Hardware Counters Based Methods for the Analysis of Shared Memory Parallel Codes

Oscar García Lorenzo
José Carlos Cabaleiro Domínguez
Tomás Fernández Pena

CITIUS
24/05/2013

Index

1 Introduction

2 Manycore Issues

3 PEBS

4 Memory Analysis

5 Performance Analysis

6 Automatic Optimisation

7 Recap and Future Work

Index

1 Introduction

2 Manycore Issues

3 PEBS

4 Memory Analysis

5 Performance Analysis

6 Automatic Optimisation

7 Recap and Future Work

Introduction
Manycore Issues

Memory Issue

Heterogeneity Issue

Memory Issue Solutions (Not touching the code)

Heterogeneity Issue Solutions (Not touching the code)

Introduction
Precise Event Based Sampling

- Hardware Counters in Intel Processors.
- We use them to identify the issues.
- Samples the state of the core (Registers and Counters).
- Saves samples in a buffer.
- Very precise and low overhead.

Introduction Tools

Memory Analysis

Visual Memory Analysis Tool

Memory Issue

Performance Analysis

Visual Performance Analysis Tool

Memory Issue

Heterogeneity Issue

Automatic Optimisation

Runtime Migration Tool

Memory Issue Solutions

Heterogeneity Issue Solutions

Index

1 Introduction

2 Manycore Issues

3 PEBS

4 Memory Analysis

5 Performance Analysis

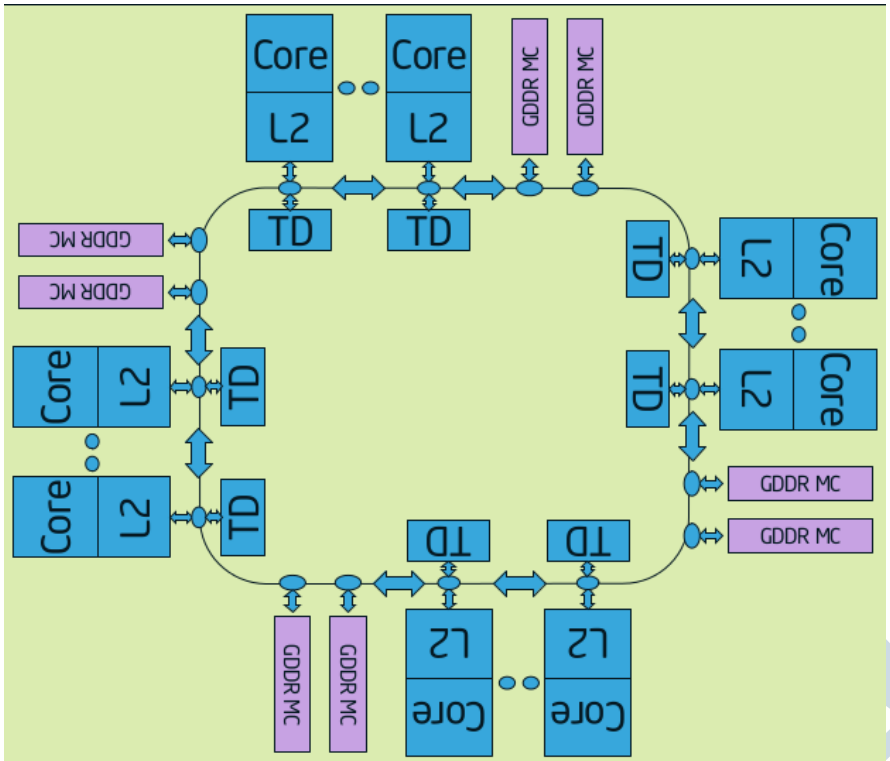
6 Automatic Optimisation

7 Recap and Future Work

Memory Issue

Memory Issue

- There is `affinity` among Cores and Memory:
  - Memory cells are at different distances.
  - There are complex interconnexions.
  - Data can be placed anywhere.
- Data has `locality`.
  - Caches store data.
  - Some levels are shared among cores.
  - Cores read from other core's caches, many hops.



Intel Xeon Phi ring.

Memory Issue
Memory Issue Solutions

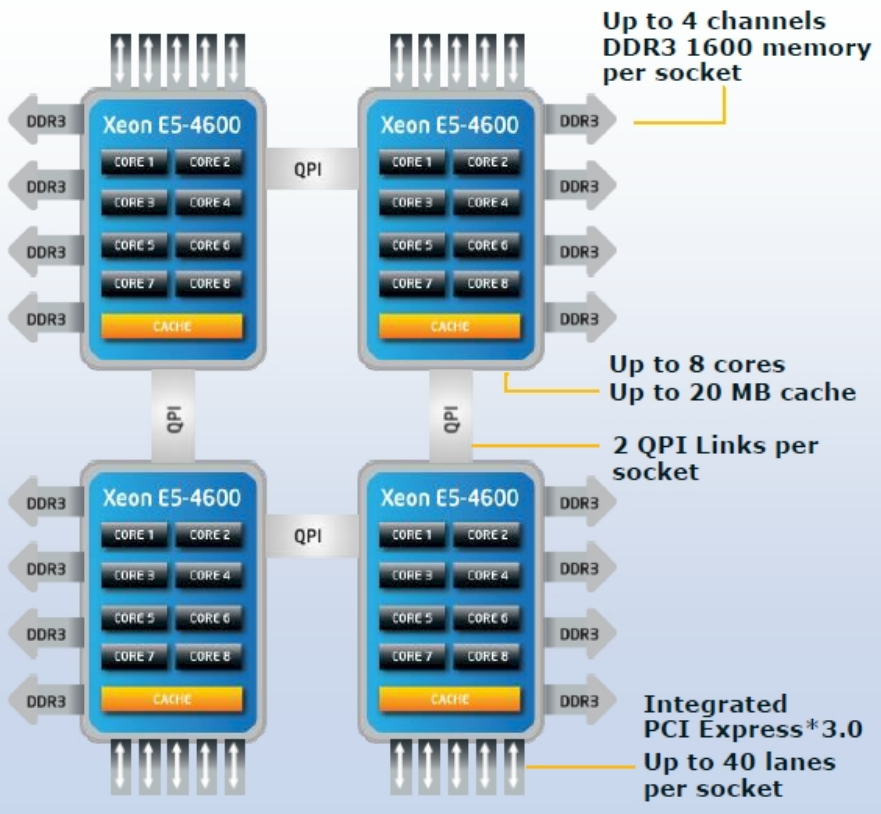
Memory Issue Solutions (Not touching the code)

- For the `affinity` among Cores and Memory:
  - Place threads in cores near the Memory cells that store their data.
  - Place memory pages in cells near the cores accessing them.
- For `locality`.
  - Place threads that access same data nearby.
- To do this we need to know:
  - Which data access each thread.
  - Where threads are at (core).
  - Where data is (relative to core).

Heterogeneity Issue

Heterogeneity Issue

- Cores are not equal.
  - May be by design.
  - Power scaling.
  - Noise.
  - Something else.
- Threads doing the same work take different times.
- Linux just balances threads among processors, if at all.



Intel Xeon E5-4600.
4 processor SMP.

Heterogeneity Issue

Heterogeneity Issue Solutions

Heterogeneity Issue Solutions (Not touching the code)

- To help deal with heterogeneity:
  - Place threads in cores adapted to their work.
  - Balance the load among cores.
  - Move threads in runtime.
- To do this we need to know:
  - Where are all threads at (all cores).
  - What is each thread doing.
  - What is the performance of each thread in each core.

