# Performance Analysis of DOE Workloads in HPC and Cloud Environments

Karl Fuerlinger
Nick Wright
David Skinner

dskinner@cal.berkeley.edu

# Outline

- IPM Overview and Philosophy
- IPM2 Design and Implementation
  - Event signatures and hashing
  - IPM2's modularized design
- Using IPM
  - Banners, logs, and reports
  - Example: profiling GTC at scale
- Efficiency and Scalability
  - Overheads, perturbation of IPM
- New Areas of Research
  - IPM in the cloud
  - Execution flow graphs and trace recovery
  - Workload analysis

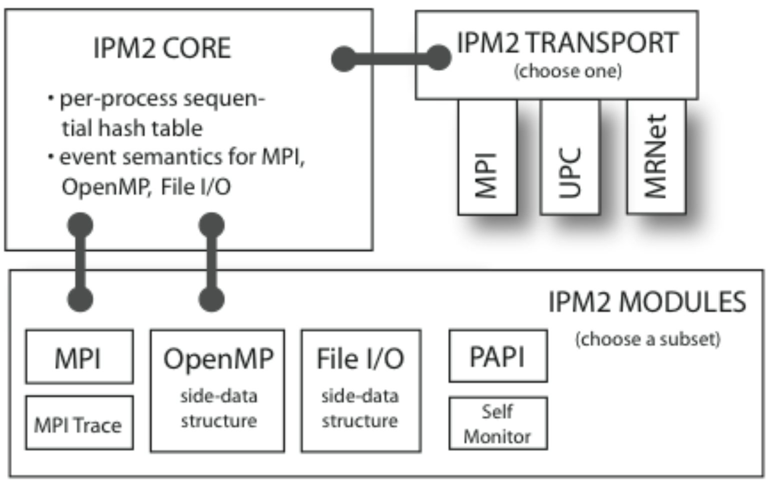
# IPM Overview

History

- IPM started as POE+ at NERSC
  - "How to profile apps from 400 projects asking for time?"
- IPM = **Integrated Performance Monitoring**
- Lightweight scalable profiling layer
- IPM2 is a from scratch re-write that will replace IPM this Fall

IPM2 is a re-architected implementation

- Same overall ideas w/ modularized design
- Extended monitoring capabilities and domains (beyond MPI)
- Autotools build / New parser / dynamic HTML charting

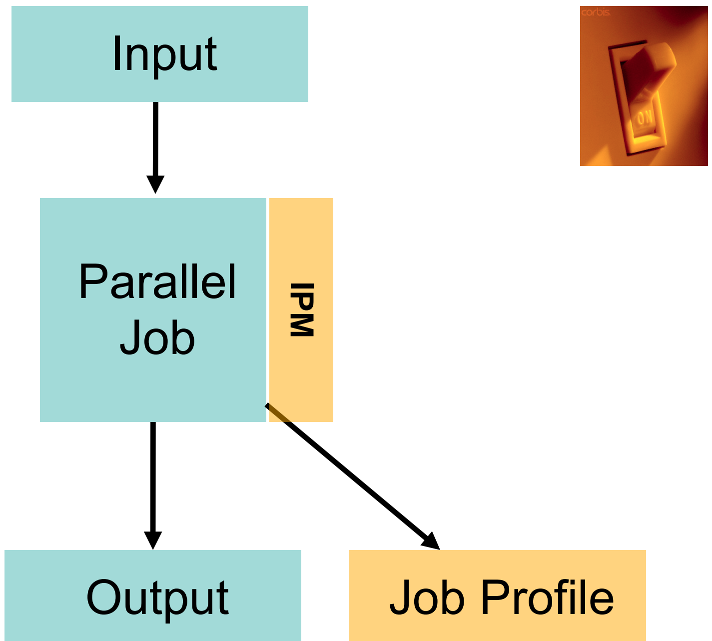


## What is IPM

- IPM is a thin measurement layer
  - Sitting between the application and the runtime/OS

- Goals
  - **Efficient** gathering of **high-level** performance metrics
  - Event inventories not traces
  - Determination of resource requirements and first order identification of performance problems
  - Less focus on drill-down into application
    - Currently no automatic function-level instrumentation
    - Manual region instrumentation supported

# IPM Philosophy




- •Banner on stderr
- •Detailed profiling log file
- •Profiling report (webpage)

- "Flip of a switch" monitoring
  - Resource consumption (used virtual memory, hw counter data)
  - Application execution event statistics
- Using /proc, other OS services, and PAPI for the resource consumption
- Efficient collection of event statistics in a hash table

## Events and Event Signatures

- Goal is to get an event inventory of the application
  - We map from events to signatures
  - Discard information we are not interested in


σ : E -> S
σ(e) … Signature of e

Event Signatures S

Usually not injective

```
…
MPI_Send(&buf, 2048, MPI_BYTE,
              33, 1,
MPI_COMM_WORLD)
....
```

