PerfMiner: Cluster-Wide Collection, Storage and Presentation of Application Level Hardware Performance Data

Philip J. Mucci, Daniel Ahlin
Lars Malinowski, Per Ekman, Johan Danielsson

Center for Parallel Computers (PDC), Royal Institute of Technology (KTH), Stockholm, Sweden
Innovative Computing Laboratory, UT Knoxville

mucci at cs.utk.edu, http://www.cs.utk.edu/~mucci

Parallelldatorcentrum

September 1st, 2005
EuroPar '05, Costa de Caparica, Portugal

Outline

- Background
- Motivation
- Integration & Implementation
- Results
- Future Work

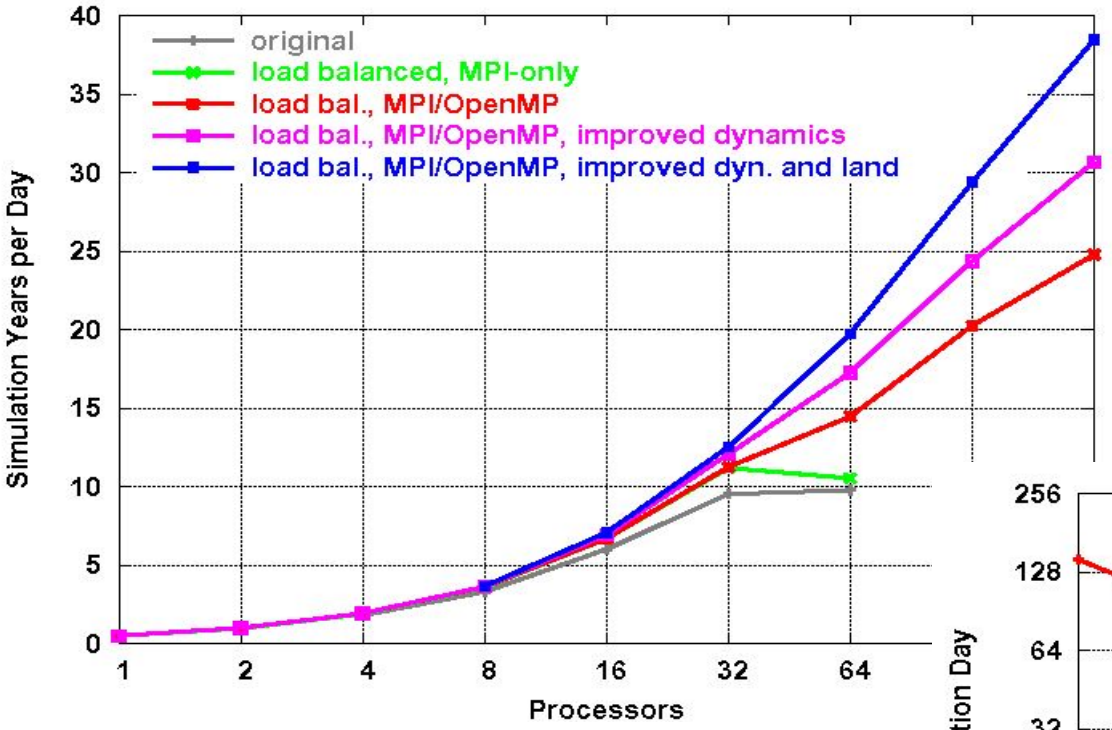
Why Performance Analysis?

- 2 reasons: Economic & Qualitative
- Economic: Time really is Money
  - Consider that the average lifetime of these large machines is ~4 years before being decommissioned.
  - Consider the cost per day of a $4,000,000 Dollar machine, with annual maintenance/electricity cost of $300,000. That's $1500.00 (US) per hour of compute time.

Why Performance Analysis? (2)

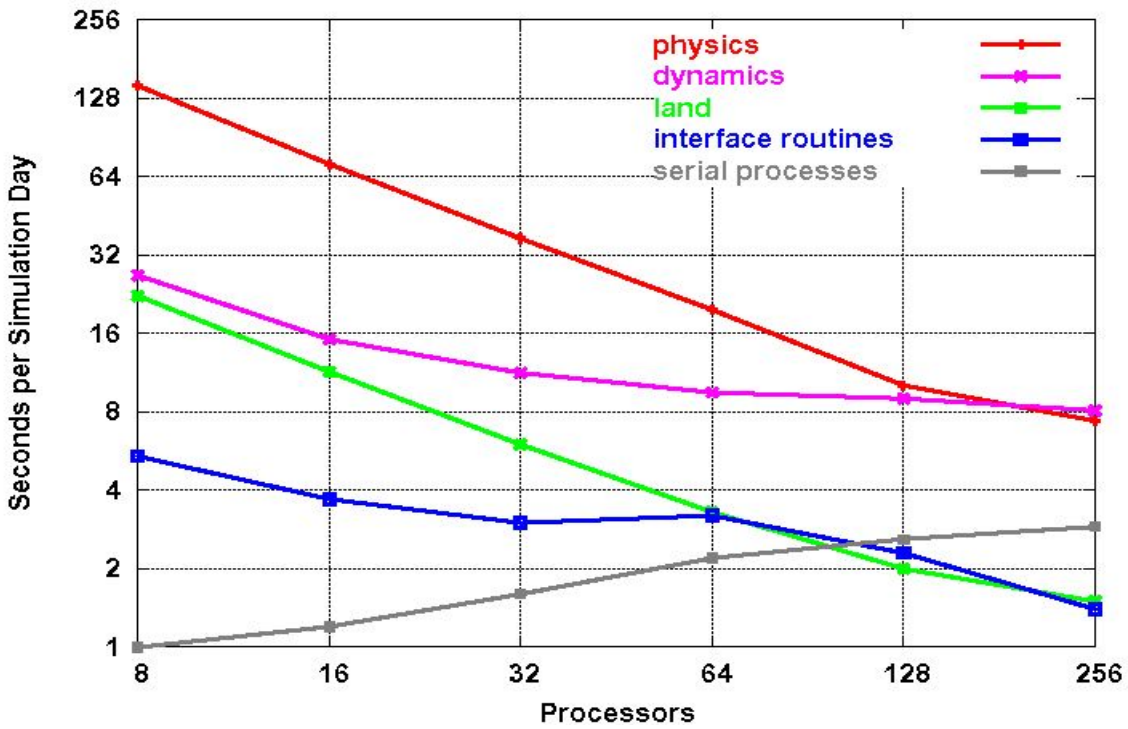
- Qualitative: Improvements in Science
  - Consider: Poorly written code can easily run 10 times worse than an optimized version.
  - Consider a 2-dimension domain decomposition of a Finite Difference formulation simulation.
  - For the same amount of time, the code can do 10 times the work. 400x400 elements vs. 1300x1300 elements
  - Or it can do 400x400 for 10 times more time-steps.
  - These could be the difference in resolving the phenomena of interest!

