# Performance Analysis and HPC

Philip J. Mucci
Visiting Scientist
mucci@pdc.kth.se

University of Tennessee, Knoxville
mucci@cs.utk.edu

August 18th, 2004
High Performance Computing
Summer School at PDC

# Outline

- Why is Performance Analysis important?
- Types of Performance Analysis
- Different types of Tools
- Hardware Performance Analysis
- 2 Example Tools to try on Lucidor with this afternoons exercises

# Performance Evaluation

- Traditionally, performance evaluation has been somewhat of an art form:
  - Limited set of tools (time & -p/-pg)
  - Major differences between systems
  - Lots of guesswork as to what was 'behind the numbers'
- Today, the situation is different.
  - Hardware support for performance analysis
  - A wide variety of Open Source tools to choose from.

## Why Performance Analysis?

- 2 reasons: Economic & Qualitative
- Economic: TIME IS MONEY
  - Average lifetime of these large machines is 4 years before being decommissioned.
  - Consider the cost per day of a 4 Million Dollar machine, with annual maintenance/electricity cost of $300,000 (US). That's $1500.00 (US) per hour of compute time.
  - Many HPC centers charge departments by the CPU hour

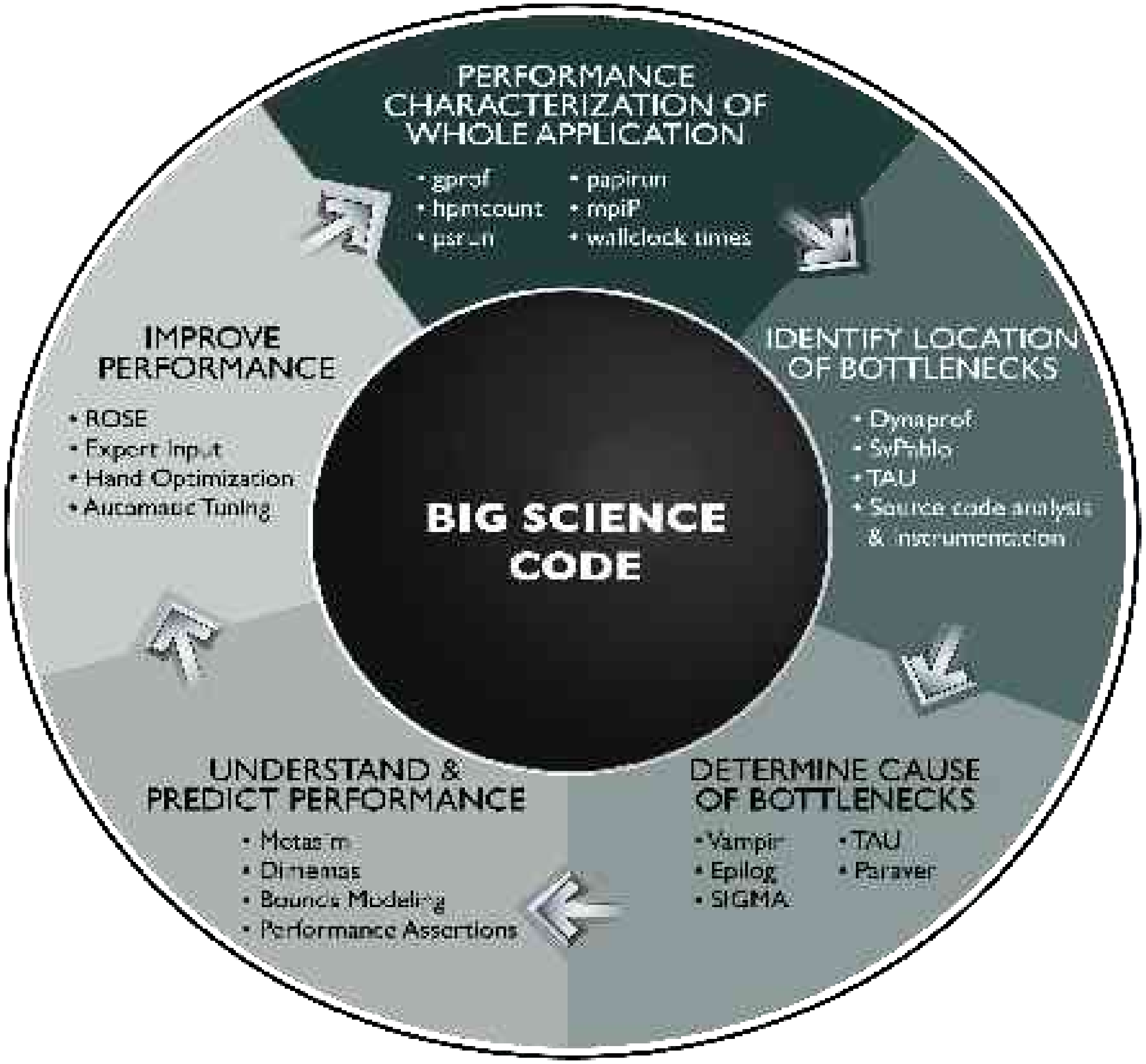
# Why Performance Analysis 2?

- 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!

## Why Performance Analysis 3?

- So, we must strive to evaluate how our code is running.
- Learn to think of performance during the entire cycle of your design and implementation.
- Systems will be in place at PDC to recognize a 'slow' code that is consuming large amounts of CPU time.

