Making Performance Understandable: Towards a Standard for Performance Counters on Manycore Architectures

Sarah Bird, Andrew Waterman, Kevin Klues, Sam Williams, Kaushik Datta, Rajesh Nishtala, Krste Asanovic, James Demmel, Dave Patterson
Par Lab Research Overview

Easy to write correct programs that run efficiently on manycore
Par Lab Research Overview

Easy to write correct programs that run efficiently on manycore

- Personal Health
- Image Retrieval
- Hearing, Music
- Speech
- Parallel Browser

Design Patterns/Motifs

Composition & Coordination Language (C&CL)

C&CL Compiler/Interpreter

- Parallel Libraries
- Parallel Frameworks

Efficiency Languages

- Sketching
- Autotuners

- Legacy Code
- Schedulers
- Communication & Synch. Primitives

Efficiency Language Compilers

- Legacy OS
- OS Libraries & Services
- Hypervisor
- Multicore/GPGPU
- RAMP Manycore

Static Verification

- Type Systems
- Directed Testing
- Dynamic Checking
- Debugging with Replay

Applications

Productivity Layer

Efficiency Layer

Diagnosing Power/Performance

OS

Arch.
Par Lab Research Overview

Easy to write correct programs that run efficiently on manycore

Applications

Productivity Layer

Efficiency Layer

Diagnosing Power/Performance

Personal Health

Image Retrieval

Hearing, Music

Speech

Parallel Browser

Design Patterns/Motifs

Composition & Coordination Language (C&CL)

C&CL Compiler/Interpreter

Parallel Libraries

Parallel Frameworks

Efficiency Languages

Legacy Code

Sketching

Autotuners

Schedulers

Communication & Synch. Primitives

Efficiency Language Compilers

Legacy OS

OS Libraries & Services

Hypervisor

Multicore/GPGPU

RAMP Manycore

Static Verification

Type Systems

Directed Testing

Dynamic Checking

Debugging with Replay

Correctness
Outline

- Motivation
- Current State of Performance Counters
- Proposed Solution
  - Framework
  - Activity Counters
- Motivating Applications
- RAMP and Future Work
Parallel Programming

- Parallel Programming is Challenging
  - Efficiency Programmers have struggled for years
  - Now we expect Productivity Programmers to write parallel code?
    - Correct
    - Power Efficient
    - Reasonable Performance
    - Quickly Written

What is the solution?

There isn’t a quick fix and programmers need help!
Programmer Tools

- More insight into application behavior
  - Better Debugging Tools
  - Better Performance Analysis Tools
  - Tools which help make scheduling and resource decisions
  - Must be portable
  - Must work in Realtime

- Claim:
  Accurate, useful performance counters more important to business success of multicore bet than clock rate, cache size, transactional memory support, ...
Outline

- Motivation
- Current State of Performance Counters
- Proposed Solution
  - Activity Counters
  - Framework
- Motivating Applications
- RAMP and Future Work
5 problems with Current Systems

- State of performance counters today is lousy
  - Intended for use by chip designers not users
  - Low priority since they are intended for internal use
  - Opportunistic bottom-up measurements

1. Inaccurate
2. Non-functional
3. Incomplete
4. Overly Complex
5. Inconsistent
Current Uses

- Only use the simple counters
  - Rely on simple performance models like CPI

- Get a graduate student to analyze the application, architecture, and perf. data
  - Doesn’t scale

- Use machine learning on all the counters
  - Complex and unclear if it is useful.. particularly if some of the counters are non-functional or inaccurate

- Software solutions like PAPI
  - Can’t overcome inconsistent, inaccurate or incomplete counters
Outline

- Motivation
- Current State of Performance Counters
- Proposed Solution
  - Activity Counters
  - Framework
- Motivating Applications
- RAMP and Future Work
Proposal

- Create a standard for counters on all future architectures
  - Places pressure on chip designers to require them to be functional, accurate, and available
  - Proactive
    - Avoid the problems of PAPI
  - Allows the creation of portable software
    - Autotuners & other Performance Analysis tools
    - Dynamically adjusting applications (Music)
    - Operating System Schedulers
Our Approach

- Measure Computation, Communication, and Energy for all components
Computation

- Efficient Execution of each core still important
  - Power/Energy
  - Overall system performance

- Measure instructions retired
  - Used by Applications, Scheduler, Productivity Programmers

- Measure instruction mix
  - Floating Point Ops, Loads, Stores, etc
  - Used by Efficiency Programmers, Program Analysis tools
Communication

- Network behavior can have a big impact on manycore performance
  - Access to DRAM and I/O
  - Communication between cores

- Measure Traffic on Each Edge
  - Used by Applications, Scheduler, Productivity Programmers

- Break traffic into types
  - Prefetch, Compulsory, Coherency, etc
  - Used by Efficiency Programmers, Program Analysis tools
Energy Counters

- Energy information can affect some non-obvious tradeoffs for applications
  - Client Server Split

- Counters to measure energy of all components
  - Everything in units of energy
    - Affects battery life
    - Works with DVFS

- Shared Resources must attribute energy to apps
  - DRAM provides a power model
  - Memory controller uses the model and accesses
Resources

This seems like a lot of hardware...