Biology and Environmental Sciences
CAM



CAM performance measurements on IBM p690 cluster (and other platforms) were used to direct development process. Graph shows performance improvement from performance tuning and recent code modifications.

Profile of current version of CAM indicates that improving the serial performance of the physics is the most important optimization for small numbers of processors, and introducing a 2D decomposition of the dynamics (to improve scalability) is the most important optimization for large numbers of processors.



Why Performance Analysis? (3)

- So, we must strive to evaluate how our code, software, hardware and systems are running.
- Learn to think about performance during the entire life-cycle of an application, a software stack, a cluster, an architecture etc...

Center for Parallel Computers (PDC)

- The biggest of the centers in Sweden that provides HPC resources to the scientific community. (~1000 procs, ~2TF)
  - Vastly different user bases, from bioinformatics to CCM.
  - One large resource is part of Swegrid.
- Wanted to purchase a new machine. (3-4x)
  - Lack of explicit knowledge of the dominant applications and their bottlenecks. No tool!

Related Work Didn't Help

- The same problem over and over: utter lack of detail
  - Batch logs
  - SuperMon, CluMon, Ganglia, Nagios, PCP, NWPerf
  - Vendor specific monitoring software...
- Only NCSA's internal system (from Rick Kufrin) met our needs. But that system has not been made public! So....

PerfMiner: Bottom Up Performance Monitoring

- Allow performance characterization of all aspects of a technical compute center:
  - Application Performance
  - Workload Characterization
  - System Performance
  - Resource Utilization
- Provide users, managers and administrators with a quick and easy way to track and visualize performance of their jobs/system.
- Full integration from batch system to database to web interface.
  - Completely transparent to the user.

3 Audiences

- Users: Integrating Performance into the Software Development Life-cycle
  - Quick and elegant way to obtain and maintain standardized perf. information about one's jobs.
- Administrators: Performance Focused System Administration
  - Efficient use of HW, SW and personnel resources.
- Managers: Characterization of True Usage
  - Purchase of a new compute resource.

Rising Processor Complexity

- No longer can we easily trace or model the execution of a code.
  - Static/Dynamic Branch Prediction
  - Hardware/Software Prefetching
  - Out-of-order scheduling
  - Predication
  - Non-overlapping caches
- So, just a measure of 'wallclock' time is not enough.
- What's really happening under the hood?

The Trouble with Timers

- They depend on the load on the system.
  - Elapsed wall clock time does not reflect the actual time the program is doing work due to:
    - OS interference.
    - Daemon processes.
- The solution?
  - We need measurements that are accurate yet independent of external factors. (Help from the OS)

Hardware Performance Counters

- Performance Counters are hardware registers dedicated to counting certain types of events within the processor or system.
  - Usually a small number of these registers (2,4,8)
  - Sometimes they can count a lot of events or just a few
  - Symmetric or asymmetric
  - May be on or off chip
- Each register has an associated control register that tells it what to count and how to do it. For example:
  - Interrupt on overflow
  - Edge detection (cycles vs. events)
  - User, kernel, interrupt mode

Availability of Performance Counters

- Most high performance processors include hardware performance counters.
  - AMD
  - Alpha
  - Cray MSP/SSP
  - PowerPC
  - Itanium
  - Pentium
  - MIPS
  - Sparc
  - And many others...

Available Performance Data

- Cycle count
- Instruction count
  - All instructions
  - Floating point
  - Integer
  - Load/store
- Branches
  - Taken / not taken
  - Mispredictions
- Pipeline stalls due to
  - Memory subsystem
  - Resource conflicts
- Cache
  - I/D cache misses for different levels
  - Invalidations
- TLB
  - Misses
  - Invalidations

Hardware Performance Counter Virtualization by the OS

- Every process appears to have its own counters.
- OS accumulates counts into 64-bit quantities for each thread and process.
  - Saved and restored on context switch.
- All counting modes are supported (user, kernel and others).
- Explicit counting, sampling or statistical histograms based on counter overflow.
- Counts are largely independent of load.

PAPI

- **P**erformance **A**pplication **P**rogramming **I**nterface
- The purpose of PAPI is to implement a standardized portable and efficient API to access the hardware performance monitor counters found on most modern microprocessors.
- The goal of PAPI is to facilitate the optimization of parallel and serial code performance by encouraging the development of cross-platform optimization tools.

Design

- Integration into the User's Environment
- Collection of Hardware Performance Data
- Post-processing and Storage of Performance Data
- Presentation of the Data to the Various Communities

Site Wide Performance Monitoring

- Integrate complete job monitoring in the batch system itself.
- Track every cluster, group, user, job, node all the way down to individual threads.
- Zero overhead monitoring, no source code modifications.
- Near 100% accuracy.

Batch System Integration

- PDC runs a heavily modified version of the Easy scheduler. (ANL)
  - Reservation system that twiddles /etc/passwd.
  - Multiple points of entry to the compute nodes
  - Kerberos authentication
- Monitoring must catch all forms of usage.
  - MPI, Interactive, Serial, rsh, etc...

Batch System Integration (2)

- Need to a shell script before and after every job.
  - Bash prevents this behavior!
- We must use /etc/passwd as the entry point!
  - Custom wrapper that runs a prologue and execs the real shell.
  - The prologue sets up data staging area and monitoring infrastructure.
- Batch system runs the epilogue.

Batch System Integration (3)

- Data is dumped into a job specific directory and flagged as BUSY.
- Data about the batch system and job are collected into a METADATA file.

```
JOBID:111714450953
CLUSTER:j-pop
USER:lama
CHARGE:ta.lama
ACCEPTTIME:1100702861
PROCS:4
FINALTIME:1100703103
```

Data Collection with PAPIEX

- PapiEx: a command line tool that collects performance metrics along with PAPI data for each thread and process of an application.
  - No recompilation required.
- Based on PAPI and Monitor libraries.
- Uses library preloading to insert shared libraries before the applications. (via Monitor)
  - Does not work on statically linked or SUID binaries.