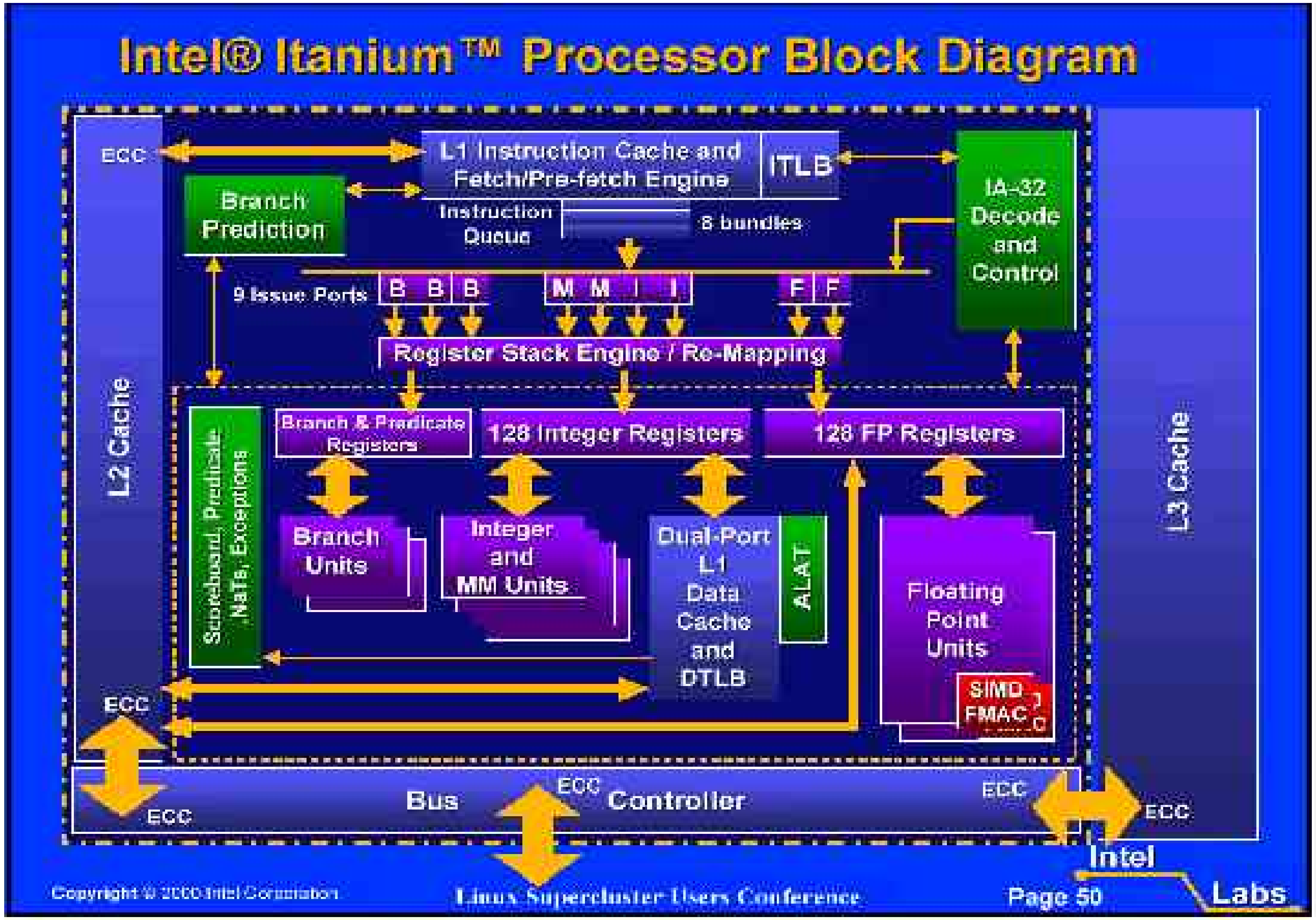
# The Performance Lifecycle



## Rising Processor Complexity

- No longer can we easily trace the execution of a segment of code.
  - Static/Dynamic Branch Prediction
  - Prefetching
  - Out-of-order scheduling
  - Predication
- So, just a measure of 'wallclock' time is not enough.
- Need to know what's really happening under the hood.

# Processor Complexity


Intel® Itanium™ Processor Block Diagram
ECC
L1 Instruction Cache and Fetch/Pre-fetch Engine
ITLB
Branch Prediction
Instruction Queue
8 bundles
IA-32 Decode and Control
9 Issue Ports
B B B
M M I I
F F
Register Stack Engine / Re-Mapping
L2 Cache
Scoreboard, Predicate, NaTs, Exceptions
Branch & Predicate Registers
128 Integer Registers
128 FP Registers
Branch Units
Integer and MM Units
Dual-Port L1 Data Cache and DTLB
ALAT
Floating Point Units
SIMD FMAC
L3 Cache
ECC
Bus
ECC
Controller
ECC
ECC
ECC
Copyright © 2000 Intel Corporation
Linux Supercluster Users Conference
Page 50
Intel Labs

## Measurement Methods

- Direct methods requires explicit instrumentation in some form.
  - Tracing
    - Generate a record for each measured event.
    - Useful only when evidence of performance anomalies is present due to the large volume of data generated.
  - Aggregate
    - Reduce data at run-time avg/min/max measurements.
    - Useful for application and architecture characterization and optimization.

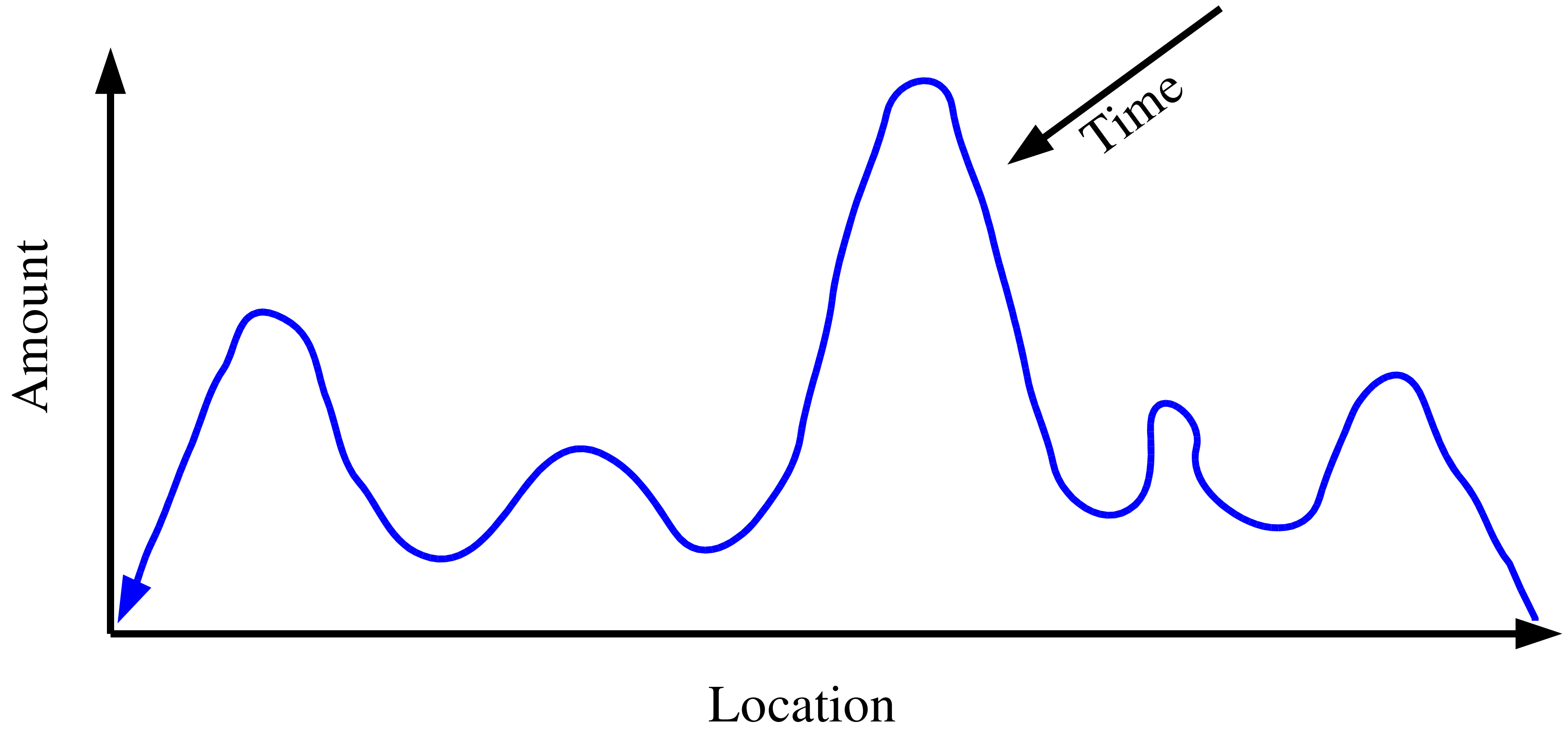
## Measurement Methods 2

- Indirect methods requires no instrumentation and can be used on unmodified applications.
- The reality is that the boundary between indirect and direct is somewhat fuzzy.
  - gprof (no source mods, but requires relink or recompile)

