type</th><th colspan="5">Fields</th><th>Mode</th><th>Unit</th></tr>
<tr><td>R</td><td rowspan="3">opcode</td><td rowspan="2">dst</td><td rowspan="2">src[0]</td><td>src[1]</td><td>--</td><td rowspan="4">all</td><td rowspan="3">PEs
SIMD Control</td></tr>
<tr><td>I</td><td colspan="2">#immediate</td></tr>
<tr><td>J</td><td colspan="4">#address</td></tr>
<tr><td rowspan="3">S</td><td>loop</td><td>--</td><td>src</td><td colspan="2">--</td><td rowspan="3">SIMD Control</td></tr>
<tr><td>rep</td><td colspan="4">#times</td></tr>
<tr><td>opcode</td><td>dst</td><td>src[0]</td><td colspan="2">#immediate_src[1]</td><td>SIMD</td></tr>
</table>

Figure 3.10: Instruction format to control the Processing Element.

tion field, the former is calculated using the dual Address Generation Unit, which works in the same manner that the I/O processors. The quad-registers configure the pattern for data accessing inside the Memory Element. Internally, this unit is similar to the PIP or POP, so except the *move* operation which has no sense in this context, the same operations are available, including arithmetic and flow operations. An immediate addressing mode is available by setting the desired value in the base address register and unsetting the increment value to zero. All the above is intended for address generation and runs in the SIMD Control unit. The rest of arithmetic operations are executed in the array of PEs. They include signed/unsigned integers and fixed point support, besides immediate operands.

In MIMD mode, the operands can come from the Register Bank, the Stream Queues or be an immediate value. In contrast to SIMD mode, in all cases the source and destination addresses are encoded in the instruction fields, so no additional tasks are required for address calculation or for quad-register configuration as the SIMD Control unit is not longer employed. The Stream Queues are accessed in the same manner as the Register Bank, using the adequate queue for the desired direction of communication, without latency or additional operations for synchronization. However, there is a major difference. When the source or destination queues are empty or full, the PE halts until more data or space is available to continue operating. This ensures data coherence and simplifies algorithm implementation. However, care must be taken to avoid chaining PEs of *fast-paths* with PEs of *slow-paths* as unbalanced processing speeds will result in an excess of halts. When the size of the partial results set exceeds the Register Bank size, *load*/*store* operations are available to transfer data between the Register Bank and the Memory Element. As each PE controls its own program flow, additional conditional and unconditional branches are available. The *loop* and *rep* instructions for fast-looping are not available, so special care has to be taken to avoid branch hazards by

inserting bubbles (*nop* instruction) or independent operations. The same applies to load/store operations.

3.2 Performance evaluation

In order to validate the feasibility of the architecture, it was prototyped on an FPGA. To evaluate its performance, a set of algorithms were executed. They include some representative tasks of the low-level and mid-level stages, covering the SIMD and MIMD operation modes. Finally and aiming to evaluate not only individual operations but also complete algorithms, a feature extraction and matching technique are thoroughly discussed. This algorithm includes mixed SIMD/MIMD tasks which permit us to discuss which paradigm select to avoid changing the operation mode too many times and to increase the performance.

3.2.1 FPGA prototyping and validation

Nowadays, FGPAs offer a large amount of resources without a significant increment in NRE costs. It is possible to prototype full SoCs into a single FPGA keeping a high throughput. This permits to emulate the architecture in real hardware, avoiding to develop cycle-accurate emulators with considerably less performance. In fact, FPGAs may even become the target platform due to the reduction of manufacturing cost, replacing a custom chip.

The proposed architecture has been prototyped on an FPGA to evaluate its feasibility and performance. The target FPGA we select is an Xilinx Virtex-6 XC6VLX240T-1, included on the Xilinx ML605 Base Board [142], described above in Section 2.4.3. It provides enough resources to evaluate the architecture without sacrificing features when adapting the design to the device. As discussed previously, the Image Coprocessor is intended to lighten the workload of the main CPU. An AXI4-based standard MicroBlaze System-on-Chip was implemented. Among other modules, it includes a Multi-Port Memory Controller and a 10/100-Ethernet units. The coprocessor was described using VHDL and synthesized with Xilinx Design Suite tools [142].

The proposed architecture is highly configurable, including the number of processing units, the size of the internal registers, queues or even the arithmetic operations, being able to include application-specific extensions for the most demanding algorithms without major modification in the design. In order to implement a prototype that can run a significant variety

of algorithms, the critical parameters that balance the computational power of each module and the degree of parallelism are detailed below:

- Instructions and data are represented using 32-bit words.
- PIP and POP have 4 quad-registers of 24-bit wide each for off-chip RAM address generation and two caches for instructions and data of 32Kbit (1024 words) each.
- Each PE has a Memory Element of 32Kbit which stores up to 1024 data-words (SIMD mode) or 512 data and 512 instructions (MIMD mode). The Register Bank and each of the four Stream Queues store 8 and 4 words respectively. The ALU includes 20 DSP/Boolean and 12 control and data movement operations.
- The array of PEs is made of 128 units. For SIMD mode, both SIPO and PISO queues are 128-word, and the SIMD Control unit includes an instruction cache of 1024 instructions plus 4 quad-registers of 10-bit wide. In MIMD mode, the PEs form an 8×16 torus, so PIP and POP are interfaced with 8 PEs each and the minimum route between them is 16 PEs.

The ML605 board includes 512MB DDR3 SO-DIMM clocked at 400MHz. Ports are configured with 64-bit width and, when employing 32-word burst length, providing a maximum data throughput of 1400 MB/s (reading) and 1140 MB/s (writing). The cache block size of PIP and POP direct-mapped data caches are configured to 32 words in SIMD mode and 16 words in MIMD mode. This way, each burst transfer also preloads adjacent data to the current address location, reducing the memory accesses by transferring more data in each transaction. Two different structures to control cache coherence, *hits* and *miss* are implemented using the distributed memory resources.

Table 3.1 summarizes the synthesized data for the coprocessor in number of LUTs and Register slices, Block RAMs and DSP slices. The MicroBlaze system, clocked to 150 MHz using the performance profile, additionally takes around 7500 slice LUTs, 6700 slice registers, 10 Block RAMs and 3 DPS48E1 slices. A 128-unit coprocessor fits on the selected FPGA leaving enough space to include other modules of the SoC. The theoretical peak performance of the coprocessor is 19.6 GOP/s at 150MHz (around 130 operations per clock cycle). As data transferring occurs in parallel if a carefully schedule is done, the amount of halts in the pipeline is reduced, so the real performance is expected to be close to this value. The power consumption was determined with the Xilinx XPower Estimator [142], resulting in 7.197 W

Module	FF	LUT	36K-BRAM	DSP48E1	f_{max}
PIP	1118	1197	2	1	175
	0.4%	0.8%	0.5%	0.1%	
POP	1080	1257	2	1	173
	0.4%	0.8%	0.5%	0.1%	
PE	1130	1023	1	2	165
	0.4%	0.7%	0.2%	0.3%	
Array	144994	131043	134	264	153
	48.1%	86.9%	32.2%	34.4%	
Total	148323	134520	139	268	153
	49.2%	89.3%	33.4%	34.9%	

Table 3.1: Summary of the synthesized data for a 128-unit 32-bit Image Coprocessor in the Virtex-6 XC6VLX240T-1 FPGA.

at peak performance on standard ambient conditions. 46% corresponds to logic resources and 30% to device static consumption.

3.2.2 Algorithm evaluation

The first step to evaluate the suitability and the performance of the architecture is to select a set of representative operations to use as benchmark. Below it is shown the implementation of different operations usually required in image processing with the aim of illustrating how the coprocessor works and different issues found when porting the algorithms.

SIMD mode

Window-based operands are key during the low-level stage in Computer Vision applications. Many operands employ the neighborhood of a given pixel to make a decision about how to modify its value. In particular, **image filtering or convolution** is one of the most basic and employed operations for tasks such as noise removal, image enhancing or edge detection. It involves intensive computation (multiply-add) and neighbor access which slows down computation. This kind of operations have an inherent massively spatial parallelism, so the large SIMD array can take advantage of this fact to easily implement it.

The approach done here is to store one pixel in each Processing Element, so that each one is able to access to the neighborhood of a given pixel using de *side-to-side* network. This has direct consequences. First, it eases memory management as each row is directly mapped

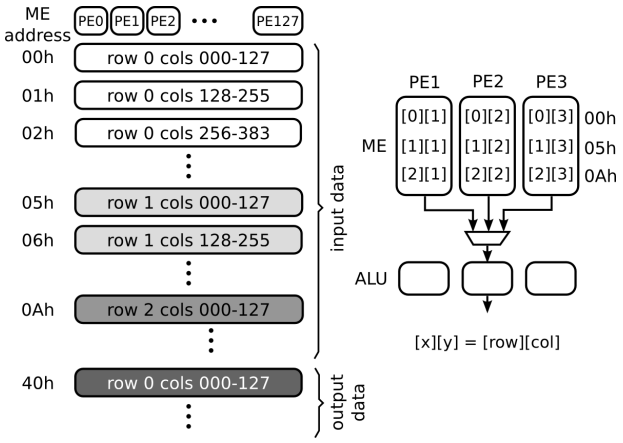


Figure 3.11: Memory organization for a 3×3 convolution on an image of 640px width. Each PE can access directly adjacent pixels both in vertical and horizontal directions.

into a 1-dimensional array. Second, the SIMD network has no latency, so the performance is not affected. Figure 3.11 shows how to organize the Memory Element of each PE in SIMD mode to speed-up a 3×3 convolution. Considering an image of 640px width and 128 PEs, each row of the image has to be split into 5 blocks of 128 words each. Therefore, each PE stores 5px per row. For simplicity we are considering each 32-bit word only stores one pixel. Additional rows of the image are stored in the same manner. It is possible to directly access to the neighborhood of each pixel both in vertical and horizontal directions employing the *side-to-side* network and knowing that rows are separated by 5 positions in the Memory Element. As all PEs execute the same instruction, the global effect is a *row shift* towards left or right directions as needed. The MAC operations have to be performed as many times as blocks the image is split to process completely a single image row, in this case 5 times.

The data access pattern is very regular so its implementation is straightforward. The PIP employs two quad-registers for source and destination addresses. The first is configured to go over the input image stored in the off-chip memory sequentially (increment is set to $+1$ and modifier, to linear addressing). The destination has the same configuration, although the parameters refer to the Memory Element storage space. The *rep* operation is very useful for fast-looping when transferring the 5 blocks each row is divided in. Additional instructions are employed for transferring the different rows. The SIMD Control unit employs the *side-to-side* network for horizontal accessing. This is directly encoded on each instruction, as seen

in Figure 3.10. For vertical communication, immediate addressing simplifies the program as a 3×3 neighborhood implies only a few instructions. However, it is a better option to employ the Address Generation Unit since the Memory Element is able to store tens of full rows. This way, setting the modifier register to modulo addressing, it is possible to program a computing kernel and iterate over all stored rows, employing a few lines of code. This scheme also permits to manage the chunks in which the rows are split (5 blocks in this case) as they behave as additional rows of the image. Finally, the POP extracts the results and write them back in the off-chip memory. The program is essentially the same as the PIP although the source and destination addresses are interchanged.

To synchronize this process, two synchronization points are employed by setting horizontal and vertical flags. PIP and POP do not transfer data until the horizontal flag is asserted, so that a row is fully processed. However, this approach can be slightly different if the processing time is much lower than data transferring. In this case, the flag can be asserted after a number of rows are processed, exploiting the memory bandwidth by performing intensive data transferring. This point will be discussed later.

MIMD mode

MIMD mode is much more flexible than SIMD as the *local network* permits to better adapt the datapath to the algorithm in execution. **Color to gray-scale** conversion is an operation which illustrates this capability. Among the plethora of possible conversions, we will consider the following: $Gray = 0.21R + 0.71G + 0.07B$. For simplicity, we assume color image is stored in memory uncompressed and using RGBA format (red-green-blue-alpha) and 8-bit per channel (32-bit word). Figure 3.12 shows how to implement this operation in MIMD mode.

PIP is continuously broadcasting the RGBA value of each pixel of the image. In the same manner as in the convolution operation, the image is read linearly. As a difference, the destination address is a mask which simply indicates de destination PEs of the first row of the 2-dimensional torus. The first column of PEs extracts the values of each channel using bitwise operations, transferring the output to the adjacent PEs, which perform the conversion employing the mentioned equation by adding the input values. Finally, the processed pixels are transferred to the off-chip memory by the POP. It reads pixels from the PE is further to the right by using a mask, and employs the Address Generation Unit to store this value in the external memory.

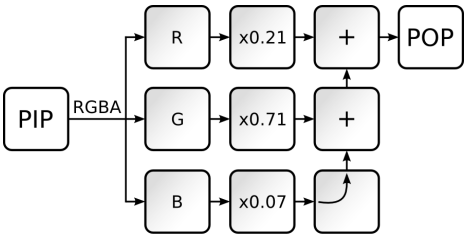


Figure 3.12: Color (RGB) to Gray conversion using MIMD mode.

This procedure enables a very high throughput using just a few processors. Also, additional steps can be included. For instance, it is possible to pack data again storing 4 pixels per word. In fact, to perform additional operations on the data stream such as intensity changes, median or histogram calculations permits to take full advantage of this mode. A throughput of a pixel-per-cycle is easily achievable with a small set of PEs. However, more complex algorithms will require a large amount of resources, so it could be convenient to perform intensive computing on each PE. For example, the whole color to gray conversion can be executed in a single PE, freeing up resources for other tasks. It should be noted that the minimum route between PIP and POP consists of 16 PEs. If no additional operations are performed, these processors have to run a program that simple acts as a router.