Some PapiEx Features

- Automatically detects multi-threaded executables.
- Supports PAPI counter multiplexing; use more counters than available hardware provides.
- Full memory usage information.
- Simple instrumentation API.
  - Called PapiEx Calipers.

Monitor

- Generic Linux library for preloading and catching important events.
  - Process/Thread creation, destruction.
  - fork/exec/dlopen.
  - exit/_exit/Exit/abort/assert.
  - User can easily add any number of wrappers.
- Weak symbols allow transparent implementations of dependent tool libraries.

```
PapiEx Version:         0.99rc2
Executable:             /afs/pdc.kth.se/home/m/mucci/summer/a.out
Processor:              Itanium 2
Clockrate:              900.000000
Parent Process ID:      8632
Process ID:             8633
Hostname:               h05n05.pdc.kth.se
Options:                MEMORY
Start:                  Wed Aug 24 14:34:18 2005
Finish:                 Wed Aug 24 14:34:19 2005
Domain:                 User
Real usecs:             1077497
Real cycles:            969742309
Proc usecs:             970144
Proc cycles:            873129600
PAPI_TOT_CYC:           850136123
PAPI_FP_OPS:            40001767
Mem Size:               4064
Mem Resident:           2000
Mem Shared:             1504
Mem Text:               16
Mem Library:            2992
Mem Heap:               576
Mem Locked:             0
Mem Stack:              32

Event descriptions:
Event: PAPI_TOT_CYC
        Derived: No
        Short Description: Total cycles
        Long Description: Total cycles
        Developer's Notes:
Event: PAPI_FP_OPS
        Derived: No
        Short Description: FP operations
        Long Description: Floating point operations
        Developer's Notes:
```

PapiEx Sample Output

The Back End

- Directory is marked BUSY with a file.
- After termination of every thread, PapiEX writes a file. (Max 2K in length.)
- At job termination, epilogue script removes BUSY file.
- Data is consumed by an offline process that imports the data to the database and archives the original data on secondary storage.

Scalable Database Design

- Now in version 2, implemented in Postgres.
  - Portable to other back ends.
- May contain many millions of rows for a production system.
- Population by the epilogue is done through Perl scripts and DBI.
  - All DB structure contained in the scripts.
  - No external schemas or DB setup required.

Scalable Database Design (2)

- 4 keys: cluster, job-id, process-id, thread-id
- First version was implemented with a base table of 'standard' metrics and individual tables for specific metrics.
  - Version 2 has a separate table for every metric.
- Each table has a scope (or is a node in an ontology).

Scalable Database Design (3)

- Direct measurements.
  - Events that are measured directly by the underlying performance tool (and METADATA).
- Derived measurements:
  - Events that are explicitly constructed from complex queries.
  - SQL for constructing them is embedded in the database. Measurements can be hidden as VIEWS.
    - Rates, Ratios, etc.