Index

1 Introduction

2 Manycore Issues

3 PEBS

4 Memory Analysis

5 Performance Analysis

6 Automatic Optimisation

7 Recap and Future Work

PEBS Precise Event Bases Sampling

- Hardware Counters in Intel Processors.
- PEBS:
  - Gets one sample each X events (most hardware events supported).
  - Each sample gets the state of the whole core (Registers and Counters).
  - Saves samples in a buffer.
  - Low overhead.
  - Use supported by Linux kernel, nothing else needed.
- In addition, very precise information about memory usage:
  - When sampling load or store instructions.
  - Returns the virtual address of the data involved.
  - Returns the latency (in cycles) of the memory access.

PEBS Precise Event Bases Sampling

PEBS supports only a subset of precise, non-architected events.

MSR Mach specific regs

IA32_DS_AREA

DS Buffer Management	
Offset	64bit/8 bytes Field
0H	BTS Buffer base
8H	BTS index
10H	BTS absolute max
18H	BTS intr threshold
20H	PEBS Buffer base
28H	PEBS index
30H	PEBS absolute max
38H	PEBS intr threshold
40H	PEBS PMC0 reset
48H	PEBS PMC1 reset
50H	PEBS PMC2 reset
58H	PEBS PMC3 reset
60H	reserved

BTS Buffer
PEBS record 0
PEBS record 1
PEBS record N

PEBS Buffer
PEBS record 0
PEBS record 1
PEBS record N

PEBS record format	
Offset	64bit/8 bytes Field
0x0	R/EFLAGS
0x8	E/REIP
0x10	R/EAX
0x18	E/EBX
0x20	R/ECX
0x28	R/EDX
0x30	R/ESI
0x38	R/EDI
0x40	W/EBP
0x48	R/ESP
0x50	R8
0x58	R9
0x60	R10
0x68	R11
0x70	R12
0x78	R13
0x80	R14
0x88	R15
0x90	IA32_PERF_GLOBAL_STATUS
0x98	Data linear Address
0xA0	Data Source encoding
0xA8	Latency (core cycles)

PEBS Buffer.

PEBS Information we read

Memory Issue Solutions

To do this we need to know:

- Which data access each thread.
  - Sample load and store instructions.
  - Virtual address of the data, ADDR (OK).
- Where threads are at (core).
  - Any instruction.
  - PID, TID, CPU (OK).
- Where data is (relative to core).
  - Sample load and store instructions.
  - Latency of the load instruction, LATENCY, plus ADDR (OK).

PEBS Information we read (2)

Heterogeneity Issue Solutions

To do this we need to know:

- Where are all threads at (all cores).
  - Sample system wide, all cores (OK).
- What is each thread doing.
  - Sample Flops/sec and number of cache lines loaded.
  - We know if the code is computation bound or memory bound (OK).
- What is the performance of each thread in each core.
  - Sample Flops/sec, number of cache lines loaded, instructions for all cores.
  - We know the entire system performance, we can compare (OK).

Index

1 Introduction

2 Manycore Issues

3 PEBS

4 Memory Analysis

5 Performance Analysis

6 Automatic Optimisation

7 Recap and Future Work

Memory Analysis Description

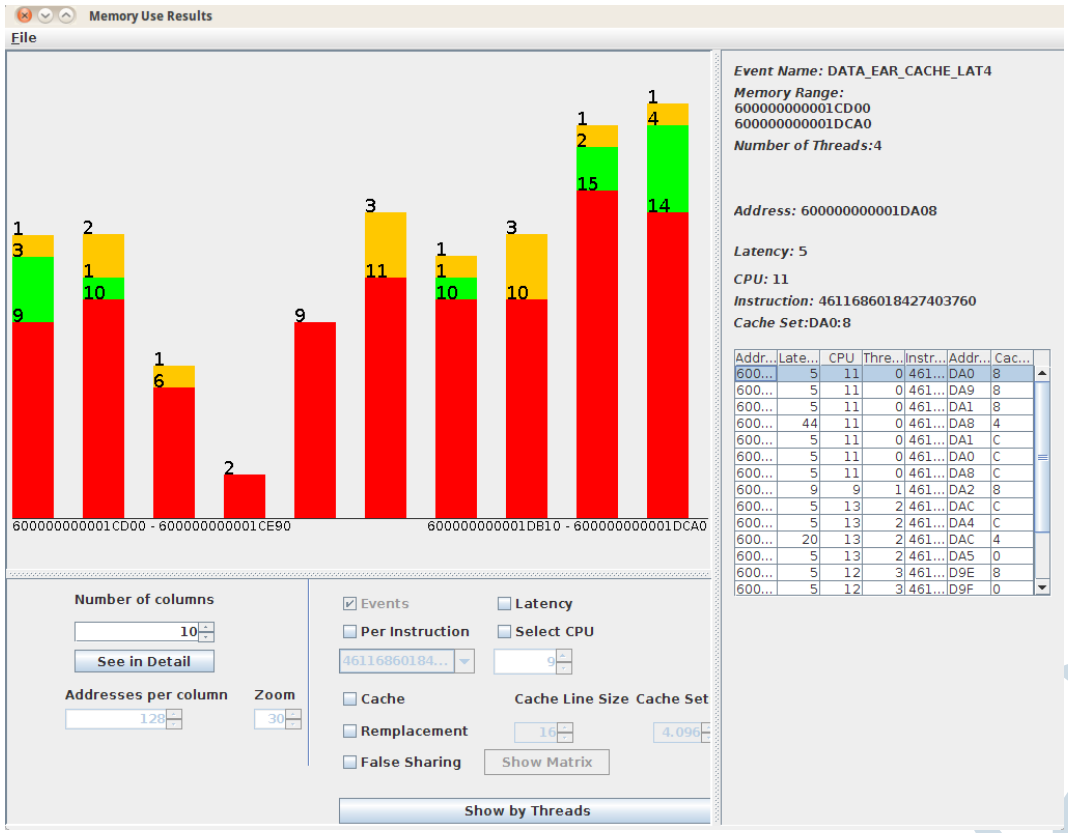
- The data is captured from PEBS.
- It allows us to see the captured accesses to memory.
  - Diverse levels of detail available.
  - Can see data per thread or whole system.
  - Can see data per instruction.
  - Can see data per core.
- It shows the latency of each access.
- It can cross-check data:
  - It can find false sharing.
  - It can simulate caches.

