PRECISION: A reconfigurable SIMD/MIMD coprocessor for Computer Vision Systems-on-Chip

Alejandro Nieto, David L. Vilariño and Victor M. Brea

Version: accepted article

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2493527, IEEE Transactions on Computers

PRECISION: A reconfigurable SIMD/MIMD coprocessor for Computer Vision Systems-on-Chip

Alejandro Nieto, David L. Vilariño and Victor M. Brea

Abstract—Computer vision applications have a large disparity in operations, data representation and memory access patterns from the early vision stages to the final classification and recognition stages. A hardware system for computer vision has to provide high flexibility without compromising performance, exploiting massively spatial-parallel operations but also keeping a high throughput on data-dependent and complex program flows. Furthermore, the architecture must be modular, scalable and easy to adapt to the needs of different applications. Keeping this in mind, a hybrid SIMD/MIMD architecture for embedded computer vision is proposed. It consists of a coprocessor designed to provide fast and flexible computation of demanding image processing tasks of vision applications. A 32-bit 128-unit device was prototyped on a Virtex-6 FPGA which delivers a peak performance of 19.6 GOP/s and 7.2 W of power dissipation.

Index Terms—Reconfigurable hardware, parallel processors, FPGA, embedded computer vision

1 Introduction

Computer vision applications are increasingly robust and accurate offering new possibilities for the treatment of visual information. However, this entails an increase in the computing requirements, and their implementation on devices beyond the scope of prototypes is becoming more complex. Novel approaches are required to avoid trimming of the characteristics of the algorithms and to meet trade-offs such as speed, power consumption or cost. 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.

The mathematical operations of the computer vision algorithms can be classified into three groups:

- Repetitive and simple operations over a large set of data, where massive spatial parallelism is key for performance.
- Relatively simple operations with data-dependent program flow, where temporal and task parallelism are easier to exploit than spatial parallelism.
- High-level operations and sets of data organized in a complex manner, where the program flow is very irregular. The computational complexity is reduced when compared with previous groups and suitable for conventional CPUs.

- *The authors are with the Centro de Investigación en Tecnoloxías da Información (CITIUS), University of Santiago de Compostela, Campus Vida, 15782 Santiago de Compostela, Spain.*

The objective of this work is to provide a coprocessor architecture for computer vision with adaptive performance on the heaviest computational steps of the algorithms alleviating the workload of the main CPU. From the flexibility point of view, the system is modular and scalable in order to address different targets and meet different trade-offs such as performance, power dissipation and cost. This coprocessor, named PRECISION (PRrocessor for Embedded Computer vISION), along with the CPU, provides enough flexibility and performance to run efficiently a broad variety of computer vision algorithms on a single device.

1.1 Related Work

Efficient embedded computer vision requires to exploit different levels of parallelism across the different stages of an algorithm in a small footprint. A natural solution to this issue is designing a reconfigurable architecture capable, on a single chip, to find the most suitable type of configuration or parallelism at compilation or runtime [1]. Fixed-type architectures exploiting only one type of parallelism to speed up a given stage of computer vision algorithms might fail to optimize the overall throughput of the computer vision algorithm. Also, their limited programmability hinders their adaptability to changing conditions in the intended application. Next, we outline a representative set of recent hardware architectures for computer vision related to our solution and sorted out according to the taxonomy of fixed-type or reconfigurable solutions with one or multiple levels of parallelism.

CMOS vision sensor chips, which feature acquisition and next to the sensor processing, like SCAMP [2], or chips like those in the Xetal family [3] run only one type of parallelism. The former exploits pixel-level parallelism, while the latter executes row-level parallelism. Both of them follow the SIMD paradigm, and they are very suitable for the lowest

image level processing stages. On the other hand, there are examples of stream processors closer to the pure MIMD paradigm, allowing to execute different tasks at a time. This is the case of the MORA processor [4], which is intended for multimedia processing, or the stream-based Ambric processor [5]. Both of them comprise coarse-grain compute objects working in synchronous mode. Their downside for computer vision applications comes out during early vision when fine-grain data parallelism matches the processing of this stage. The difficulty of stream-based processors to adapt to this type of processing degrades their performance during this stage.

The base architecture with several SIMD units or Processing Elements (PEs) with a varying degree of customization from general purpose to specialized kernels for common computer vision tasks in every SIMD unit (e.g. convolution, gradient or max-min suppression operators) is the approach followed today in many intelligent vision systems (see [6] and references therein). In this context, general purpose SIMD units could be configured as many core clusters for MIMD configuration. Differences with our approach lie in the microarchitecture level of every SIMD unit. Also, specialized kernels permit to increase performance, but at the cost of larger area and possibly of a more complex toolchain, making it difficult to both map them onto an FPGA without a decline in performance metrics, and to use them in practical applications.

In the line of customization of the data path, the memory or the control there are System-on-Chip (SoC) solutions like those reported in [7], [8], [9]. These are state-of-the-art fixed-type architectures, although with multiple levels of parallelism. The SoC reported in [7] comprises an array of 64×64 vision sensors connected to a reconfigurable array of processing elements (PEs) working in SIMD mode, and a set of row parallel (RP) processors. Higher levels of parallelism are tackled with a dual core on-chip processor that permits thread parallelism, and with a Self-Organizing Map (SOM) for pattern recognition tasks. Similarly, the SoC in [8] features SIMD for data parallelism and MIMD for task parallelism with a pipeline multicore architecture. Also, the SoC in [9] contains two set of processors; one of them to exploit row parallelism, and another set for full pixel-level parallelism. Although these solutions achieve impressive performance, e.g. the chip reported in [7] yields more than 1000 fps in face recognition, the dedicated hardware for every type of parallelism on the Silicon substrate makes them very bulky, being very challenging to map them onto a reconfigurable device.

