CS594: Performance Evaluation and Tuning

Vince Weaver
vweaver1@eecs.utk.edu

15 February 2012

Some slides based on previous years presentations by Heike Jagode
Introduction to Performance Analysis
What is Performance?

- Getting results as quickly as possible?
- Getting *correct* results as quickly as possible?
- What about Budget?
- What about Development Time?
- What about Hardware Usage?
- What about Power Consumption?
Motivation

HPC environments are expensive:

- Procurement costs: $\sim$40 million
- Operational costs: $\sim$5 million/year
- Electricity costs: 1 MW / year $\sim$1 million
- Air Conditioning costs: ??
PART II — Computer Architecture Review

You don’t have to be a computer architect to optimize code. . .

But it helps.
In an ideal world...
Compiler Should Not be Neglected

```c
int init_array(double c) {
    int i, j;

    for (i = 0; i < 100; i++) {
        for (j = 0; j < 100; j++) {
            A[i][j] = (double)(i * j) + c;
        }
    }
}
```
gcc -S

no optimization

init_array:
pushq %rbp
movq %rsp, %rbp
movsd %xmm0, -24(%rbp)
movl $0, -4(%rbp)
jmp .L2
.L5: movl $0, -8(%rbp)
jmp .L3
.L4: movl -4(%rbp), %edx
movl -8(%rbp), %ecx
movl -4(%rbp), %eax
imull -8(%rbp), %eax
cvtsi2sd %eax, %xmm0
addsd -24(%rbp), %xmm0
movslq %ecx,%rcx
movslq %edx,%rdx
movq %rdx, %rax
salq $2, %rax
...

-O3 -march=native

init_array:
movl $A, %eax
xorl %ecx, %ecx
movdqa .LC0(%rip), %xmm5
movddup %xmm0, %xmm3
movdqa .LC1(%rip), %xmm6
movd %ecx, %xmm0
movdqa %xmm6, %xmm2
pshufd $0, %xmm0, %xmm4
leaq 800(%rax), %rdx
movdqa %xmm2, %xmm0
padd %xmm5, %xmm2
pmulld %xmm4, %xmm0
cvtdq2pd %xmm0, %xmm1
pshufd $238, %xmm0, %xmm0
addpd %xmm3, %xmm1
cvtdq2pd %xmm0, %xmm0
movapd %xmm1, (%rax)
...

PAPI
ICL
7
Simple CPU

1 instruction at a time

Simple CPU
(multiple cycles)

Optimization primarily cycle-counting.
Pipelined CPU

5 instructions "in Flight"

| IF | ID | EX | MEM | WB |

Optimization: avoid Load and Branch Delays, “Bubbles” in pipeline.
Branch Prediction

- As pipelines get deeper, more work lost if branch mispredicted
- Modern branch predictors very good ( >95% accuracy) on regular workloads
- Various implementations; from simple static (backward taken) to 2-bit saturating counters, to multi-level “tournament” predictors
- Optimizations: Loop Unrolling
Loop Unrolling

```c
for(i=0;i<10;i++) {
    A[i]=i*i;
}
```

```
A[0]=0;
A[1]=1;
A[8]=64;
```
Super-Scalar CPU

Optimization: pair instructions that can run together, avoid data hazards
Out of Order

Tries to solve performance issues in hardware.
Memory Hierarchy

There’s never enough memory, so a hierarchy is created of increasingly slow storage.

