VI-HPS Santiago 2015

PAPI: Performance API

Andrés Ávila
Centro de Modelación y Computación Científica
Universidad de La Frontera
andres.avila@ufrontera.cl

October 27th, 2015

Motivation

Motivation

PERFORMANCE ANALYSIS/ENGINEERING

- **Optimization of codes: continuous process**
- **Depend mainly on Algorithms**
- **Detailed data for analysis:**
  - **Profiling: summary of data ("Statistics")**
  - **Tracing: time and space details**

Motivation

COSTS

top500 june 2015: 30% cluster with less than 63% peak performance

Worst cases: IBM manufacturer+Sandy Bridge??

Best cases: IBM and SGI manufactures +Ivy Bridge

Best energy efficiency: IBM+PowerPC

Motivation

PERFORMANCE TOOLS USING PAPI

- **TAU**
- **Vampir**
- **Periscope**
- **Scalasca**
- **HPCToolkit**
- **PerfSuite**
- **ompP**
- **Blue Gene Perf. tool**

PAPI Goals

Hardware Performance Counters

- Small set of registers
- Counting ***events/signals***

PAPI GOALS

- *To ease accessing HWPC*
- *To define a standard/ portability*
- *To aid performance analysis, modelling and tuning*

Motivation

SOME COMMENTS

- **Lots of detailed information**
- **Data mining for analysis**
- **Continuously supported**
- **Tuning is recommended**

Motivation

PERFORMANCE ANALYSIS

- Use of OS timers
  - Granularity:=0
  - Size problem: *as much as I can wait*
- Use of language libraries
  - Time between two lines
  - Hand made instrumentation
  - Algorithm hint: *$O(n^2)$ in ops*
- Other SW tools
  - Pipelining
  - Optimizers
  - Different OS commands

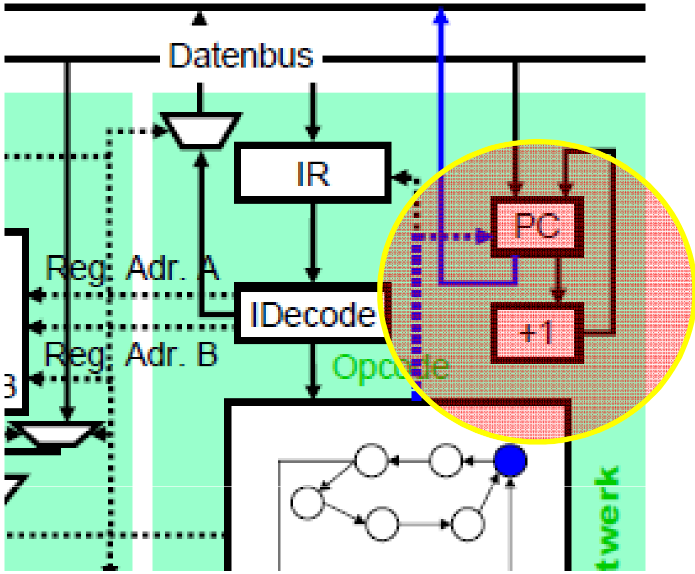
```
>time ./a.out
real    8m43.858s
user    8m26.445s
sys     0m0.616s
```

```
#include "time.h"
int main()
{
    time_t start, end;
……
 time(&end);
printf("Time is %.5f
\n",difftime(end,start));
time(&start);
….
time(&end);
printf("Time is %.5f
\n",difftime(end,start));
time(&start);
  …..
time(&end);
}
```

Motivation

PERFORMANCE ANALYSIS

- **Processor architecture**
  - Very different choices
  - PC and Buses
  - Memory hierarchy
  - *Memory wall*
  - *Energy wall*
- **Hardware performance events**
  - For collecting processor information
  - For monitoring threads
  - Appeared in 1995(?)
  - Intel Pentium Pro: two 40-bit counters for secondary cache misses
  - Ivy Bridge: over 50 counters
- **Access through low level programming**



http://oprofile.sourceforge.net/docs/intel-ivybridge-events.php

PAPI Architecture

Architecture

- **Based on perfctr from Linux**
- **List of counters**
  - **Cycles**
  - **Integer and floating points**
  - **Memory hierarchy (L1, L2, TBL)**
  - **Branch prediction**
  - **Load and store**
- **Most used counters: memory misses**

New Architectures

- **GPU CUDA**
  - **CUDA performance tool instrument: CUPTI**
  - **Three levels of counters: processor, GPU and cluster**
  - **Some events:**
    - **Local and global access**
    - **Warp counters**

TABLE I
A PORTION OF CUDA EVENTS AVAILABLE ON GEFORCE GTX 480 AND TESLA C870 DEVICES.