An example of architecture which combines on the same device both SIMD and MIMD modes is IMAPCAR [10]. Although implemented on a SoC, the IMAPCAR architecture could be mapped onto an FPGA with a loss of performance. It comprises a set of 128 processors with their own memory that can be reconfigured at runtime to work in SIMD or MIMD mode. The downside of this approach is that when running MIMD operations, several processing elements are merged into only one MIMD processor, being the number of MIMD processing elements inferior to the whole initial set of 128 processing elements for SIMD. This, along with the overhead time for such a reconfiguration, leads to a degradation in performance. In our architecture, the number of processing elements remains the same for both SIMD and MIMD modes. Also, at microarchitecture level every processing element in our architecture has a higher degree of customization, containing a separate memory for SIMD and MIMD modes, falling down memory access time.

1.2 Overview

The processor addressed in this paper is introduced in Section 2. It includes a detailed description of the main components (I/O processors and processing elements) and the operation modes (SIMD and MIMD). In this section, the instruction set for the coprocessor is also addressed. In Section 3, results from an FPGA prototyping of the coprocessor are presented, including the execution of some usual image processing algorithms and applications together with a discussion on the coprocessor capabilities compared with other architectures. Finally, the conclusions are gathered in Section 4.

1.3 Contributions

The primary contributions of this work are:

- A modular and scalable architecture for embedded computer vision systems, which focuses on easing algorithm migration by natively parallelizing the processing threads and data transfer. The architecture presents capabilities for runtime reconfigurability allowing to adapt the internal datapath to the algorithm execution.
- An FPGA implementation of the architecture and the evaluation of the potential value of reconfigurable datapaths to face computer vision applications.

This paper represents an extended version of [11]. Particularly, a more detailed description of the proposed architecture is included; new sections for instructions sets and scheduling are added; an extended section of performance evaluation and power consumption is presented; and a comparison with other architectures based on the computation of practical image processing algorithms is discussed.

2 PRECISION ARCHITECTURE

Fig. 1 illustrates the main modules of a System-on-Chip including the proposed coprocessor, PRECISION. The embedded CPU is intended to manage irregular data and program flows. These high-level operations do usually represent a small fraction of the whole computation so control tasks are feasible to be executed on the same processor without compromising performance. Although the 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.

The coprocessor architecture consists of three main modules: a Programmable Input Processor (PIP), a Programmable Output Processor (POP) and an array 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,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2493527, IEEE Transactions on Computers

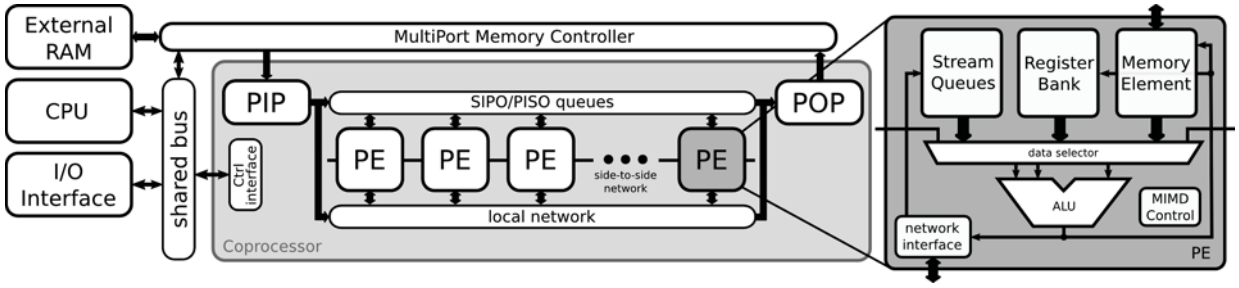


Fig. 1. Simplified scheme of the PRECISION coprocessor integrated on a SoC.

all PEs execute the same instruction over their own data set, located in a local memory. 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 a local memory. However, they are able to exchange information using a *local network*, a 2-dimensional auto-synchronized and distributed network. The modules of the coprocessor architecture are described below.

2.1 Processor datapath

2.1.1 I/O Processors

The I/O processors manage the data transfer between the external memory and the processing elements. The PIP supplies data to the processing array, while the POP collects the results and stores them in the external memory. Both PIP and POP have their own program space and work in parallel, overlapping in/out operations with computation when possible. In addition, data transfer does not need to be synchronous, improving performance when the size of the input data stream is different from the output data stream.

The I/O Processors comprise 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. External RAM memory is interfaced with PIP using a read-only port, while POP employs a write-only port.

I/O processors use the 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 done with a set of quad-registers which configure each pattern and enable to manage several data streams simultaneously. 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 memory bank contains the instructions that PIP or POP run. Each pattern is defined with the four parameters aforementioned. It requires a single instruction to perform the transfer, which sets which one of the quad-registers is employed to calculate source and destination addresses. In order to increase flexibility, the quad-registers can also be managed as independent registers. This permits to modify their value at runtime 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.

2.1.2 Processing Elements Array

Processing Elements in the array are connected through a reduced and programmable network (see Fig. 1). Each PE comprises a Memory Element, an Arithmetic and Logic Unit (ALU), a Register Bank and a set of *Stream Queues* (only used in MIMD mode). Fig. 1 also illustrates the PE architecture, which adopts different working modes (SIMD and MIMD) according to the operation to be performed.

The instruction set includes the standard signal processing and logic operations with 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 (bit-wise 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. The *data selector* drives the operands to the ALU. Input data and output data from previous computations are stored in the two-ports Memory Element or the smaller Register Bank but provided with three independent ports (two read-only, one write-only). The output data can also be transferred to neighboring PEs through the network interface. 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 as showed in Fig. 1.