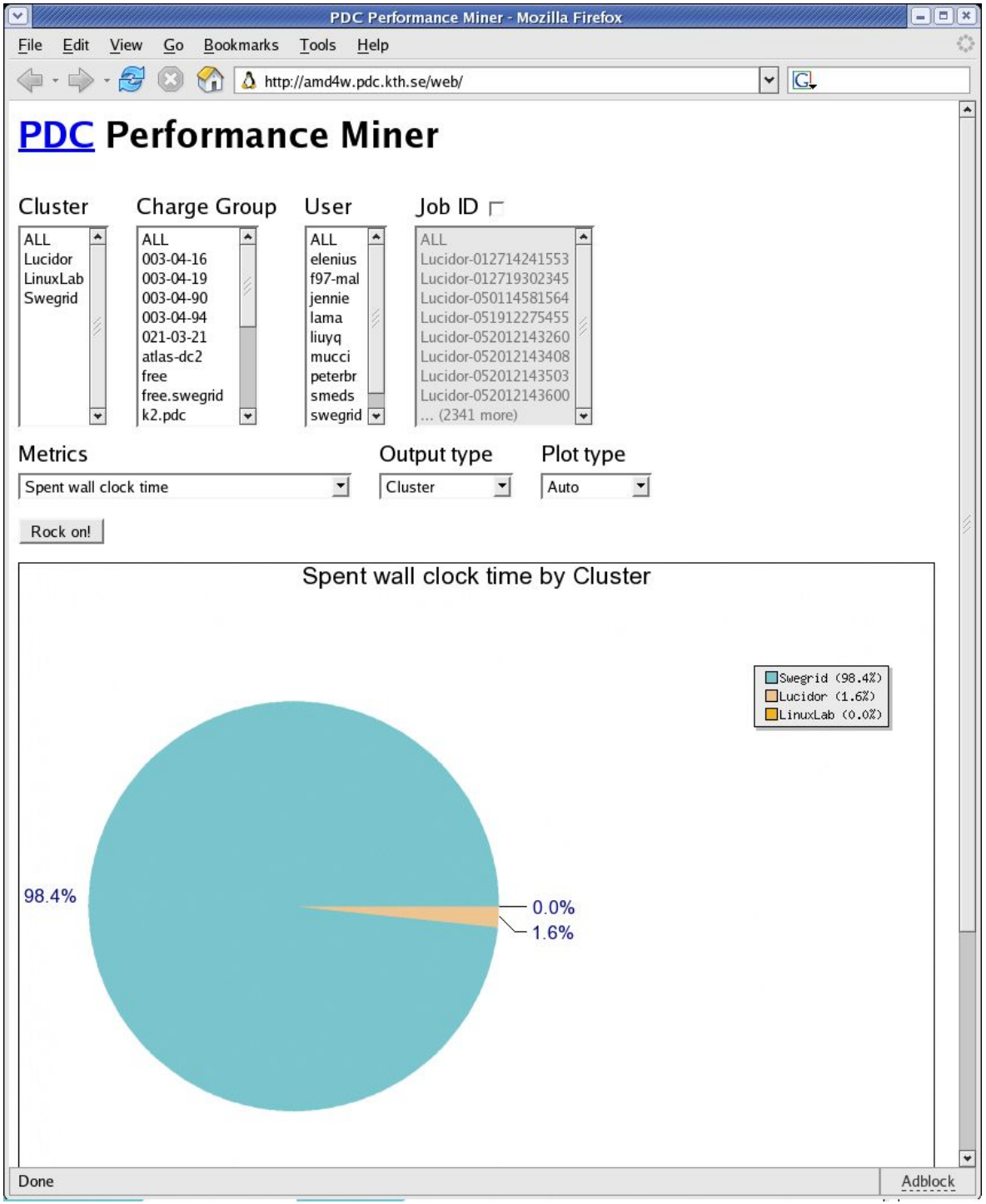
PerfMiner Interface

- Straight HTML fed by PHP scripts.
  - JpGraph/GD and PHP DBI.
- Proof of concept interface: lots more work to do.
- 3 selection criteria:
  - What event to visualize?
  - What range/scope to select?
  - What range/scope to display over?

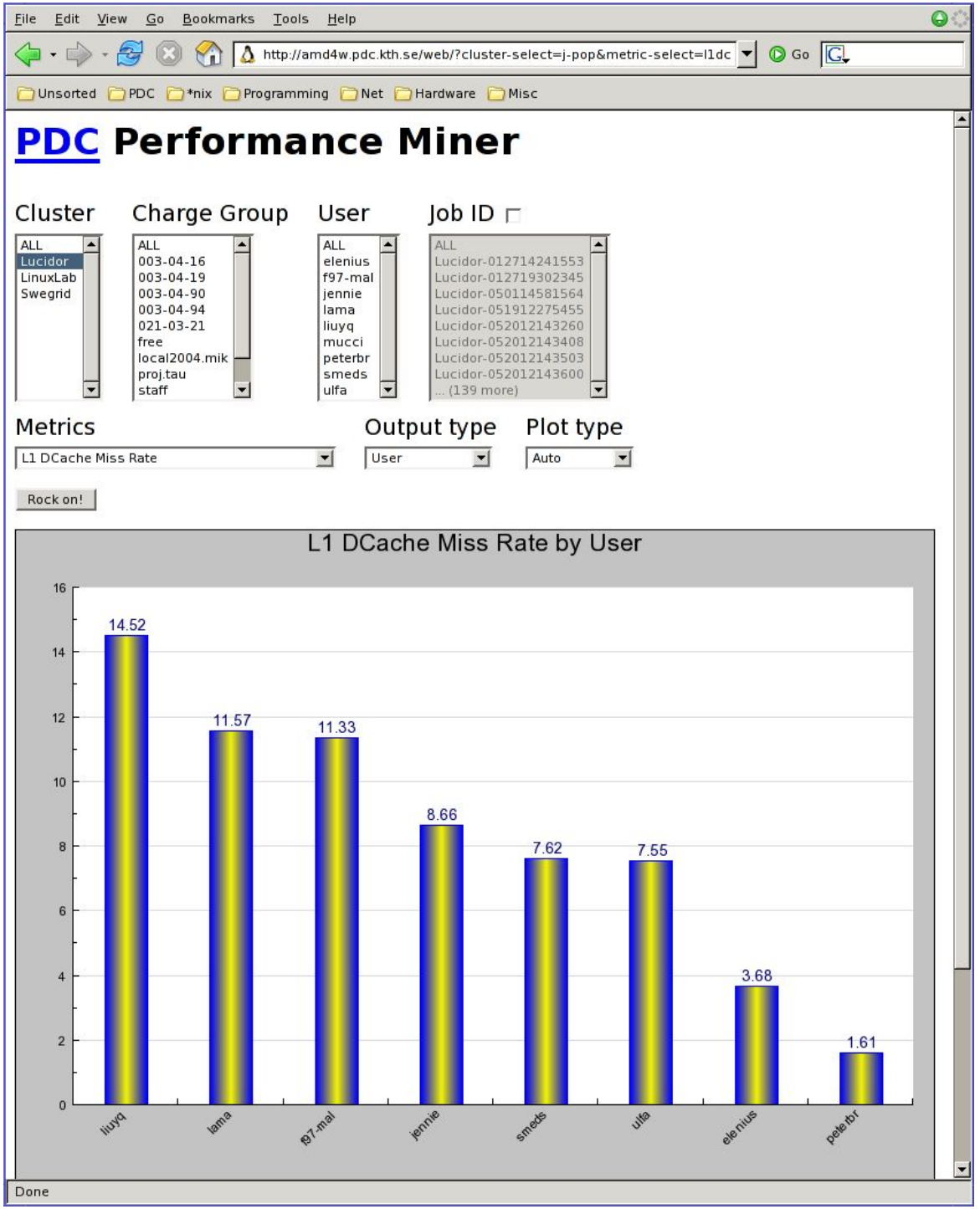
Test Deployment

- Run for 3 weeks on 3 different clusters.
  - Lucidor: 90 Dual HP-ZX1 Nodes (IA64)
  - Swegrid: 100 Pentium 4 Nodes (x86)
  - Linux Labs: 16 Dual Pentium III (x86)
- ~2.5 Million Threads in the database.