- Older: CPU → Memory → Disk → Tape
- Old: CPU → L1 Cache → Memory → Disk
- Now?: CPU → L1/L2/L3 Cache → Memory → SSD
  Disk → Network/Cloud
Caches

CPU

L1I$ 32kB

L1D$ 32kB

L2$ 256kB

L3 8MB

Main Memory
## Cache Example

<table>
<thead>
<tr>
<th>Line</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x02</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x03</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0xff</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Cache Example

<table>
<thead>
<tr>
<th>Line</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x02</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x03</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0xff</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Access: 0x00120 01 2
Cache Example

Access: 0x00120 01 2

<table>
<thead>
<tr>
<th>Line</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x01</td>
<td>0x00120</td>
<td>A B C D</td>
</tr>
<tr>
<td>0x02</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x03</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0xff</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Cache Example

<table>
<thead>
<tr>
<th>Line</th>
<th>Tag</th>
<th>Data</th>
<th>Data</th>
<th>Data</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x01</td>
<td>0x00120</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td>0x02</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x03</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0xff</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Access: 0x99abc 02 03
## Cache Example

### Access:

<table>
<thead>
<tr>
<th>Line</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x01</td>
<td>0x00120</td>
<td>A</td>
</tr>
<tr>
<td>0x02</td>
<td>0x99abc</td>
<td>Z</td>
</tr>
<tr>
<td>0x03</td>
<td>0x99abc</td>
<td>Q</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T</td>
</tr>
</tbody>
</table>

**MISS**
## Cache Example

Access:

<table>
<thead>
<tr>
<th>Line</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x01</td>
<td>0x00120</td>
<td>A B C D</td>
</tr>
<tr>
<td>0x02</td>
<td>0x99abc</td>
<td>Z Q R T</td>
</tr>
<tr>
<td>0x03</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
Cache Example

<table>
<thead>
<tr>
<th>Line</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x01</td>
<td>0x00120</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B</td>
</tr>
<tr>
<td></td>
<td></td>
<td>C</td>
</tr>
<tr>
<td></td>
<td></td>
<td>D</td>
</tr>
<tr>
<td>0x02</td>
<td>0x99abc</td>
<td>Z</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Q</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T</td>
</tr>
<tr>
<td>0x03</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0xff</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Access: 0x00120 01 4

HIT!
# Cache Example

<table>
<thead>
<tr>
<th>Line</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x01</td>
<td>0x00120</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B</td>
</tr>
<tr>
<td></td>
<td></td>
<td>C</td>
</tr>
<tr>
<td></td>
<td></td>
<td>D</td>
</tr>
<tr>
<td>0x02</td>
<td>0x99abc</td>
<td>Z</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Q</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T</td>
</tr>
<tr>
<td>0x03</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0xff</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Cache Example

Access: 0x00120 02 4

<table>
<thead>
<tr>
<th>Line</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x01</td>
<td>0x00120</td>
<td>A</td>
</tr>
<tr>
<td>0x02</td>
<td>0x00120</td>
<td>B</td>
</tr>
<tr>
<td>0x03</td>
<td></td>
<td>C</td>
</tr>
<tr>
<td>0xff</td>
<td></td>
<td>D</td>
</tr>
</tbody>
</table>

MISS!
## Cache Example

**Access:**

<table>
<thead>
<tr>
<th>Line</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x01</td>
<td>0x00120</td>
<td>A B C D</td>
</tr>
<tr>
<td>0x02</td>
<td>0x00120</td>
<td>E F G H</td>
</tr>
<tr>
<td>0x03</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0xff</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Cache Example

Access: 0x99abc 02 1

<table>
<thead>
<tr>
<th>Line</th>
<th>Tag</th>
<th>Data</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x01</td>
<td>0x00120</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>0x02</td>
<td>0x99abc</td>
<td>Z</td>
<td>Q</td>
</tr>
<tr>
<td>0x03</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0xff</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

MISS
Cache Miss Types

- Compulsory — can’t avoid, first time seen
- Capacity — wouldn’t have been a miss with larger cache
- Conflict — miss caused by conflict with another address
- Coherence — miss caused by other processor
Fixing Compulsory Misses