- Counters designed to be
  - Fixed Function
- As a result they can easily be made to be
  - Simple
  - Small
  - Low Power

It’s worth it if we improve significantly improve performance
Counter Characteristics

- Fixed Function
  - Small
  - Low Power
- Wide (64 bits)
- Accurate
- Always On
- Accessed in a *Reasonable* time
  - Latches to quickly record values
  - Buffers & DMA to save values to memory
Counter Framework

- Atomically Snapshot Set of Counters
  - Use a 100 Mhz Global Realtime Clock (GRTC)
    - Helps solve DVFS
  - Triggers to the OS and User Level Latches
Counter Framework

- Composable

  - Different levels of the system are interesting
    - Application -> Cores, Partitions
    - OS -> Partitions, System
Outline

- Motivation
- Current State of Performance Counters
- Proposed Solution
  - Activity Counters
  - Framework
- Motivating Applications
- RAMP and Future Work
Performance Counter Uses

- Applications
  - Dynamic Execution Adjustment
  - Debugging Tools
  - Performance Analysis Tools
- Autotuners
- Operation System
  - Scheduler/Resource Allocation
The Music Application

- Strong realtime requirements
  - Realtime clock
  - Packet timestamping and logging

- Lots of I/O
  - Need good bandwidth
    - Traffic measurements
  - Novel I/O Devices
    - Ethernet AVB
    - Interface to get counters from devices
  - Extremely latency sensitive
    - I/O logging with timestamping using the global clock
Autotuning

- Autotuning with Machine Learning
  - Standard counters to create a portable system
  - Compare the performance of on application with different architectures

- Roofline
  - Model to represent the performance of application on an architecture
  - Autogenerate model using performance counters
Space-Time Scheduling

- Portable
  - Standard for all architectures
- Track ALL resource usage and compute performance-bandwidth-energy curves on the fly
  - Computation, Communication and Energy
- Profile resource usage in different application phases
  - Atomic snapshot a whole application at once
- Energy constraints on applications
  - Energy Counter and Energy models for shared resources
Outline

- Motivation
- Current State of Performance Counters
- Proposed Solution
  - Activity Counters
  - Framework
- Motivating Applications
- RAMP and Future Work
RAMP Gold and Performance Counters

- Research Accelerator for Multiple Processors
- Manycore emulation on FPGAs
- Use RAMP to implement performance counters
  - Study application behavior using our activity counters
  - Do complete tracing of dependency behavior to extract useful information

- Write new applications/tools that use the counters
- Experiment with new counters
Future Work: Diagnostic Tools

- Goals
  - Recreate Dependency Graphs and Calculate Slack
  - Logging of I/O latency for novel I/O devices

- Implementation:
  - Add simple non-intrusive hardware to record information
  - Software can recreate program information from logged data
  - Lots of compression can be done
Future Work: Starting Points

- Dependency Graphs
  - Use something similar to the Shotgun Approach
  - Keep track of the last writer on each cache line
    - Allows communication arcs to be recorded on reads

- I/O Information
  - Allow I/O devices to timestamp packets using the GRTC
  - System packets are also timestamped with the GRTC
  - All packets can be logged
  - Create a standard way for the network interface to access relevant performance counters on novel I/O devices
Conclusions

- Must have a standard for performance counters
  - Always On
  - Accurate and Accessible
  - Same across all architectures

- Big impact on future software systems
  - Aid programmer, autotuner, scheduler, OS in adapting system
  - Help turn data into useful information that can help *efficiency*-level programmer improve system
    - Why not getting 100% of memory bandwidth? Conflict misses?
  - Help turn data into useful information that can help *productivity*-level programmer improve app
    - Where am I spending my time in my program?
    - If I change it like this, impact on performance?
Acknowledgements

In addition to all the authors at the beginning of this talk, I would like to thank

- Ras Bodik
- David Wessel
- The BeBOP group

Questions?

slbird@eecs.berkeley.edu
Extra Slides
Solving the Parallel Problem

- A lot of solutions have been proposed
  - New programming languages
  - Parallel Frameworks
  - Better Compilers
  - Speculative Execution/Transaction Memory
  - Better Hardware
  - ....

There isn’t a quick fix and programmers need help!
## Utilization Meters

<table>
<thead>
<tr>
<th>Communication</th>
<th>Computation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compulsory Misses</td>
<td>Floating Point Instructions</td>
</tr>
<tr>
<td>Capacity Misses</td>
<td>Atomic Instructions</td>
</tr>
<tr>
<td>Conflict Misses</td>
<td>Branches</td>
</tr>
<tr>
<td>Coherency Misses</td>
<td>Integer Ops</td>
</tr>
<tr>
<td>Prefetch Data</td>
<td>Load/Stores</td>
</tr>
<tr>
<td>Write Allocation Data</td>
<td>SIMD/Vector</td>
</tr>
<tr>
<td>Misc Data</td>
<td>Misc Instructions</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>Sum (All Traffic)</td>
<td>Sum (All Retired Instrs)</td>
</tr>
</tbody>
</table>
Communication

- Measure Traffic on Each Edge
- Compulsory Traffic
  - TLB/Page Tables maintain a reference bit per cache line
  - 4KB pages and 64B blocks => 64 new bits per PTE
- Conflict Misses
  - Approximate using a tag victim cache
- Coherency Traffic
  - Invalidations