2.2 Configuration of the processing array

The coprocessor is intended to execute different sub-tasks of a given algorithm in either SIMD or MIMD mode, depending on the type of operations and the algorithm sequence.

2.2.1 SIMD mode

Fig. 2 shows the array and the PEs configured in SIMD mode. The *data selector* drives the operands to the ALU,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2493527, IEEE Transactions on Computers

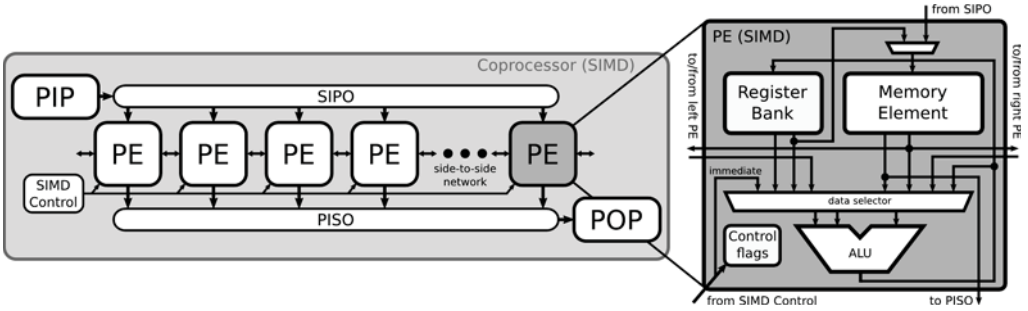


Fig. 2. Coprocessor array layout and Processing Element in SIMD mode configuration.

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 SIMD 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.

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. The Serial-In-Parallel-Out (SIPO) and Parallel-In-Serial-Out (PISO) queues were added to introduce or retrieve data from the Memory Element of each PE (see Fig. 2). 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 (see the Appendix).

The PEs are arranged in 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. 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. This network is made of synchronous direct links with zero-latency, so additional operations or clock cycles are not necessary to share data.

When working in SIMD mode all PEs execute the same instruction but retrieve data from their own data storage, the Memory Element. Therefore, only one controller for all the array is needed, the SIMD Control unit, which stores the 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 the 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 one additional instruction and one address are 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.

2.2.2 MIMD mode

MIMD mode aims to handle irregular program executions with data-dependent execution to take advantage of the task-parallelism. The processing array is configured as an *enhanced pipeline*, where each stage does not execute a single operation but a micro-kernel. Therefore, a suitable algorithm partitioning and the design of an efficient communication pattern between each of the parts is required.

In MIMD mode every PE handles its own code including computation, flow control and network access by the MIMD Control. Each PE works as an independent and encapsulated processor connected to the network, as Fig. 3 depicts. It uses the Memory Element for data and instructions storage. 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 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 through the local network.

The Stream Queues are employed during network access to buffer and synchronize communication. For communi-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2493527, IEEE Transactions on Computers

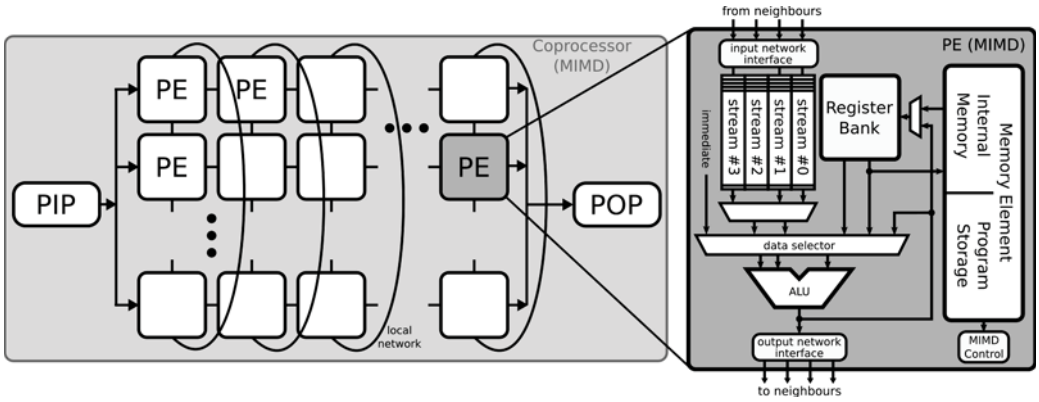


Fig. 3. Coprocessor array layout and Processing Element in MIMD mode configuration.

cation among PEs, 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 communication between processors [12] [13]. 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 are available to be read or there is a memory position available to write the result, ensuring data coherence.

The data access patterns in MIMD mode are the same as in SIMD mode. 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 interface. As shown in Fig. 3, PIP and POP are only connected to the first and last columns of PEs. In MIMD mode, a configurable amount of data are transferred to a single PE, indicated in each transfer. 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. POP works in the same manner.

2.3 SIMD/MIMD configuration

From the operational point of view, the selection of either SIMD or MIMD mode involves two main steps. First, the CPU selects the mode by setting a global flag which directly indicates if the processor is in SIMD or MIMD mode. Then the CPU has to load the programs of all involved modules, this is, the PIP, POP, and the SIMD Control unit (SIMD mode) or the used PEs (MIMD mode). This is done through the Control Interface shown in Fig. 1, which also offers status and performance information.

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 a drop in 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 executed.