- Prefetching (Hardware and Software)
Fixing Capacity Misses

- Build Bigger Caches
Fixing Conflict Misses

- More Ways in Cache
- Code/Variable Alignment
Fixing Coherence Misses

- False Sharing
## Virtual Memory

<table>
<thead>
<tr>
<th>Operating System</th>
<th>0xffff ffff</th>
</tr>
</thead>
<tbody>
<tr>
<td>Executable Info</td>
<td>0xbfff ffff</td>
</tr>
<tr>
<td>Stack</td>
<td></td>
</tr>
<tr>
<td>shared libraries</td>
<td></td>
</tr>
<tr>
<td>vdso</td>
<td></td>
</tr>
<tr>
<td>mmap</td>
<td></td>
</tr>
<tr>
<td>Heap</td>
<td>0x0804 8000</td>
</tr>
<tr>
<td>BSS</td>
<td></td>
</tr>
<tr>
<td>Data</td>
<td></td>
</tr>
<tr>
<td>Text (Executable)</td>
<td>0x0000 0000</td>
</tr>
<tr>
<td>Null Guard Page</td>
<td></td>
</tr>
</tbody>
</table>

### Diagram

```
Operating System
<table>
<thead>
<tr>
<th>Environment Strings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cmd Line Arg Strings</td>
</tr>
<tr>
<td>Executable Name</td>
</tr>
<tr>
<td>Padding</td>
</tr>
<tr>
<td>ELF Auxiliary Vectors</td>
</tr>
<tr>
<td>Environment Pointers</td>
</tr>
<tr>
<td>Command Line Pointers</td>
</tr>
<tr>
<td>Cmd Line Arg Count</td>
</tr>
<tr>
<td>Stack</td>
</tr>
</tbody>
</table>
```

---

*Note: This diagram illustrates the organization of virtual memory in an operating system.*
Virtual/Physical Mapping

Virtual Address Space
- Operating System
- Executable Info
- Stack
- Shared libraries
- Vdso
- Mmap
- Heap
- Bss
- Data
- Text (Executable)
- Null Guard Page

Physical Memory
- 0xffff ffff
- 0xbfff ffff
- 0x0804 8000
- 0x0000 0000
- 0x0000 0000
- 0x0000 0000
Page Tables
## TLB

<table>
<thead>
<tr>
<th>Virt Addr</th>
<th>Phys Addr</th>
<th>perm</th>
<th>v</th>
<th>pid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x12345000</td>
<td>0x1ee70000</td>
<td>x</td>
<td>y</td>
<td>456</td>
</tr>
<tr>
<td>0xdeadb000</td>
<td>0xf00d4000</td>
<td>rw</td>
<td>y</td>
<td>123</td>
</tr>
<tr>
<td>0xffff0000</td>
<td>0x00000000</td>
<td>−</td>
<td>n</td>
<td>0</td>
</tr>
<tr>
<td>0xfeedf0000</td>
<td>0xc0000000</td>
<td>r</td>
<td>y</td>
<td>456</td>
</tr>
</tbody>
</table>
Vector Units

- Operate on many values in parallel
- MMX/SSE/SSE2/SSE3/SSSE3/SSE4
- AVX
- Altivec, NEON
Multi-Threading

Not using all hardware? Split into two.
Multi-core

Nehalem CPU, as in Hydra machines
Memory Latency Concerns

- Cache Coherency — MESI
- NUMA — Non-Uniform Memory Access
More Info

- Performance Analysis Guide for Intel Core i7 Processors and Intel Xeon 5500 Processors by David Levinthal
- Intel 64 and IA-32 Architectures Optimization Reference Manual
- Software Optimization Guide for AMD Family 15h Processors
PART III — Types of Performance Analysis
Know Your Limitation

- CPU Constrained
- Memory Constrained (Memory Wall)
- I/O Constrained
- Thermal Constrained
- Energy Constrained
Performance Optimization Cycle

1. Develop Code
2. Measure
3. Analyze
4. Modify / Tune
5. Usage / Production

- Functionally Complete / Correct Code
- Functionally Complete / Correct/Optimized Code
Wisdom from Knuth

“We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil.