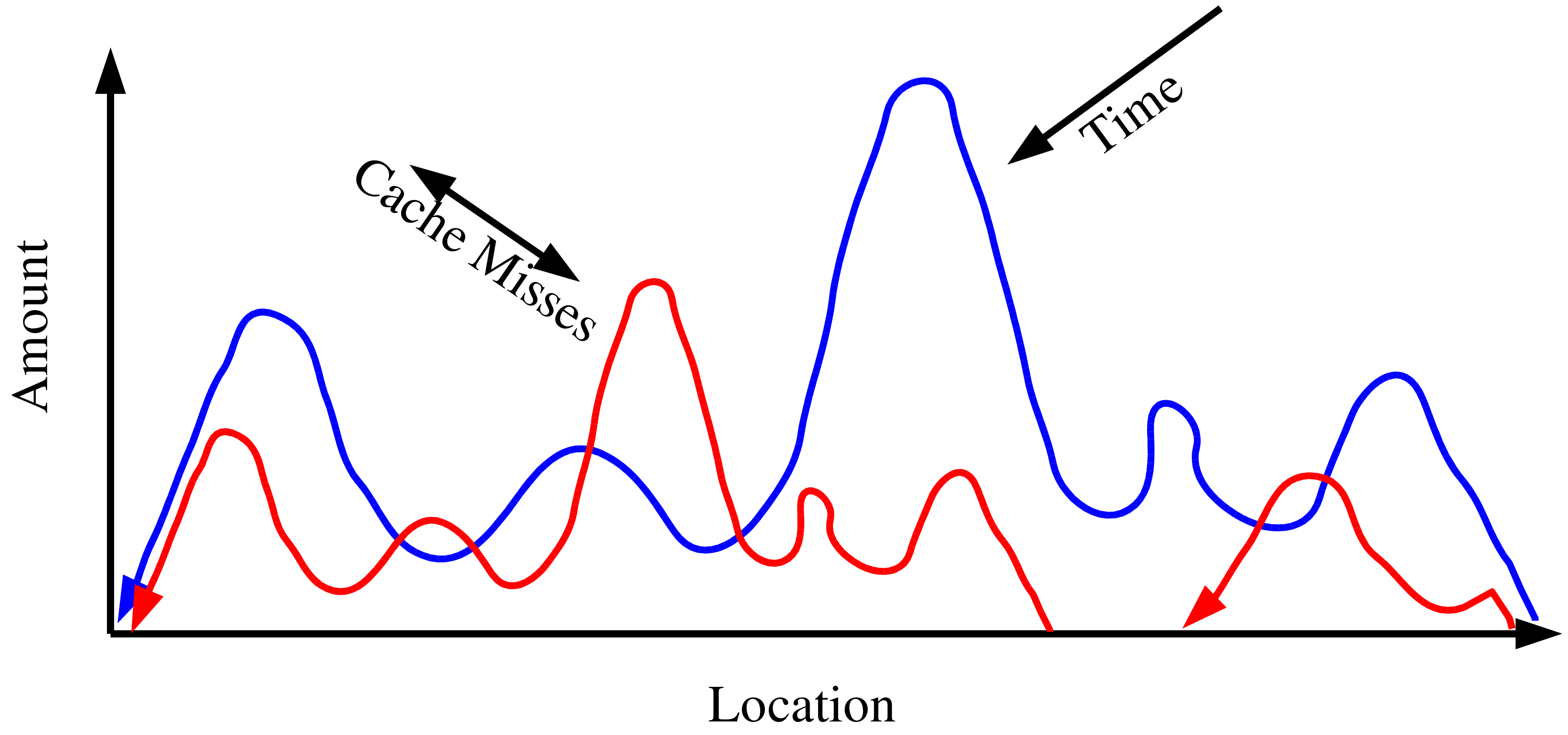
## Statistical Profiling

- Dr. Andersson introduced you to gprof.
- At a defined interval (interrupts), record WHERE in the program the CPU is.
- Data gathered represents a probabilistic distribution in the form of a histogram.
- Interrupts can be based on time or hardware counter events with the proper infrastructure like...

# Statistical Profiling



# Hardware Statistical Profiling



## 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
- Each register has an associated control register that tells it what to count and how to do it.
  - Interrupt on overflow
  - Edge detection (cycles vs. events)
  - User vs. kernel mode

# Performance Counters

- Most high performance processors include hardware performance counters.
  - AMD Athlon and Opteron
  - Compaq Alpha EV Series
  - CRAY T3E,  X1
  - IBM Power Series
  - Intel Itanium, Itanium 2, Pentium
  - SGI MIPS R1xK Series
  - Sun UltraSparc II+
  - 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

## 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.

PARALLEL TOOLS CONSORTIUM

## PAPI Preset Events

- PAPI supports around preset events
- Proposed set of events deemed most relevant for application performance tuning
- Preset events are mappings from symbolic names to machine specific definitions for a particular hardware resource.
  - Total Cycles is PAPI_TOT_CYC
- Mapped to native events on a given platform
- PAPI also supports presets that may be derived from the underlying hardware metrics

## Linux Performance Infrastructure

- Contrary to popular belief, the Linux infrastructure is well established.
- PAPI is +7 years old.
- Wide complement of tools from which to choose.
- Some are production quality.
- Sun, IBM and HP are now focusing on Linux/HPC which means a focus on performance.