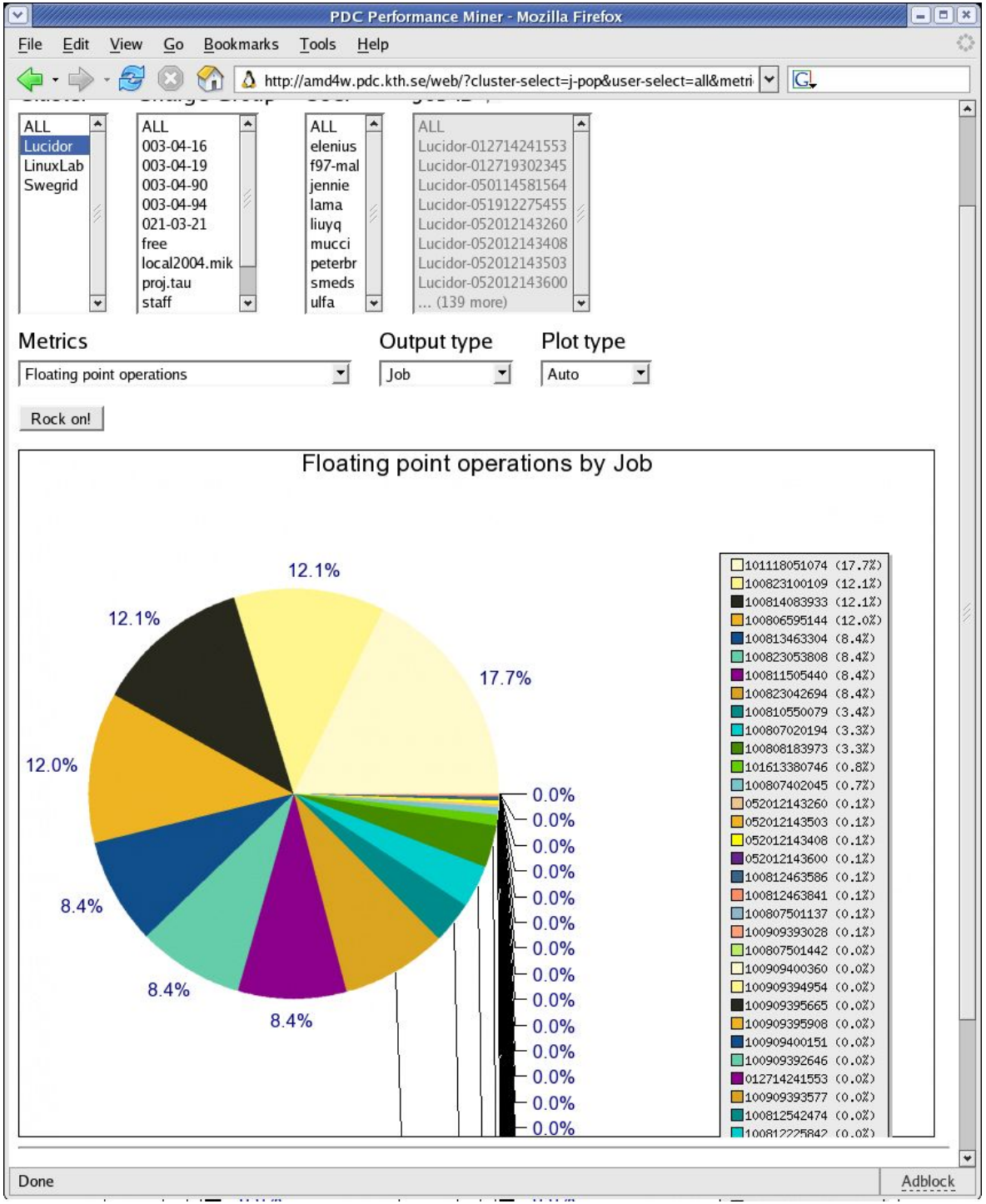
PerfMiner: Main Window



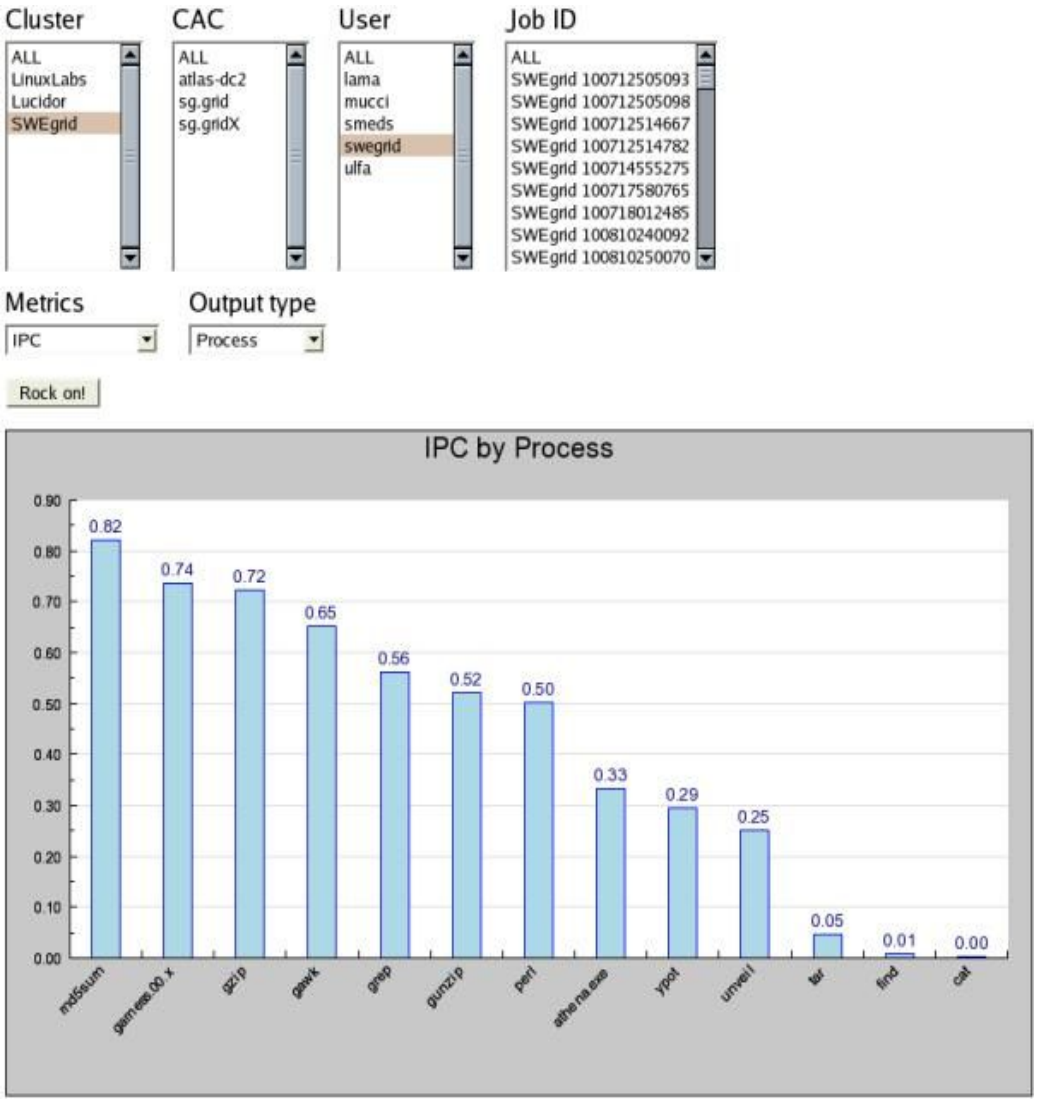
PerfMiner: L1 Miss Rate by User



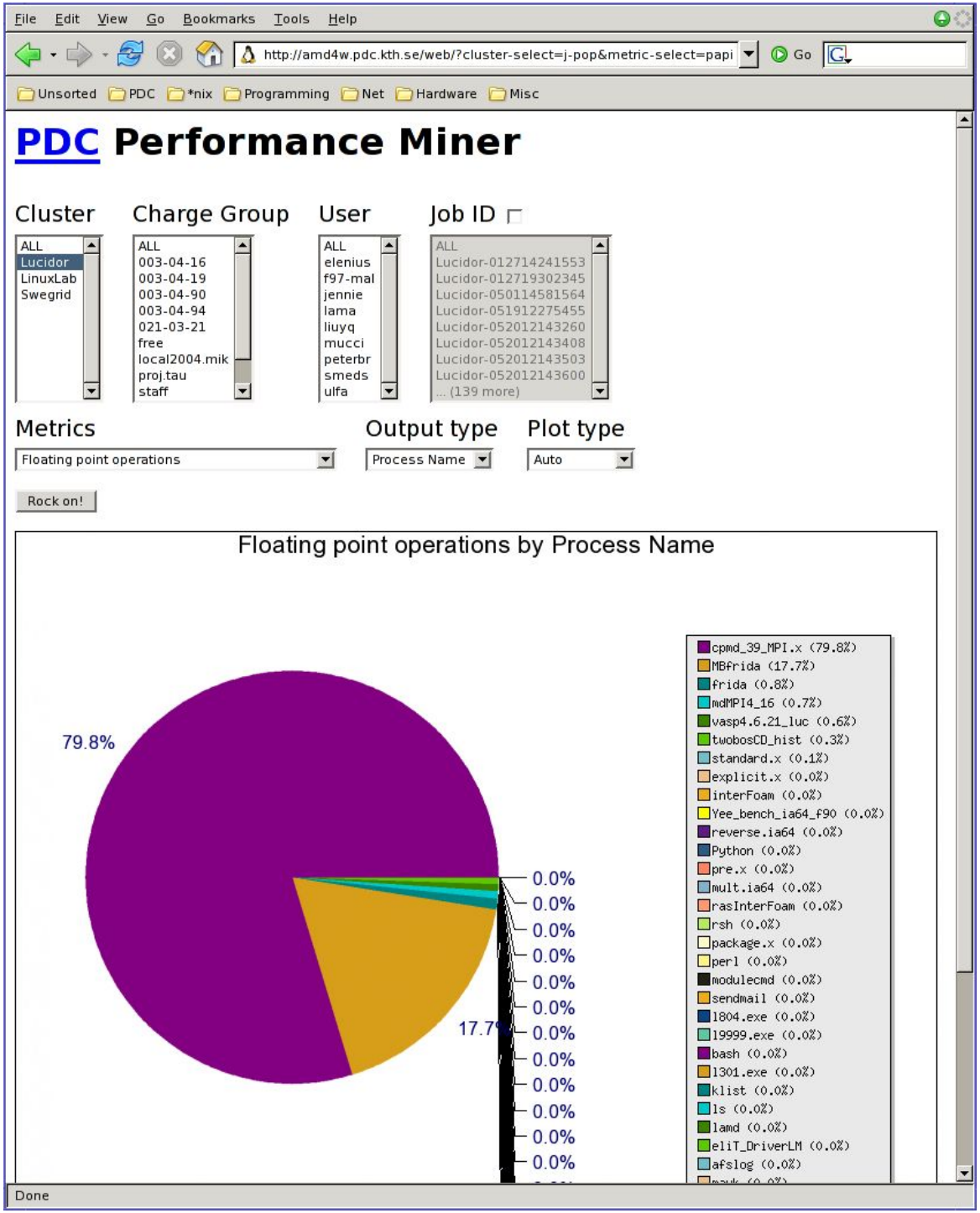
PerfMiner: FP Ops by Job ID



PerfMiner: IPC by Process



PerfMiner: FP Ops by Process Name



Many Additional Queries

- Degree of thread and process parallelism.
  - Wasted nodes, processors?
- Wait time in queue.
- Number of threads in each application.
  - Wasted processor?
- Memory usage.
- Relatively easy to add new complex queries through the derived metrics mechanism.

Issues

- Some queries can take a very long time.
  - Do the queries need to be realtime?
- Interference with instrumented codes that make use of the PM hardware.
  - Hints to the batch script to disable monitoring.
- Browser and JpGraph overhead of rendering:
  - Drop down boxes
  - Graphs with hundreds of thousands of points.

Extensions

- KSOM's, Principle Component Analysis and Data Clustering Techniques.
- Integration with a User and Group Accounting System.
- Additional monitoring modules could be specified by the user.
  - MPI, OpenMP, UPC entry points.
  - User defined entry points.