Memory Analysis

Visual Memory Analysis Tool

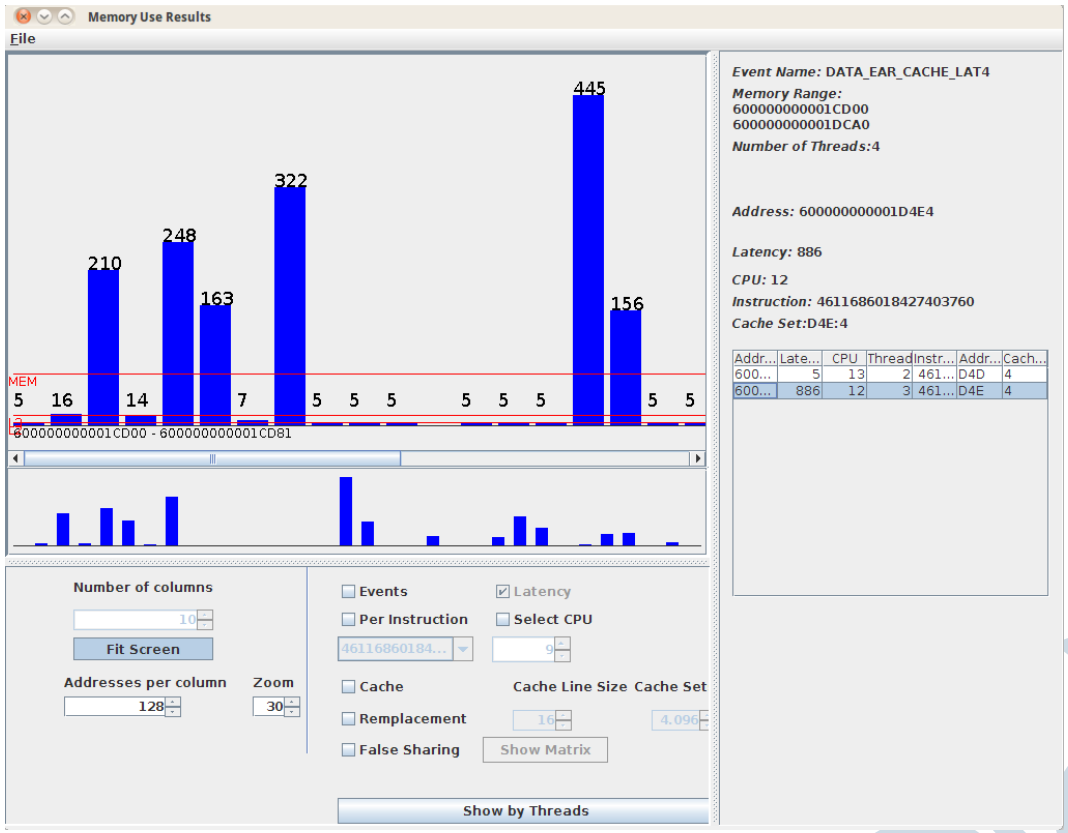
Memory Issue

Memory Analysis General Occurrence histogram



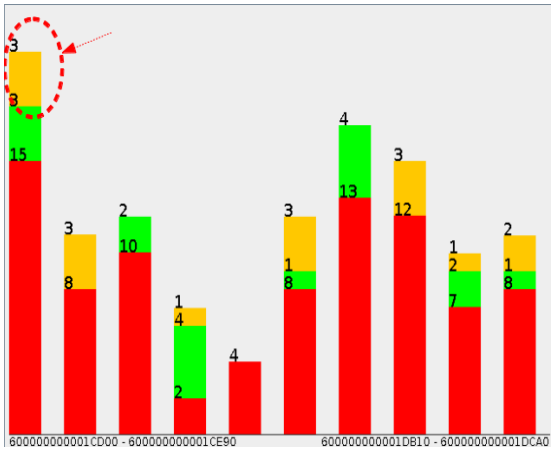
General Occurrence histogram. In the Histogram L2 misses are shown in red, L3 in green, and Main Memory in orange.

Memory Analysis
Detailed Latency histogram

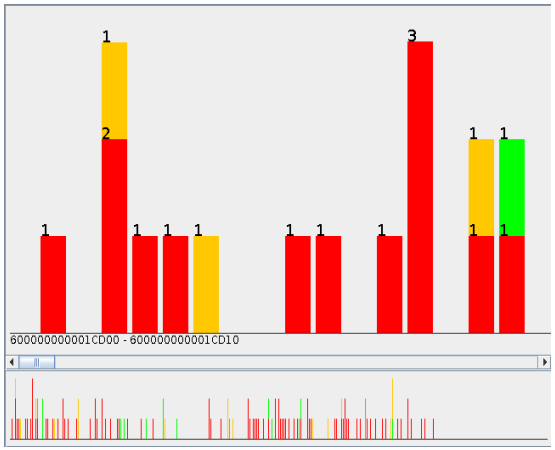


Detailed Latency histogram.

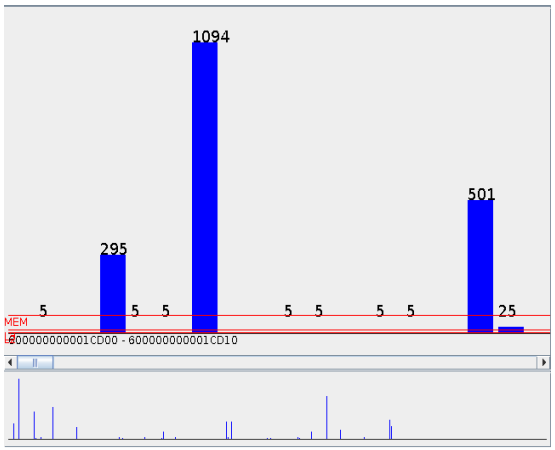
Memory Analysis Normal Use



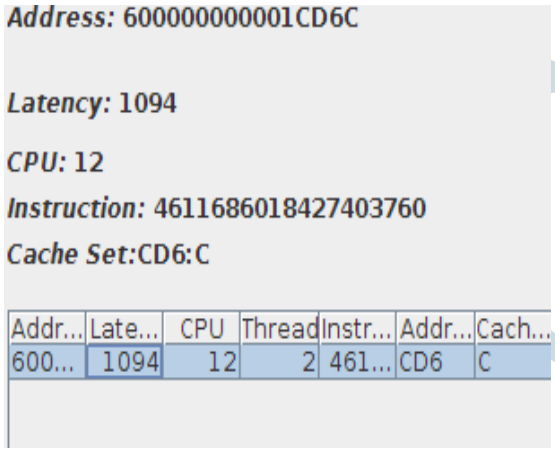
Cache misses, each bar represents 400 consecutive addresses.



Cache misses, each bar represents 16 addresses.

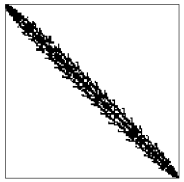


Latencies, each bar represents 16 addresses.

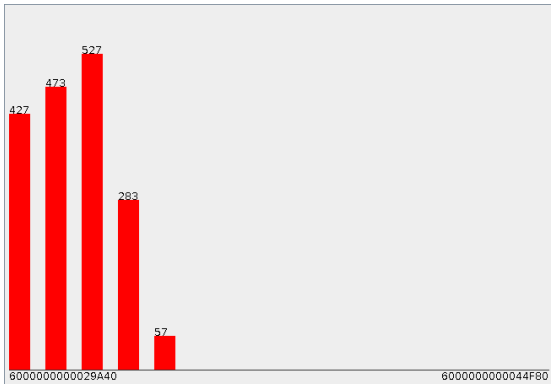


Event table.