Yet we should not pass up our opportunities in that critical 3%. A good programmer will not be lulled into complacency by such reasoning, he will be wise to look carefully at the critical code; but only after that code has been identified” — Donald Knuth
Amdahl’s Law

- Original
- Speed up Blue 100x
- Speed up Red 2x

Time
Profiling and Tracing

Profiling
● Records aggregate performance metrics
● Number of times routine invoked
● Structure of invocations

Tracing
● When and where events of interest took place
● Time-stamped events
● Shows when/where messages sent/received
Profiling Details

- Records summary information during execution
- Usually Low Overhead
- Implemented via **Sampling** (execution periodically interrupted and measures what is happening) or **Measurement** (extra code inserted to take readings)
Tracing Details

- Records information on significant events
- Provides timestamps for events
- Trace files are typically *huge*
- When doing multi-processor or multi-machine tracing, hard to line up timestamps
Performance Data Analysis

Manual Analysis
- Visualization, Interactive Exploration, Statistical Analysis
- Examples: TAU, Vampir

Automatic Analysis
- Try to cope with huge amounts of data by automatic analysis
- Examples: Paradyn, KOJAK, Scalasca, Perf-expert
Automated Performance Analysis

• Reason for Automation
  0 Size of systems: several tens of thousand of processors
  0 ORNL’s Jaguar Cray XT5: 224,256 compute cores (2010)
  0 LLNL Sequoia (IBM, based on future Blue Gene arch.): ~1.6 million compute cores (2011-2012)
  0 Trend to multi-core

• Large amounts of performance data
  0 Several gigabytes or even terabytes
  0 Overwhelms user

• Not all programmers are performance experts
  0 Scientists want to focus on their domain
  0 Need to keep up with new machines

• Automation can solve some of these issues
Automation Example

This is a situation that can be detected automatically by analyzing the trace file.

-> *late sender* pattern
Different Low-Level Tools on Linux

- Oprofile, Vtune — System Wide Sampled Profiling
- perf — Local and System-wide profiling as well as aggregate counts
- PAPI — Low-level library providing support for profiling as well as aggregate counting and self monitoring
**perf_event on Linux**

vweaver1:hydra10 ~> perf stat /bin/ls  
Performance counter stats for '/bin/ls':

<table>
<thead>
<tr>
<th>Counter</th>
<th>Value</th>
<th>Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>task-clock-msecs</td>
<td>1.202602</td>
<td>0.119 CPUs</td>
</tr>
<tr>
<td>context-switches</td>
<td>9</td>
<td>0.007 M/sec</td>
</tr>
<tr>
<td>CPU-migrations</td>
<td>0</td>
<td>0.000 M/sec</td>
</tr>
<tr>
<td>page-faults</td>
<td>281</td>
<td>0.234 M/sec</td>
</tr>
<tr>
<td>cycles</td>
<td>3471913</td>
<td>2887.001 M/sec</td>
</tr>
<tr>
<td>instructions</td>
<td>1995325</td>
<td>0.575 IPC</td>
</tr>
<tr>
<td>cache-references</td>
<td>40271</td>
<td>33.487 M/sec</td>
</tr>
<tr>
<td>cache-misses</td>
<td>13670</td>
<td>11.367 M/sec</td>
</tr>
</tbody>
</table>

0.010071165 seconds time elapsed
The Performance API Library (PAPI)

- Low-level Crossplatform Performance Measurement Interface
- C, C++, Fortran (or attach to running process)
- Basis for more advanced visualization tools. Vampir, Tau, PerfExpert, etc.
PAPI Features

- Provides high-level access to timers
- Provides high and low-level access to performance counters
- Provides profiling support
- Provides system information
What are Hardware Performance Counters?