## Parallel Performance

*"The single most important impediment to good parallel performance is still poor single-node performance."*

*- William Gropp*

*Argonne National Lab*

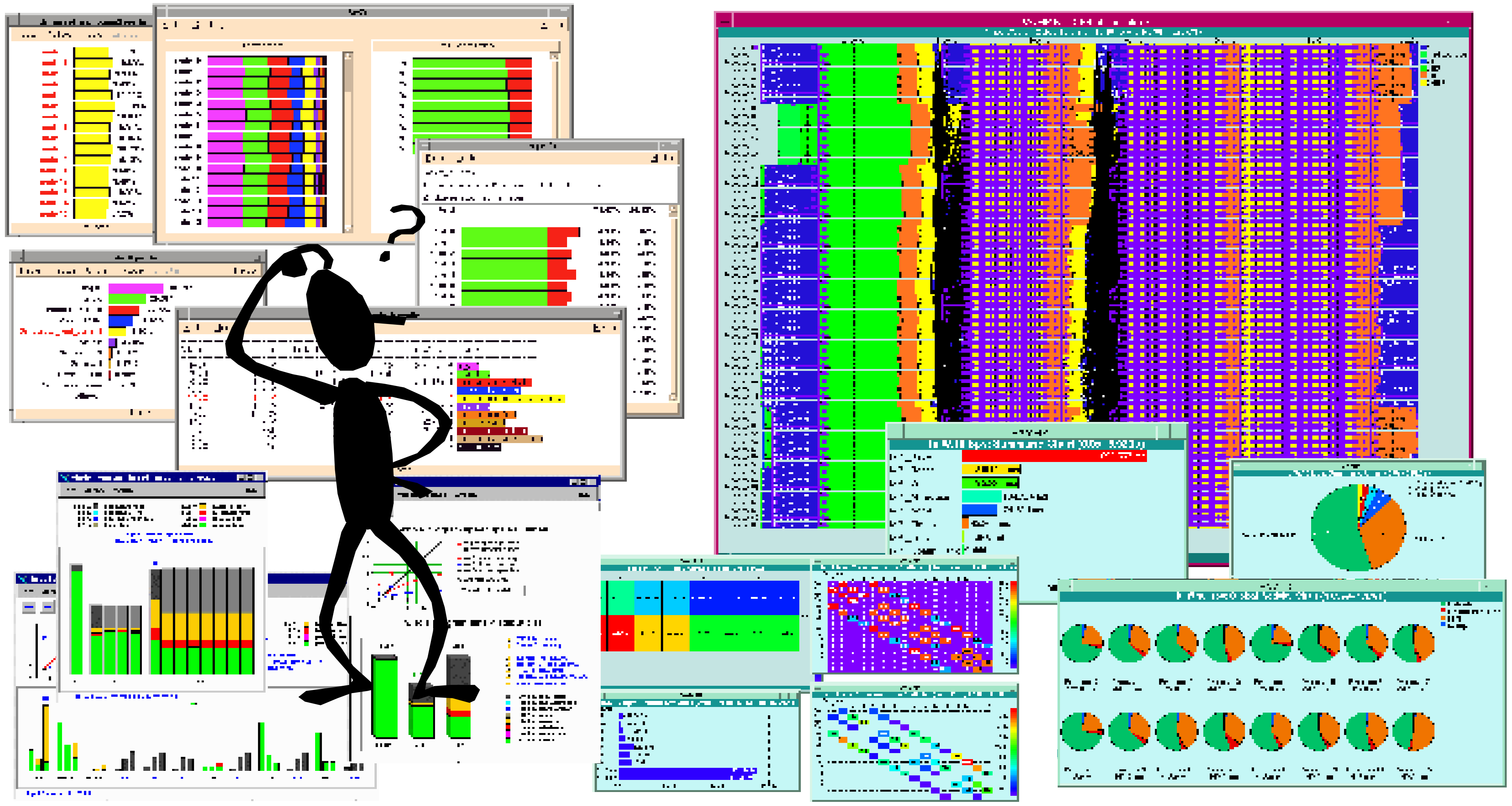
## What is Good Parallel Performance?

- Single CPU performance is high.
- The code is scalable out to more than a few nodes.
- The network is not the bottleneck.
- In parallel computation, algorithm design is the key to good performance.
- You must reduce the amount of data that needs to be sent around.

## Beware The Fallacy Linear Scalability

- But what about per/PE performance?
- With a slow code, overall performance of the code is not vulnerable to other system parameters like communication bandwidth, latency.
- Very common on tightly integrated systems where you can simple add PE's for performance.

# Which Tool?



## The Right Performance Tool

- You must have the right tool for the job.
- What are your needs? Things to consider:
  - User Interface
    - Complex Suite
    - Quick and Dirty
  - Data Collection Mechanism
    - Aggregate
    - Trace based
    - Statistical

# The Right Performance Tool 2

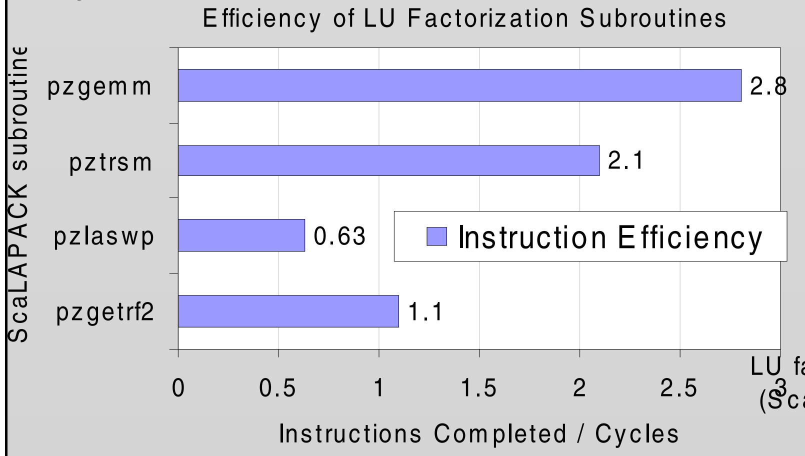
- Performance Data
  - Communications (MPI)
  - Synchronization (Threads and OpenMP)
  - External Libraries
  - User code
- Data correlation
  - Task Parallel (MPI)
  - Thread Parallel
- Instrumentation Mechanism
  - Source
  - Binary (DPCL/DynInst)
  - Library interposition

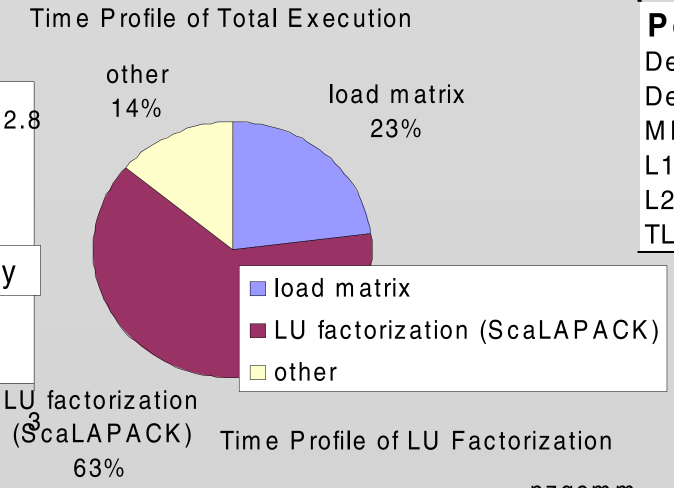
## The Right Performance Tool 3

- Data Management
  - Performance Database
  - User (Flat file)
- Data Visualization
  - Run Time
  - Post Mortem
  - Serial/Parallel Display
  - ASCII