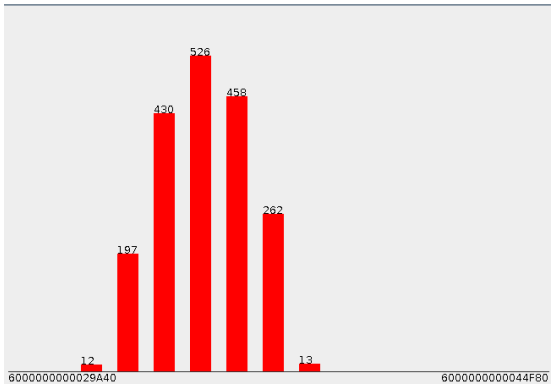
Memory Analysis Data split



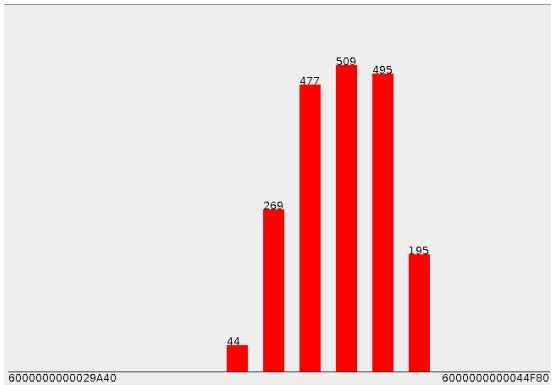
bcsstk29



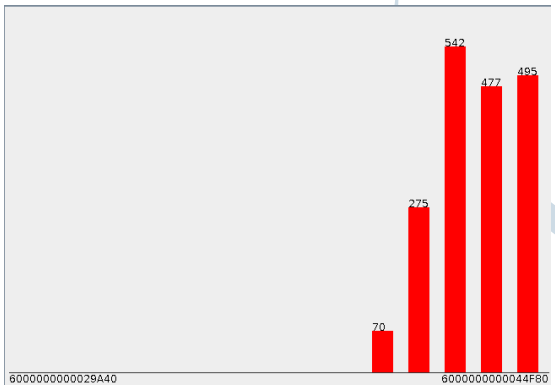
Thread 0



Thread 1

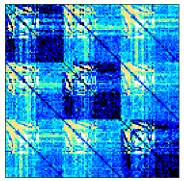


Thread 2

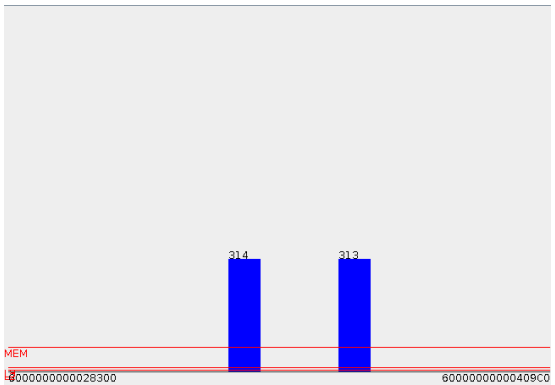


Thread 3

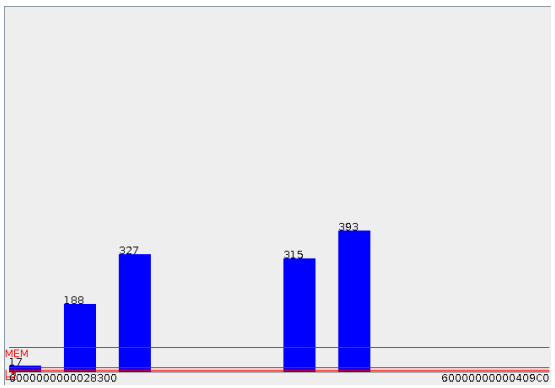
Memory Analysis
Memory Cells Latencies



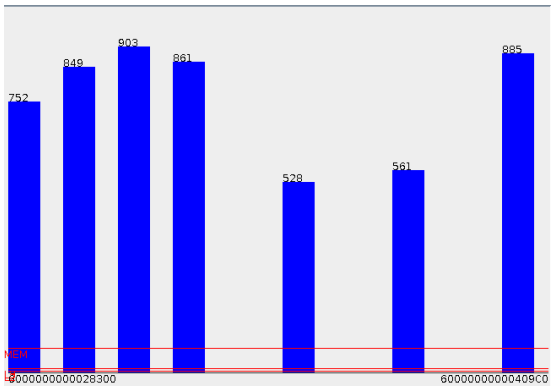
sme3Da



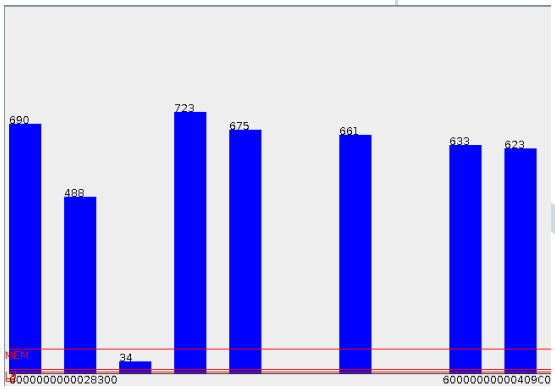
Core 0



Core 1



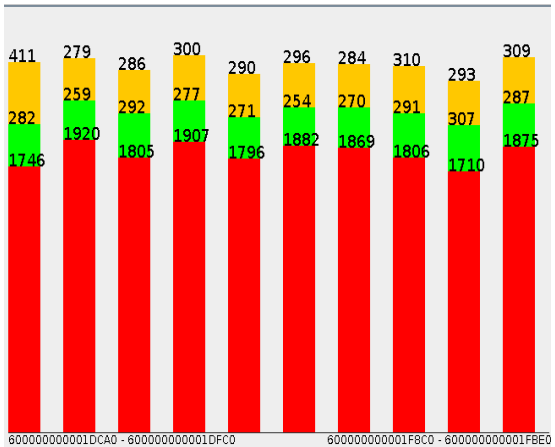
Core 8



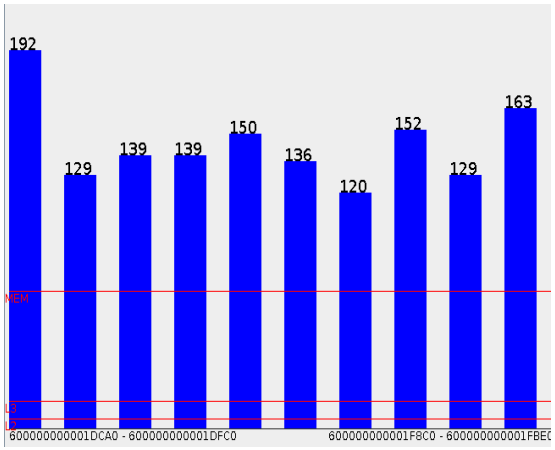
Core 9

Memory Analysis

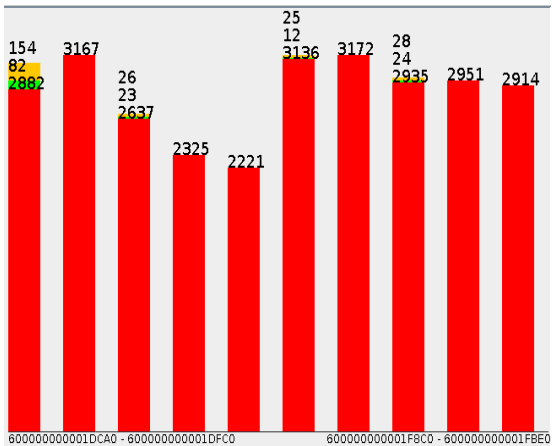
Difference between cyclic and block distribution.