- Registers on CPU that measure low-level system performance
- Available on most modern CPUs; increasingly found on GPUs, network devices, etc.
- Low overhead to read
Learning About the Counters

• Number of counters varies from machine to machine
• Available events different for every vendor and every generation
• Available documentation not very complete (Intel Vol3b, AMD BKDG)
• PAPI tries to hide this complexity from users
Example Implementation – Hydra Lab Machines

- Nehalem Processor
- Counters: 4 general purpose, 3 fixed
- Events: 90 General (with many umasks), 3 Offcore, 44 Uncore, 26 Linux Software
- PEBS precise sampling and load latency monitoring
- LBR Last Branch recording
vweaver1:hydra10 ~> papi_avail | less
Available events and hardware information.

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAPI Version</td>
<td>4.1.3.0</td>
</tr>
<tr>
<td>Vendor string and code</td>
<td>GenuineIntel (1)</td>
</tr>
<tr>
<td>Model string and code</td>
<td>Intel(R) Xeon(R) CPU X5550</td>
</tr>
<tr>
<td>CPU Revision</td>
<td>5.000000</td>
</tr>
<tr>
<td>CPUID Info</td>
<td>Family: 6  Model: 26  Stepping: 5</td>
</tr>
<tr>
<td>CPU Megahertz</td>
<td>2666.822998</td>
</tr>
<tr>
<td>CPU Clock Megahertz</td>
<td>2666</td>
</tr>
<tr>
<td>Hdw Threads per core</td>
<td>1</td>
</tr>
<tr>
<td>Cores per Socket</td>
<td>4</td>
</tr>
<tr>
<td>Total CPU’s</td>
<td>4</td>
</tr>
<tr>
<td>Number Hardware Counters</td>
<td>16</td>
</tr>
<tr>
<td>Max Multiplex Counters</td>
<td>512</td>
</tr>
</tbody>
</table>
### papi_avail continued

<table>
<thead>
<tr>
<th>Name</th>
<th>Code</th>
<th>Avail</th>
<th>Deriv</th>
<th>Description (Note)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAPI_L1_DCM</td>
<td>0x80000000</td>
<td>Yes</td>
<td>No</td>
<td>Level 1 data cache misses</td>
</tr>
<tr>
<td>PAPI_L1_ICM</td>
<td>0x80000001</td>
<td>Yes</td>
<td>No</td>
<td>Level 1 instruction cache</td>
</tr>
<tr>
<td>PAPI_L2_DCM</td>
<td>0x80000002</td>
<td>Yes</td>
<td>Yes</td>
<td>Level 2 data cache misses</td>
</tr>
<tr>
<td>PAPI_L2_ICM</td>
<td>0x80000003</td>
<td>Yes</td>
<td>No</td>
<td>Level 2 instruction cache</td>
</tr>
<tr>
<td>PAPI_L3_DCM</td>
<td>0x80000004</td>
<td>No</td>
<td>No</td>
<td>Level 3 data cache misses</td>
</tr>
<tr>
<td>PAPI_L3_ICM</td>
<td>0x80000005</td>
<td>No</td>
<td>No</td>
<td>Level 3 instruction cache</td>
</tr>
<tr>
<td>PAPI_L1_TCM</td>
<td>0x80000006</td>
<td>Yes</td>
<td>Yes</td>
<td>Level 1 cache misses</td>
</tr>
<tr>
<td>PAPI_L2_TCM</td>
<td>0x80000007</td>
<td>Yes</td>
<td>No</td>
<td>Level 2 cache misses</td>
</tr>
<tr>
<td>PAPI_L3_TCM</td>
<td>0x80000008</td>
<td>Yes</td>
<td>No</td>
<td>Level 3 cache misses</td>
</tr>
</tbody>
</table>
Selected Native Events

UNHALTED_CORE_CYCLES
INSTRUCTIONS RETIRED
LAST_LEVEL_CACHE_REFERENCES
LAST_LEVEL_CACHE_MISSES
BRANCH_INSTRUCTIONS RETIRED
DTLB_LOAD_MISSES
...
FP_COMP_OPS EXE:SSE_FP_PACKED
SQ_MISC:FILL_DROPPED
vweaver1:hydra10 ~> papi_native_avail | less