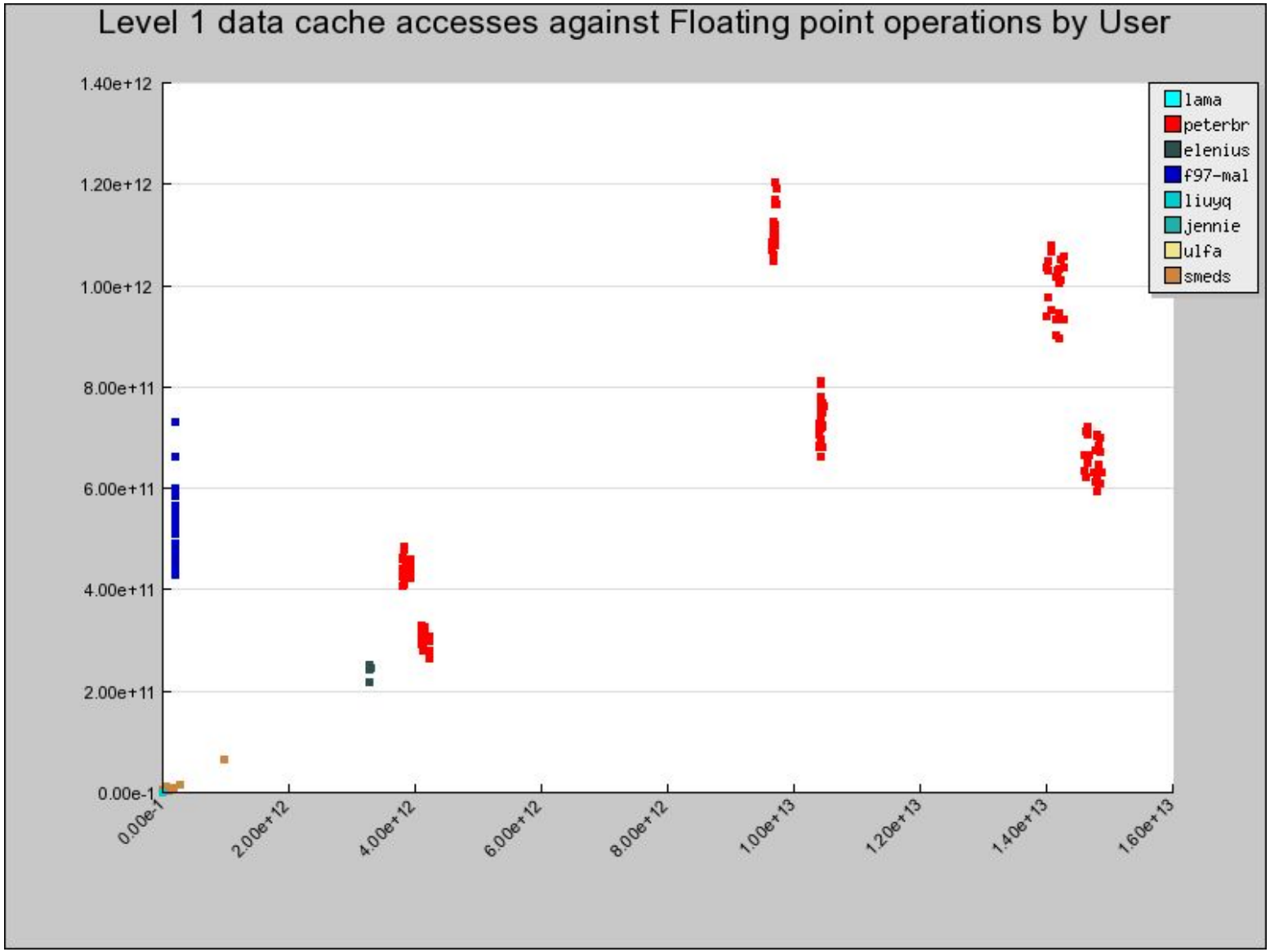
Portability and Availability

- CVS tree is currently private, but open soon.
  - Contact us if you'd like to be notified.
- System is reasonably portable, but:
  - Datatypes?, VIEW syntax?
  - Batch system and METADATA file are rather specific to the site.
  - You need LD_PRELOAD. (AIX, Unicos)

Scatter Plots and Correlation

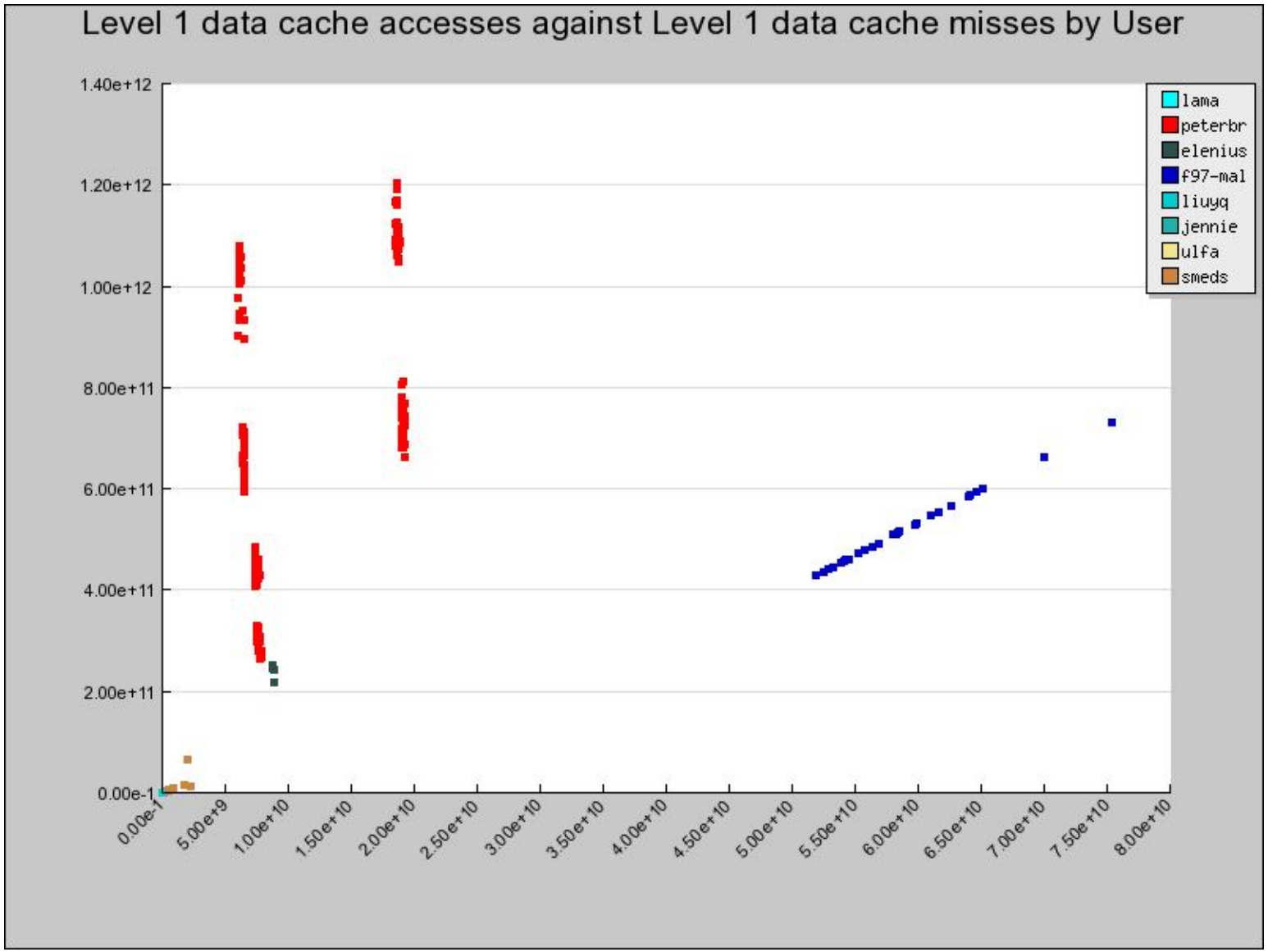
- We expect many performance metrics to be more or less correlated in some fashion.
  - (instructions vs. cycles, flops vs. misses)
- Scatter plots provide one method of exposing this.
- Data points that cluster in non-obvious fashion can provide guidance for focused attention.