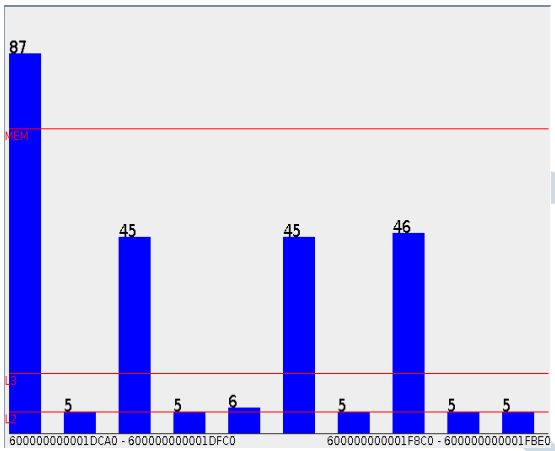
b = 4, Occurrences



b = 4, Latencies



Block distribution, Occurrences



Block distribution, Latencies

Index

1 Introduction

2 Manycore Issues

3 PEBS

4 Memory Analysis

5 Performance Analysis

6 Automatic Optimisation

7 Recap and Future Work

Performance Analysis
Performance Analysis Tool Description

- The data is captured from PEBS, visualised in R.
- It allows to see the GFlops/sec, Flops/Byte and mean MemoryLatency.
  - Can see data per thread or per core.
  - Can see the evolution of a code in time.
  - Can detect phases in execution.
- It captures the noise in the system.
- Can capture data from many concurrent processes.
  - It can find differences among cores.
  - It can find differences among threads.

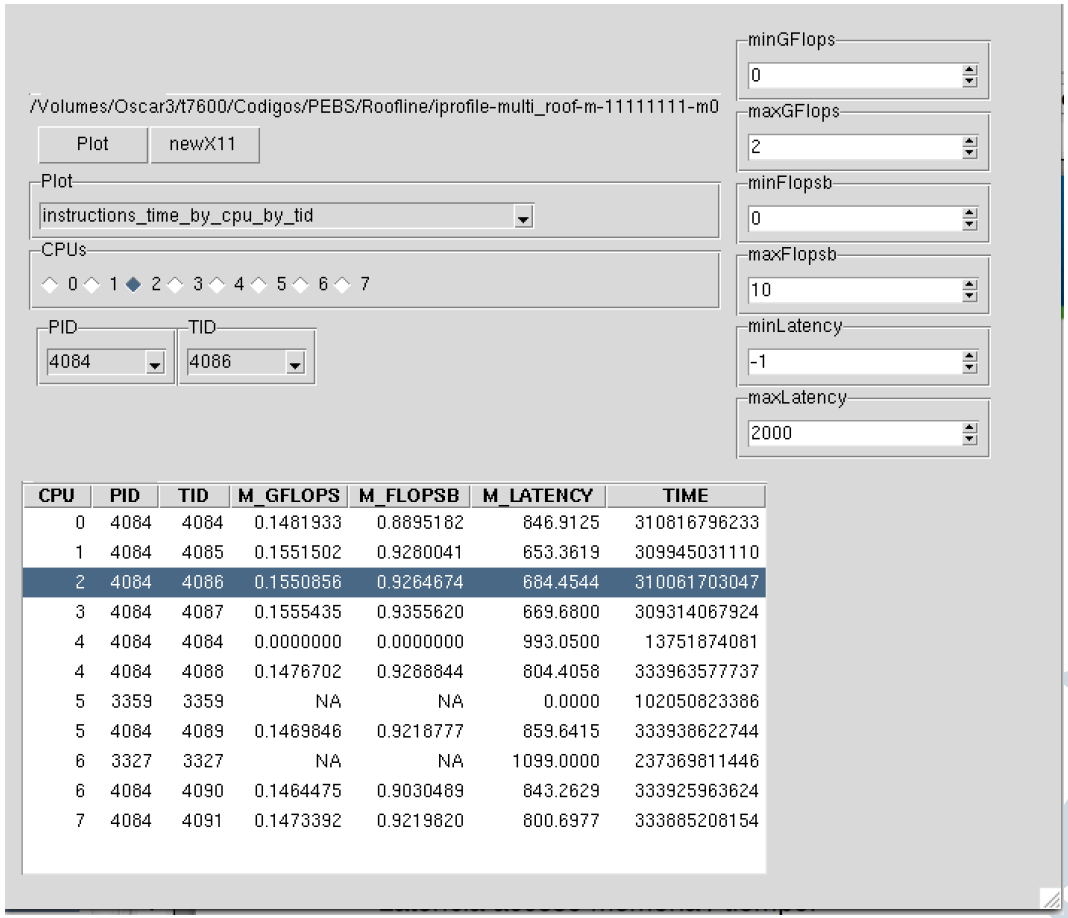
Performance Analysis

Visual Performance Analysis Tool

Memory Issue

Heterogeneity Issue

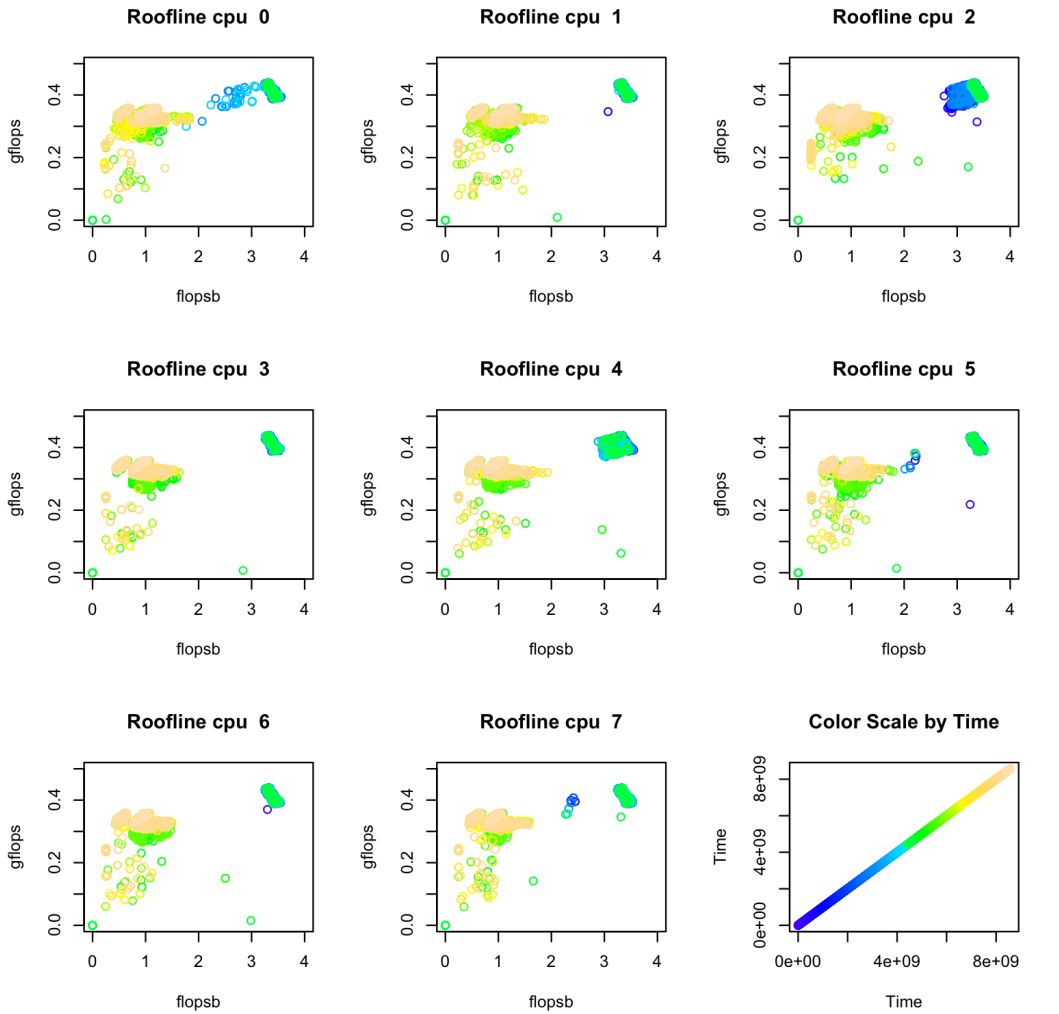
Performance Analysis The R Environment GUI