<table>
<thead>
<tr>
<th>Event Code</th>
<th>Symbol</th>
<th>Long Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x40000000</td>
<td>UNHALTED_CORE_CYCLES</td>
<td>count core clock cycles whenever the clock signal on the specific core is running (not halted).</td>
</tr>
<tr>
<td>0x40000001</td>
<td>INSTRUCTION RETIRED</td>
<td>count the number of instructions at retirement.</td>
</tr>
<tr>
<td>0x40000004</td>
<td>LLC_REFERENCES</td>
<td>count each request originating from the core to reference a cache line in the last level cache. The count may include speculation, but excludes cache line fills due to hardware prefetch.</td>
</tr>
</tbody>
</table>
int main(int argc, char** argv) {
    matrix_multiply(n, A, B, C);
    return 0;
}
/* Compile with gcc -O2 -Wall mmm_papi_timer.c -lpapi */

#include <papi.h>

int main(int argc, char** argv) {

    retval = PAPI_library_init(PAPI_VER_CURRENT);
    if (retval != PAPI_VER_CURRENT) {
        fprintf(stderr,"PAPI_library_init\%d\n", retval);
    }
    start_usecs=PAPI_get_real_usec();
    matrix_multiply(n, A, B, C);
    stop_usecs=PAPI_get_real_usec();
    printf("Elapsed\%d\n",stop_usecs-start_usecs);
    return 0;
}
PAPI\_get\_real\_usec() vs PAPI\_get\_virt\_usec()

- PAPI\_get\_real\_usec()
  wall-clock time
  maps to clock\_gettime(CLOCK\_REALTIME)
- PAPI\_get\_virt\_usec()
  only time process is actually running
  maps to clock\_gettime(CLOCK\_THREAD\_CPUTIME\_ID)
Real vs Virt Timer Results

Time to run MMM, Actual Core2 Hardware

- **PAPI\_get\_real\_usec()**
- **PAPI\_get\_virt\_usec()**

Other CPU-hogging Apps Running

- **Time (us)**
  - 0
  - 500000
  - 1000000
Don’t forget the man pages!

vweaver1:hydra10 ~> man PAPI_get_real_usec
PAPI_get_real_usec(3) PAPI

NAME

PAPI_get_real_cyc - get real time counter in clock cycles

PAPI_get_real_usec - get real time counter in microseconds

DESCRIPTION

Both of these functions return the total real time passed since some arbitrary starting point. The time is returned in clock cycles or microseconds respectively. These calls are equivalent to wall clock time.
Measuring Floating Point Usage

- We’ll use the `PAPI_FP_OPS` pre-defined counter
- On Nehalem this maps to
  \[
  \text{FP\_COMP\_OPS\_EXE:}\text{SSE\_FP} + \text{FP\_COMP\_OPS\_EXE:}\text{X87}
  \]
PAPI FP Measurement

#include <papi.h>

int main(int argc, char** argv) {

    long long values[1];
    int event_set=PAPI_NULL;
    int ntv;

    retval = PAPI_library_init(PAPI_VER_CURRENT);
    if (retval != PAPI_VER_CURRENT) {
        fprintf(stderr,"PAPI_library_init\n");
    }
retval = PAPI_event_name_to_code("PAPI_FP_OPS", &ntv);
if (retval != PAPI_OK) {
    fprintf(stderr, "Error converting PAPI_FP_OPS\n");
}

retval = PAPI_create_eventset(&event_set);
if (retval != PAPI_OK) {
    fprintf(stderr, "Error creating event_set\n");
}

retval = PAPI_add_event(event_set, ntv);
if (retval != PAPI_OK) {
    fprintf(stderr, "Error adding PAPI_FP_OPS\n");
}
retval = PAPI_start( event_set );

matrix_multiply(n, A, B, C);

retval = PAPI_stop( event_set, values );

printf("PAPI_FP_OPS=\%lld\n",values[0]);

return 0;
}
Results

vweaver1:hydra10 ~/class/cs340/src> ./mmm_papi_fp
PAPI_FP_OPS = 2012334

- Code: naive 100x100 matrix-matrix multiply.
- Expected value: 100x100 multiplies + 100x100 additions = 2,000,000 floating point operations.
- The result 2,012,334 seems reasonable.
GigaFLOP/s