2.4 Instruction set

The selection of the instruction set is important to meet some of the trade-offs of the architecture. The coprocessor uses a RISC architecture with regular and compact instructions, permitting to decrease hardware requirements and to ease instruction decoding, a critical aspect for a functional MIMD mode. In addition, RISC architectures are generally faster than their CISC counterpart, with shallower datapaths. 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. Considering that 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. Fig. 4 shows the instruction format indicating the size of the different fields. An optimal set of operands matches 32-bits instructions, allowing particular implementations of the coprocessor with different number of registers. The set of instructions does not depend on the number of PEs.

2.4.1 I/O Processors

PIP and POP are responsible for data transferring. Fig. 4(a) 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

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2493527, IEEE Transactions on Computers

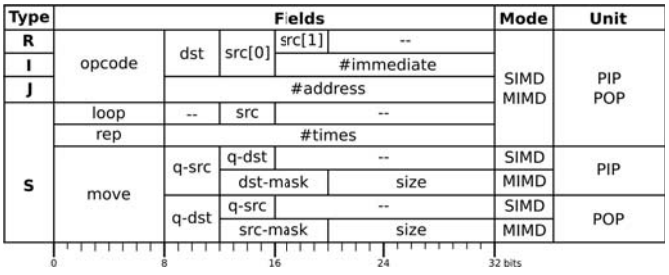
address is a mask which sets the destination PEs of the first row of the array. Data transferring is equivalent for POP, although inverting *source* and *destination* fields.

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 as many times as indicated 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.

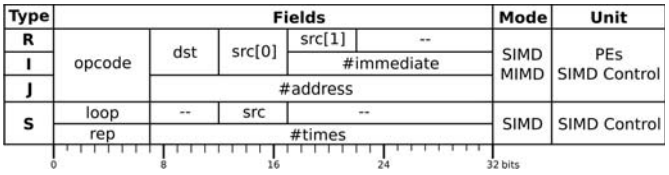
2.4.2 Processing Element

The instruction set for the PEs is the same for both operation modes to ease decoding, although separate control units handle each mode. Therefore, some operations related with flow control are not available in SIMD mode because the SIMD Control unit stores and decodes the program, acting the PEs as stand-alone computation units, while in MIMD mode each PE stores and decodes its own program. Fig. 4(b) shows the instruction format for the PEs.

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 instruction field, the former is calculated using the dual Address Generation Unit, which works in the same manner as the I/O processors do. Except the *move* operation, 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



(a) Data I/O: PIP and POP



(b) Data processing: PEs and SIMD Control

Fig. 4. Instruction format of the different PRECISION units.

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. 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, when the source or destination queues are empty or full, the PE halts until more data or space is available to continue operating (see the Appendix). This ensures data coherence and simplifies algorithm implementation. 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 FPGA PROTOTYPING AND VALIDATION

3.1 FPGA prototyping

PRECISION has been prototyped on an FPGA to evaluate its feasibility and performance, although this architecture is not limited to FPGAs, being an ASIC implementation ideal to show its full potential. The target FPGA was a Xilinx Virtex-6 XC6VLX240T-1, included on the Xilinx ML605 Base Board [14] which provides enough resources to validate the architecture. 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 unit. The coprocessor was described using VHDL and synthesized with Xilinx Design Suite tools [14].

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 the following:

- Instructions and data are represented using 32-bit words. Therefore, the width of all buses in Figs. 2 and 3 is 32-bits.
- PIP and POP have 4 quad-registers of 24-bit wide for off-chip RAM address generation and two caches for instructions and data of 32Kbit (1024 words).
- 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

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

TABLE 1
Synthesized results for a 128-unit 32-bit implementation of PRECISION on a Virtex-6 XC6VLX240T-1 FPGA.

	Power (W)
Static	2.142
Logic	3.850
Clock	0.643
36k-BRAM	0.298
DSP48E1	0.158
Others	0.106
Total	7.197

	Power (W)
PIP	0.044
POP	0.045
PE (each)	0.039
Array of 128 PEs	4.967
Total	5.056

TABLE 2
Power consumption of the 128-unit 32-bit implementation of PRECISION on a Virtex-6 XC6VLX240T-1 FPGA. Left table shows power consumption per FPGA component and right table shows the dynamic power consumption per coprocessor unit.

8 and 4 words respectively. The ALU includes 20 DSP/Boolean and 12 control and data movement operations.

- The array of PEs is made up 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 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 PRECISION is 19.6 GOP/s at 150MHz (around 130 operations per clock cycle). As data transfer occurs in parallel if a careful 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 [14], resulting in 7.197 W at peak performance in standard ambient conditions. 46% corresponds to logic resources and 30% to device static consumption. Table 2 details the power consumption per FPGA component and per coprocessor unit.

3.2 Algorithm evaluation

To evaluate the performance of the proposed architecture, a set of algorithms and operations were executed. They include some representative image processing tasks, covering the SIMD and MIMD operation modes.

3.2.1 SIMD mode

Image filtering or convolution are the most basic and employed operations for tasks such as noise removal, image enhancing or edge detection. They involve intensive computation and neighbor accesses which penalize the overall performance. These window-based operations can be implemented by storing one pixel in each PE, so that each one is able to access the neighborhood of a given pixel using the *side-to-side* network. This has direct consequences. First, it eases memory management as each row is directly mapped onto a 1-dimensional array. Second, the SIMD network has no latency, so the performance is not affected. Fig. 5 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 that 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 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 complete processing of a single image row requires to perform a number of MAC operations equivalent to the number of blocks the image is split to, 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