CPU	PID	TID	M_GFLOPS	M_FLOPSB	M_LATENCY	TIME
0	4084	4084	0.1481933	0.8895182	846.9125	310816796233
1	4084	4085	0.1551502	0.9280041	653.3619	309945031110
2	4084	4086	0.1550856	0.9264674	684.4544	310061703047
3	4084	4087	0.1555435	0.9355620	669.6800	309314067924
4	4084	4084	0.0000000	0.0000000	993.0500	13751874081
4	4084	4088	0.1476702	0.9288844	804.4058	333963577737
5	3359	3359	NA	NA	0.0000	102050823386
5	4084	4089	0.1469846	0.9218777	859.6415	333938622744
6	3327	3327	NA	NA	1099.0000	237369811446
6	4084	4090	0.1464475	0.9030489	843.2629	333925963624
7	4084	4091	0.1473392	0.9219820	800.6977	333885208154

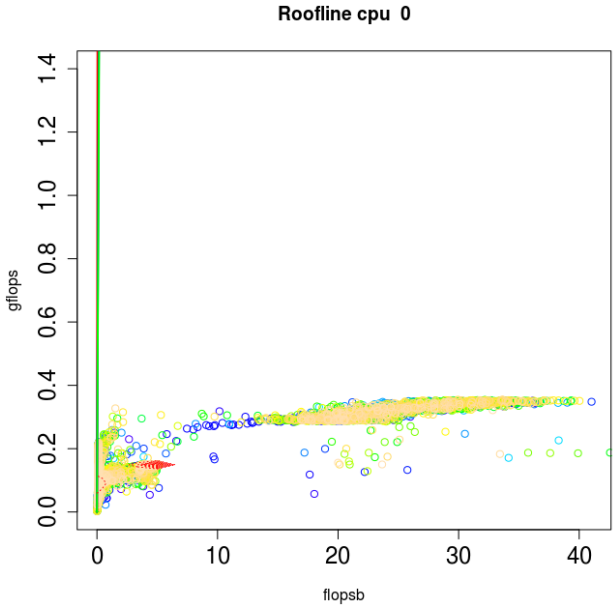
The R Environment GUI.

Performance Analysis
System with 8 Cores

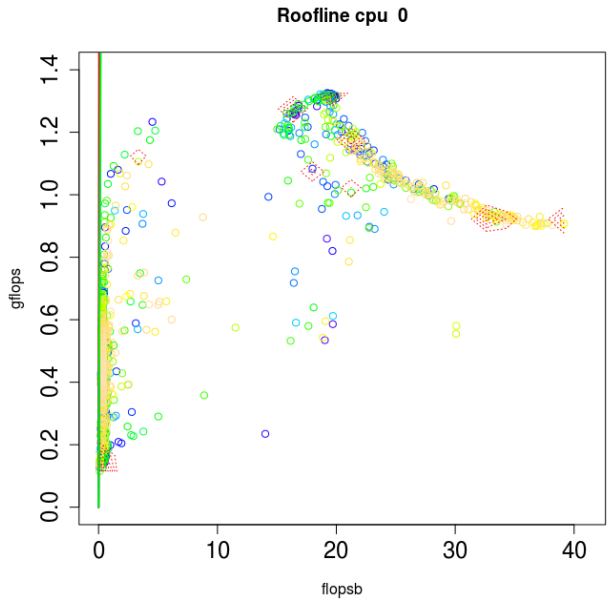


Dynamic Roofline for system with 8 cores.