Scatter Plot (1)



- Is the user optimizing his code?
- These are probably not different applications inside of each cluster.

Scatter Plot (2)



- Peterbr is in cache
- f97-mal is not

Additional Plots

- Scatter plots do not:
  - Easily display patterns with many data points
  - Scale to multiple dimensions (metrics)
  - Immediately expose relations made by distinct clusters.

Kohonen Self Organizing Maps

- Each set of metrics (sample) can be seen as a vector in N-dimensional space.
- A KSOM is a grid of cells each representing a vector of equal dimension.
- Entries in the map can be considered attractors.
  - Randomly initialized.
  - The cell representing the closest point (Euclidean) in adjusted to be closer to the sample.
  - Neighbors also trained although to a lesser degree.

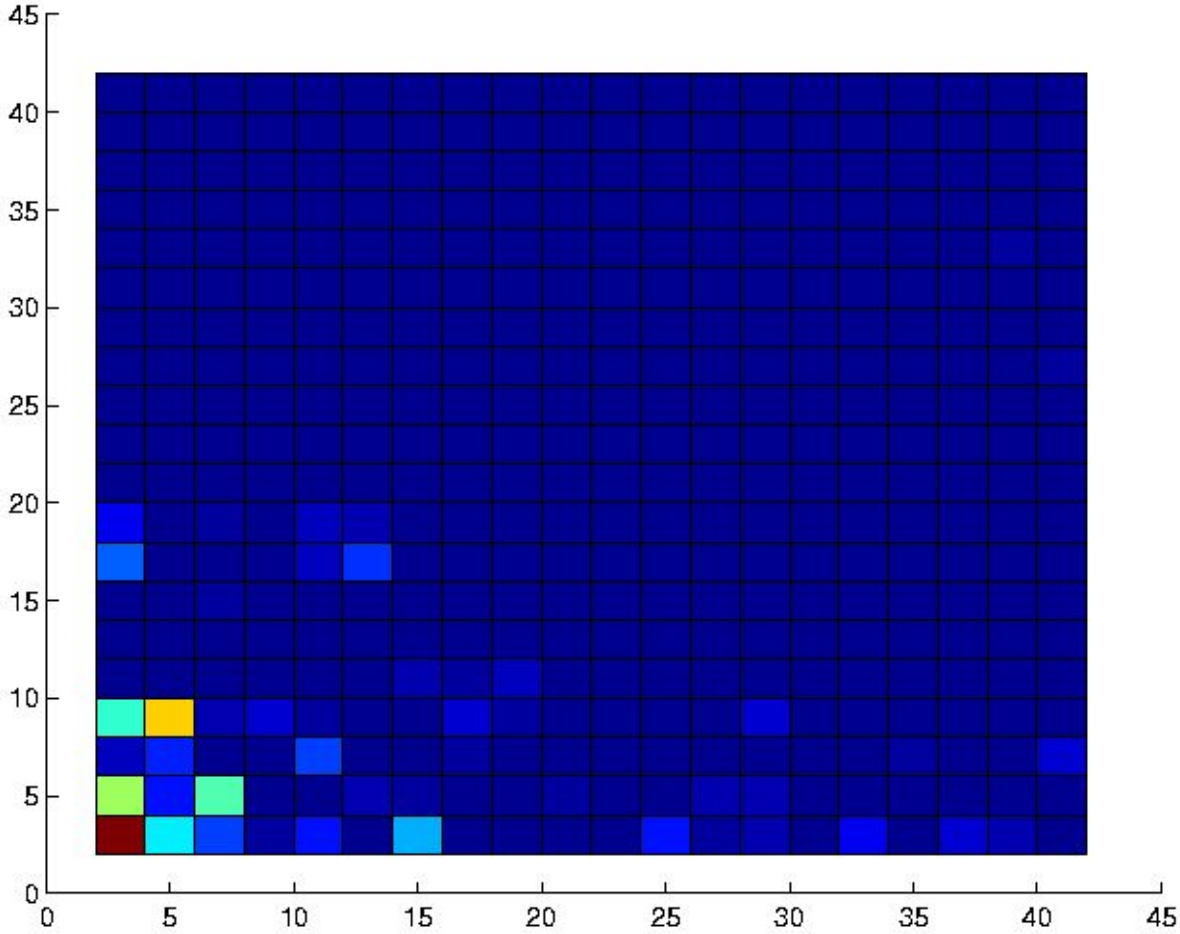
Kohonen Self Organizing Maps (2)

- The map is trained for multiple generations.
  - A generation is just a pass through the data set.
- When finished, the map is colored per cell in some “meaningful” way. i.e.
  - Number of samples that map to the cell. (histogram)
  - Total distance.
- KSOMs can be sliced, where each component of each vector are visualized separately.

Kohonen Self Organizing Maps (3)

- The goal is to classify entities (users, processes, threads, etc) across this multidimensional space.
  - Identify distinct patterns that could lead to specialization of:
    - Architecture
    - Code
    - Resources
  - Ultimately a greater understanding of the dependent measurements (components) present in the workload.

KSOM Sample



Links

- http://icl.cs.utk.edu/~mucci/mucci_talks.html
- Software:
  - http://icl.cs.utk.edu/~mucci/monitor
  - http://icl.cs.utk.edu/~mucci/papiex
  - http://icl.cs.utk.edu/papi
- Sample web database:
  - http://amd4w.pdc.kth.se/web

Questions: mucci at cs.utk.edu & dah at pdc.kth.se