The previous example shows a *many-to-one* conversion, where many input streams (in this case the same although replicated) are combined into a single one. It should be noted that if pixel packing is programmed, input and output streams do not have the same length. As PIP and POP are independent and autonomous; this becomes a benefit and the performance increases. Color space converting shows other possibilities. **RGB to YUV conversion** can be done considering the following equations:

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.147 & -0.289 & 0.437 \\ 0.615 & -0.515 & -0.100 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

In this case, it is a *many-to-many* conversion. Under the same assumptions as the previous conversion, the different channels of a RGBA value are combined to produce three different channels on the YUV color space. In this case, the POP has to manage three output streams, one per channel. However, they can be packed into a single word depending on the goals of the following processing steps. Figure 3.13 shows how to implement this operation in

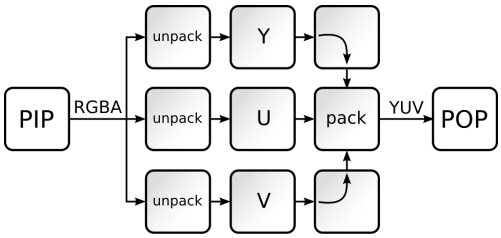


Figure 3.13: Color (RGB) to YUV conversion using MIMD mode.

MIMD mode. In the RGB to Gray conversion, each arithmetic operation is performed by a PE. Although it is also possible to implement it in the same way, the RGB to YUV conversion follows a different approach: all conversion operations are performed by a single PE, employing one for each channel of the output format. An additional PE unpacks the RGBA input stream, providing the R,G and B values in a serial manner, as opposite in the previous conversion. The final result is then packed into a single 24-bit work. While the first scheme permits a high throughput for simple operations which can be expanded along the array, the last eases algorithm deployment by parallelizing tasks much more complex than a simple arithmetic operation.

One advantage of this mode is that the *local network* ensures automatic synchronization and deterministic data processing. This way, no additional instructions are required for synchronization or for data coherence checking. In addition, network access is done without latency (unless there are no data available in the queues). In particular, these operations do not require to check which part of the image is being processed, so there is no needed to check boundary conditions and all instruction are arithmetic operations.

Remarks

Although it is possible to implement color conversion in SIMD mode, this illustrates how easy is to implement algorithms when written as a graph. Likewise, the convolution operation can also be implemented in MIMD mode, although the problem we face here is different. A single 3×3 convolution is easy to perform using a few PEs. However, as soon as the filter size grows, many PEs have to be employed simply to distribute data between them due to the layout of the network. This becomes critical if recursive operations are present. This was thoroughly discussed in Section 2.5. Another limitation comes when intensively access to the Memory

Element is needed in MIMD mode. The ALU has to perform address calculations and care of load/store hazards have to be taken into account. It is possible to employ the Memory Element as a shift register to store the input pixels as usual in straightforward custom implementations [190]. However, if the row lenght is larger than the Memory Element, several PEs have to be chained to emulate a larger storage, adding complexity to the program.

In SIMD mode, all PEs and the whole Memory Element can be used without restrictions. However, only a single task can be executed at a time. On the contrary, MIMD mode allows many simultaneous tasks. As detailed above, color conversion can be chained with other tasks. Other possibility is to replicate the same task several times by replicating the distribution of processors in different parts of the 2d-torus. This can greatly increase the performance avoiding PEs unused by processing several pixels in parallel. As the coprocessor contains 128-units, color conversion can be replicated more than 10 times. Depending on the algorithm, it is possible that some processors are employed as simple routers, so the implementation could be sub-optimal and the performance can not scale linearly with the number of processing units.

Although it is possible to switch mode at a very low cost (essentially the program load time), the coprocessor usage must be complete to fully take advantage of each mode. For instance, the 3×3 convolution in SIMD mode is limited by the speed we can enter data. It only requires 9 MAC operations (9 clock cycles) while transferring each 128 data block takes 129 clock cycles. This is, PEs are 93% of the time waiting for new data to process (data transferring and processing occur simultaneously). However, if the amount of computation exceeds the time the PIP or POP employ for data transferring, data I/O is no longer the bottleneck. As pointed before, employing as much as possible the on-chip memories is essential to fully take advantage of this mode. MIMD mode does not experience this problem as it process streams and data preloading is not necessary. However, it is hard to fully employ the processing array as the network does not fit perfectly all data exchanges of most algorithms.

Despite all the above, to select SIMD or MIMD mode depends on the particular algorithm and how operations are scheduled. Consider a color conversion in these two cases: a) followed by a mean calculation for further analysis, and b) followed by a filtering for noise reduction. As discussed above, color conversion suits both for SIMD and MIMD modes. In the first case it is better to perform both tasks in MIMD mode as only a few processors need to be employed freeing resources for other parallel or subsequent tasks. Performance is not compromised as a rate of 1 px per clock cycle is easily achievable. However, the second case fits better in SIMD mode. It eases algorithm implementation as large convolutions natively fits this mode

SIMD

	px/cycle	Mpx/s	#PEs
3x3 Convolution	0.9996	149.9	128
15x15 Convolution	0.5689	85.3	128
3x3 Erosion (binary)	0.9997	150.0	128
8x8 DCT	0.8136	122.1	128
Stereo Matching (SAD,9,32)	0.0455	6.8	128
Harris Corner Detector	0.3130	79.6	128

MIMD

	px/cycle	Mpx/s	#PEs
RGB to Gray	0.9998	150.0	9
6x RGB to Gray	5.8988	884.8	55
RGB to YUV	0.9997	150.0	9
Entropy Encoding (4x4)	0.5781	86.7	14
6x Entropy Encoding (4x4)	3.4108	511.6	90
Median	0.9999	150.0	1
Histogram	0.4311	64.7	1
Integral image	0.6135	92.0	5

Table 3.2: Performance results of implementing several image-processing tasks in SIMD and MIMD modes.

and it is possible to fully exploit the on-chip memories. In addition, as all PEs are employed, the performance is much higher. Switching between SIMD and MIMD modes for simple operations is not the recommended approach. As the output has to be written back to the off-chip memory when changing mode (except in some particular cases when the Memory Element is not manually reset), PIP and POP have to transfer the data twice on each change of mode, reducing the performance. Although the current mode could not be the most suitable for subsequent operations, to use the coprocessor below its capacity with small kernels has more impact in the performance.

Besides the two previous operations, other algorithms were implemented to evaluate the feasibility of the architecture. Table 3.2 shows the most relevant results of implementing several tasks in SIMD and MIMD modes. All test images are 640×480px.

As discussed below, SIMD mode is limited by the speed we can enter and extract data. The performance of the 15×15 convolution is only a half of the 3×3 convolution despite of having 25 times more MAC operations. This is because in the first case, the amount of

computation largely exceeds the time required for data preloading in the SIPO/PISO queues. This does not occur with smaller kernels, were the PEs halt until more data are available. To avoid this issue, many operations must be chained and the on-chip data caches have to be employed to store the partial results. The POP must only extract the final results. This is the case of Stereo Matching, which employs sum of absolute differences (SAD) for block matching (9×9) with a disparity of 32px.

MIMD results show the performance of several operations when implemented in *stream-like* fashion. As seen before, a rate of a pixel-per-cycle can be achievable with few PEs, so it is possible to replicate each mode several times and increasing the throughput. Results show that replicating RGB to Gray and Entropy Encoding 6 times the performance increases practically linearly. Other operations, such as median calculation, are pure arithmetic and can be performed very efficiently without consuming resources. However, others require internal storage as the Register Bank is not large enough. This results in lower throughput as load/store hazards drops the performance if there are not independent instructions to fill the pipeline, as is the case in histogram or integral image calculations.

Program loading represents a small fraction of the whole operation. In SIMD mode, the worst case comes when PIP, POP and SIMD Control instructions storages are filled completely. In the current implementation, all memories are 1024-instructions wide, so less than 22 μs are required to fully program the coprocessor (with a clock frequency of 150 MHz). In MIMD mode, each PE stores its own program, which is 512-instructions wide. In the worst case, all PEs completely fill the memory, requiring at most 460 μs in total. However, this has never happened in the test algorithms. In addition, as it is possible to randomly access each PEs, loading times are usually much lower.

3.2.3 Case of study: feature extraction and matching

In an effort to illustrate the capabilities and advantages of the architecture and how to address the different issues found when mapping an algorithm, a feature detector and matching technique was implemented [13]. This approach requires a previous training stage, where the models of the features we want to find are calculated. From a large set of viewpoint transformations of the target object, the most repeatable features are combined into a compact representation, called Histogrammed Intensity Patch model (HIP). During execution, matching can be performed quickly with bitwise operations between the detected features and the

database. This section outlines the basic steps of this method. A detailed description can be found in [13].

Algorithm description

Corner detection

Many approaches in the literature employ highly complex and accurate techniques for feature detection. The selected approach employs a simple method in order to reduce the computational load. The FAST-9 corner detector was designed with the aim of lowering computing requirements, working directly on pixel data. A 16-pixel ring surrounding the pixel under study is considered, as Figure 3.15(a) shows. If the intensity of 9 adjacent pixels is greater or lower than the center pixel, it is considered as a corner. A non-maximum suppression removes multiple responses for the same corner. Finally, orientation is computed by accumulating the differences between opposite pixels in the pixel ring. This method provides suitable results for many practical applications.

As FAST is a fixed-scale corner detector, the database of features must be built from samples of the target object with different viewpoints, including scale, rotation and affine transformations to achieve scale invariance. The repeatable features on the different viewpoints are clustered and combined into a HIP model, located at the center of each cluster. The feature clustering criterion is directly related with the robustness of the detector. Adding more different characteristics to match all possible transformations of the target object will lead to better accuracy, beyond the limitations of the corner detection and orientation assignment. A detailed discussion of the training stage can be found in [13].

Descriptor calculation

The descriptors are used to distinguish the different features. To extract a descriptor, a sparse 8×8 grid around the corner is considered. This grid is aligned with the orientation previously calculated during the corner detection stage, so rotation and interpolation steps are necessary. Then, the pixel values are quantized into l intensity levels. The boundaries of each bin depend on the mean and standard deviation of the 64 pixels of the grid to achieve invariance to illumination changes. The quantized patch will be compared with the database of HIP models to search for a match.

The HIP model summarizes the information contained on each feature cluster obtained during the training stage, which includes all considered viewpoints. For each feature, of the cluster, a quantized path is obtained. For each location on the grid, the quantized pixels intensities of each patch on the cluster are histogrammed, resulting in 64 independent histograms of l bins each. To reduce storage and computing requirements, the histogram bin values are binarized, setting to 1 the *rare bins*, those with probabilities lower than a certain threshold. This way, matching can be performed by bitwise operations uniquely.

Feature matching

Once at runtime, a set of features of the current frame are obtained using the FAST-9 detector. Then, a quantized patch around each feature is calculated. This patch will be compared with the HIP database with the purpose of finding a match. As HIP stores binary data, a similarity score can be computed with bitwise operations. The best matches will show a small number of rarely observed bins. Counting its number (encoded with ones) will identify the number of match *errors*. Using the HIP model, a feature is represented by an array of $64 \times l$ binary values, which can be compacted into a few data-words to compute a wide number of matches on large databases very fast. This approach is expected to be much faster than strategies based on distance calculations between features.

Techniques such as tree-based search can take advantage of similarity between HIPs in the database to speed-up the matching stage. It is possible to construct a binary-tree, where each child node only differentiates one bit (one bin) from the parent node. This way, similarity scores lower than a threshold eliminate the need to perform matches along that branch of the tree. Otherwise, a search on the entire database would be necessary. The cost of this solution is larger storage requirements, but the benefits typically outweigh this cost.

Algorithm mapping

As it can be extracted from the previous section, the learning stage where the database is built is carried off-line on a PC-based solution. The FPGA will perform only runtime feature detection and matching. There are three main steps in this algorithm. First, the corners have to be located and oriented. Then, the patch around each corner is rotated and quantized. Finally, the matching stage takes place. The algorithm flow is depicted in Figure 3.14. An optional pre-processing stage, such as noise reduction, suits for SIMD mode, as discussed

below, while the post-processing is usually handled by the main CPU, although it can still use the coprocessor to speed-up these tasks.

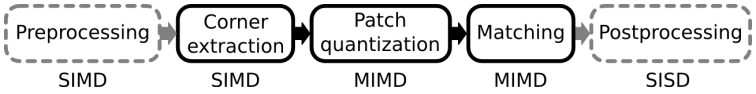


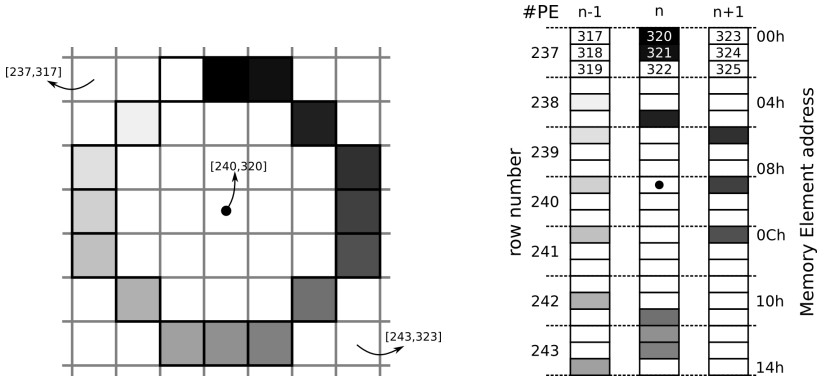
Figure 3.14: Processing modes on the different stages.