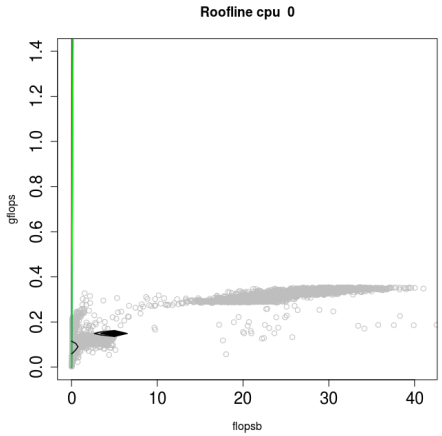
Performance Analysis Dynamic Roofline



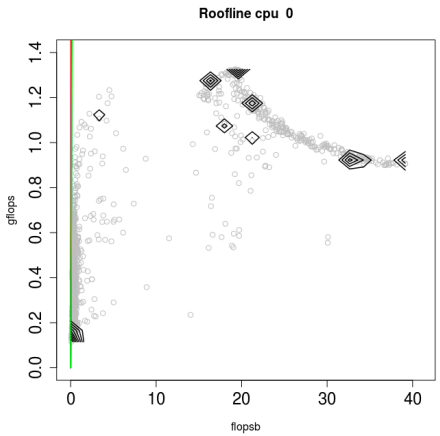
Roofline ua.A No Optimisations



Roofline ua.A

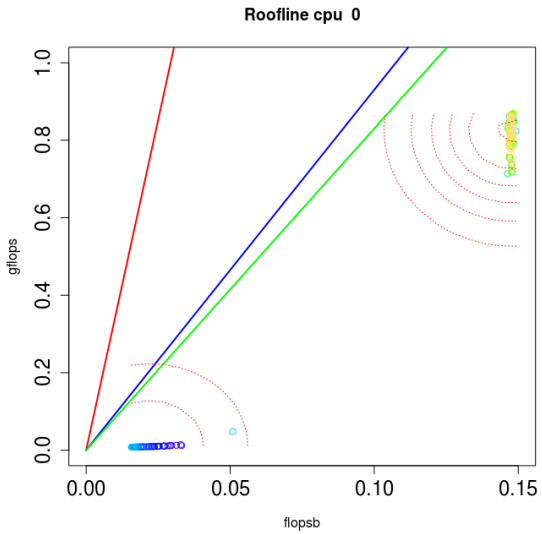


Density. ua.A No

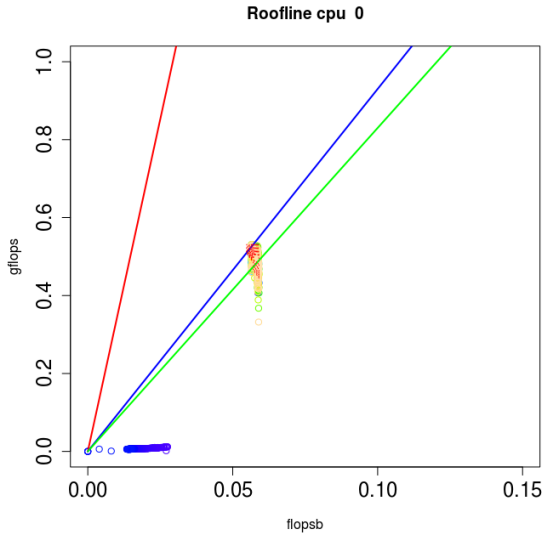


Density. ua.A

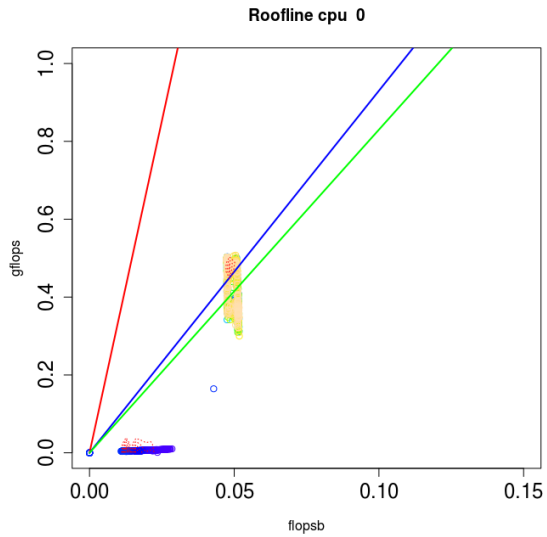
Performance Analysis
Dynamic Roofline



Roofline cg.A

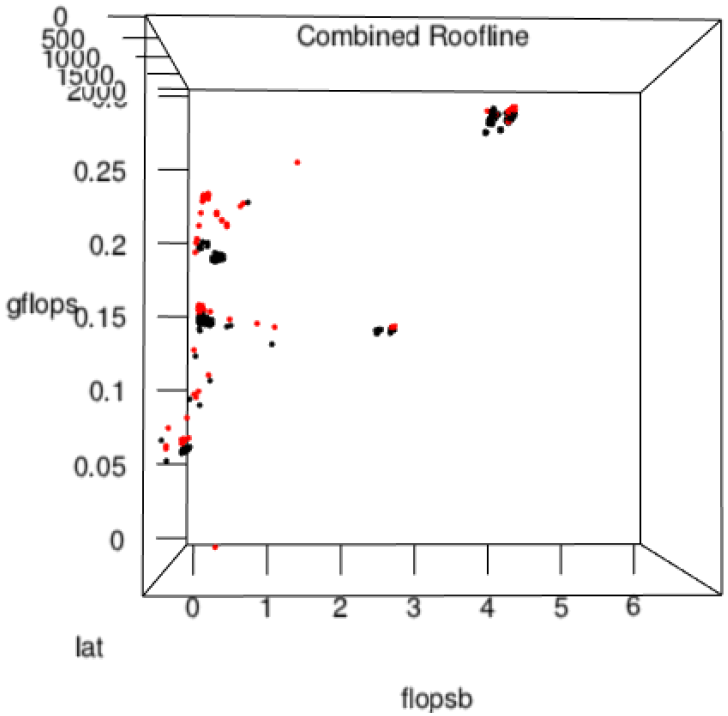


Roofline cg.B

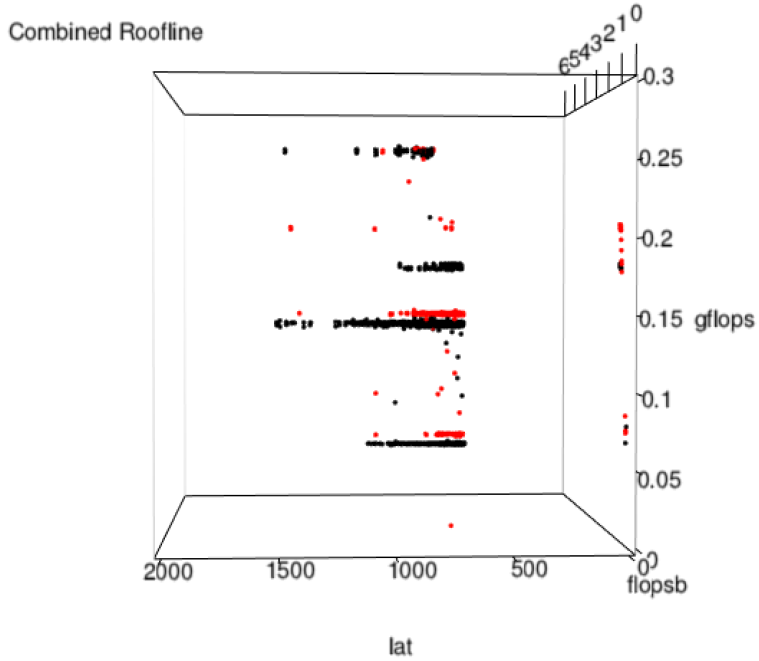


Roofline cg.C

Performance Analysis
Roofline 3D



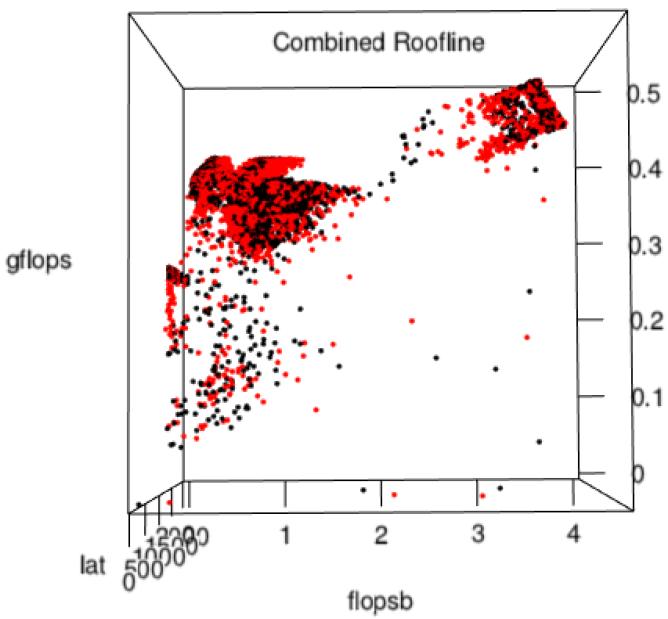
Roofline 3D, GFlops/FlopB

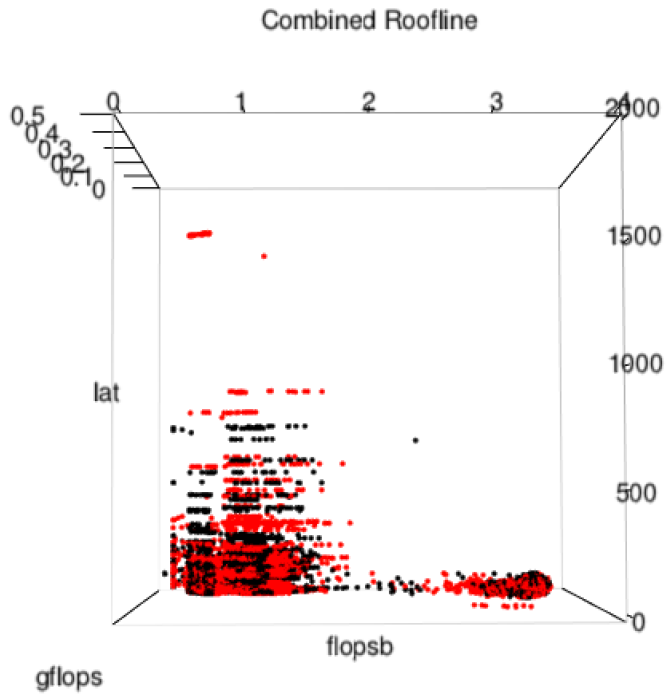


Roofline 3D, GFlops/Latency(cycles)