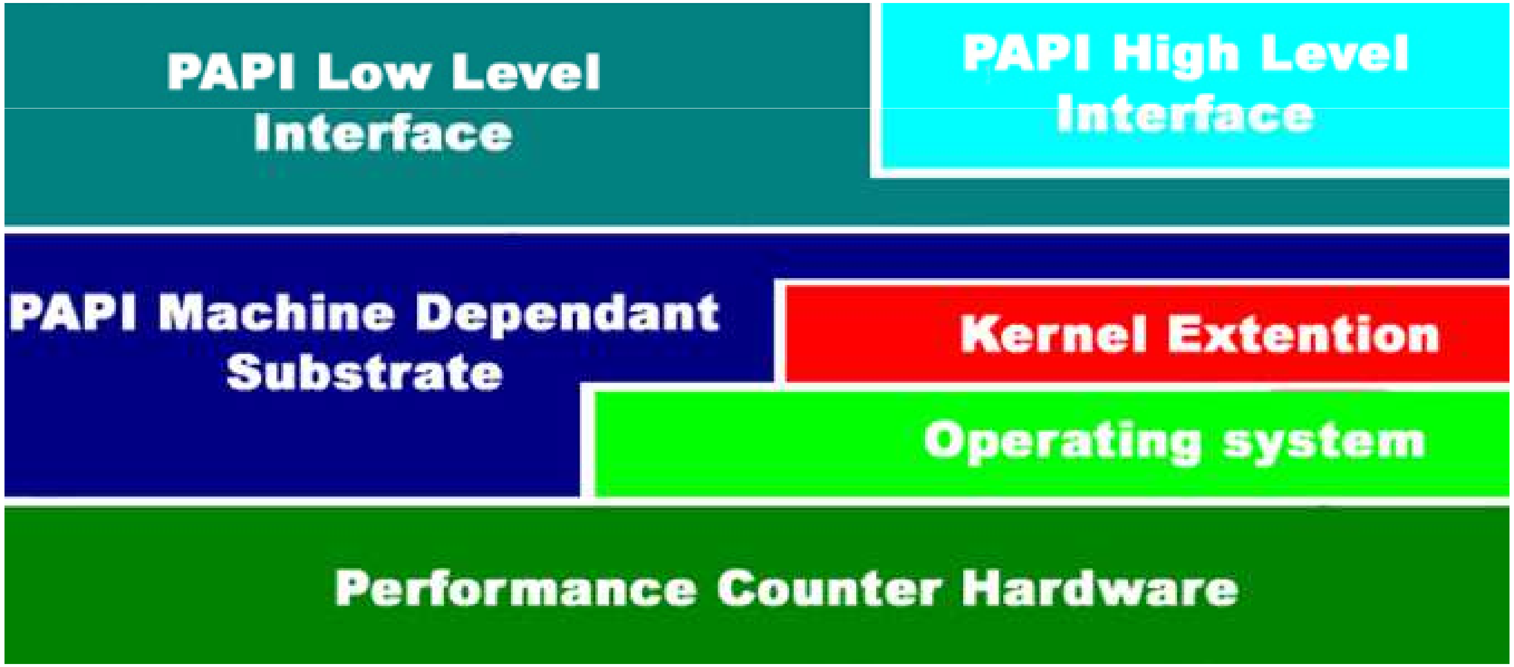
Event Code	Symbol	Long Description
0x44000000	CUDA.GeForce_GTX_480.gpc0.local_load	# executed local load instructions per warp on a multiprocessor
0x44000001	CUDA.GeForce_GTX_480.gpc0.local_store	# executed local store instructions per warp on a multiprocessor
0x44000002	CUDA.GeForce_GTX_480.gpc0.gld_request	# executed global load instructions per warp on a multiprocessor
0x44000003	CUDA.GeForce_GTX_480.gpc0.gst_request	# executed global store instructions per warp on a multiprocessor
0x44000004	CUDA.GeForce_GTX_480.gpc0.shared_load	# executed shared load instructions per warp on a multiprocessor
0x44000005	CUDA.GeForce_GTX_480.gpc0.shared_store	# executed shared store instructions per warp on a multiprocessor
0x44000006	CUDA.GeForce_GTX_480.gpc0.branch	# branches taken by threads executing a kernel
0x44000007	CUDA.GeForce_GTX_480.gpc0.divergent_branch	# divergent branches within a warp
0x4400000b	CUDA.GeForce_GTX_480.gpc0.active_cycles	# cycles a multiprocessor has at least one active warp
0x4400000c	CUDA.GeForce_GTX_480.gpc0.sm_cta_launched	# thread blocks launched on a multiprocessor
0x4400000d	CUDA.GeForce_GTX_480.gpc0.l1_local_load_hit	# local load hits in L1 cache
0x4400000e	CUDA.GeForce_GTX_480.gpc0.l1_local_load_miss	# local load misses in L1 cache
0x44000011	CUDA.GeForce_GTX_480.gpc0.l1_global_load_hit	# global load hits in L1 cache

New Architectures

- **Intel Xeon Phi**
  - **PAPI support**
  - **Few counters per core**

Architecture

- **High level routines**: specific simple counters
- **Low level routines**: several counters as an ***EventSet***
- **Reference implementation**: ***substrate layer***



Standard events

- Predefined with HPC community (1999)
- Useful for tuning
- Include:
  - Memory hierarchy
  - Cache coherence protocol events
  - Cycle and instruction counts
  - Functional units
  - Pipeline status
- Focus on improving memory utilization
- Summarized in a library for C and Fortran

Standard events

EXAMPLES MEMORY

PAPI_L1_DCM	Level 1 data cache misses
PAPI_L1_ICM	Level 1 instruction cache misses
PAPI_L2_DCM	Level 2 data cache misses
PAPI_L2_ICM	Level 2 instruction cache misses
PAPI_L3_DCM	Level 3 data cache misses
PAPI_L3_ICM	Level 3 instruction cache misses
PAPI_L1_TCM	Level 1 total cache misses
PAPI_L2_TCM	Level 2 total cache misses
PAPI_L3_TCM	Level 3 total cache misses
PAPI_TLB_DM	Data translation lookaside buffer misses
PAPI_TLB_IM	Instruction translation lookaside buffer misses
PAPI_TLB_TL	Total translation lookaside buffer misses
PAPI_L1_LDM	Level 1 load misses
PAPI_L1_STM	Level 1 store misses
PAPI_L2_LDM	Level 2 load misses
PAPI_L2_STM	Level 2 store misses

Standard events

Examples cache coherence

PAPI_CA_SNP	Snoops
PAPI_CA_SHR	Request for access to shared cache line
PAPI_CA_CLN	Request for access to clean cache line
PAPI_CA_INV	Cache line invalidation
PAPI_CA_ITV	Cache line intervention
PAPI_TLB_SD	Translation lookaside buffer shootdowns