FAST corner detection is suitable for SIMD machines. The 128-unit SIMD coprocessor can take advantage of this fact to speed-up computation. In addition, SIMD mode intends to perform as much computation as possible catching data on-chip avoiding external memory accesses. The large size of the Memory Element permits to store tens of rows of VGA images. This way, the execution time is limited by the computing capability and not by the bandwidth of the memory. This applies for other tasks which employ this mode.

In SIMD mode, only the *side-to-side* network is used, allowing to directly access uniquely to the adjacent PEs. An interlacing scheme is needed to access to the farthest pixels as Figure 3.15(a) shows. Each row of the image is read from memory with a step of 3 pixels, so the PE under study (labeled as *N*) can directly access to the complete 16-pixel ring needed for corner and orientation calculation. The PIP permits to read pixels in a non-aligned fashion, resulting in the scheme shown in Figure 3.15(b). This transfer is completely regular, so PIP can do it easily with modulo addressing arithmetic.

Access to data stored in the Memory Element is done manually as memory accesses are sparse, but fixed for all processors. This does not involve additional instructions (data source is included on each instruction) or clock cycles (SIMD mode employs a direct network without latency).

This step generates two images, one with a mask of corner locations and other with the orientation values. Considering 320×240 px images, and taking into account that each PE stores up to 1024 words in SIMD mode, it is possible to store on-chip the full image (600 pixels per PE), leaving space to store 80 full rows for both corner and orientation images. As the number of stored rows is very high, the non-max suppression can be performed directly after a few rows are processed without additional external memory access. Although in larger images the number of stored rows is lower, the same approach can still be used. The array is able to store up to 68 rows of Full HD frames (1920×1080 px), enough for corner extraction.



(a) 16 pixel ring used for FAST-9 corner detection.

(b) Pixel interlacing 16 pixel ring accessing in SIMD mode.

Figure 3.15: Storage scheme in SIMD mode for FAST-9 corner detection.

The results are stored back in the memory by the POP, and post-processed for a compact representation.

The last one is performed efficiently in MIMD mode. Reading sequentially both images, a single PE can discard non-corner pixels, creating an array which contains the corner location and orientation values. This step is shown in Figure 3.16. As PEs work in a pipeline fashion, several corners are processed simultaneously. In addition, the large number of PEs permits to speed up this task by replicating the scheme up to 8 times, thus processing up to 8 rows at a time. Although not shown, a number of PEs simply move the output along the 2D-torus towards the POP without any processing operations. We should note that the *local network* access and synchronization do not consume extra instructions or clock cycles, providing a high throughput.

If the image is blurred, the FAST detector will fail as it is not multi-scale. Although the training database includes blurred samples, an image pyramid will increase robustness. A simple pyramid with a scale factor of 2 by pixel averaging will be used. To speed up runtime execution, the next scale of the pyramid will only be built if the number of detected features is low. This process iterates for a number of scales until the desired number of features is reached. If the number of features is lower than a certain value, the system will perform the same task at the next scale, loading again the original image, downscaling to the half by pixel-averaging, and extracting new features in SIMD mode.

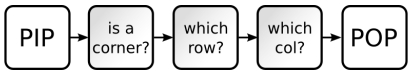


Figure 3.16: MIMD scheme for compacting data after corner detection.

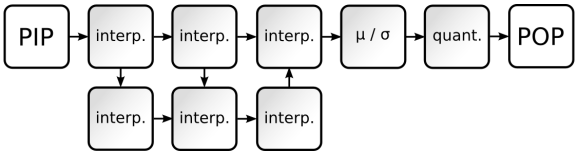


Figure 3.17: MIMD scheme for patch rotation with interpolation and HIP calculation.

To compute the HIP model of each feature in the current frame, an 8×8 neighborhood access is needed. According with the original implementation, 2 px sampling and 5 bins were selected. It is possible to exploit the on-chip memories in SIMD for patch rotation, using precomputed pixel positions and weights and a storage scheme similar to that of the corner extraction. However, as corner distribution is highly irregular along the image, MIMD offers better results. From the corner locations calculated during data packing, a grid around each feature is extracted. This grid is larger than the 8×8 target grid as a rotation must be performed, ensuring all necessary pixels are available. The CPU computes the start and end addresses of the extended grid and configures the PIP appropriately to transfer the pixels and the desired orientation towards the PEs. To ease programming, all grids have the same size, so precomputed pixel positions remain unchanged. Once the patch is rotated, the mean and standard deviation are calculated. Finally, the pixel values of the entire patch are quantized. This process is shown in Figure 3.17. As bilinear interpolation is computationally expensive, several PEs are used to speed-up the calculation. This distribution of PEs can be replicated along the 2D-torus to process several patches at a time. Data storage is not an issue, as each PE is able to handle up to 512 words.

Finally, the matching stage takes place. As mentioned before, a tree-based search can greatly reduce the number of checks for each feature found in the current frame. Clearly, the MIMD mode can take advantage of the distributed and task parallel nature of this stage. However, it is not feasible to directly translate the binary tree to the array as the number of PEs and interconnections is usually much lower than the number of features in the database. In addition, the tree depends on the database (the target object) and it is expected to be irregular, making it difficult to map it to the processing array. A hybrid approach has to be carried out.

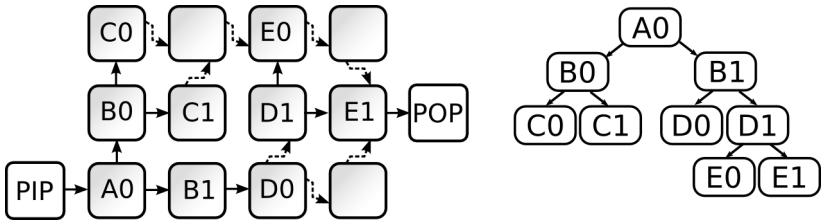


Figure 3.18: Sample tree mapping for HIPs matching in MIMD mode.

As Figure 3.18 shows, database is distributed over the PEs. The feature under study will be propagated along the PEs in one or other direction depending on the score of each match. Small branches will be compacted and executed in one PE. Larger branches will be executed in more than one PE. The *local network* is employed uniquely to map the most important branches, performing computation in the same PE for the rest of children. The PEs located at the end of each branch will lead to the correct match, being the POP responsible to store the results in the external memory. For a 500 features database, a minimum of 10 PEs are needed to fully store the features. Taking into account that a tree-based representation requires more storage space, and that a binary-tree does not match perfectly a 2D-torus arrangement, not all PEs of the array can be used. However, even although a few branches could be parallelized, this considerably reduces the execution time as several features can be checked at a time as the PEs work in a streaming fashion. A small improvement can be introduced if we use PEs of different branches as routers to move the data to the POP, as they act as bridges. These connections are shown with dashed lines in Figure 3.18. As discussed previously, this scheme can be replicated as many times as processors available to check several patches at a time. However, the tree structure depends on the database format and size, and as it usually requires many PEs, this improvement is limited to embed 1-4 trees of 500 features each. To fully store a large database (e.g. 5000 features), a flat scheme has to be selected to save memory resources.

For sake of clarity, Figure 3.19 shows just a set of representative feature matches on a test sequence. From this point, the main CPU, the MicroBlaze, is intended for implementing higher-level tasks although it can still use the coprocessor to improve performance, when possible.

Figure 3.19: Representative feature matches on a test sequence.

Performance

Table 3.3 shows the average performance on 320×240 px test images, compared with the original implementation. The matching stage depends on the number of features to be checked. In our case, a 700 features database is considered for single-target location. Program loading is included and, as mentioned above, it represents a very low overhead. The performance results only considers algorithm execution. The time taken to copy the image into the external RAM via the Ethernet port and retrieve the results back from the RAM via the Ethernet are not computed because the MicroBlaze-based SoC is a prototype conceived for test purposes, so the results would not reflect the actual power of the coprocessor. Results are comparable with those obtained on a 2.4 GHz CPU [13], being possible to locate a target in less than 1 ms. These results also show that it would be possible to process up to 7 simultaneous targets on 640×480 px images in less than 5 ms (compared to 6.03 ms on the CPU). In addition, it enables better integration on embedded devices, and post-processing tasks can also take advantage of the coprocessor to speed-up computation.

	[13] **t (ms)**	**t (ms)**	**Mode**
FAST-9 corner detection	0.45	0.353	SIMD
Corner orientation	-	0.081	SIMD
Corner image-to-array	-	0.002	MIMD
Patch quantization	-	0.193	MIMD
Subtotal	*0.25*	*0.276*	
Matching	0.32	0.251	MIMD
Total	1.02	0.880	

Table 3.3: Average performance in 320×240px images.

3.3 Comparison with other approaches

In spite of the discussed above, the different trade-offs during processor design and implementation may lead to sub-optimal results. As seeing in Section 2.5, other *a priori* less suitable solutions can provide similar or even better performance. Due to the limitations imposed by the technology, exploiting the spatial (data) parallelism may result more beneficial than taking advantage of the temporal (task) parallelism. MIMD units are usually more complex and require more hardware resources, leading to a lower degree of parallelism, so a fast sequential processor or a large SIMD unit can offer comparable performance maintaining better figures of merit in other areas. Nevertheless, large SIMD units require fast memories and a high bandwidth to supply data at optimal rate. If this requirement is not met, a smaller MIMD unit may beat it by parallelizing tasks, serializing part of the computation, although some of these tasks would fit better in a SIMD unit. This section shows a comparison of the proposed hybrid computing platform with other SIMD and MIMD architectures. This gives us a clear picture of the actual performance and the advantages and drawbacks of the proposed architecture.

3.3.1 General-purpose coarse-grain processor array

The retinal vessel-tree extraction algorithm used as benchmark to design and test the coarse-grain processor array will allow us to determine the performance level when executing operations of the low-level stage in the SIMD/MIMD hybrid processor. As SIMD is more suitable for the algorithm used as benchmark than MIMD, it should perform better. By comparing the same algorithm in both architectures, it will be possible to discuss how adding general-purpose capabilities and more flexibility impact in throughput.

The coarse-grain processor array has proved to be efficient, low resource consumer and easy to scale and to program. The low cost FPGA implementation outperforms a high-end CPU by a factor of 10 while employing a more modern and capable FPGA the difference increases up to 110 in the algorithm selected as benchmark (see Table 2.11). In addition, the last option offers similar performance to an MP-SIMD processor, which is the native architecture for the selected algorithm. The coarse-grain processor array employs a 2-dimensional grid of Processing Elements with local interconnections to exchange data. The Processing Elements are made of a dual-port RAM block and a DSP unit for integer and fixed-point arithmetics. The NEWS (north-east-west-south) network between these units permits to fit an image on the plane of processors so that a small sub-window of the image is stored on each processor. This scheme matches most of the operations of the low-level stage.

As detailed in Section 3.2, the SIMD/MIMD hybrid processor is also able to handle these operations. The mode of operation, either SIMD or MIMD, depends on the preceding and subsequent algorithm steps. This avoids an excess of mode switching taking advantage of the on-chip memories, and reducing the number of accesses to the off-chip memory. The image is stored row by row in SIMD mode and the *side-to-side* network permits to exchange data horizontally. Dedicated connection for vertical access is not required as adjacent rows are stored in the same Memory Element. In MIMD mode, these operations are carried out sequentially, employing the Memory Element as row buffers if neighbor access is needed. The most suitable configuration for this algorithm is the SIMD mode due to the algorithm inherent massively spatial parallelism. The processor array is not large enough to run the algorithm in MIMD mode without compromising performance or accuracy, so it makes no sense to employ this mode being the SIMD mode available.

Table 3.4 compares the performance of the algorithm with the previous implementations shown in Section 2.5. The coarse-grain implementation refers to the improved architecture datapath depicted in Section 2.4.3 and implemented on the same Xilinx Virtex-6 FPGA as the SIMD/MIMD hybrid architecture. The cycles-per-pixel (CPP) metric measures the number of clock cycles required to execute all the operations on each one of the pixels in the resulting image, normalizing the differences in frequency and number of processing cores. It allows us to reliably compare both architectures despite of their different implementations. Results show that the hybrid architecture requires only 3.73% more clock cycles to accomplish the same tasks. The approach employed was the same as that of the coarse-grain processor array. A large subwindow of the input retinal image is stored in the on-chip memories. All processing

	Intel Core i7	SCAMP-3	Coarse-grain	Am2045	Hybrid
Processors	4/4	16384/16384	192/192	125/360	128/128
Clock (MHz)	2930	1.25	150	333	150
Window size (px)	–	128×128	384×256	–	768×195
Window time (ms)	–	6.55	30.8	–	63.7
Required windows	1	30	4	1	3
Total time (s)	13.6	0.193	0.123	0.008	0.191
Speed-up (vs. PC)	x1	x70	x110	x1700*	x71
Speed-up (vs. SCAMP)	x0.014	x1	x1.6	x24*	x1
Cycles-per-pixel	357993	8950	7873	742*	8167
Normalized Speed-up	x0.025	x1	x1.14	x12*	x1.096

Table 3.4: Retinal vessel-tree extraction algorithm performance on different processors. * *The Ambric Am2045 runs a simplified version of the algorithm. See Section 2.5.2 for further details.*

is done without additional I/O operations due to the nature of the algorithm, as discussed previously. Therefore, data supply is not the bottleneck in this particular algorithm, representing less than a 4% of the whole computation time. However, generally the SIMD/MIMD hybrid processor will outperform the coarse-grain processor array as data I/O can be accomplished simultaneously to computation. Although this is not critical in the retinal vessel-tree extraction algorithm, it becomes a large advantage when the on-chip memory is not large enough to store all the necessary data and the number of external memory accesses grows, which is the most common case.