A more interesting benchmark might be measuring how many GigaFLOP/s (Billions of Floating Point Operations Per Second) the code can sustain.
```c
retval = PAPI_start( event_set );
start_usecs = PAPI_get_virt_usec();

matrix_multiply(n, A, B, C);

stop_usecs = PAPI_get_virt_usec();
retval = PAPI_stop( event_set, values );

flops = ((double)values[0]/
      (double)(stop_usecs - start_usecs))*1.0e6;

printf("GigaFLOP/s = %.3f\n", flops/1.0e9);
```
GFLOP/s Results

vweaver1:hydra10 ~/class/cs340/src> ./mmm_papi_gflops
GigaFLOP/s = 1.667

Theoretical Peak performance of a Xeon 5500:

85.12 GFLOP/s.

There is still some optimization that can be done here.
GFLOPS Results continued

Theoretical Peak performance of a Xeon 5500:

85.12 GFLOP/s.

Why code it yourself if a library is available?

<table>
<thead>
<tr>
<th>implementation</th>
<th>usecs</th>
<th>FLOPs</th>
<th>GFLOP/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naive</td>
<td>1200</td>
<td>2,000,635</td>
<td>1.667</td>
</tr>
<tr>
<td>ATLAS</td>
<td>560</td>
<td>1,037,425</td>
<td>1.853</td>
</tr>
<tr>
<td>GOTO</td>
<td>305</td>
<td>254,758</td>
<td>0.835</td>
</tr>
</tbody>
</table>
Parallel Code

- If using pthreads, have to enable it in PAPI
- Enable “inherit” to get combined stats for all subthreads
Cycles Per Instruction (CPI)

Most modern CPUs are super-scalar, meaning they can execute multiple instructions per cycle (this is unrelated to, but complements, multi-core). They can usually also execute instructions out-of-order but that’s another story. CPI measures how much intra-thread parallelization is happening.
Measuring CPI

- We’ll use the PAPI_TOT_CYC and PAPI_TOT_INS pre-defined counters
- On Nehalem these map to UNHALTED_CORE_CYCLES and INSTRUCTION_RETIRED
Code modified to measure CPI

```c
long long values[2];

retval = PAPI_create_eventset( &event_set );

retval = PAPI_event_name_to_code("PAPI_TOT_CYC", &ntv);
retval = PAPI_add_event( event_set, ntv );

retval = PAPI_event_name_to_code("PAPI_TOT_INS", &ntv);
retval = PAPI_add_event( event_set, ntv );
```
Code modified to measure CPI

```c
retval = PAPI_start( event_set );

matrix_multiply(n, A, B, C);

retval = PAPI_stop( event_set, values );

printf("Cycles = %lld\n", values[0]);
printf("Retired Instructions = %lld\n", values[1]);
printf("CPI = %.2f\n", (double)values[0]/(double)values[1]);
printf("IPC = %.2f\n", (double)values[1]/(double)values[0]);
```
CPI Results

vweaver1:hydra10 ~/class/cs340/src> ./mmm_papi_cpi
Cycles = 3418330
Retired Instructions = 8112910
CPI = 0.42
IPC = 2.37

Theoretical Peak performance of a Xeon 5500:

CPI = 0.25, IPC = 4.0
Other Metrics

- Branch Mispredictions: PAPI_BR_MSP
- Cache Misses: PAPI_L2_TCM
- Resource Stalls: PAPI_STL_ICY
- TLB Misses: PAPI_TLB_TL
Cautions when Using the Counters