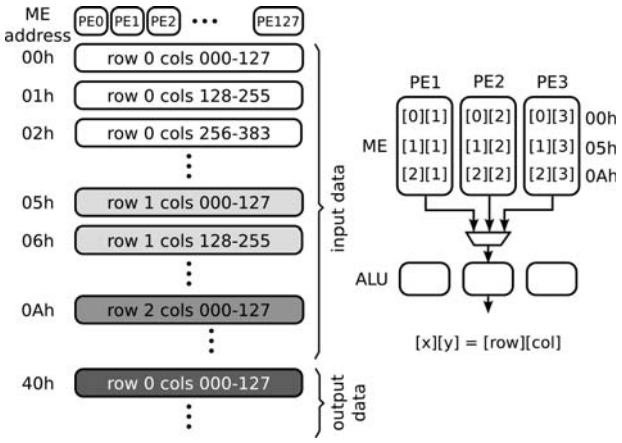


Fig. 5. Memory scheme for a 3×3 convolution on 640 pixel-wide images.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2493527, IEEE Transactions on Computers

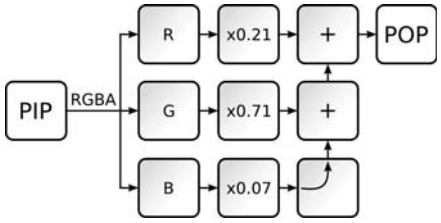
is configured to go over the input image stored in the off-chip memory sequentially (increment is set to $+1$ and modifier, to linear addressing). The destination has the same configuration, although the parameters refer to the Memory Element storage space. The *rep* operation is very useful for fast-looping 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 Fig. 4(b). 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.

3.2.2 MIMD mode

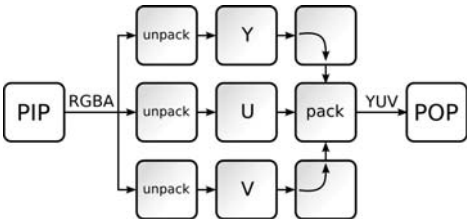
MIMD mode is much more flexible than SIMD as the *local network* permits to better adapt the datapath to the algorithm in execution. Color to gray-scale conversion is an operation which illustrates this capability. Among the plethora of possible conversions, we will consider the following: $Gray = 0.21R + 0.71G + 0.07B$. 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). Fig. 6.a 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 the 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 rightmost PE by using a mask, and employs the Address Generation Unit to store this value in the external memory.

The 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. For instance, RGB to YUV conversion which represents a *many-to-many* conversion. Under the same assumptions as the previous conversion, the different channels of an RGBA value are combined to produce three different channels on the YUV color space. In this case, the



(a) RGB to gray



(b) RGB to YUV

Fig. 6. Color space conversion using MIMD mode.

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. Fig. 6.b shows how to implement this operation in 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 word. 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 instructions are arithmetic operations.

3.2.3 Remarks

Although it is possible to implement color conversion in SIMD mode, the described MIMD-based version illustrates how easy it 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. 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 custom implementations [15]. However, if the row length 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 processes 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 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 the SIMD mode. It eases algorithm implementation as large convolutions natively fits this mode 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.

SIMD

	px/cycle	Mpx/s	#PEs	nJ/px
3x3 Convolution	0.9996	149.9	128	48.0
15x15 Convolution	0.5689	85.3	128	84.4
3x3 Binary Erosion/Dilation	0.9997	150.0	128	48.0
8x8 DCT	0.8136	122.1	128	58.9
Harris Corner Detector	0.3130	79.6	128	90.4
Stereo Matching (SAD,9,32)	0.0455	6.8	128	1058.4

MIMD

	px/cycle	Mpx/s	#PEs	nJ/px
RGB to Gray	0.9998	150.0	9	3.37
6x RGB to Gray	5.8988	884.8	55	3.50
RGB to YUV	0.9997	150.0	9	3.37
Entropy Encoding (4x4)	0.5781	86.7	14	9.08
6x Entropy Encoding (4x4)	3.4108	511.6	90	9.89
Median	0.9999	150.0	1	0.37
Histogram	0.4311	64.7	1	0.87
Integral image	0.6135	92.0	5	3.06

TABLE 3
Performance of SIMD/MIMD modes for usual image-processing tasks.

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 shows the most relevant results of several tasks in SIMD and MIMD modes. All test images are 640×480px.

Note that 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, where 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 purely 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

Operation	t (ms)	Mode
Detection	0.353	SIMD
Orientation calculation	0.081	SIMD
Image-to-array conversion	0.002	MIMD
Description	0.193	MIMD
Matching	0.251	MIMD
Total	0.880	

TABLE 4
Feature detection and matching performance in 320×240px images employing PRECISION.

completely fill the memory, requiring at most $460\ \mu s$ 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.

Note that the performance of computing those relative simple algorithms can be optimized by using custom accelerators. However, the cost of integrating dedicated accelerators is also higher. The aim of the proposed coprocessor is to perform multiple tasks in either SIMD or MIMD mode assuming a trade-off between performance and cost.

With the aim of showing the capabilities of the proposed coprocessor to perform complex computer vision algorithms, a feature detection and matching technique has been implemented. The selected algorithm, proposed in [16] has been partitioned in subtasks which were executed in either SIMD or MIMD model according with their characteristics.

The feature detection is carried out by the FAST corner detector. A 16 pixel ring surrounding the pixel under study is considered. If the intensity of 9 adjacent pixels is greater or lower than the center pixel, it is considered as a corner. FAST corner detection is suitable for SIMD computation. However, since the PEs array working in SIMD mode presents a side-to-side connectivity, an interlacing scheme is needed to access to the farthest pixels. Each row of the image is read from memory with a step of 3 pixels, so a PE can directly access to the complete 16- pixel ring. The PIP permits to read pixels in a non-aligned fashion. This transfer is completely regular, so PIP can do it easily with modulo addressing arithmetic.