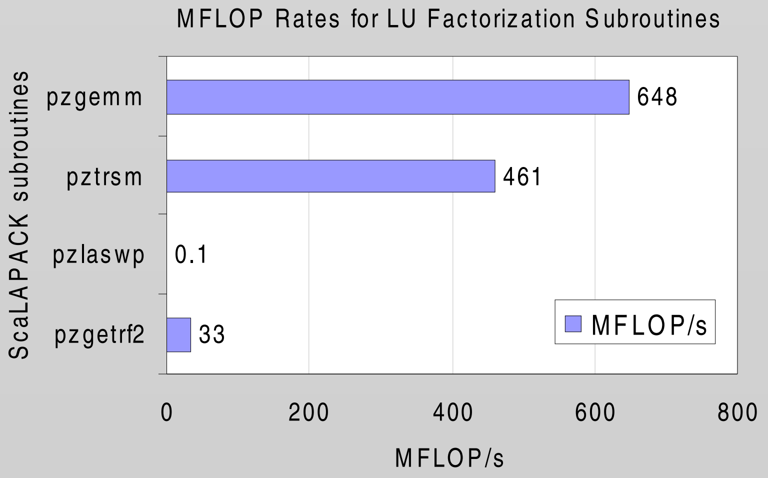
# Fusion Sciences
# AORSA3D

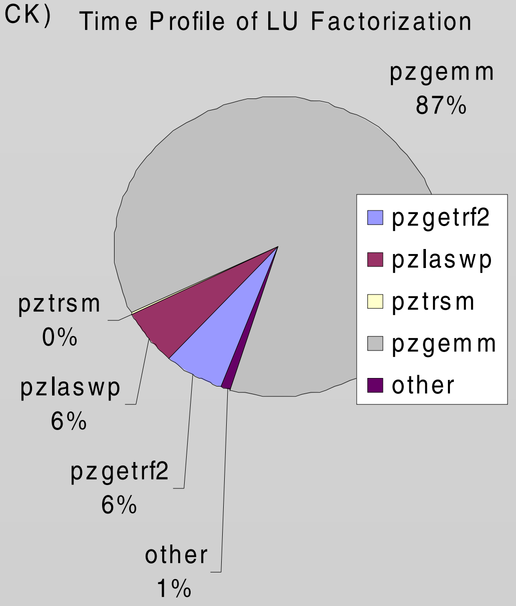
AORSA3D was ported and benchmarked on IBM and Compaq platforms. A detailed performance analysis has begun using SvPablo and PAPI. The results below are for a 400 Fourier mode run on 16 processors and 1 node of an IBM SP (Nighthawk II / 375MHz).

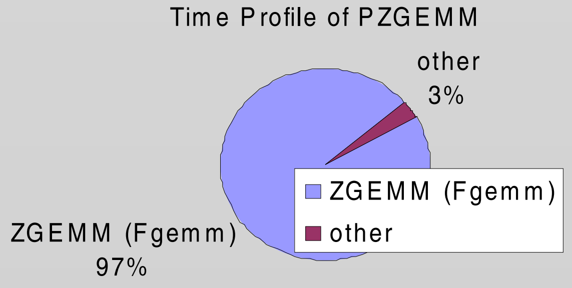




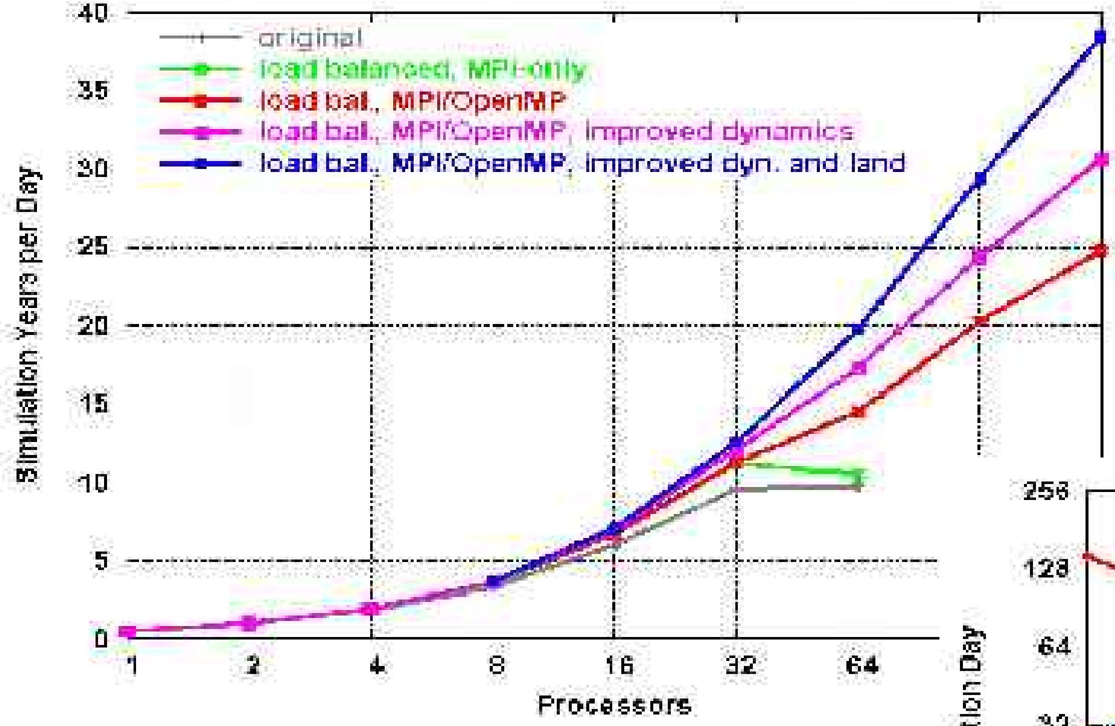
| **Performance for ZGemm** | |
|---|---|
| Denisty of Mem Access | 1 |
| Denisty of FLOPs | 1.8 |
| MFLOP/s | 664 |
| L1 cache hit rate | 0.98 |
| L2 cache hit rate | 0.96 |
| TLB misses | 285034 |