3.3.2 SCAMP-3 Vision Chip

The coarse-grain processor array has a large number of processing units, resulting in a high throughput. However, as this number grows, the clock frequency decreases dropping the performance while the amount of hardware resources grows at a rate that makes the most of implementations unpractical. The SCAMP-3 processor is an example of this statement. Although it employs analog computation (current-based arithmetics) to reduce the size of each processing unit, there is an upper limit in scalability in practical implementations. In addition, the maximum clock frequency is much lower than digital architectures. As it can be extracted from Table 3.4, there is no benefit in increasing the number of processing units if these issues are not solved. The SIMD/MIMD hybrid architecture offers much more features and a larger set of operations, including additional computation stages which are unpractical

on the SCAMP-3 processor. For instance, the size of the images are fixed for SCAMP-3 and the coarse-grain processor array. This constraint is not present on the hybrid processor, which is able to handle images of any shape and in a large variety of sizes, including resolutions greater than Full HD. However, we have to remark that focal-plane vision chips focus on early vision, where integrating the sensing stage and a basic computation stage for high frame rate and very low power consumption is a benefit. In this field, they clearly outperform the hybrid architecture, even though it should be noted that this is a very specific application and most general purpose solutions may not meet the requirements.

3.3.3 Ambric Am2045

The MIMD mode of the hybrid processor works in a similar way as the Ambric Am2045 processor. However, there is a major difference. The last is a massively parallel processor array which intends to run the algorithms by parallelizing tasks without strictly requiring a general purpose CPU. The board employed incorporates a PCI Express bridge for fast and high bandwidth communication with a desktop computer. This limits its use on embedded devices or stand-alone systems and increases the cost of the final system. The proposed hybrid processor is intended to systems-on-chip and to reduce the communication gap between CPU and off-chip memory. In addition, the coprocessor was designed to reduce the workload of the CPU in certain tasks and not to replace it.

The Ambric Am2045 is made of a large set of encapsulated processors interconnected through a large network. The processors are optimized for signal-processing operations and includes two different versions with different capabilities in order to save hardware resources and to increase the parallelism. The processing elements of the hybrid architecture are all the same in order to meet the requirements of the SIMD mode. Regarding Ambric's network, it is made of 1-word depth queues which employs a simple handshake protocol. This permits to include a large network with few resources, something necessary when the algorithm complexity grows and more intrincate communications are required. One disadvantage of this network is the small depth of the communication channels, leading to halts on the processors which block the processing. It is possible to configure the distributed RAM in the Ambric device as a buffer to avoid this issue (see Section 2.5.2). The SIMD/MIMD hybrid processor permits to configure this parameter according to the trade-offs of each particular implementation. As the network is not so complex, the hardware resources do not represent a high cost. In addition, it is also possible to employ the Memory Element as a buffer, although in a more

limited way as the access to the network is being blocked, and the control of this buffer has to be done by software, programming the Processing Element adequately.

Considering the retinal-vessel tree extraction algorithm and as it can be drawn from Table 3.4, Ambric's architecture offers a very high throughput when performing low-level operations. However, this was achieved after the reduction of algorithm complexity. Otherwise, the required hardware resources would have to be too large for a practical implementation, and the throughput would be considerably lower. This point was deeply discussed in Section 2.5.3. The MIMD mode of the hybrid processors also experience similar issues as all processing is done on-chip. However, the existence of the SIMD mode makes unnecessary employ the MIMD computing paradigm to implement the algorithm, disappearing all the mentioned issues. This is one clear advantage of using hybrid processors, as it is possible to choose the optimal computing paradigm according to the characteristics of the algorithm.

3.3.4 EnCore processor

The major difference between the MIMD mode of the hybrid processor and the Ambric's architecture is the the kind of computation they were designed to perform. As commented above, Ambric architecture aims to perform all computation on-chip, requiring a more complex dapatah. However, the MIMD mode of the hybrid processor aims to work as an *enhanced pipeline of arithmetic units*, in which each stage applies a small kernel to the input stream, thus parallelizing the processing.

The EnCore processor [191] was designed with a similar philosophy. It is a configurable 32-bit single-issue RISC core which implements the ARCompact instruction set [192]. For its evaluation in silicon, it was integrated within a system-on-chip, including an extension interface to integrate function accelerators. The reconfigurable set extension has a Configurable Flow Accelerator (CFA) architecture, which allows certain customizations in application-specific instruction-set processors (ASIPs) by including a static logic which defines a minimum instruction set architecture (ISA) and a configurable logic for user-defined instruction set extensions (ISE). Custom instructions enable to adapt the instruction set to the requirements of the current algorithm by programming the CFA, including additional arithmetic operations or a combination of them to speed-up critical tasks. In addition, as CFAs are made of several single-function ALUs for resource sharing and are usually pipelined, they can exploit both spatial and temporal parallelism. The combination of these factors permits a large increase of the performance within a limited increase in hardware resources and power consumption.

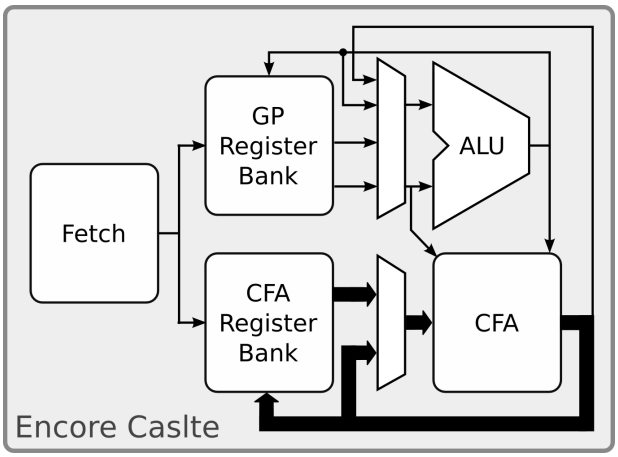


Figure 3.20: Schematic view of the EnCore Castle processor.

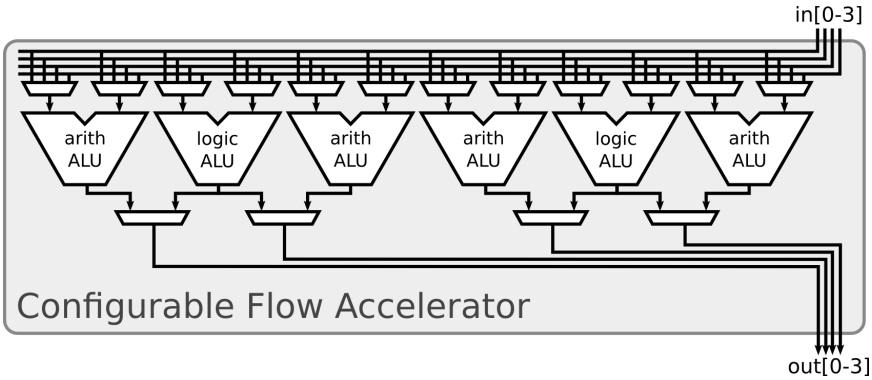


Figure 3.21: Simplified schematic of the EnCore Configurable Flow Accelerator.

Figure 3.20 shows a simplified schematic of the EnCore Castle datapath. Although not shown, the current implementation has a 5-stage pipeline. The Fetch block manages instruction supply. There are two banks of registers. The first, the general-purpose register bank (GP) is employed for the standard ALU of the CPU. The second, stores data for the CFA extension datapath. A simplified schematic of CFA unit is depicted in Figure 3.21. It is made of a set of ALUs and a set of multiplexers which permit to shuffle operands for data alignment. This unit

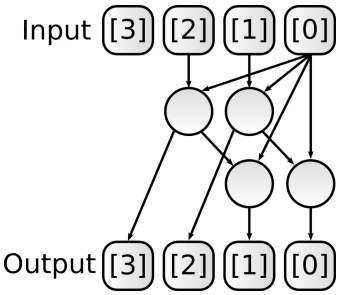


Figure 3.22: Example of custom instruction on the EnCore Configurable Flow Accelerator.

is highly configurable and its datapath runs multi-cycle operations. The CFA register bank supplies a vector of 4 elements to the CFA, storing up to 10 of them. The CFA has a 3-stage pipeline and is able to handle 4 independent arithmetic operations according to the configuration of the particular ISE under execution. The whole design, including the CFA and two 32KB caches, occupies 2.25 mm^2 in 90 nm Silicon and the standard cell library. The design operate at 600 MHz, consuming 70 mW at this clock frequency under typical conditions.

In order to exploit the CFA, a set of ISEs has to be defined. This process allows to analyze the application to identify candidate instructions and modify the source code accordingly to handle these new instructions. This process can be done manually or in an automated way, and can be used to generate a custom CFA or to map a series of templates in an existing one. Figure 3.22 shows how a custom ISE is mapped in the CFA unit. By analyzing the variety of operations of the algorithm, the different operations can be chained and mixed to increase the performance. The EnCore processor employs a design flow for automated construction of ISEs [191]. The design flow employs the compiler as the center of the process to identify and exploit the ISEs, reusing the results between applications. In addition, it includes an additional step to synthesize extensible designs from the extracted ISEs and libraries of standard cells. The process of employing ISEs in existing programs is arduous because of their inherent complexity. They are larger and more complex than standard RISC instructions. In addition, the CFA does not match all ISEs in order to save hardware resources by sharing them between instructions, so they have to be adapted and extra instructions included. Employing a graph representation, several sub-graphs can be combined to maximize performance and reduce latency. However, this makes the CFA very specific, being necessary more hardware resources. When limiting the complexity of the resulting CFA sharing resources, the latency

increases as more ISEs are necessary to perform the same tasks. The design space must be explored to meet a trade-off [193]. In addition, data allocation becomes critical to maximize performance.

To evaluate the performance gain due to the use of CFA, a set of widely used operations were employed as benchmark. Table 3.5 shows the main results of this study. It includes several time-consuming operations of both low- and mid-level stages, including global operations, point and neighbor arithmetics, image transforms and more complex algorithms such as optical flow, disparity map extraction or k-means clustering. Regarding to the parameters shown in Table 3.5, SAD-based stereo matching considers three maximum disparities (16, 32 and 64) and two different block sizes for matching (7 and 11). All images are 640×480px. k-means clustering considers 300 observations and 2, 4 and 6 clusters. Data-parallel operations such as image convolution or pixel arithmetics are largely accelerated, but as the complexity of the operator is reduced, no additional gain is obtained due to the operation chaining. However, more intense operations such as Horn-Schunck optical-flow [194] or SAD-based stereo matching can take advantage of both facts, greatly increasing the performance. Despite of this, the overall performance remains low to be used in video applications, although it may be suitable for off-line applications or systems where response time is not critical. The CFA is not able to provide a large increase of the throughput in other algorithms such as k-means clustering for data mining or global operations for image statistics as mean, maximum/minimum or histogram calculation. Compared to the data intensive test algorithms, the average speed-up is much lower. The test carried out also shown that the memory bandwidth is also limiting the possibilities to achieve optimal performance. As shown in the image transforms, the cache seems not to be able to take advantage of the high degree of spatial parallelism of these operations to correctly preallocate the data in the cache or the CFA register bank. In addition, due to the difference of stages between the EnCore datapath and the CFA datapath, independent instruction must fill the pipeline to fully take advantage of the architecture. In the tested scenarios, this is not always possible, reducing even more the performance.

Although the achieved speed-up is high compared with the unaccelerated execution, the results show that the processor is limited to applications where the CPU workload is not highly intensive. However, the power-consumption against performance ratio is very low, indicating that this processor suits for low power embedded SoCs, where power supply becomes more important than performance. In fact, this processor can be a suitable candidate to replace the MicroBlaze processor on the tested FPGA-based SoC when implemented it on silicon. The

high-level stage can take advantage of the CFA to speed-up certain tasks without employing an external bus to communicate with a coprocessor. In addition, memory management is greatly eased when dealing with programs more complex than the previous stages.

The proposed hybrid processor is not intended to play the same role as the CFA. It was conceived as an *enhanced pipeline* of ALUs but, on the contrary to the CFA, it works independently of the CPU and has their own memory controllers and ports. This permits to improve memory access performance in compute intensive algorithms, which becomes a requirement when upscaling the array of PEs. The memory hierarchy of the Castle processor is more limited and does not permit to scale the CFA easily. Other aspect to take into account is code overhead. The hybrid approach employs very little overhead for synchronization and data transferring between the computing units, permitting to achieve speed-ups close to the peak performance. The EnCore processor needs to manage the CFA unit for each ISE under execution, behaving as a single issue processor, while the hybrid approach has completely independent units for control, computing and data transferring. This results in higher and deterministic performance, as Table 3.5 shows. Despite this, some operations perform better on the EnCore processor due to the integration of the CFA in the processor datapath. Image transformations, although accelerated, are not suitable for the hybrid processor due to the type of memory access. It is possible to perform simple transformations in SIMD mode taking advantage of the large on-chip memories. However, when the data access patterns become irregular, to completely run on the main CPU is a more suitable solution. The results shown in Table 3.5 for image rotation and scaling were obtained using the MIMD mode of the hybrid processor, which only computes the source addresses of the pixels to be copied in the destination image. The output of this process is an array of addresses which will be used by the main CPU to copy the pixels, without any address calculation. This is a sub-optimal implementation as a complete on-chip implementation does not offer satisfactory results. The EnCore processor can fully take advantage of the CFA as it is embedded in the pipeline, so memory access is not an issue.