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. Therefore, two output images are provided: one containing the corner locations and another with the corner orientations. These images are compacted in a vector representation by the PEs working in a pipelining configuration (MIMD mode). Before the matching stage it is needed to extract the descriptors for the previously detected features. To this end, an extended patch around every feature location is rotated and interpolated according with the feature orientation. Then the intensity pixels in an 8×8 grid around the feature location is considered to compose the feature descriptor. 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.

Finally, the matching between the detected features and those stored in a database are carried out. The strategy for matching depends on the number of features to be checked. For relative small databases a tree-based approach allows to split similarities among feature models and then accelerate the computation. We have followed this approach for a database of 700 features. Table 4 summarizes the average performance on QVGA test images. This implementation which combines different steps working in SIMD and MIMD modes permits to process up to 7 simultaneous targets on 640×480px images in less than 5 ms. Further details of the algorithm implementation and mapping issues can be found in [17].

3.3 Discussion

Most of the image processing tasks fit better on SIMD computation than on MIMD computation. However, from the perspective of the architecture design is not always clear what the best paradigm is. 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 outperform 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 PRECISION coprocessor with other SIMD and MIMD architectures. This gives us a clearer picture of the actual performance and the advantages and drawbacks of the proposed architecture.

In [18] an algorithm for retinal vessel segmentation has been deeply analyzed from the computation point of view. The algorithm, proposed in [19], was originally designed to be executed on massively parallel processor arrays following the SIMD paradigm. All the steps of the algorithm consist of simple local dynamic convolution and morphological hit and miss operations together with simple arithmetic and logical operations. The algorithm was tested in four different architectures, a standard PC (Intel Core i7 940 2.93 GHz), a coarse-grain processor array (custom FPGA implementation [20]), a focal-plane processor array (SCAMP vision system [21]) and a MIMD processor array (Ambric Am2045 [5]).

With the aim of comparing the proposed system with those architectures under the same conditions this retinal vessel-tree extraction algorithm has been implemented on the proposed coprocessor. The most suitable configuration for this algorithm is the SIMD mode due to the algorithm inherent massively spatial parallelism. On the other hand, the processor array is not large enough to run the algorithm in MIMD mode without compromising performance or accuracy. Table 3 shows performance data from the algorithm execution on the coprocessor together with those addressed in [18] for the mentioned architectures.

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

	Intel Core i7	SCAMP-3	Coarse-grain	Am2045*	PRECISION
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
Cycles-per-pixel	357993	8950	7873	742*	8167
Maximum Average Accuracy	0.9202	0.9180	0.9192	0.8132*	0.9191

TABLE 5
Retinal vessel-tree extraction algorithm performance on different processors.
**The Ambric Am2045 runs a simplified version of the algorithm. See Sec. 3.3 for further details.*

on the plane of processors so that a small sub-window of the image is stored on each processor. Both the coarse-grain implementation and the proposed coprocessor were implemented on the same Xilinx Virtex-6 FPGA. Results show that the SIMD/MIMD architecture requires only 3.73% more clock cycles to accomplish the same tasks. All processing is done without additional I/O operations due to the nature of the algorithm. Therefore, data supply is not the bottleneck in this particular algorithm, representing less than 4% of the whole computation time. However, generally PRECISION 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.

The SCAMP processor employs analog computation to reduce the size of each processing unit. However, there is an upper limit in scalability in practical implementations. In addition, the maximum clock frequency is much lower than digital architectures. The proposed architecture offers much more features and a larger set of operations, including additional computation stages which are unpractical on the SCAMP processor. For instance, the size of the images are fixed for SCAMP and the coarse-grain processor array. This constraint is not present in PRECISION, 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 proposed architecture, even though it should be noted that this is a very specific application and most general purpose solutions may not meet the requirements.

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 distinct capabilities in order to save hardware resources and to increase the parallelism. The processing elements of PRECISION are all the same in order to meet the requirements of the SIMD mode. Regarding Ambric network, it is made of 1-word depth queues which employ a simple handshake protocol. This permits to include a large network with few resources. One disadvantage of this network is the small depth of the communication channels, leading to halts on the processors which block the processing. However, it is possible to configure the distributed RAM in the Ambric device as a buffer to avoid this issue. The proposed coprocessor 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.

Ambric architecture offers a very high throughput when performing low-level operations, as can be also seen from the performance on the retinal-vessel tree extraction algorithm, reported in Table 5. However, this was achieved after the reduction of the algorithm complexity which affects the segmentation accuracy. Otherwise, the required hardware resources would have to be too large for a practical implementation, and the throughput would be considerably lower. The MIMD mode of PRECISION also experiences 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 this algorithm. 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.

The major difference between the MIMD mode of the coprocessor and the Ambric architecture is the kind of computation they were designed to perform. Ambric architecture aims to perform all computation on-chip, requiring a more complex datapath. However, the MIMD mode of the coprocessor 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 [22] was designed with a similar philosophy. It is a configurable 32-bit single-issue RISC core which implements the ARCompact instruction set. For its evaluation in Silicon, it was integrated within a System-on-Chip, including an extension interface for reconfigurable accelerators. The specific reconfigurable accelerator of the Encore processor, called Configurable Flow Accelerator (CFA), allows customizations in application-specific instruction-set processors (ASIPs) through the use of user-defined instruction set extensions. The CFA entails only a limited increase in hardware resources and power consumption, but usually implies a large increase in performance. This is achieved by making use of several single-function ALUs that allow spatial and temporal parallelism through resource sharing