Examples cycles and instructions

PAPI_TOT_CYC	Total cycles
PAPI_TOT_IIS	Total instructions issued
PAPI_TOT_INS	Total instruction completed
PAPI_INT_INS	Integer instructions completed
PAPI_FP_INS	Floating point instructions completed
PAPI_LD_INS	Load instructions completed
PAPI_SR_INS	Store instructions completed
PAPI_LST_INS	Total load/store instructions completed
PAPI_FMA_INS	FMA instructions completed
PAPI_VEC_INS	Vector/SIMD instructions completed
PAPI_BR_UCN	Unconditional branch instructions completed
PAPI_BR_CN	Conditional branch instructions completed
PAPI_BR_TKN	Conditional branch instructions taken
PAPI_BR_NTK	Conditional branch instructions not taken
PAPI_BR_MSP	Conditional branch instructions mispredicted

....

How to use it

Memory hierarchy

- Load/store: memory addressing process
- First, check L1 cache.
  - If it is present, L1 cache hit and out
  - Else, L1 cache miss and goto L2 cache
- Check L2 cache
- Check L3 cache
- Check TLB

How to use it

Old latencies for MIPS 10000

CPU register	0 cycles
L1 cache hit	2 or 3 cycles
L1 cache miss satisfied by L2 cache hit	8 to 10 cycles
L2 cache miss satisfied from main memory, no TLB miss	75 to 250 cycles
TLB miss requiring only reload of TLB	2000 cycles
TLB miss requiring virtual page to be loaded from backing store	Hundreds of millions of cycles

L1 data cache hit rate

=1- PAPI_L1_DCM/(PAPI_LD_INS+PAPI_SR_INS)

DCM: data cache miss

Hint 1: Over 0.95, good cache performance

Hint 2: large PAPI_TLB_DM data spread over many pages

Other hints

- ***Hint 3:*** PAPI_TOT_INS/PAPI_TOT_CYC must be low or stalling
- ***Hint 4:*** PAPI_LST_INS/PAPI_TOT_CYC density of memory in the program
- ***Hint 5:*** PAPI_FP_INS/PAPI_TOT_CYC density of floating point operations
- ***Hint 6:*** high PAPI_LD_INS/PAPI_L1_DCM is a dense numerical program

Hardware counters on Linux

- **IA-32 P6 family**
  - Two 40-bit counters
  - Either count events or measure time
  - More events and greater control
  - Based on **perfctr** linux command
- **AMD Athlon**
  - Four 48-bit counters
  - Either count events or measure time
  - Not guaranteed to be fully accurate
- **IA-64 family**
  - Four counters and four overflow status registers
  - Events grouped by: basics, execution, memory, cycle counting, branch events, hierarchy, system
  - Focus on L3 properties

Other characteristics

- Record data either by counting or sampling modes
- Graphical interfaces
- MPI
- Scalability
- Problems
  - How to choose events?
  - How to analyze data?

Examples

First example: Itanium 2 registers

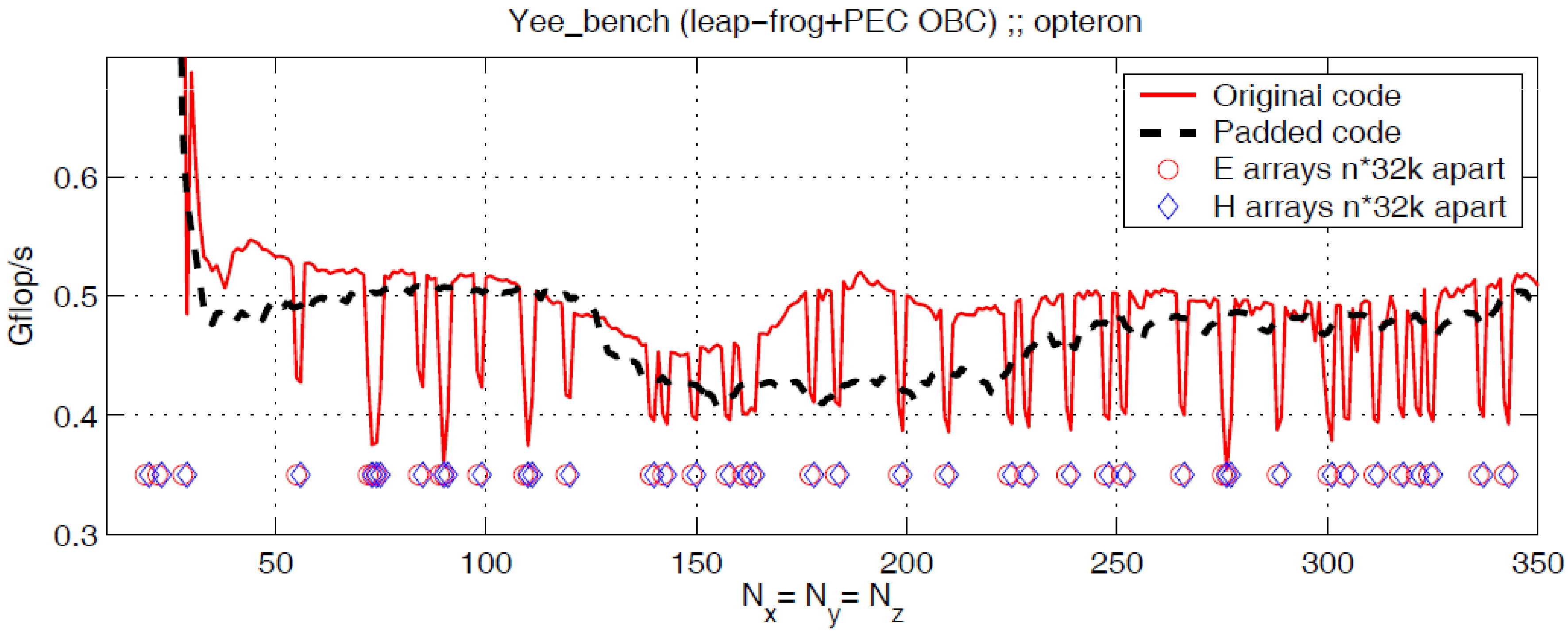
- Hardware: performance monitor control registers
  - **PMC4-PMC7: counting occurences**
  - PMC10-11: instruction/data event addresses
  - PMC12: branch trace buffer
  - Focus on counting occurrences
- 300 different counters
  - CPU
  - Stalls
  - TLB
  - Cache hierarchy
  - Memory subsystem
- Several runs to get relevant performance aspects: ***drill down*** approach