Operation	Parameters	CPU (ms)	CFA (ms)	Speed-up	Hybrid (ms)	Speed-up (vs. CFA)
Pixel arithmetics	addition/subtraction	42.9	10.2	x4.0	2.11	x5.1
	alpha blending	44.2	10.8		2.11	
2d-convolution	3×3	102.0	29.0	x3.5	2.11	x70.9
	7×7	368.4	104.4		2.11	
	11×11	682.1	194.3		2.11	
	17×17	1665.2	475.8		3.71	
Image displacement	horizontal 20px	26.0	23.6	x1.1	2.34	x9.2
	horizontal 60px	26.0	23.6		2.37	
	vertical 20px	25.3	23.0		2.64	
	vertical 60px	22.9	20.8		2.68	
Image rotation	20^o	28.2	18.8	x1.5	20.71	x0.84
	45^o	28.1	18.7		22.49	
	75^o	28.2	18.8		24.42	
Image scaling	50% (nearest-neighbor)	7.7	7.0	x1.1	9.19	x0.76
	150% (nearest-neighbor)	46.4	4.9		55.66	
	200% (nearest-neighbor)	82.4	74.9		98.94	
	50% (bilinear)	78.0	32.5	x2.5	38.99	x0.76
	150% (bilinear)	1051.4	420.55		504.66	
	200% (bilinear)	2432.8	935.7		1122.83	
Global	max/min	28.7	13.7	x2.1	2.11	x3.8
	mean	12.4	5.9	x2.0	2.11	
	histogram	14.8	11.1	x1.3	4.89	
Horn-Schunck optical-flow	-	3278.4	630.5	x5.1	32.5	x19.4
SAD-based stereo matching	16/7	5327.8	1087.3	x4.9	15.77	x47
	16/11	8746.3	1785.0		35.54	
	32/7	9000.2	1836.8		31.54	
	32/11	14703.7	3000.7		72.09	
	64/7	11571.6	2361.5		63.07	
	64/11	18426.5	3760.5		144.18	

Table 3.5: EnCore Castle and SIMD/MIMD hybrid processor comparison. Average performance in 640×480px images.

3.4 Summary

This chapter has shown an extensive optimization of the architecture proposed in Chapter 2. This first approach was designed to address the low-level stage of most Computer Vision applications, which consumes most of the CPU workload. The coarse-grain processor array has proved to be very efficient in these tasks. However, its flexibility is not the most adequate to handle efficiently the subsequent stages. Computing cores distribution, data representation and I/O operations are not suitable to address the mid- and high-level stages.

The architecture described in this chapter takes the best characteristics of SIMD and MIMD paradigms, and combines them in order to save hardware resources. The proposed SIMD/MIMD coprocessor lights the workload of the main CPU of a system-on-chip. This way, the new architecture can take advantage of both spatial and temporal parallelism, executing, massive operations in SIMD mode, task-parallel operations in MIMD mode and the high-level steps on a sequential processor. Computing paradigm can be changed at runtime as needed according to the requirements of the particular algorithm under execution. I/O management is improved by overlapping computation and data transferring, reducing the memory bottleneck and easing off-chip memory management. One of the goals of this architecture is to provide enough flexibility to handle many different situations, including different data types, image sizes or more abstract representations. For this purpose, two different networks are employed to exchange data between the computing units. Their characteristics ease data distribution and reduce overhead in order to maximize performance. Besides flexibility, another goal is to make the architecture highly configurable, so it does not depend on the number of computing cores, memory sizes or arithmetic operations. This allows to easily create a family of processors to match different applications or different markets, fitting not only performance or power consumption but also cost.

The proposed SIMD/MIMD hybrid processor was tested on an FPGA-based System-on-Chip. A set of operators and algorithms were tested to evaluated its performance and feasibility. Results prove that a high throughput is achievable when the different operation modes are selected appropriately. The architecture was compared with other related approaches, including the coarse-grain processor array proposed in Chapter 2. The results show that the price to pay due to the increase of hardware requirements and the addition of general-purpose capabilities which could penalize the performance of specific tasks, greatly boosts the flexibility without compromising the performance.

Conclusions

Thesis summary

In this work, a novel hardware architecture to speed-up Computer Vision algorithms on embedded systems has been presented. This hardware architecture provides a single-chip device able to run most of the image processing and related algorithms present in the Computer Vision applications.

First, the introduction of the Computer Vision problem and the analysis of different algorithms were done. After reviewing their characteristics, type of operations and program complexity, the different computing paradigms were evaluated. Based on this knowledge, a massively parallel processor array was proposed in order to evaluate its feasibility employing the current technology. This processor embeds a very large number of computing cores arranged in a two dimensional array employing local connections to exchange data. Although processing only binary images, the results show that practical implementations are limited to applications which process low resolution images and where the bandwidth is the major bottleneck of the system. A later version of this processor, extended to deal with gray and color images, improves the results by reducing the parallelism but enhancing the computing units. This architecture, as the previous one, was prototyped on an FPGA and the results validate the improvements included into the architecture. The prototype was evaluated using an algorithm to extract the retinal-vessel tree from retinal images. This high demanding application is representative of the algorithms the processor must be able to address, mainly on the low- and mid-level stages, and includes most of the mathematical operations. The proposed architecture, a scalable coarse-grain processor array which exploits the data parallelism of the algorithms, was compared with a general-purpose desktop CPU, a focal-plane processor

which features a processor-per-pixel scheme and a stream processor, which exploits the task parallelism of the algorithm.

The results obtained from the comparison of the different architectures were very valuable in order to expand the rage of operation of the architecture. One of the conclusions drawn is that a two-dimensional arrangement of the computing units does not provide enough flexibility to implement operations beyond the low- and mid-level stages. In addition, to exploit the task-level parallelism of the algorithms is not possible. The previous comparative showed that this is very important even in algorithms were its presence is reduced. This leads us to design a new architecture, based on the previous one, where the different modules allow to reconfigure on demand and at runtime the computing paradigm.

The latest upgrades greatly increase the flexibility and the performance of the architecture. It can run two operation modes, SIMD and MIMD. In the first mode, all processors run the same instruction. The computing units are arranged into a one-dimensional array with local connection between adjacent units. In this mode, data parallelism can be fully exploited employing the large on-chip storage each computing unit includes. Window-based operators, very important in the first steps of almost all Computer Vision algorithms, can be also implemented by storing several rows of the image adjacently. In MIMD mode, each computing unit runs a small program, a kernel of the whole algorithm. The different kernels are then connected employing a local network in order to build the flow diagram of the algorithm. This mode implements task parallelism natively as all computing units work concurrently. This network is self-managed, so no additional control is needed and the algorithm migration and debugging steps are greatly cut. In both modes, two independent processors managed data I/O between the computing units and the external memory. This permits to operate concurrently data I/O and computation, increasing the overall performance of the system.

The final architecture was prototyped on an FPGA and several algorithms were implemented in order to evaluate the benefits of the included upgrades. Although it features general-purpose capabilities, the results show it is possible to achieve similar performance to the previous version of the architecture for low-level image processing tasks, so it was possible to greatly increase the flexibility without compromise the performance. Besides this review, a comparison with other architectures were also carried out. The results also show the same conclusion; the price to pay due to the increase of hardware requirements and the addition of general-purpose capabilities which could penalize the performance of specific tasks, greatly boosts the flexibility without compromising the performance.

Future work

The architecture proposed in this work provides a solid basis on which it is possible to add new characteristics in order to increase the flexibility and performance, among other figures of merit. In particular and based on the obtained results, a multi-core architecture provides major improvements. To run simultaneously both SIMD and MIMD computing paradigms, besides an internal interconnection to drive data among the different cores, is expected to provide better results than to increase the number of computing units on the single-core proposed architecture.

Beyond structural modifications and besides reconfigurable hardware such as FPGAs, the architecture is intended to be implemented on custom chips. The architecture can be optimized in a much deeper level, specially the arithmetic units, constrained by the available resources of the FPGAs. In addition, this opens new research directions and allows to evaluate more accurately parameters such as power consumption or area requirements.

APPENDIX A

SIMD/MIMD HYBRID PROCESSOR TIMING DIAGRAMS

This appendix shows how the different operations of each module of the dynamically reconfigurable processor described in Chapter 3 are scheduled. The time scale does not indicate the real duration of each operation but how the different tasks are scheduled on each operation mode. A sample task for each mode is described below.

SIMD mode

In SIMD mode, all Processing Elements execute the same instruction and are controlled by the SIMD Control module. The timing diagram refers to them employing the label *PE Array*. In this example task, three blocks of data are copied from the external RAM to the array of PEs and then the results are extracted.

- The main CPU loads the program which PIP, POP and SIMD Control will execute.
- All modules are idle until a *start* flag is asserted by the main CPU.
- The PIP starts filling the SIPO queue according to the program and employing the AGU for address calculation. If the data is cached, it is not necessary program accesses to the external RAM. Data are copied serially in the SIPO until it is filled. The rest of modules wait until this process ends. Once the SIPO is full, the stored data are moved to the ME of each PE. This only takes one clock cycle, leaving the PEs ready to continue

processing. This process is executed as many times as blocks of data are needed by the PEs to start processing (one block, in this example).

- Now, the PEs can start computing. The PIP is able to continue filling the SIPO with a new block of data but it only can be copied when the PE Array finishes processing, as all PEs are busy.
- Once the PE Array finish processing a number of blocks of data, the POP reads the results from the ME of each PE and copies them in the PISO queue. This only takes one clock cycle, leaving the PEs ready to continue processing.
- The POP extracts the results from the PISO serially and move them to the data cache, and then to the external RAM, according to the program it is running.
- This process iterates until all data are processed. At this point, all modules assert an *end* flag which is monitored by the main CPU. It should be noted that, as all modules work in parallel, simultaneous data transfers between the PIP, POP and PE Array are possible, greatly increasing the performance.

Diagram nomenclature: E=empty; F=fill

MIMD mode

In MIMD mode, each Processing Element executes a small task of the whole algorithm. All modules operate concurrently, so task parallelism is exploited natively by the coprocessor. Each one executes the operations programmed and transfers the data to other adjacent PE using the local network. This network is automatically synchronized by the Stream Queues. The size of each program is application-dependent, so it is possible that one PE would be waiting for input data or due to the destination PE is not processing fast enough and its input queue is full. In this example, a data stream is processed by the PEs, acting PIP and POP as the supplier and the receiver of the results, respectively.

- The main CPU loads the program which PIP, POP and all the PEs involved will execute.
- All modules are idle until a *start* flag is asserted by the main CPU.

- The PIP starts supplying data according to the program and employing the AGU for address calculation. If the data are cached, it is not necessary program accesses to the external RAM. Data are copied serially in the input Stream Queues of the destination PE. This process is executed iteratively until the whole stream is transferred.

- Once there are data available in the input Stream Queues of the different PEs, they start processing, exchanging data according to the program they are running.

- The POP extracts the results from the PEs once they are reaching the end of the *pipeline*, i.e., the end of the 2D Torus. Results are stored in the data cache, and then copied to the external RAM, according to the program it is running.

- As soon as the different modules end the processing, they assert an *end* flag which is monitored by the main CPU. It should be noted that all units operate in parallel. There are not conflicts for memory access as the Stream Queues automatically manages synchronization and permits simultaneous read and write access. If one queue is full or empty, the unit which is trying to access remains in *wait state* until more data or space is available.

a) Example: Timing diagram for the SIMD mode.

b) Example: Timing diagram for the MIMD mode.

List of acronyms

AGU Address Generation Unit

ALU Arithmetic Logic Unit

ASIC Application Specific Integrated Circuit

ASIP Application-Specific Instruction Set Processor

CNN Cellular Neural Network

CPU Central Processing Unit)

CWCL Cellular Wave Computing Library

DSP Digital Signal Processor

FAST Features from Accelerated Segment Test

FFT Fast Fourier transform

FPGA Field Programmable Gate Array

GPGPU General-Purpose Graphics Processing Unit

GPU Graphics Processing Unit

HDL Hardware Description Language

IC Integrated Circuit

ITRS International Technology Roadmap for Semiconductors

LUT Look-up Table

MAC Multiply-Accumulate

ME Memory Element

MIMD Multiple Instruction Multiple Data

MISD Multiple Instruction Single Data

MPMC Multi-Port Memory Controller

MPPA Massively Parallel Processor Array

MP-SIMD Massively Parallel Simple Instruction Multiple Data

PLS Pixel-Level Snakes

PE Processing Element

RAM Random-Access Memory

SCAMP SIMD Current-mode Analogue Matrix Processor

SIMD Single Instruction Multiple Data

SIFT Scale-Invariant Feature Transform

SiP System in Package

SISD Single Instruction Single Data

SoC System-on-a-Chip

SURF Speeded Up Robust Feature

SSE Streaming SIMD Extensions

TTM Time to market

VHDL Very-High-Speed Integrated Circuits Hardware Description Language

Bibliography

[1] BCC Research. Machine Vision: Technologies and Global Markets. Report IAS010C, BCC Research, June 2010.

[2] R. Szeliski. *Computer vision: Algorithms and applications*. Springer-Verlag New York Inc, 2010.