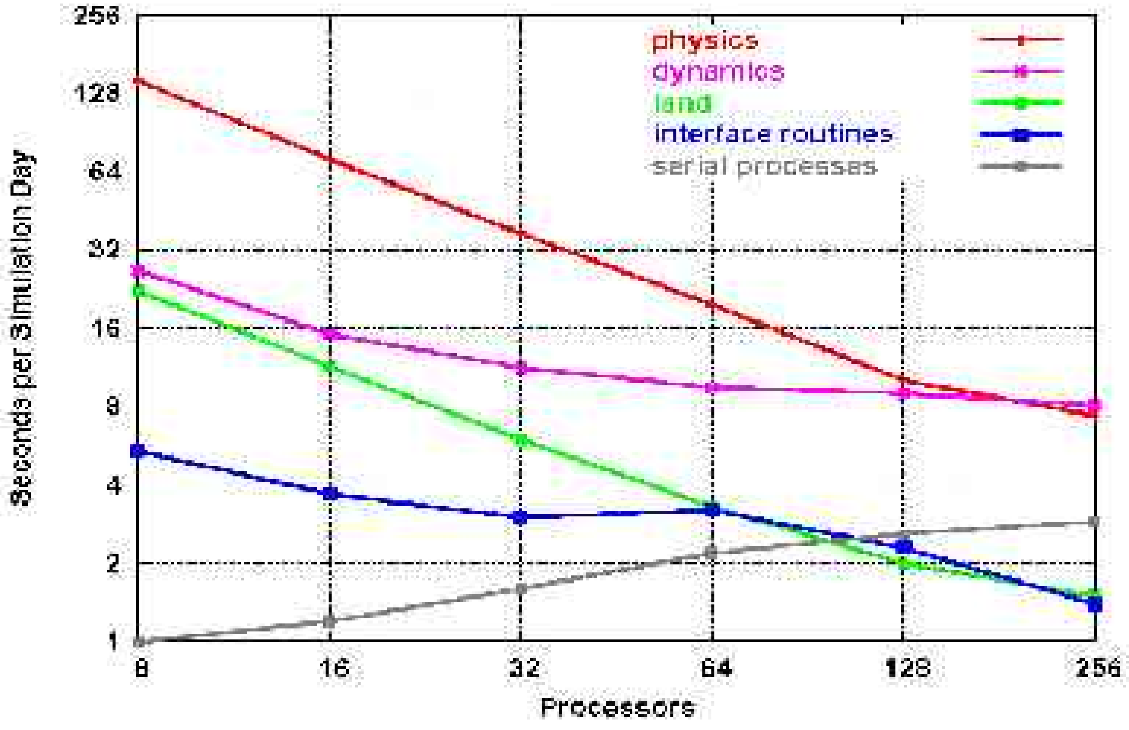


# Biology and Environmental Sciences
# CAM



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.



## Hardware Profiling and papiex

- A simple tool that generates performance measurements for the entire run of a code.
- Requires no recompilation.
- Monitors all subprocesses/threads.
- Output goes to stderr or a file.
- Try running your code under papiex to measure IPC or MFLOPS (the default).

## Papiex v0.9 Example

```
> module load perftools/1.1

> papiex <application>

> papiex -e PAPI_TOT_CYC -e PAPI_TOT_INS --
  <application>

> mpirun -np 4 `which papiex` -f --
  <application>
```

## papiex v0.9 Output

```
--- papiex 0.9  hardware counter report ---.
Executable:             /afs/pdc.kth.se/home/m/mucci/mpiP-2.7/testing/a.out
Parent Process ID:      18115
Process ID:             18116
Hostname:               h05n05.pdc.kth.se
Start:                  Tue Aug 17 17:45:36 2004
Finish:                 Tue Aug 17 17:45:40 2004
Domain:                 User
Real usecs:             3678252 (3s.)
Real cycles:            3310413694
Proc usecs:             16592 (0s.)
Proc cycles:            14932800
PAPI_TOT_CYC:           13962873
PAPI_FP_INS:            285847

Event descriptions:
Event: PAPI_TOT_CYC
        Derived: No
        Short: Total cycles
        Long: Total cycles
        Vendor Symbol: CPU_CYCLES
        Vendor Long: CPU_CYCLES
Event: PAPI_FP_INS
        Derived: No
        Short: FP instructions
        Long: Floating point instructions
        Vendor Symbol: FP_OPS_RETIRED
        Vendor Long: FP_OPS_RETIRED
```

# Papiex v0.9 Usage

```
Usage: papiex [-lLihvtmnukord] [-f [prefix]] [-e event]... -- <cmd> <cmd
   options>

 -l                List the available events.

 -L                List all information about the available events.

 -i                Print information about the host machine.

 -h                Print this message.

 -v                Print version information.

 -t                Enable monitoring of multiple threads.

 -m                Enable multiplexing of hardware counters.

 -n                Do not follow fork()'s.

 -u                Monitor user mode events. (default)

 -k                Monitor kernel mode events.

 -o                Monitor transient mode events.

 -r                Report getrusage() information. (no children included)

 -d                Enable debugging output.

 -f[prefix]        Output to <prefix><cmd>.papiex.<host>.<pid>.<tid>.

 -e event          Monitor this hardware event.
```

## Parallel Profiling

- Often we want to see how much time we are spending communicating.
- Many tools to do this via “Tracing” the MPI calls.
- A very good and simple tool available on Lucidor is mpiP v2.7, it does online trace reduction.

# MpiP v2.7 Example

```
> module load perftools/1.1
> module show perftools
```

- Follow the instructions to link your C/C++/F77/F90 codes with mpiP.
- Run your code and examine the output in <*.mpiP>.

# MpiP v2.7 Output

```
@ Command : /afs/pdc.kth.se/home/m/mucci/mpiP-2.7/testing/./sweep-ops-stack.exe
/tmp/SPnodes-mucci-0
@ Version                  : 2.7
@ MPIP Build date          : Aug 17 2004, 17:04:36
@ Start time               : 2004 08 17 17:08:48
@ Stop time                : 2004 08 17 17:08:48
@ MPIP env var             : [null]
@ Collector Rank           : 0
@ Collector PID            : 17412
@ Final Output Dir         : .
@ MPI Task Assignment      : 0 h05n05-e.pdc.kth.se
@ MPI Task Assignment      : 1 h05n35-e.pdc.kth.se
@ MPI Task Assignment      : 2 h05n05-e.pdc.kth.se
@ MPI Task Assignment      : 3 h05n35-e.pdc.kth.se

@--- MPI Time (seconds) ---------------------------------------------------
---------------------------------------------------------------------------
Task    AppTime    MPITime     MPI%
   0      0.084     0.0523    62.21
   1     0.0481      0.015    31.19
   2      0.087     0.0567    65.20
   3     0.0495     0.0149    29.98
   *      0.269      0.139    51.69

@--- Aggregate Time (top twenty, descending, milliseconds) ----------------
---------------------------------------------------------------------------
Call                 Site       Time    App%    MPI%
Barrier                 1        112   41.57   80.42
Recv                    1       26.2    9.76   18.89
Allreduce               1      0.634    0.24    0.46
Bcast                   1        0.3    0.11    0.22
Send                    1      0.033    0.01    0.02

@--- Aggregate Sent Message Size (top twenty, descending, bytes) ----------
---------------------------------------------------------------------------
Call                 Site      Count      Total       Avrg  Sent%
Allreduce               1          8    4.8e+03        600  46.15
Bcast                   1          8    4.8e+03        600  46.15
Send                    1          2        800        400   7.69
```