Second example: AMD Opteron

- Yee_Bench Fortran 90 benchmark
- Finite difference time domain algorithm for Maxwell equations
- Performance of serial code depending on size
- Scaling up of algorithm means memory problems
- HW:
  - 2.0 GHz clock, 8Gb memory DDR 333,
  - L1 64 Kb and 64-byte lenght two-way associativity
  - L2 1Mb, associativity 16
- SW: compiler pgf90 5.2-4

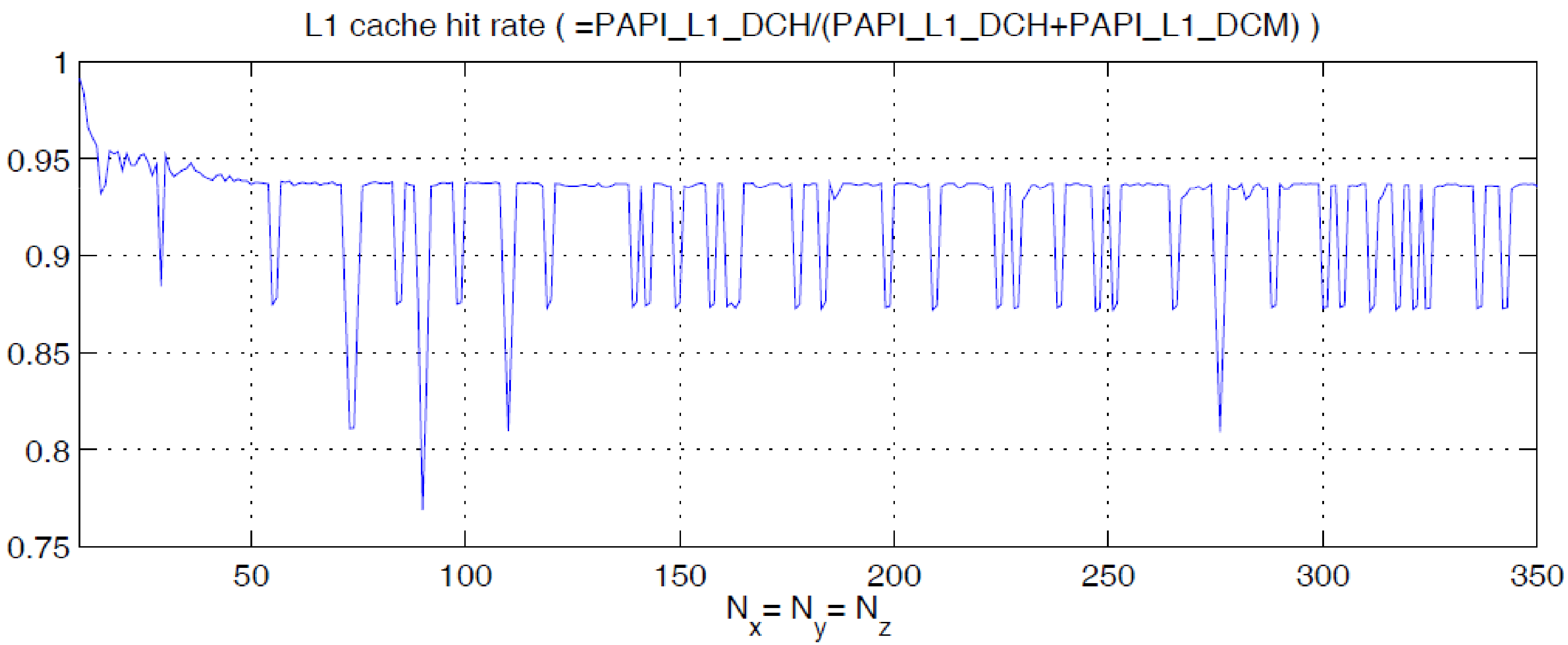
Second example: AMD Opteron

Results: poor performance forN or N+1 power of two and no padding, memory problems