[3] Daniel Castaño-Díez, Dominik Moser, Andreas Schoenegger, Sabine Pruggnaller, and Achilleas S. Frangakis. Performance evaluation of image processing algorithms on the GPU. *Journal of Structural Biology*, 164(1):153 – 160, 2008.

[4] Mukul Shirvaikar and Tariq Bushnaq. A comparison between DSP and FPGA platforms for real-time imaging applications. volume 7244, page 724406. SPIE, 2009.

[5] M. Kolsch and S. Butner. Hardware Considerations for Embedded Vision Systems. *Embedded Computer Vision, Springer*, pages 3–26, 2009.

[6] M.J. Flynn. Some computer organizations and their effectiveness. *Computers, IEEE Transactions on*, 100(9):948–960, 1972.

[7] C. Harris and M. Stephens. A combined corner and edge detector. In *Alvey vision conference*, volume 15, page 50. Manchester, UK, 1988.

[8] D.G. Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[9] H. Bay, T. Tuytelaars, and L. Van Gool. Surf: Speeded up robust features. *Computer Vision–ECCV 2006*, pages 404–417, 2006.

[10] K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. *IEEE transactions on pattern analysis and machine intelligence*, pages 1615–1630, 2005.

[11] M. Trajkovi and M. Hedley. Fast corner detection. *Image and Vision Computing*, 16(2):75–87, 1998.

[12] Stephen M. Smith and J. Michael Brady. Susan, a new approach to low level image processing. *International Journal of Computer Vision*, 23:45–78, 1997.

[13] S. Taylor and T. Drummond. Binary histogrammed intensity patches for efficient and robust matching. *International journal of computer vision*, pages 1–25, 2011.

[14] Nikhil R. Pal and Sankar K. Pal. A review on image segmentation techniques. *Pattern Recognition*, pages 1277–1294, 1993.

[15] Robert M. Haralick and Linda G. Shapiro. Image segmentation techniques. *Computer Vision, Graphics, and Image Processing*, 29(1):100 – 132, 1985.

[16] Barbara Zitova and Jan Flusser. Image registration methods: a survey. *Image and Vision Computing*, 21(11):977 – 1000, 2003.

[17] Bill Triggs, Philip McLauchlan, Richard Hartley, and Andrew Fitzgibbon. Bundle adjustment: A modern synthesis. In Bill Triggs, Andrew Zisserman, and Richard Szeliski, editors, *Vision Algorithms: Theory and Practice*, volume 1883 of *Lecture Notes in Computer Science*, pages 153–177. Springer Berlin / Heidelberg, 2000.

[18] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, 1:511, 2001.

[19] N.M. Oliver, B. Rosario, and A.P. Pentland. A bayesian computer vision system for modeling human interactions. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(8):831 –843, aug 2000.

[20] G.R. Bradski and A. Kaehler. *Learning OpenCV*. O'Reilly, 2008.

[21] R. Singhal. Inside intel® next generation nehalem microarchitecture. In *Hot Chips*, volume 20, 2008.

[22] F. Franchetti, S. Kral, J. Lorenz, and C.W. Ueberhuber. Efficient utilization of simd extensions. *Proceedings of the IEEE*, 93(2):409–425, 2005.

[23] S. Naffziger. Microprocessors of the future: Commodity or engine growth? *Solid-State Circuits Magazine, IEEE*, 1(1):76 –82, 2009.

[24] B. Chapman. The multicore programming challenge. *Lecture Notes in Computer Science*, 4847:3, 2007.

[25] C. Urmson, J. Anhalt, D. Bagnell, C. Baker, R. Bittner, MN Clark, J. Dolan, D. Duggins, T. Galatali, C. Geyer, et al. Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics*, 25(8):425–466, 2008.

[26] C.Y. Chu and S.W. Chen. Parallel implementation for cone beam based 3d computed tomography (ct) medical image reconstruction on multi-core processors. In *World Congress on Medical Physics and Biomedical Engineering, September 7-12, 2009, Munich, Germany*, pages 2066–2069. Springer, 2009.

[27] L. Meier, P. Tanskanen, F. Fraundorfer, and M. Pollefeys. Pixhawk: A system for autonomous flight using onboard computer vision. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 2992–2997. IEEE, 2011.

[28] T. Deselaers, D. Keysers, and H. Ney. Features for image retrieval: an experimental comparison. *Information Retrieval*, 11(2):77–107, 2008.

[29] L.J. Li, R. Socher, and L. Fei-Fei. Towards total scene understanding: Classification, annotation and segmentation in an automatic framework. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 2036–2043. IEEE, 2009.

[30] A. Ahmed and K. Terada. A general framework for multi-human tracking. *Journal of Software*, 5(9):966–973, 2010.

[31] W.T. Chen, P.Y. Chen, W.S. Lee, and C.F. Huang. Design and implementation of a real time video surveillance system with wireless sensor networks. In *Vehicular Technology Conference, 2008. VTC Spring 2008. IEEE*, pages 218–222. IEEE, 2008.

[32] Simon Taylor, Edward Rosten, and Tom Drummond. Robust feature matching in 2.3μs. In *IEEE CVPR Workshop on Feature Detectors and Descriptors: The State Of The Art and Beyond*, June 2009.

[33] J. Wither, Y.T. Tsai, and R. Azuma. Mobile augmented reality: Indirect augmented reality. *Computers and Graphics*, 35(4):810–822, 2011.

[34] F. Ren, J. Huang, M. Terauchi, R. Jiang, and R. Klette. Lane detection on the iphone. *Arts and Technology*, pages 198–205, 2010.

[35] John D. Owens, David Luebke, Naga Govindaraju, Mark Harris, Jens Krüger, Aaron E. Lefohn, and Timothy J. Purcell. A survey of general-purpose computation on graphics hardware. *Computer Graphics Forum*, 26(1):80–113, 2007.

[36] S. Che, M. Boyer, J. Meng, D. Tarjan, J.W. Sheaffer, and K. Skadron. A performance study of general-purpose applications on graphics processors using cuda. *Journal of Parallel and Distributed Computing*, 68(10):1370–1380, 2008.

[37] L. Seiler, D. Carmean, E. Sprangle, T. Forsyth, M. Abrash, P. Dubey, S. Junkins, A. Lake, J. Sugerman, R. Cavin, et al. Larrabee: a many-core x86 architecture for visual computing. *ACM Transactions on Graphics (TOG)*, 27(3):1–15, 2008.

[38] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym. Nvidia tesla: A unified graphics and computing architecture. *Micro, IEEE*, 28(2):39–55, 2008.

[39] A. Thall. Extended-precision floating-point numbers for gpu computation. In *ACM SIGGRAPH 2006 Research posters*, pages 52–es. ACM, 2006.

[40] S. Rixner, W.J. Dally, U.J. Kapasi, B. Khailany, A. López-Lagunas, P.R. Mattson, and J.D. Owens. A bandwidth-efficient architecture for media processing. In *Proceedings of the 31st annual ACM/IEEE international symposium on Microarchitecture*, pages 3–13. IEEE Computer Society Press, 1998.

[41] V. Podlozhnyuk. Image convolution with cuda. *NVIDIA Corporation white paper, June*, 2097(3), 2007.

[42] N.K. Govindaraju and D. Manocha. Cache-efficient numerical algorithms using graphics hardware. *Parallel Computing*, 33(10-11):663–684, 2007.

[43] S. Heymann, K. Maller, A. Smolic, B. Froehlich, and T. Wiegand. Sift implementation and optimization for general-purpose gpu. In *Proceedings of the International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*. Citeseer, 2007.

[44] W.L.D. Lui and R. Jarvis. Eye-full tower: A gpu-based variable multibaseline omnidirectional stereovision system with automatic baseline selection for outdoor mobile robot navigation. *Robotics and Autonomous Systems*, 58(6):747–761, 2010.

[45] R. Dolan and G. DeSouza. Gpu-based simulation of cellular neural networks for image processing. In *Proceedings of the 2009 international joint conference on Neural Networks*, pages 2712–2717. IEEE Press, 2009.

[46] V.W. Lee, C. Kim, J. Chhugani, M. Deisher, D. Kim, A.D. Nguyen, N. Satish, M. Smelyanskiy, S. Chennupaty, P. Hammarlund, et al. Debunking the 100x gpu vs. cpu myth: an evaluation of throughput computing on cpu and gpu. In *ACM SIGARCH Computer Architecture News*, volume 38, pages 451–460. ACM, 2010.

[47] R. Schneiderman. Dsps evolving in consumer electronics applications [special reports]. *Signal Processing Magazine, IEEE*, 27(3):6–10, 2010.

[48] T.J. Lin, C.N. Liu, S.Y. Tseng, Y.H. Chu, and A.Y. Wu. Overview of itri pac project-from vliw dsp processor to multicore computing platform. In *VLSI Design, Automation and Test, 2008. VLSI-DAT 2008. IEEE International Symposium on*, pages 188–191. IEEE, 2008.

[49] M. Wang, Y. Wang, D. Liu, Z. Qin, and Z. Shao. Compiler-assisted leakage-aware loop scheduling for embedded vliw dsp processors. *Journal of Systems and Software*, 83(5):772–785, 2010.

[50] G. Talavera, M. Jayapala, J. Carrabina, and F. Catthoor. Address generation optimization for embedded high-performance processors: A survey. *Journal of Signal Processing Systems*, 53(3):271–284, 2008.

[51] Texas Instruments. Tms320c6000 programmer's guide. *White Paper*, 2002.

[52] D. Baumgartner, P. Roessler, W. Kubinger, C. Zinner, and K. Ambrosch. Benchmarks of low-level vision algorithms for dsp, fpga, and mobile pc processors. *Embedded Computer Vision*, pages 101–120, 2009.

[53] C.Y. Lin and Y.P. Chiu. The dsp based catcher robot system with stereo vision. In *Advanced Intelligent Mechatronics, 2008. AIM 2008. IEEE/ASME International Conference on*, pages 897–903. IEEE, 2008.

[54] T.Y. Sun and Y.H. Yu. Memory usage reduction method for fft implementations on dsp based embedded system. In *Consumer Electronics, 2009. ISCE'09. IEEE 13th International Symposium on*, pages 812–815. IEEE, 2009.

[55] S. Shah, T. Khattak, M. Farooq, Y. Khawaja, A. Bais, A. Anees, and M. Khan. Real time object tracking in a video sequence using a fixed point dsp. *Advances in Visual Computing*, pages 879–888, 2008.

[56] K. Suzuki, H. Ikeda, K. Ishimaru, J. Suzuki, F. Adachi, and Xinlei Wang. New image retrieval system utilizing image directory on gigabit network for distributing industrial product information. *Industry Applications, IEEE Transactions on*, 43(4):1099 –1107, july-aug. 2007.

[57] C. Neri, G. Baccarelli, S. Bertazzoni, F. Pollastrone, and M. Salmeri. Parallel hardware implementation of radar electronics equipment for a laser inspection system. *Nuclear Science, IEEE Transactions on*, 52(6):2741 –2748, dec. 2005.

[58] F. Rinnerthaler, W. Kubinger, J. Langer, M. Humenberger, and S. Borbély. Boosting the performance of embedded vision systems using a dsp/fpga co-processor system. In *Systems, Man and Cybernetics, 2007. ISIC. IEEE International Conference on*, pages 1141–1146. IEEE, 2007.

[59] M. Bramberger and B. Rinner. An embedded smart camera on a scalable heterogeneous multi-dsp system. In *In Proceedings of the European DSP Education and Research Symposium (EDERS 2004*. Citeseer, 2004.

[60] P.H.W. Leong. Recent trends in fpga architectures and applications. In *Electronic Design, Test and Applications, 2008. DELTA 2008. 4th IEEE International Symposium on*, pages 137–141. IEEE, 2008.

[61] P.P. Fasang. Prototyping for industrial applications [industry forum]. *Industrial Electronics Magazine, IEEE*, 3(1):4–7, 2009.

[62] I.B. Djordjevic, M. Arabaci, and L.L. Minkov. Next generation fec for high-capacity communication in optical transport networks. *Journal of Lightwave Technology*, 27(16):3518–3530, 2009.

[63] S. Craven and P. Athanas. Examining the viability of fpga supercomputing. *EURASIP Journal on Embedded systems*, 2007(1):13–13, 2007.

[64] P. Coussy, A. Takach, M. McNamara, and M. Meredith. An introduction to the systemc synthesis subset standard. In *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 183–184. ACM, 2010.

[65] F. Moreno, I. Lopez, and R. Sanz. A design process for hardware–software system co-design and its application to designing a reconfigurable fpga. In *Digital System Design - Architectures, Methods and Tools (DSD), 2010 13th Euromicro Conference on*, pages 556–562. IEEE, 2010.

[66] S. Jin, J. Cho, X. Dai Pham, K.M. Lee, S.K. Park, M. Kim, and J.W. Jeon. Fpga design and implementation of a real-time stereo vision system. *Circuits and Systems for Video Technology, IEEE Transactions on*, 20(1):15–26, 2010.

[67] S. Franchini, A. Gentile, F. Sorbello, G. Vassallo, and S. Vitabile. An embedded, fpga-based computer graphics coprocessor with native geometric algebra support. *Integration, the VLSI Journal*, 42(3):346–355, 2009.

[68] M. Martineau, Z. Wei, D.J. Lee, and M. Martineau. A fast and accurate tensor-based optical flow algorithm implemented in fpga. In *Applications of Computer Vision, 2007. WACV'07. IEEE Workshop on*, pages 18–18. IEEE, 2007.