Operation	Parameters	CFA (ms)	PRECISION (ms)	Average Speed-up
Global	max/min	13.7	2.11	x3.8
	mean	5.9	2.11	
	histogram	11.1	4.89	
Image displacement	horizontal 20px	23.6	2.34	x9.2
	horizontal 60px	23.6	2.37	
	vertical 20px	23.0	2.64	
	vertical 60px	20.8	2.68	
Image rotation	20º	18.8	20.71	x0.84
	45º	18.7	22.49	
	75º	18.8	24.42	
Image scaling	50% (nearest-neighbor)	7.0	9.19	x0.76
	150% (nearest-neighbor)	4.9	55.66	
	200% (nearest-neighbor)	74.9	98.94	
	50% (bilinear)	32.5	38.99	x0.76
	150% (bilinear)	420.55	504.66	
	200% (bilinear)	935.7	1122.83	
Horn-Schunck optical-flow	-	630.5	32.5	x19.4
SAD-based stereo matching	16/7 (disparity/window)	1087.3	15.77	x47
	16/11	1785.0	35.54	
	32/7	1836.8	31.54	
	32/11	3000.7	72.09	
	64/7	2361.5	63.07	
	64/11	3760.5	144.18	

TABLE 6
EnCore (with CFA) and PRECISION coprocessor comparison. Average performance in 640×480px images.

and pipelining. In addition, the CFA is fully programmable, supporting up to 64 reconfigurable extension instructions.

PRECISION working in MIMD mode can be placed midway between the CFA and the Ambric architecture. It was conceived as an enhanced pipeline of ALUs but, contrary to 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 Encore processor is more limited and does not permit to scale the CFA easily. Another aspect to take into account is code overhead. PRECISION 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 coprocessor has completely independent units for control, computing and data transferring. Table 6 shows the performance of some typical image processing algorithms executed in both the EnCore and PRECISION. Note that 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 PRECISION 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 run them completely on the main CPU is a more suitable solution. The results shown in Table 6 for image rotation and scaling were obtained using the MIMD mode of the PRECISION processor, which only computes the source addresses of the pixels to be copied in the destination image.

4 Conclusion

In this work, a hardware architecture intended to efficiently compute embedded vision applications named PRECISION is introduced. It consists of a reconfigurable processor array which can be dynamically arranged for SIMD or MIMD computation depending on the operation to be performed. The architecture shares functional units to reduce hardware consumption and includes a flexible network to adapt the datapath to the algorithm to be executed. The proposed coprocessor was tested on a System-on-Chip deployed on a Virtex-6 FPGA, providing 128-units of 32-bit each with a peak performance of 19.6 GOP/s consuming around 7.2W.

Results indicate that this solution provides enough flexibility to take advantage of different data- and instruction-parallel processing, making it suitable as coprocessor of high-throughput vision systems.

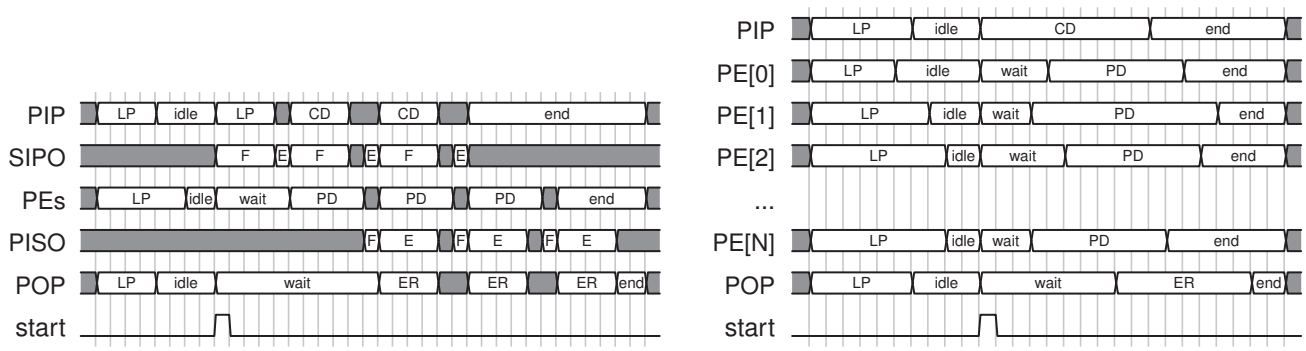
Appendix

This section shows how the different operations of each module of the coprocessor 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 *PEs*. In this example, three data blocks are copied from the external RAM to the array of PEs and then the results are extracted.

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



(a) SIMD mode (b) MIMD mode

Fig. 7. Timing diagram for SIMD and MIMD operating modes. (LD)=Load Program, (CD)=Copy Data, (PD)=Process Data, (ER)=Extract Results, (E)=Empty, (F)=Fill.

- The PIP starts filling the SIPO queue according to the program and employing the AGU for address calculation. 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 Memory Element 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 can only be copied when the PE Array finishes processing, as all PEs are busy.
- Once the PE Array finishes processing a number of blocks of data, the POP reads the results from the Memory Element 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 being run.
- This process iterates until all data are processed. At this point, all modules assert an *end* flag which is monitored by the CPU. Since all modules work in parallel, simultaneous data transfers between the PIP, POP and PE Array are possible.

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 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. 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.
- The POP extracts the results from the PEs once they are reaching the end of the *pipeline*. Results are stored in the data cache, and then copied to the external RAM.
- As soon as the different modules end the processing, they assert an *end* flag which is monitored by the CPU. It should be noted that all units operate in parallel. There are not conflicts for memory access as the Stream Queues automatically manage synchronization and permit 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.

ACKNOWLEDGMENT

This work is funded by the Ministry of Science and Innovation, Government of Spain (projects TIN2013-41129-P and TEC2012-38921-C02-02) and the Xunta de Galicia (contract GRC 2014/008).