Second example: AMD Opteron

Results: L1 cache hit rate: drops by L1 misses



Second example: AMD Opteron

Algorithm

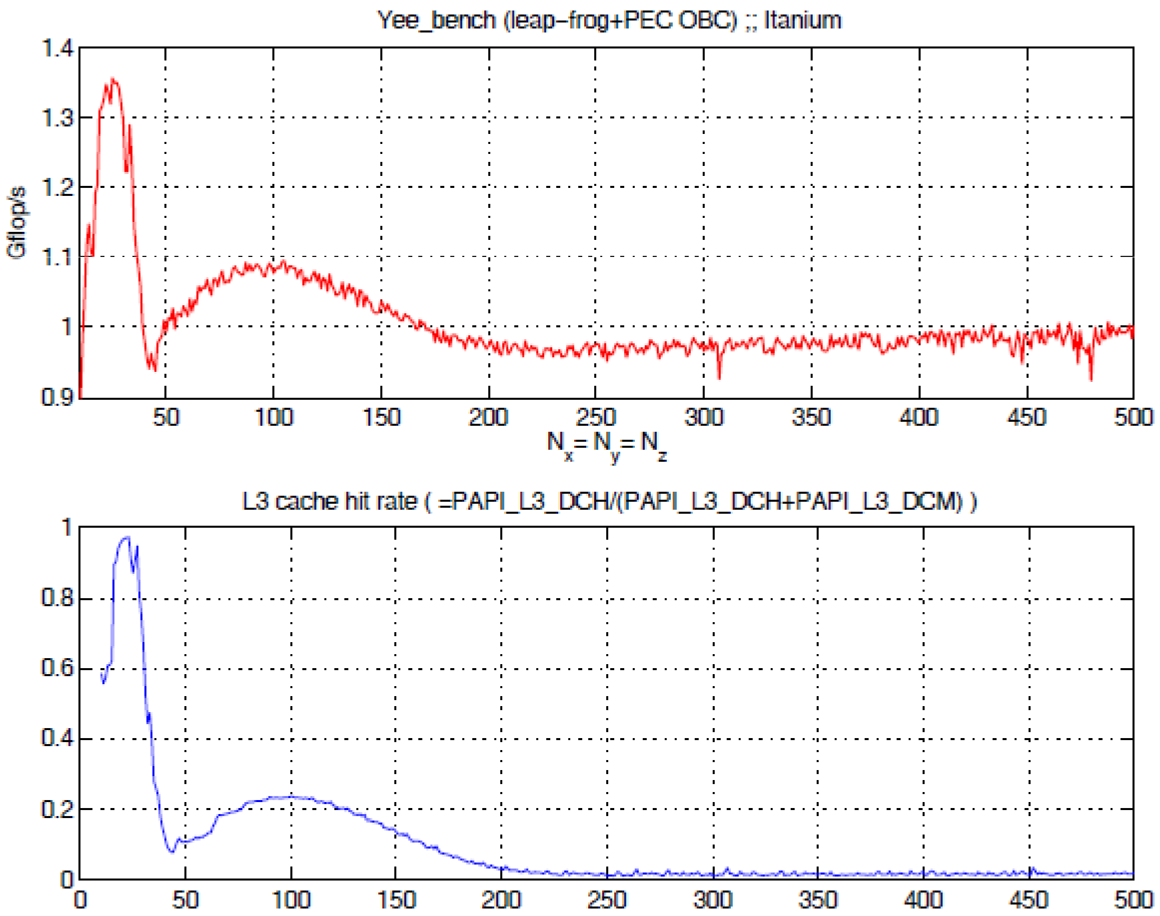
```
do k=1,nz ; do j=1,ny ; do i=1,nx
   Hx(i,j,k) = Hx(i,j,k) + (   (Ey(i,j,k+1)-Ey(i,j  ,k))*Cbdz +    &
                               (Ez(i,j,k  )-Ez(i,j+1,k))*Cbdy   )
   Hy(i,j,k) = Hy(i,j,k) + (   (Ez(i+1,j,k)-Ez(i,j,k  ))*Cbdx +    &
                               (Ex(i  ,j,k)-Ex(i,j,k+1))*Cbdz   )
   Hz(i,j,k) = Hz(i,j,k) + (   (Ex(i,j+1,k)-Ex(i  ,j,k))*Cbdy +    &
                               (Ey(i,j  ,k)-Ey(i+1,j,k))*Cbdx   )
end do ; end do ; end do
```

- Four loads previously used
- L1 cache line eight 64-bit FP values
- One miss for each six new values
- L1 cache hit rate= 94.2%