[69] H. Meng, K. Appiah, A. Hunter, and P. Dickinson. Fpga implementation of naive bayes classifier for visual object recognition. IEEE Computer Vision and Pattern Recognition, 2011.

[70] V. Nair, P.O. Laprise, and J.J. Clark. An fpga-based people detection system. *EURASIP journal on applied signal processing*, 2005:1047–1061, 2005.

[71] M.A.M. Salem, K. Klaus, F. Winkler, and B. Meffert. Resolution mosaic-based smart camera for video surveillance. In *Distributed Smart Cameras, 2009. ICDSC 2009. Third ACM/IEEE International Conference on*, pages 1–7. IEEE, 2009.

[72] R. Lysecky and F. Vahid. A study of the speedups and competitiveness of fpga soft processor cores using dynamic hardware/software partitioning. In *Proceedings of the conference on Design, Automation and Test in Europe-Volume 1*, pages 18–23. IEEE Computer Society, 2005.

[73] B.F. Veale, J.K. Antonio, M.P. Tull, and S.A. Jones. Selection of instruction set extensions for an fpga embedded processor core. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, page 8 pp., april 2006.

[74] Y. Shi. Smart cameras for machine vision. *Smart Cameras*, pages 283–303, 2010.

[75] I. Kuon and J. Rose. Measuring the gap between fpgas and asics. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 26(2):203 –215, feb. 2007.

[76] Kwanho Kim, Joo-Young Kim, Seungjin Lee, Minsu Kim, and Hoi-Jun Yoo. A 211 gops/w dual-mode real-time object recognition processor with network-on-chip. In *Solid-State Circuits Conference, 2008. ESSCIRC 2008. 34th European*, pages 462 –465, sept. 2008.

[77] G.P. Stein, E. Rushinek, G. Hayun, and A. Shashua. A computer vision system on a chip: a case study from the automotive domain. In *Computer Vision and Pattern Recognition - Workshops, 2005. CVPR Workshops. IEEE Computer Society Conference on*, page 130, june 2005.

[78] B.K. Khailany, T. Williams, J. Lin, E.P. Long, M. Rygh, D.W. Tovey, and W.J. Dally. A programmable 512 gops stream processor for signal, image, and video processing. *Solid-State Circuits, IEEE Journal of*, 43(1):202 –213, jan. 2008.

[79] S.M. Garrido, J. Listán, L. Alba, C. Utrera, S.Á. Rodríguez-Vázquez, R. Domínguez-Castro, F. Jiménez-Espejo, and R. Romay. The Eye-RIS CMOS Vision System. *Analog circuit design: sensors, actuators and power drivers; integrated power amplifiers from wireline to RF; very high frequency front ends*, pages 15–32, 2008.

[80] A. Lopich and P. Dudek. Aspa: Focal plane digital processor array with asynchronous processing capabilities. In *Circuits and Systems, 2008. ISCAS 2008. IEEE International Symposium on*, pages 1592–1595. IEEE, 2008.

[81] P. Foldesy, Á. Zarándy, and C. Rekeczky. Configurable 3d-integrated focal-plane cellular sensor–processor array architecture. *International Journal of Circuit Theory and Applications*, 36(5-6):573–588, 2008.

[82] M. Koyanagi, Y. Nakagawa, K.W. Lee, T. Nakamura, Y. Yamada, K. Inamura, K.T. Park, and H. Kurino. Neuromorphic vision chip fabricated using three-dimensional integration technology. In *Solid-State Circuits Conference, 2001. Digest of Technical Papers. ISSCC. 2001 IEEE International*, pages 270–271. IEEE, 2001.

[83] T.G. Constandinou, J. Georgiou, and C. Toumazou. Towards a bio-inspired mixed-signal retinal processor. In *Circuits and Systems, 2004. ISCAS'04. Proceedings of the 2004 International Symposium on*, volume 5, pages V–493. IEEE, 2004.

[84] Ã. Zarándy. *Focal-plane sensor-processor chips*. Springer Verlag, 2011.

[85] A. Duller, G. Panesar, and D. Towner. Parallel processing-the picochip way. *Communicating Processing Architectures*, pages 125–138, 2003.

[86] S. Bell, B. Edwards, J. Amann, R. Conlin, K. Joyce, V. Leung, J. MacKay, M. Reif, L. Bao, J. Brown, et al. Tile64-processor: A 64-core soc with mesh interconnect. In *Solid-State Circuits Conference, 2008. ISSCC 2008. Digest of Technical Papers. IEEE International*, pages 88–598. IEEE, 2008.

[87] M. Butts, A.M. Jones, and P. Wasson. A structural object programming model, architecture, chip and tools for reconfigurable computing. In *Field-Programmable Custom Computing Machines, 2007. FCCM 2007. 15th Annual IEEE Symposium on*, pages 55–64. IEEE, 2007.

[88] N. Zhang. Computing optimised parallel speeded-up robust features (p-surf) on multi-core processors. *International Journal of Parallel Programming*, 38(2):138–158, 2010.

[89] D. Bouris, A. Nikitakis, and I. Papaefstathiou. Fast and efficient fpga-based feature detection employing the surf algorithm. In *2010 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines*, pages 3–10. IEEE, 2010.

[90] PK Aby, A. Jose, B. Jose, LD Dinu, J. John, and G. Sabarinath. Implementation and optimization of embedded face detection system. In *Signal Processing, Communication, Computing and Networking Technologies (ICSCCN), 2011 International Conference on*, pages 250–253. IEEE, 2011.

[91] C. Arth and H. Bischof. Real-time object recognition using local features on a dsp-based embedded system. *Journal of Real-Time Image Processing*, 3(4):233–253, 2008.

[92] T.B. Terriberry, L.M. French, and J. Helmsen. Gpu accelerating speeded-up robust features. In *Proc. Int. Symp. on 3D Data Processing, Visualization and Transmission (3DPVT)*, pages 355–362. Citeseer, 2008.

[93] M. Murphy, K. Keutzer, and H. Wang. Image feature extraction for mobile processors. In *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*, pages 138–147. IEEE, 2009.

[94] D. Takahashi. Implementation and evaluation of parallel fft using simd instructions on multi-core processors. In *Innovative architecture for future generation high-performance processors and systems, 2007. iwia 2007. international workshop on*, pages 53–59. IEEE, 2007.

[95] Y. Ogata, T. Endo, N. Maruyama, and S. Matsuoka. An efficient, model-based cpu-gpu heterogeneous fft library. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pages 1–10. IEEE, 2008.

[96] H. He and H. Guo. The realization of fft algorithm based on fpga co-processor. In *Second International Symposium on Intelligent Information Technology Application*, pages 239–243. IEEE, 2008.

[97] X. Guan, H. Lin, and Y. Fei. Design of an application-specific instruction set processor for high-throughput and scalable fft. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 1302–1307, 2009.

[98] K. Pauwels, M. Tomasi, J.D. Alonso, E. Ros, and M.M. Van Hulle. A comparison of fpga and gpu for real-time phase-based optical flow, stereo, and local image features. *IEEE Transactions on Computers*, 2011.

[99] P. Dudek and P.J. Hicks. A general-purpose processor-per-pixel analog simd vision chip. *Circuits and Systems I: Regular Papers, IEEE Transactions on*, 52(1):13–20, 2005.

[100] A. Lopich and P. Dudek. Global operations in simd cellular processor arrays employing functional asynchronism. In *Computer Architecture for Machine Perception and Sensing, 2006. CAMP 2006. International Workshop on*, pages 18–23. IEEE, 2007.

[101] B.K. Khailany, T. Williams, J. Lin, E.P. Long, M. Rygh, D.F.W. Tovey, and W.J. Dally. A programmable 512 gops stream processor for signal, image, and video processing. *Solid-State Circuits, IEEE Journal of*, 43(1):202–213, 2008.

[102] A. Fijany and F. Hosseini. Image processing applications on a low power highly parallel simd architecture. In *Aerospace Conference, 2011 IEEE*, pages 1–12. IEEE, 2011.

[103] N.S. Kim, T. Austin, D. Baauw, T. Mudge, K. Flautner, J.S. Hu, M.J. Irwin, M. Kandemir, and V. Narayanan. Leakage current: Moore's law meets static power. *Computer*, 36(12):68–75, 2003.

[104] Peter Ramm Philip Garrou, Chistopher Bower. *Handbook of 3D Integration*. Wiley-VCH, 2008.

[105] Intel. Intel 22nm 3-d tri-gate transistor technology. *Intel Documentation*, 2011.

[106] G. Koch. Discovering multi-core: extending the benefits of moore's law. *Technology*, 1, 2005.

[107] Nathan Brookwood. Amd fusion family of apus – enabling a superior, immersive pc experience. *AMD white paper*, 2010.

[108] Arnon Friedmann. Enabling small cells with ti's new multicore soc. *Texas Instruments White Paper*, 2010.

[109] Nathan Brookwood. Amd fusion family of apus – enabling a superior, immersive pc experience. *AMD white paper*, 2010.

[110] NVIDIA. Variable smp – a multi-core cpu architecture for low power and high performance. *NVIDIA Corporation white paper*, 2011.

[111] Desh Singh. Higher level programming abstractions for fpgas using opencl. *ALTERA*, 2011.

[112] Keith DeHaven. Extensible processing platform ideal solution for a wide range of embedded systems. *Xilinx White Paper*, 2010.

[113] M. Hassaballah, S. Omran, and Y.B. Mahdy. A review of simd multimedia extensions and their usage in scientific and engineering applications. *The Computer Journal*, 51(6):630–649, 2008.

[114] L.J. Karam, I. AlKamal, A. Gatherer, G.A. Frantz, D.V. Anderson, and B.L. Evans. Trends in multicore dsp platforms. *Signal Processing Magazine, IEEE*, 26(6):38–49, 2009.

[115] M. Ilic and M. Stojcev. Address generation unit as accelerator block in dsp. In *Telecommunication in Modern Satellite Cable and Broadcasting Services (TELSIKS), 2011 10th International Conference on*, volume 2, pages 563–566. IEEE, 2011.

[116] R.P. Kleihorst, AA Abbo, A. van der Avoird, MJR Op de Beeck, L. Sevat, P. Wielage, R. van Veen, and H. van Herten. Xetal: a low-power high-performance smart camera processor. In *Circuits and Systems, 2001. ISCAS 2001. The 2001 IEEE International Symposium on*, volume 5, pages 215–218. IEEE, 2001.

[117] A.A. Abbo, R.P. Kleihorst, V. Choudhary, L. Sevat, P. Wielage, S. Mouy, B. Vermeulen, and M. Heijligers. Xetal-ii: a 107 gops, 600 mw massively parallel processor for video scene analysis. *Solid-State Circuits, IEEE Journal of*, 43(1):192–201, 2008.

[118] S. Kyo, S. Okazaki, and T. Arai. An integrated memory array processor for embedded image recognition systems. *Computers, IEEE Transactions on*, 56(5):622–634, 2007.

[119] J. Poikonen, M. Laiho, and A. Paasio. Mipa4k: A 64$\times$ 64 cell mixed-mode image processor array. In *Circuits and Systems, 2009. ISCAS 2009. IEEE International Symposium on*, pages 1927–1930. IEEE, 2009.

[120] A. Lopich and P. Dudek. Aspa: Asynchronous–synchronous focal-plane sensor-processor chip. *Focal-Plane Sensor-Processor Chips*, page 73, 2011.

[121] J. Fernández-Berni, R. Carmona-Galán, and L. Carranza-González. Flip-q: A qcif resolution focal-plane array for low-power image processing. *Solid-State Circuits, IEEE Journal of*, 46(3):669–680, 2011.

[122] H. Singh, M.H. Lee, G. Lu, F.J. Kurdahi, N. Bagherzadeh, and E.M. Chaves Filho. Morphosys: an integrated reconfigurable system for data-parallel and computation-intensive applications. *Computers, IEEE Transactions on*, 49(5):465–481, 2000.

[123] K. Yamaguchi, Y. Watanabe, T. Komuro, and M. Ishikawa. Design of a massively parallel vision processor based on multi-simd architecture. In *Circuits and Systems, 2007. ISCAS 2007. IEEE International Symposium on*, pages 3498–3501. IEEE, 2007.

[124] L.O. Chua and L. Yang. Cellular neural networks: Applications. *Circuits and Systems, IEEE Transactions on*, 35(10):1273–1290, 1988.

[125] Ian Overington. *Computer Vision: A Unified, Biologically-Inspired Approach*. Elsevier Science Inc., New York, NY, USA, 1992.

[126] A. Rodríguez-Vázquez, G. Liñán-Cembrano, L. Carranza, E. Roca-Moreno, R. Carmona-Galán, F. Jiménez-Garrido, R. Domínguez-Castro, and S.E. Meana. Ace16k: the third generation of mixed-signal simd-cnn ace chips toward vsocs. *Circuits and Systems I: Regular Papers, IEEE Transactions on*, 51(5):851–863, 2004.

[127] Z. Nagy and P. Szolgay. Configurable multilayer cnn-um emulator on fpga. *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on*, 50(6):774–778, 2003.

[128] J.Y. Kim, M. Kim, S. Lee, J. Oh, K. Kim, and H.J. Yoo. A 201.4 gops 496 mw real-time multi-object recognition processor with bio-inspired neural perception engine. *Solid-State Circuits, IEEE Journal of*, 45(1):32–45, 2010.