Roofline 3D. GFlops/FlopsB/Latency. Processor 0 is shown in red, Processor 1 in black

Performance Analysis
Roofline 3D





Roofline 3D, GFlops/FlopB

Roofline 3D, FlopB/Latency(cycles)

Roofline 3D of the ep.A and ft.A benchmarks. GFlops/FlopsB/Latency. Processor 0 is shown in red, Processor 1 in black

Index

1 Introduction

2 Manycore Issues

3 PEBS

4 Memory Analysis

5 Performance Analysis

6 Automatic Optimisation

7 Recap and Future Work

Automatic Optimisation
Runtime Migration Tool

Automatic Performance Optimisation

Runtime Migration Tool

- Memory Issue Solutions
- Heterogeneity Issue Solutions

Automatic Optimisation What needs to be done

Memory Issue Solutions

- For the `affinity` among Cores and Memory:
  - Place threads in cores near the Memory cells that store their data.
  - Place memory pages in cells near the cores accessing them.
- For `locality`.
  - Place threads that access same data nearby.

Heterogeneity Issue Solutions

- To help deal with heterogeneity:
  - Place threads in cores adapted to their work.
  - Balance the load among cores.
  - Move threads in runtime.

Automatic Optimisation What needs to be done

Memory Issue Solutions

- For the `affinity` among Cores and Memory:
  - Place threads in cores near the Memory cells that store their data.
  - Place memory pages in cells near the cores accessing them.
    - Page Migration, Juan A. Lorenzo PhD dissertation.
- For `locality`.
  - Place threads that access same data nearby.

Heterogeneity Issue Solutions

- To help deal with heterogeneity:
  - Place threads in cores adapted to their work.
  - Balance the load among cores.
  - Move threads in runtime.

Automatic Optimisation
Some Results

- A code with many memory accesses.
- All threads do the same work.
- Two processors (0 and 1), each connected to a 8GB memory cell.
- All data is store near the procesor 0.
- Threads in processor 1 take longer.
- We measure memory access latency for each thread.
- Simple Strategy: The worst thread is moved.
  - To an empty core if possible.
  - Interchange with the best.
- Times:
  - Without migration: 4:31 min
  - With thread migration: 4:16 min
- Works somewhat.

Automatic Optimisation
Current Work

- We can move threads between cores.
- We can move memory pages between cells.
- We can capture data about memory usage, per core and thread.
- We can capture data about performance, per core and thread.
- The problem now is to combine everything.
  - We need strategies and algorithms.
- Then we will have to test it.
  - We need access to more machines and systems.
  - Software support is not ready.

Index

1 Introduction

2 Manycore Issues

3 PEBS

4 Memory Analysis

5 Performance Analysis

6 Automatic Optimisation

7 Recap and Future Work

Recap

Memory Analysis

Visual Memory Analysis Tool

- Memory Issue

Performance Analysis

Visual Performance Analysis Tool

- Memory Issue
- Heterogeneity Issue

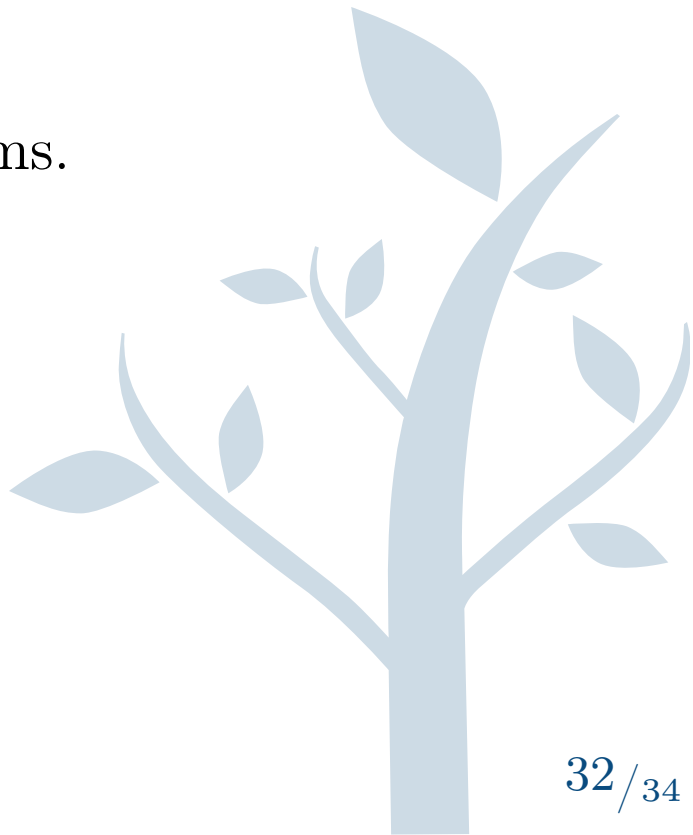
Automatic Optimisation

Runtime Migration Tool

- Memory Issue Solutions
- Heterogeneity Issue Solutions

Future Work Future Work

- We need algorithms.
- We need to test them.
- We need to validate them in more systems.



Publications

- Oscar G. Lorenzo, Juan A. Lorenzo, Dora B. Heras, Juan C. Pichel, Francisco F. Rivera, "Herramientas para la Monitorización de los Accesos a Memoria de Códigos Paralelos Mediante Contadores Hardware", Actas XXII Jornadas de Paralelismo (JP2011), La Laguna 2011, pages 651–656.
- Oscar G. Lorenzo, Juan A. Lorenzo, José C. Cabaleiro, Dora B. Heras, Marcos Suarez, Juan C. Pichel “A Study of Memory Access Patterns in Irregular Parallel Codes Using Hardware Counter-Based Tools", Int. Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA), Las Vegas (USA), 2011, pages 920–923
- Oscar G. Lorenzo; Tomás F. Pena; José C. Cabaleiro; Juan C. Pichel; Juan A. Lorenzo; Francisco F. Rivera, "Hardware Counter Based Analysis of Memory Accesses in SMPs",ISPA2012 Madrid July 2012

Publications Publications

- Oscar G. Lorenzo; Tomás F. Pena; José C. Cabaleiro; Juan C. Pichel; Francisco F. Rivera, "DyRM: A Dynamic Roofline Model Based on Runtime Information", CMMSE 2013 (Accepted)
- Oscar G. Lorenzo; Tomás F. Pena; José C. Cabaleiro; Juan C. Pichel; Juan A. Lorenzo; Francisco F. Rivera, "A Hardware Counters Based Toolkit for the Analysis of Memory Accesses in SMPs",Concurrency and Computation: Practice and Experience (Under Review)

Thank you!!