Second example: AMD Opteron

On Intel Itanium 2: L3 cache hit rate

Data travelled in one iteration: $64*100^2$=0.61Mb

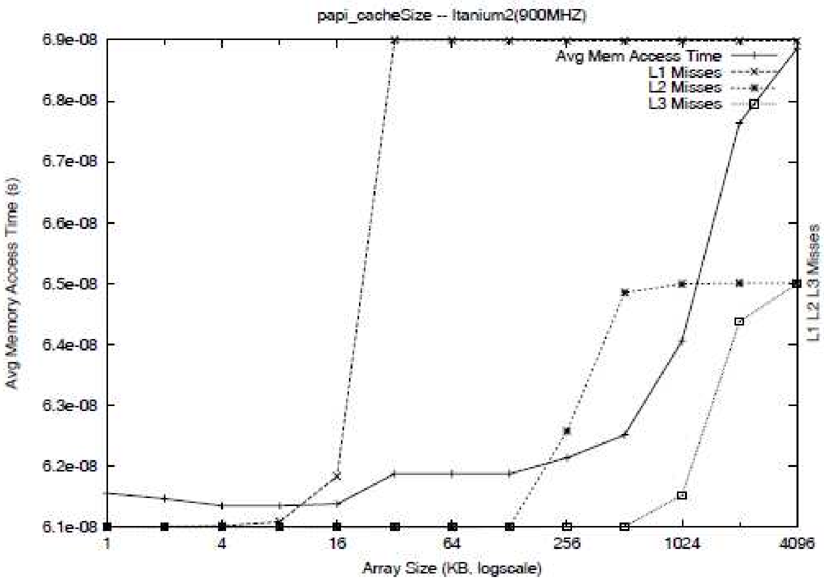


Third example: cache and TLB

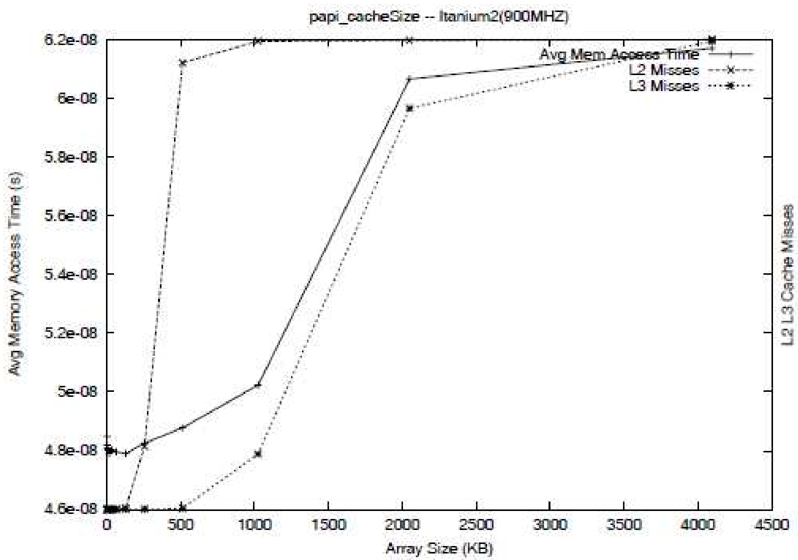
Events

PAPI_L1_DCM	Level 1 data cache misses
PAPI_L2_DCM	Level 2 data cache misses
PAPI_L3_DCM	Level 3 data cache misses
PAPI_TLB_DM	Data TLB misses

Two benchmarks: papi_cacheSize and papi_cacheBlock



Integer data



Real data

New Ivy Bridge

3 Fixed Function Counters

- Unhalted Core Cycles
- Unhalted Reference Cycles
- Instructions Retired

8 Programmable Counters

- unless you're Hyperthreading (4 per thread)
- or using an NMI watchdog timer (3 per thread)

4 Uncore Counters

- chip wide; not core specific
- unified cache measurement (L3)
- shared resources

http://www.netlib.org/utk/people/JackDongarra/WEB-PAGES/SPRING-2013/Lect08-counters.pdf

Using PAPI High Level Counters

High level counters

- ***high-level.c*** are the available counters
- **Initialize:** *PAPI_start_counters(*events, array_length)*
  - **events -- an array of codes for events such as PAPI_INT_INS or a native event code.*

    int Events[NUM_EVENTS] = {PAPI_TOT_INS};

  - *array_length -- the number of items in the events array.*
- **Execution:** floating point or total instruction rates

```
PAPI_flips(*real_time, *proc_time, *flpins, *mflips)
PAPI_flops(*real_time, *proc_time, *flpins, *mflops)
PAPI_ipc(*real_time, *proc_time, *ins, *ipc)
```

- **Read, accumulate, stop:**

```
PAPI_read_counters(*values, array_length)
PAPI_accum_counters(*values, array_length)
PAPI_stop_counters(*values, array_length)
```

Using PAPI Low Level Counters

Low level counters

- **EventSets**
- **Initialization:** *PAPI_library_init(version)*
- **Create evet set:** *PAPI_create_eventset (*EventSet)*
- **Add event**

```
PAPI_add_event(EventSet, EventCode)
PAPI_add_events(EventSet, *EventCode, number)
```

- **Start, read, add, stop**

```
PAPI_start(EventSet)
PAPI_read(EventSet, *values)
PAPI_accum(EventSet, *values)
PAPI_stop(EventSet, *values)
```

- **Reset:** PAPI_reset(*EventSet)*
- **Remove, empty, destroy, state**

Instrumentation

```
#include <papi.h>
………………..
int main()
{
EventSet = PAPI_NULL;
long_long values[1] = {(long_long) 0};
……

/* 1. Initialize the PAPI library */
  retval =
      PAPI_library_init(PAPI_VER_CURRENT);
  if (retval != PAPI_VER_CURRENT) {
    printf("PAPI library init error!\n");
    exit(1); }
```

```
/* 2. Create an EventSet */
  if (PAPI_create_eventset(&EventSet)
!= PAPI_OK){
    printf("PAPI library create error!\n");
    exit(1); }

/* 3. Add Total Instructions Executed to
our EventSet */
  if (PAPI_add_event(EventSet,
PAPI_L1_DCM) != PAPI_OK){
    printf("PAPI add PAPI_L1_DCM
error!\n");
    exit(1); }

/* 4. Start counting */
  if (PAPI_start(EventSet) != PAPI_OK){
    printf("PAPI library start error!\n");
    exit(1); }
```

Instrumentation

```
/**************************/
/*  COMPUTING …… */
/**************************/

  /* 5. Read the counters*/
 if (PAPI_read(EventSet, values) !=
PAPI_OK){
    printf("PAPI library read error!\n");
    exit(1); }

  data[n]= values[0];


  /* 6. Stop the counters */
  if (PAPI_stop(EventSet, values) !=
PAPI_OK){
    printf("PAPI library stop error!\n");
    exit(1); }
}
```

An example from Analysis

MKL subroutines

- Check performance of some BLAS routines by

1. L1DCA Level 1 Data Cache Access
2. L1DCM Level 1 Data Cache Misses
3. L2DCA Level 2 Data Cache Access
4. L2DCM Level 2 Data Cache Misses
5. L3DCA Level 3 Data Cache Access
6. L3DCM Level 3 Data Cache Misses
7. FPOPS Floating Point Operations
8. TOTCYC Total Cycles

MKL subroutines

- ***BLAS 1 dasum***: take a double vector and returns the sum of components as a single real
- The experiment is from 100Kb to 200Kb with 1280 steps, about 80 bytes.
- For L1DCA and L1DCM, we notice that there are three types of results:
  - The most common is L1DCA 43 and L1DCM 32.
  - Next, there are three cases of L1DCA 44 and L1DCM 33 at KB=100.016, 126.64 and 150
  - Finally there is only one case of L1DCA 45 and L1DCM 33 at 100.08.