[129] B. Khailany, W.J. Dally, U.J. Kapasi, P. Mattson, J. Namkoong, J.D. Owens, B. Towles, A. Chang, and S. Rixner. Imagine: Media processing with streams. *Micro, IEEE*, 21(2):35–46, 2001.

[130] W.J. Dally, F. Labonte, A. Das, P. Hanrahan, J.H. Ahn, J. Gummaraju, M. Erez, N. Jayasena, I. Buck, T.J. Knight, et al. Merrimac: Supercomputing with streams. In *Proceedings of the 2003 ACM/IEEE conference on Supercomputing*, page 35. ACM, 2003.

[131] J.C. Chen and S.Y. Chien. Crisp: Coarse-grained reconfigurable image stream processor for digital still cameras and camcorders. *Circuits and Systems for Video Technology, IEEE Transactions on*, 18(9):1223–1236, 2008.

[132] M. Lanuzza, S. Perri, P. Corsonello, and M. Margala. A new reconfigurable coarse-grain architecture for multimedia applications. In *Adaptive Hardware and Systems, 2007. AHS 2007. Second NASA/ESA Conference on*, pages 119–126. IEEE, 2007.

[133] D. Kim, K. Kim, J.Y. Kim, S. Lee, and H.J. Yoo. An 81.6 gops object recognition processor based on noc and visual image processing memory. In *Custom Integrated Circuits Conference, 2007. CICC'07. IEEE*, pages 443–446. IEEE, 2007.

[134] H.J. Siegel, J.B. Armstrong, and D.W. Watson. Mapping computer-vision-related tasks onto reconfigurable parallel-processing systems. *Computer*, 25(2):54–63, 1992.

[135] X. Wang and S.G. Ziavras. Hera: A reconfigurable and mixed-mode parallel computing engine on platform fpgas. In *Parallel and Distributed Computing and Systems*. ACTA Press, 2004.

[136] P. Bonnot, F. Lemonnier, G. Edelin, G. Gaillat, O. Ruch, and P. Gauget. Definition and simd implementation of a multi-processing architecture approach on fpga. In *Proceedings of the conference on Design, automation and test in Europe*, pages 610–615. ACM, 2008.

[137] A. Prengler and K. Adi. A reconfigurable simd-mimd processor architecture for embedded vision processing applications. In *SAE World Congress*, 2009.

[138] B. Kisacanin. Examples of low-level computer vision on media processors. In *Computer Vision and Pattern Recognition-Workshops, 2005. CVPR Workshops. IEEE Computer Society Conference on*, pages 135–135. IEEE, 2005.

[139] M. Wnuk. Remarks on hardware implementation of image processing algorithms. *International Journal of Applied Mathematics and Computer Science*, 18(1):105–110, 2008.

[140] S. Asano, T. Maruyama, and Y. Yamaguchi. Performance comparison of fpga, gpu and cpu in image processing. In *Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on*, pages 126–131. IEEE, 2009.

[141] Celoxica. `http://www.celoxica.com/`.

[142] Xilinx, inc. `http://www.xilinx.com/`.

[143] Itrs. 2011. International Technology Roadmap for Semiconductors, *http://public.itrs.net/*.

[144] George Varghese and Jan M. Rabaey. *Low-Energy FPGAs, Architecture and Design*. Kluwer Academic Publishers, 2001.

[145] L. Kék, K. Karacs, and T. Roska. Cellular wave computing library: Templates, algorithms, and programs. *MTA-SZTAKI, Budapest, version*, 2, 2007.

[146] V.M. Brea, M. Laiho, DL Vilarino, A. Paasio, and D. Cabello. A binary-based on-chip cnn solution for pixel-level snakes. *International journal of circuit theory and applications*, 34(4):383–407, 2006.

[147] C. Rekeczky. Skeletonization and the shortest path problem—theoretical investigation and algorithms for cnn universal chips. In *Proceedings of International Symposium on Non-linear Theory and its Applications (NOLTA'99)*, volume 1, pages 423–426, 1999.

[148] M. Eldesouki, O. Marinov, M.J. Deen, and Q. Fang. Cmos active-pixel sensor with in-situ memory for ultrahigh-speed imaging. *Sensors Journal, IEEE*, (99):1–1, 2011.

[149] C. Alonso-Montes et al. Arteriolar-to-venular diameter ratio estimation: A pixel-parallel approach. *11th International Workshop on Cellular Neural Networks and Their Applications*, pages 86–91, 2008.

[150] C. Mariño, et al. Personal authentication using digital retinal images. *Pattern Analysis & Applications*, 9(1):21–33, 2006.

[151] C. Alonso-Montes, M. Ortega, MG Penedo, and DL Vilarino. Pixel parallel vessel tree extraction for a personal authentication system. In *IEEE International Symposium on Circuits and Systems, 2008*, pages 1596–1599, 2008.

[152] C. Alonso-Montes, DL Vilarino, P. Dudek, and MG Penedo. Fast retinal vessel tree extraction: A pixel parallel approach. *International Journal of Circuit Theory and Applications*, 36(5-6):641–651, 2008.

[153] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, 1(4):321–331, 1988.

[154] L.D. Cohen and I. Cohen. Finite-element methods for active contour models and balloons for 2-D and 3-D images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(11):1131–1147, 2002.

[155] D.L. Vilariño and C. Rekeczky. Pixel-level snakes on the CNNUM: algorithm design, on-chip implementation and applications. *International Journal of Circuit Theory and Applications*, 33(1):17–51, 2005.

[156] P. Dudek, L. Vilarino, et al. A cellular active contours algorithm based on region evolution. In *10th International Workshop on Cellular Neural Networks and Their Applications, 2006. CNNA 2006.*, pages 1–6. IEEE, 2006.

[157] D.L. Vilarino and P. Dudek. Evolution of Pixel Level Snakes towards an efficient hardware implementation. In *IEEE International Symposium on Circuits and Systems, 2007. ISCAS 2007.*, pages 2678–2681.

[158] Carmen Alonso Montes. Automatic Pixel-Parallel Extraction of the Retinal Vascular-Tree: Algorithm Design, On-Chip Implementation and Applications. PhD Thesis, Faculty of Informatics, University of A Coruna, 2008.

[159] J.J. Staal, M.D. Abramoff, M. Niemeijer, M.A. Viergever, and B. van Ginneken. Ridge based vessel segmentation in color images of the retina. *IEEE Transactions on Medical Imaging*, 23(4):501–509, 2004.

[160] Mathworks, inc. `http://www.mathworks.com/`.

[161] Opal kelly. `http://www.opalkelly.com/`.

[162] Cypress semiconductor corporation. `http://www.cypress.com/`.

[163] Virtex-6 Family Overview. In Inc. Xilinx, editor, *DS099 White Paper*. 2010.

[164] H. Feng-Cheng, H. Shi-Yu, K. Ji-Wei, and C. Yung-Chang. High-Performance SIFT Hardware Accelerator for Real-Time Image Feature Extraction. *IEEE Transactions on Circuits and Systems for Video Technology*, 22(3):340–351, 2012.

[165] W.J. MacLean. An evaluation of the suitability of FPGAs for embedded vision systems. In *Computer Vision and Pattern Recognition-Workshops, 2005. CVPR Workshops. IEEE Computer Society Conference on*, page 131. IEEE, 2005.

[166] HM Rode, AS Chiddarwar, and SJ Darak. Suitability of FPGA for computationally intensive image processing algorithms. 2009.

[167] D. Foty. Perspectives on scaling theory and CMOS technology - understanding the past, present, and future. In *Proceedings of the 2004 11th IEEE International Conference on Electronics, Circuits and Systems, 2004. ICECS 2004.*, pages 631 – 637, dec. 2004.

[168] W.J. Dally, U.J. Kapasi, B. Khailany, J.H. Ahn, and A. Das. Stream processors: Progammability and efficiency. *Queue*, 2(1):52–62, 2004.

[169] Á. Rodríguez-Vázquez, R. Domínguez-Castro, F. Jiménez-Garrido, S. Morillas, J. Listán, L. Alba, C. Utrera, S. Espejo, and R. Romay. The eye-RIS CMOS vision system. *Analog Circuit Design*, pages 15–32, 2008.

[170] F. Paillet, D. Mercier, and T.M. Bernard. Second generation programmable artificial retina. In *Proceedings of the Twelfth Annual IEEE International ASIC/SOC Conference, 1999.*, pages 304 –309, 1999.

[171] Alexey Lopich and Piotr Dudek. Asynchronous cellular logic network as a co-processor for a general-purpose massively parallel array. *International Journal of Circuit Theory and Applications*, 2010.

[172] T. Komuro, I. Ishii, M. Ishikawa, and A. Yoshida. A digital vision chip specialized for high-speed target tracking. *IEEE Transactions on Electron Devices*, 50(1):191 – 199, jan 2003.

[173] A. W. Topol, D. C. La Tulipe, L. Shi, D. J. Frank, K. Bernstein, S. E. Steen, A. Kumar, G. U. Singco, A. M. Young, K. W. Guarini, and M. Ieong. Three-dimensional integrated circuits. *IBM Journal of Research and Development*, 50(4.5):491 –506, july 2006.

[174] H. Kurino, M. Nakagawa, K.W. Lee, T. Nakamura, Y. Yamada, K.T. Park, and M. Koyanagi. Smart vision chip fabricated using three dimensional integration technology. *Advances in Neural Information Processing Systems*, pages 720–726, 2001.

[175] P. Foldesy, A. Zarandy, C. Rekeczky, and T. Roska. 3D integrated scalable focal-plane processor array. In *18th European Conference on Circuit Theory and Design, 2007. ECCTD 2007.*, pages 954 –957, aug. 2007.

[176] P. Dudek. Implementation of simd vision chip with 128x128 array of analogue processing elements. In *Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium on*, pages 5806 – 5809 Vol. 6, may 2005.

[177] P. Dudek. A Processing Element for an Analogue SIMD Vision Chip. In *European Conference on Circuit Theory and Design. ECCTD 2003*, volume 3, pages 221–224, 2003.

[178] C. Alonso-Montes, P. Dudek, DL Vilarifio, and MG Penedo. On chip implementation of a pixel-parallel approach for retinal vessel tree extraction. In *18th European Conference on Circuit Theory and Design, 2007. ECCTD 2007.*, pages 511–514. IEEE, 2008.

[179] W. Schroder-Preikschat and G. Snelting. Invasive Computing: An Overview. *Multiprocessor System-on-Chip: Hardware Design and Tool Integration*, page 241, 2010.

[180] M. Butts, A.M. Jones, and P. Wasson. A Structural Object Programming Model, Architecture, Chip and Tools for Reconfigurable Computing. In *15th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 2007. FCCM 2007.*, pages 55 –64, april 2007.

[181] Keith DeHaven. Extensible Processing Platform Ideal Solution for a Wide Range of Embedded Systems. In *Extensible Processing Platform Overview White Paper*. 2010.

[182] B. Hutchings, B. Nelson, S. West, and R. Curtis. Comparing fine-grained performance on the Ambric MPPA against an FPGA. In *International Conference on Field Programmable Logic and Applications, 2009. FPL 2009.*, pages 174 –179, sept 2009.

[183] A. Agarwal. The Tile processor: A 64-core multicore for embedded processing. In *Proceedings of HPEC Workshop*, 2007.

[184] F. Hannig, H. Ruckdeschel, H. Dutta, and J. Teich. Paro: Synthesis of hardware accelerators for multi-dimensional dataflow-intensive applications. *Reconfigurable Computing: Architectures, Tools and Applications*, pages 287–293, 2008.

[185] C.D. Resco, A. Nieto, R.R. Osorio, V.M. Brea, and D.L. Vilarino. A digital cellular-based system for retinal vessel-tree extraction. In *European Conference on Circuit Theory and Design, 2009. ECCTD 2009.*, pages 835 –838, aug. 2009.

[186] M. Niemeijer, J. Staal, B. van Ginneken, M. Loog, and M.D. Abramoff. Comparative study of retinal vessel segmentation methods on a new publicly available database. In *Proceedings of SPIE*, volume 5370, page 648, 2004.

[187] M. Geilen and T. Basten. Requirements on the execution of Kahn process networks. *Programming Languages and Systems*, pages 319–334, 2003.

[188] J. Lee and L. Shannon. The effect of node size, heterogeneity, and network size on FPGA based NoCs. In *International Conference on Field-Programmable Technology*, pages 479 –482, dec. 2009.

[189] N. Alaraje, J.E. DeGroat, and H. Jasani. SoFPGA (System-on-FPGA) architecture: Performance analysis. In *IEEE International Conference on Electro/Information Technology*, pages 551 –556, may. 2007.

[190] A. Benedetti, A. Prati, and N. Scarabottolo. Image convolution on fpgas: the implementation of a multi-fpga fifo structure. In *Euromicro Conference, 1998. Proceedings. 24th*, volume 1, pages 123–130. IEEE, 1998.

[191] O. Almer, R. Bennett, I. Böhm, A. Murray, X. Qu, M. Zuluaga, B. Franke, and N. Topham. An end-to-end design flow for automated instruction set extension and complex instruction selection based on gcc. In *International Workshop on GCC Research Opportunities*, 2009.

[192] Arc international. `http://http://www.synopsys.com/`.

[193] M. Zuluaga and N. Topham. Resource sharing in custom instruction set extensions. In *Application Specific Processors, 2008. SASP 2008. Symposium on*, pages 7–13. IEEE, 2008.

[194] B.K.P. Horn and B.G. Schunck. Determining optical flow. *Artificial intelligence*, 17(1-3):185–203, 1981.

List of Figures

List of Tables