# MpiP v2.7 Output 2

```
---------------------------------------------------------------------------
@--- Callsite Time statistics (all, milliseconds): 16 --------------------
---------------------------------------------------------------------------
Name              Site Rank  Count      Max      Mean       Min   App%   MPI%
Allreduce            1    0      2    0.105     0.087     0.069   0.21   0.33
Allreduce            1    1      2    0.118      0.08     0.042   0.33   1.07
Allreduce            1    2      2     0.11     0.078     0.046   0.18   0.27
Allreduce            1    3      2    0.102     0.072     0.042   0.29   0.97
Barrier              1    0      3     51.9      17.3     0.015  61.86  99.44
Barrier              1    1      3    0.073    0.0457     0.016   0.29   0.91
Barrier              1    2      3     54.9      18.8     0.031  64.90  99.53
Barrier              1    3      3     1.56      1.02     0.035   6.20  20.68
Bcast                1    0      2    0.073    0.0535     0.034   0.13   0.20
Bcast                1    1      2    0.037     0.023     0.009   0.10   0.31
Bcast                1    2      2    0.084     0.046     0.008   0.11   0.16
Bcast                1    3      2     0.03    0.0275     0.025   0.11   0.37
Recv                 1    1      1     14.6      14.6      14.6  30.48  97.71
Recv                 1    3      1     11.6      11.6      11.6  23.37  77.98
Send                 1    0      1    0.013     0.013     0.013   0.02   0.02
Send                 1    2      1     0.02      0.02      0.02   0.02   0.04
Send                 1    *     32     54.9      4.34     0.008  51.69 100.00
---------------------------------------------------------------------------
@--- Callsite Message Sent statistics (all, sent bytes) ------------------
---------------------------------------------------------------------------
Name              Site Rank   Count       Max      Mean       Min       Sum
Allreduce            1    0       2       800       600       400      1200
Allreduce            1    1       2       800       600       400      1200
Allreduce            1    2       2       800       600       400      1200
Allreduce            1    3       2       800       600       400      1200
Bcast                1    0       2       800       600       400      1200
Bcast                1    1       2       800       600       400      1200
Bcast                1    2       2       800       600       400      1200
Bcast                1    3       2       800       600       400      1200
Send                 1    0       1       400       400       400       400
Send                 1    2       1       400       400       400       400
Send                 1    *      18       800     577.8       400  1.04e+04
---------------------------------------------------------------------------
@--- End of Report --------------------------------------------------------
```

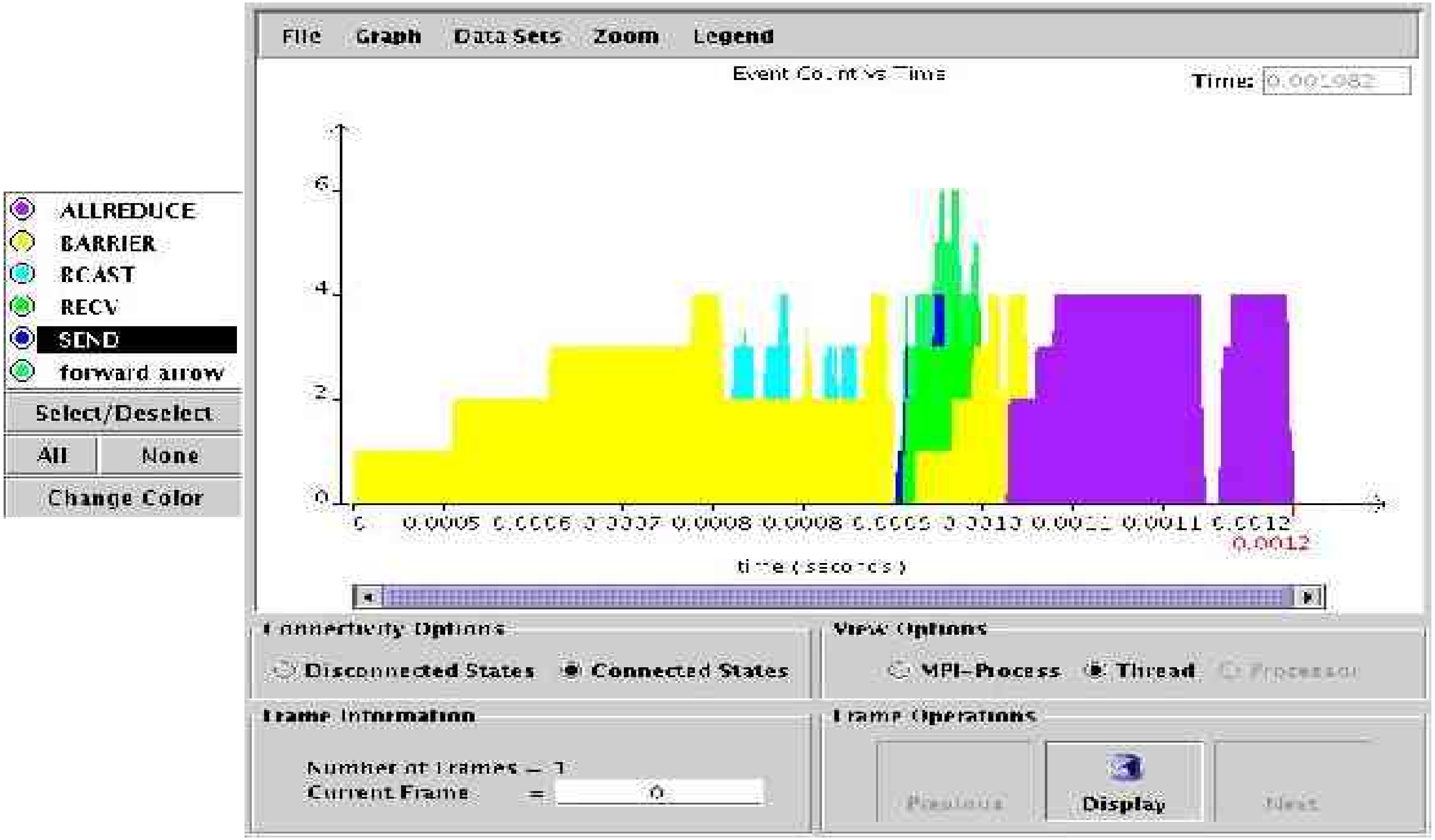
## MPI Tracing and Jumpshot

- Sometimes we need to see the exact sequence of messages exchanged between processes.
- For this, we can enable MPI tracing by relinking our application and using the Jumpshot tool.
- Use the -mpilog option to the mpi compile line.
- After running, you will see a <app>.clog
- Next, translate the logfile to a scalable format with slog.