MKL subroutines

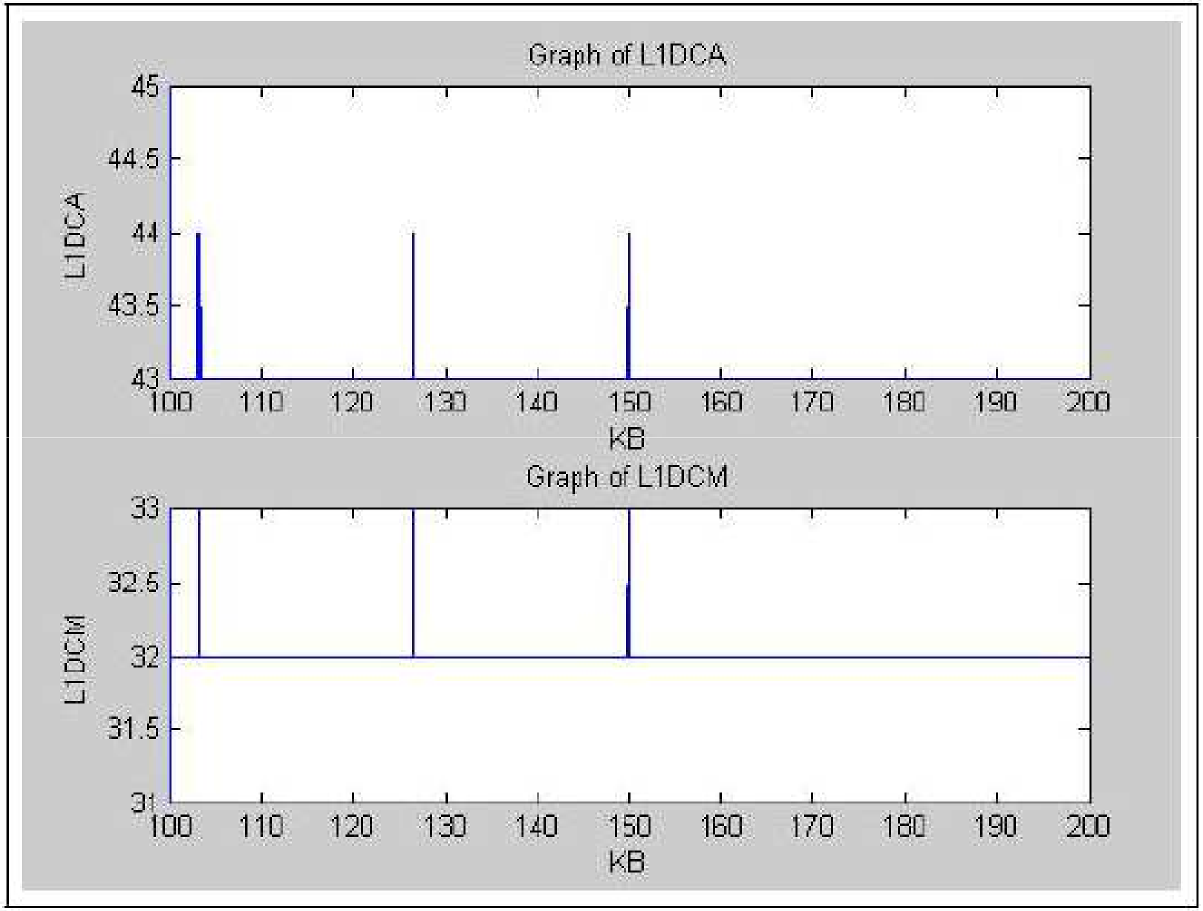


Figure 1: *Graphs of L1DCA and L1DCM.*

MKL subroutines

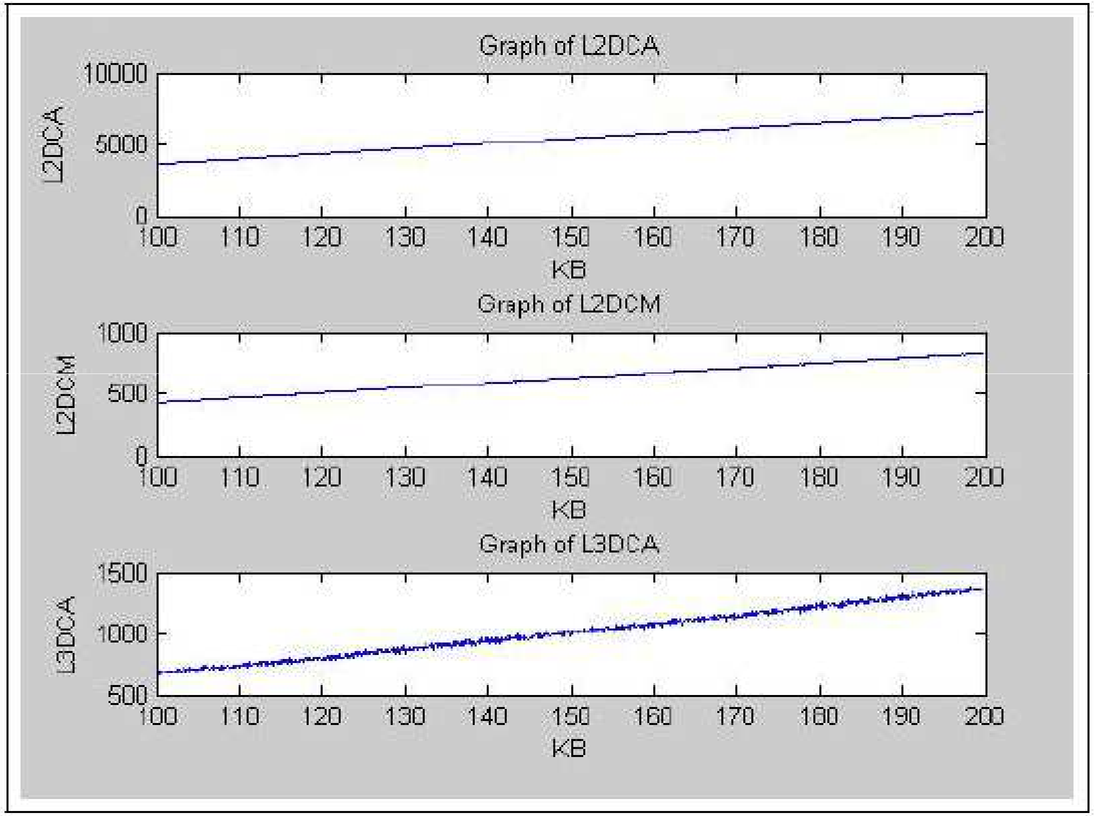


Figure 2: *Graphs of L2DCA, L2DCM, and L3DCA.*

MKL subroutines

- Fitting a line,

$$L2DCA(KB) = 45.1534 + 36.0008KB,$$

$$L2DCM(KB) = 24.8951 + 4.0079KB,$$

with RMS 0.000967. Also,

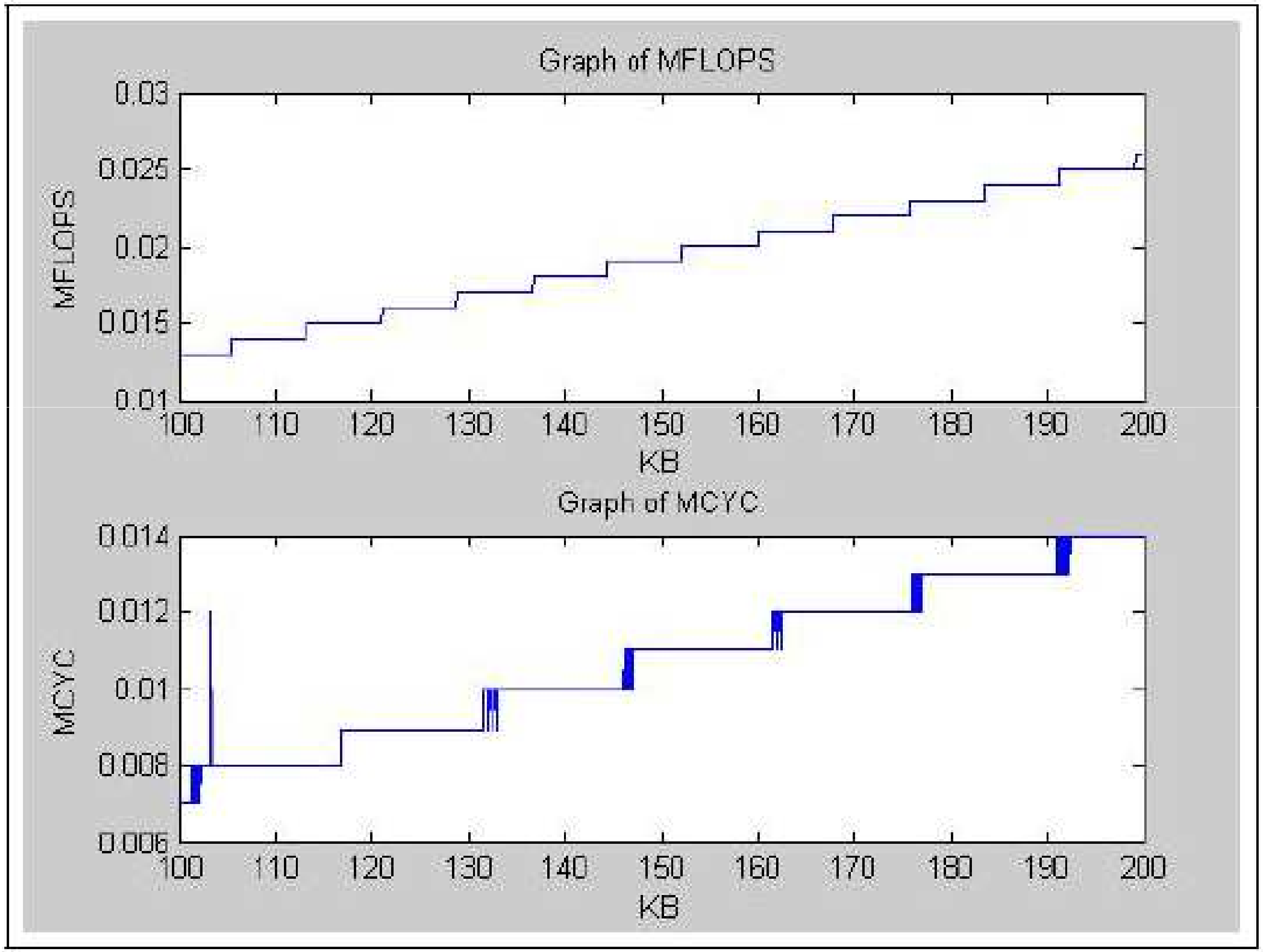
$$L3DCA(KB) = -21.3985 + 6.8827KB,$$

with RMS 0.003052.

Intel Itanium II cache architecture:

  - 16Kb L1D and 16Kb L1I
  - 1Mb L2I, 256Kb L2D
  - 12MB L3 per processor

MKL subroutines



Final words

Final Words

- *On-line* performance data
- Several events
- Off-line Data Analysis
  - No visualization
  - Large datasets
- Increasing Hardware monitors
- Complex architectures
- Accuracy: *Heisenberg principle*

References

- PAPI publications
  http://icl.cs.utk.edu/papi/pubs/index.html
- Personal reports