- Not-well Validated
- Low-priority CPU feature
- Not-well Documented
Pentium 4 Retired Stores Result

Measured Loads on Pentium D vs Value in RCX for the rep movsb instruction
Non-CPU Counters

- GPU Counters
- Network, Infiniband
- Disk I/O
- Power, Energy, Temperature
Energy/Thermal, Low-End
Energy/Thermal, High-End

- Thousands of processors: every Joule counts
- Running the Air Conditioning could cost more than running the machine
PAPI-V Issues

- Time?
- Exporting VM Statistics?
- Virtualized Counters
Some Tools that Use PAPI

- **TAU** (U Oregon) [http://www.cs.uoregon.edu/research/tau/](http://www.cs.uoregon.edu/research/tau/)
- PerfSuite (NCSA) [http://perfsuite.ncsa.uiuc.edu/](http://perfsuite.ncsa.uiuc.edu/)
- HPCToolkit (Rice Univ) [http://hipersoft.cs.rice.edu/hpctoolkit/](http://hipersoft.cs.rice.edu/hpctoolkit/)
- **KOJAK and SCALASCA** (FZ Juelich, UTK) [http://icl.cs.utk.edu/kojak/](http://icl.cs.utk.edu/kojak/)
- VampirTrace and Vampir (TU Dresden) [http://www.vamir.eu](http://www.vamir.eu)
- SvPablo (UNC Renaissance Computing Institute) [http://www.renci.unc.edu/Software/Pablo/pablo.htm](http://www.renci.unc.edu/Software/Pablo/pablo.htm)
- ompP (UTK) [http://www.ompp-tool.com](http://www.ompp-tool.com)
PART V — High Level Tools
VAMPIR: Performance Analysis Suite

Consists of:
• VampirTrace part for instrumentation, monitoring and recording
• VampirServer part for visualization and analysis

VampirTrace
• Supports a variety of performance aspects: e.g. MPI comm events, subroutine calls from user code, HW perf counter, I/O events, etc.

VampirServer
• Implements client / server model with distributed server
• Allows very scalable interactive visualization for traces
Instrumentation with VAMPIRTRACE

Edit – Compile – Run Cycle

Source Code → Compiler → Binary → Run → Results

Edit – Compile – Run Cycle with VampirTrace

Source Code → Compiler → Binary → Run → Results

VT Wrapper → Traces
VAMPIR Example

1. Shows all selected processes
2. Shows state changes (activity color)
3. Shows messages, collective and MPI–IO operations
PAPI FP Exception Example
Tau Parallel Performance System

• Multi-level performance instrumentation
  • Multi-language automatic source instrumentation

• Flexible and configurable performance measurement

• Widely-ported parallel performance profiling system
  • Computer system architectures and operating systems
  • Different programming languages and compilers

• Support for multiple parallel programming paradigms
  • Multi-threading, message passing, mixed-mode, hybrid

• Integration in complex software, systems, applications
 Tau Instrumentation

Flexible instrumentation mechanisms at multiple levels:

- **Source code**
  - manual (TAU API, TAU Component API)
  - automatic
    - C, C++, F77/90/95 (Program Database Toolkit *(PDT)*)
    - OpenMP (directive rewriting *(Opari)*, *(POMP spec)*)
- **Object code**
  - pre-instrumented libraries (e.g., MPI using *(PMPI)*)
  - statically-linked and dynamically-linked
- **Executable code**
  - dynamic instrumentation (pre-execution) *(DynInstAPI)*
  - virtual machine instrumentation (e.g., Java using *(JVMPI)*)
  - Python interpreter based instrumentation at runtime
Tau Example

- Click left mouse button
- Click right mouse button
Tau Example

Paraprof viewer
(from the TAU toolset)
Tau Example

- Four significant events automatically selected (from 16K processors)
- Clusters and correlations are visible
Conclusions

- Performance Analysis is important in HPC
- Performance Analysis tools are available that can ease optimization
Questions?