## Jumpshot 3 Example

```
> module load perftools/1.1

> mpicc -mpilog example.c -o example

> mpirun -np 4 example

> clog2slog example.clog

> jumpshot example.slog
```

# Jumpshot Main Window



Time: 0.001982

Connectivity Options

( ) Disconnected States [✓ Connected States]

View Options

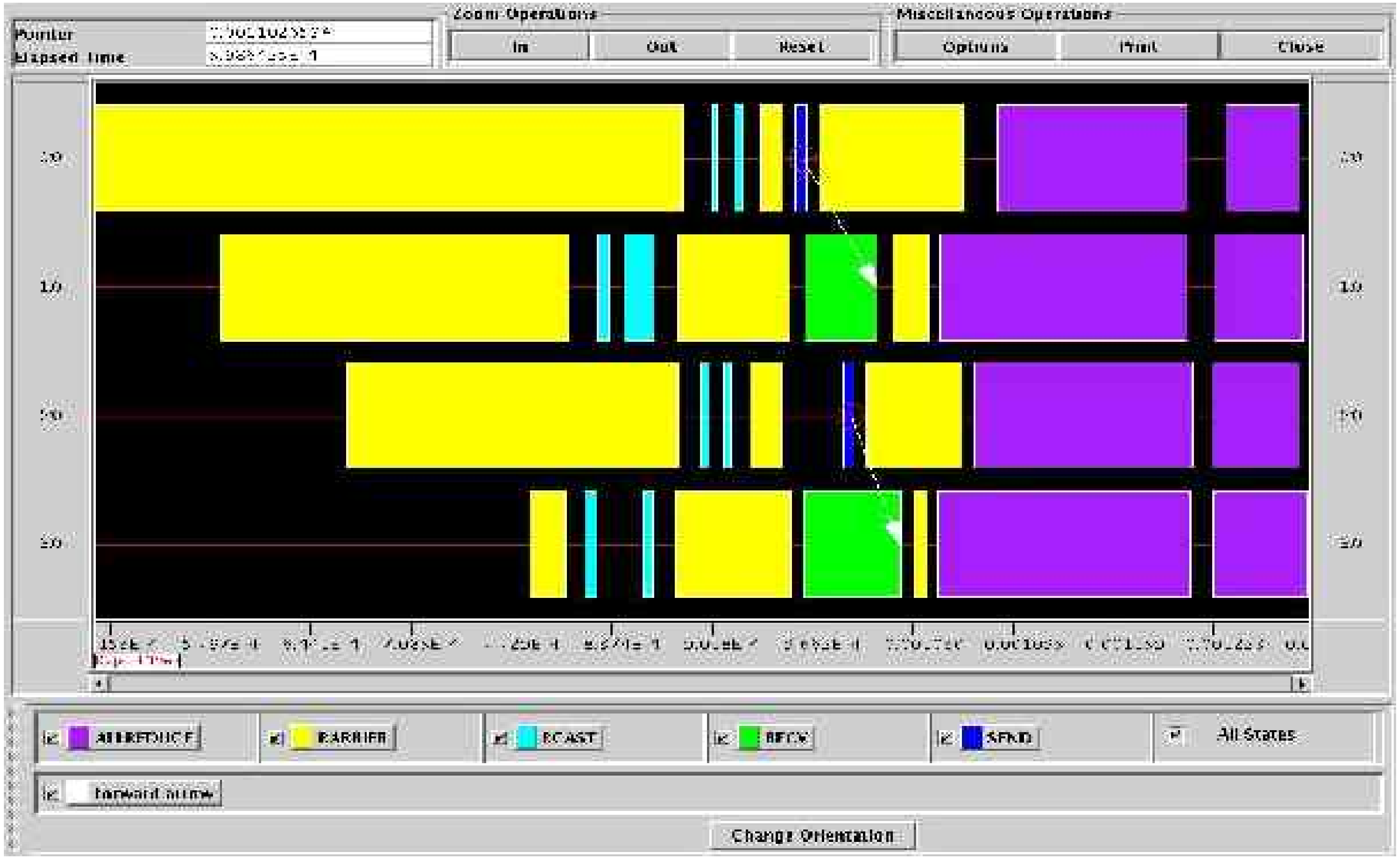
( ) MPI-Process [✓ Thread] ( ) Processor

Frame Information

Number of Frames = 1
Current Frame = 0

Frame Operations

# Jumpshot Timeline



## 5 Ways to Avoid Performance Problems: Number 1

*Never, ever, write your own code unless you absolutely have to.*

- Libraries, libraries, libraries!
- Spend time to do the research, chances are you will find a package that suits your needs.
- Often you just need to do the glue that puts the application together.
- The 90/10 Rule! 90% of time is spent in 10% of code.

## 5 Ways to Avoid Performance Problems: Number 2

*Never violate the usage model of your environment.*

- If something seems impossible to accomplish in your language or programming environment, you're probably doing something wrong.
- Consider such anomalies as:
  - Matlab in parallel on a cluster of machines.
  - High performance Java.
- There probably is a better way to do it, ask around.

## 5 Ways to Avoid Performance Problems: Number 3

*Always let the compiler do the work.*

- The compiler is much better at optimizing most code than you are.
- Gains of 30-50% are reasonable common when the 'right' flags are thrown.
- Spend some time to read the manual and ask around.

## 5 Ways to Avoid Performance Problems: 4

*Never use more data than absolutely necessary.*

- C: float vs. double.
- Fortran: REAL*4, REAL*8, REAL*16
- Only use 64-bit precision if you NEED it.
- A reduction in the amount of data the CPU needs ALWAYS translates to a increase in performance.
- Always keep in mind that the memory subsystem and the network are the ultimate bottlenecks.

# 5 Ways to Avoid Performance Problems: 5

*Make friends with Computer Scientists*

- Learning even a little about modern computer architectures will result in much better code.
- 2 Challenges: Learn why the following statements are almost always horribly slow on modern CPUs when placed inside loops where Index is the loop variable.
- 1) Var = Var + DataArray[IndexArray[Index]]
- 2) IF (VAR2 .EQ. I) THEN
- 2) DOESN'T REALLY MATTER
- 2) ENDIF

# Some Performance Tools

- TAU (U. Oregon)
  - Source/dynamic instrumentation and tracing system
  - http://www.cs.uoregon.edu/research/paracomp/tau/
- HPCToolkit (Rice U.)
  - Command line statistical profiling (including shlibs)
  - http://hipersoft.cs.rice.edu/hpctoolkit/
- PerfSuite and PSRUN (NCSA)
  - Command line aggregate and statistical profiling
  - http://perfsuite.ncsa.uiuc.edu

## More Performance Tools

- KOJAK (Juelich, UTK)
  - Instrumentation, tracing and analysis system for MPI, OpenMP and Performance Counters.
  - http://www.fz-juelich.de/zam/kojak/
- SvPablo (UIUC)
  - Instrumentation system for Performance Counters
  - http://www-pablo.cs.uiuc.edu/Project/SVPablo
- Q-Tools (HP) (non-PAPI)
  - Statistical profiling of system and user processes
  - http://www.hpl.hp.com/research/linux/q-tools

## More Performance Tools

- PapiEx: PAPI Execute
  - Passive aggregate counter measurement tool.
  - http://www.cs.utk.edu/~mucci/papiex
- DynaProf (P. Mucci, U Tenn)
  - Dynamic instrumentation tool.
  - http://www.cs.utk.edu/~mucci/dynaprof

# Questions?

- This talk:
  - http://www.cs.utk.edu/~mucci/latest/mucci_talks.html
- PAPI Homepage:
  - http://icl.cs.utk.edu/papi
- How to reach me:
  - mucci@pdc.kth.se
- For those here at KTH, many on the PDC staff are well versed in the art of performance. Use them!