REFERENCES

[1] R. Tessier, K. Pocek, and A. DeHon, "Reconfigurable Computing Architectures," *Proceedings of the IEEE*, vol. 103, no. 3, pp. 332–354, 2015.

[2] S. J. Carey, A. Lopich, D. R. Barr, B. Wang, and P. Dudek, "A 100,000 fps vision sensor with embedded 535GOPS/W 256× 256 SIMD processor array," in *VLSI Circuits (VLSIC), 2013 Symposium on*. IEEE, 2013, pp. C182–C183.

[3] Y. Pu, Y. He, Z. Ye, S. M. Londono, A. A. Abbo, R. Kleihorst, and H. Corporaal, "From Xetal-II to Xetal-Pro: On the road toward an ultralow-energy and high-throughput SIMD processor," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 21, no. 4, pp. 472–484, 2011.

[4] 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*. IEEE, 2007, pp. 119–126.

[5] 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*. IEEE, 2007, pp. 55–64.

[6] N. Chandramoorthy, G. Tagliavini, K. Irick, A. Pullini, S. Advani, S. Al Habsi, M. Cotter, J. Sampson, V. Narayanan, and L. Benini, "Exploring architectural heterogeneity in intelligent vision systems," in *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*. IEEE, 2015, pp. 1–12.

[7] C. Shi, J. Yang, Y. Han, Z. Cao, Q. Qin, L. Liu, N.-J. Wu, and Z. Wang, "A 1000 fps Vision Chip Based on a Dynamically Reconfigurable Hybrid Architecture Comprising a PE Array Processor and Self-Organizing Map Neural Network," *Solid-State Circuits, IEEE Journal of*, vol. 49, no. 9, pp. 2067–2082, 2014.

[8] G. Kim, K. Lee, Y. Kim, S. Park, I. Hong, K. Bong, and H.-J. Yoo, "A 1.22 TOPS and 1.52 mW/MHz Augmented Reality Multicore Processor With Neural Network NoC for HMD Applications," *Solid-State Circuits, IEEE Journal of*, vol. 50, no. 1, pp. 113–124, 2015.

[9] H. Zhu and T. Shibata, "A Real-Time Motion-Feature-Extraction VLSI Employing Digital-Pixel-Sensor-Based Parallel Architecture," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 24, no. 10, pp. 1787–1799, 2014.

[10] S. Kyo and S. Okazaki, "Imapcar: A 100 gops in-vehicle vision processor based on 128 ring connected four-way vliw processing elements," *Journal of Signal Processing Systems*, vol. 62, no. 1, pp. 5–16, 2011.

[11] A. Nieto, D. Vilariño, and V. Brea, "SIMD/MIMD dynamically-reconfigurable architecture for high performance embedded vision systems," in *23rd IEEE International Conference on Application Specific Systems, Architectures and Processors (ASAP2012)*, 2012.

[12] 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*, dec. 2009, pp. 479 –482.

[13] N. Alaraje, J. DeGroat, and H. Jasani, "SoFPGA (System-on-FPGA) architecture: Performance analysis," in *IEEE International Conference on Electro/Information Technology*, may. 2007, pp. 551 –556.

[14] Xilinx, inc. http://www.xilinx.com/. [Online; accessed 25-July-2015].

[15] 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*, vol. 1. IEEE, 1998, pp. 123–130.

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

[17] A. Nieto, D. Vilariño, and V. Brea, "Feature Detection and Matching on a SIMD/MIMD Hybrid Embedded Processor," in *8th IEEE Workshop on Embedded Vision (EVW2012)*, 2012.

[18] A. Nieto, V. Brea, D. Vilariño, and R. Osorio, "Performance analysis of massively parallel embedded hardware architectures for retinal image processing," *EURASIP Journal on Image and Video Processing*, vol. 2011, no. 1, p. 10, 2011.

[19] C. Alonso-Montes, D. Vilariño, P. Dudek, and M. Penedo, "Fast retinal vessel tree extraction: A pixel parallel approach," *International Journal of Circuit Theory and Applications*, vol. 36, no. 5-6, pp. 641–651, 2008.

[20] A. Nieto, V. Brea, and D. Vilariño, "FPGA-accelerated retinal vessel-tree extraction," in *Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on*. IEEE, 2009, pp. 485–488.

[21] 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*, 2005, pp. 5806 – 5809 Vol. 6.

[22] 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 *Proceedings of 1st International Workshop on GCC Research Opportunities*, 2009.

Alejandro Nieto received the PhD. degree from the University of Santiago de Compostela (USC) in 2012. Previously, he obtained the M.S. degree in Information Technology and the B.S. degree in Physics Science specializing in Electronics, both from USC. His current research is focused on the development of new reconfigurable architectures optimized for image and video processing with emphasis on performance. Other research interests include multi-core/many-core and custom parallel architectures, reconfigurable computing, parallel programming, embedded systems, FPGAs and digital ASIC design, besides Computer vision applications.

David López Vilariño received the Ph.D. degree from in 2001 from the University of Santiago de Compostela, Spain, where he is currently associate professor in Electronics. He is the author or co-author of around 80 publications on the design and hardware implementation of algorithms for computer vision and medical image processing. His current research is on the design and implementation of image processing algorithms for embedded processor and FPGA-based architectures with fast and efficient computation as main concerns.

Víctor M. Brea received his PhD in Physics in 2003. Currently he is an Associate Professor at Centro de Investigación en Tecnoloxías da Información (CiTIUS), University of Santiago de Compostela, Spain. His main research interests lie in the design of efficient architectures and CMOS solutions for computer vision, especially in early vision, as well as micro energy harvesting.