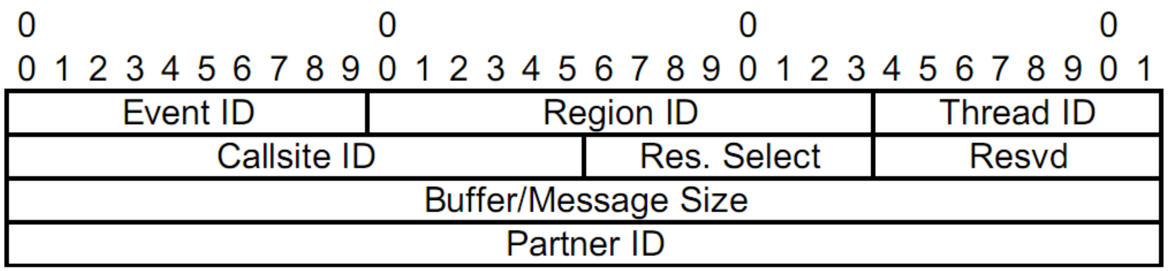
[MPI_SEND, 2048, 33, 21, 2, ...]

Comm. Partner
Call-Site ID
Region- or phase ID

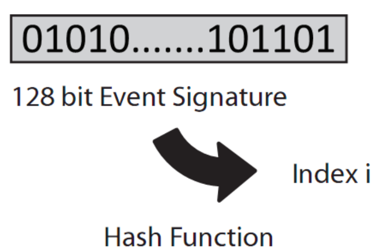
Signature vector

# IPM2 Signatures

- IPM2 uses 128 bit event signatures
  - 64 bit context key (where, what)
  - 64 bit resource key (buffer sizes, partners, …)




- The Signatures are keys into a hash table
- Table holds event statistics
  - Event count
  - Minimum duration
  - Maximum duration
  - Average duration

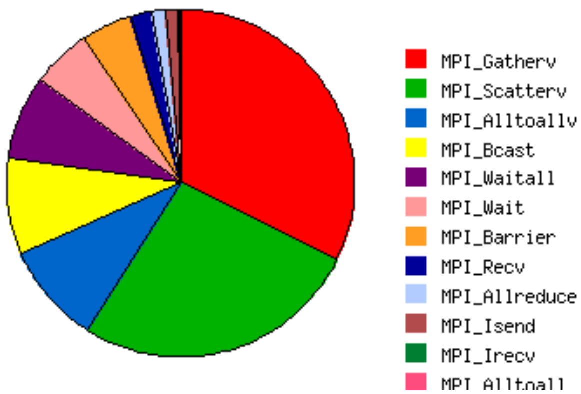



| Signature | #events, tmin, tmax, tavg |
|---|---|
| … | |
| 010…101 | 728, 3.20, 5.61, 4.41 |
| … | |

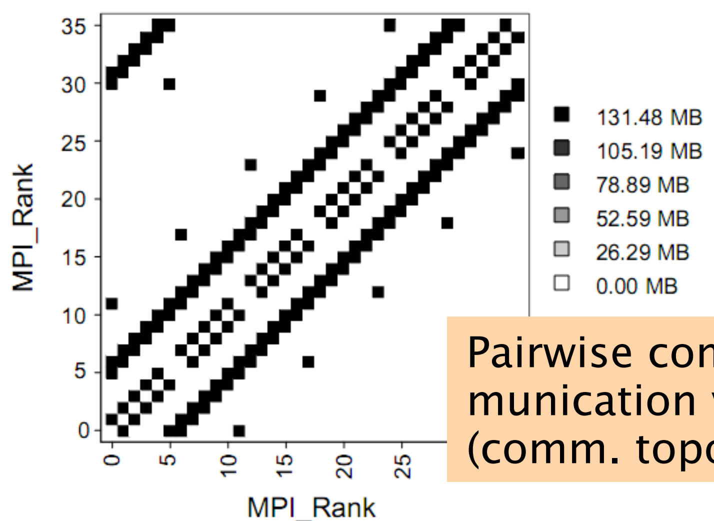
Performance Data Hash Table

# Analyzing the Event Signatures

- The hash table of event signatures contains a lot of interesting data

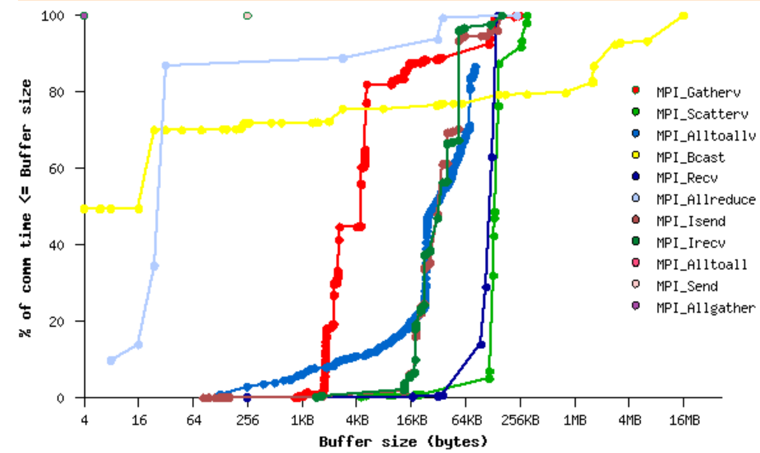



Communication time per type of MPI call



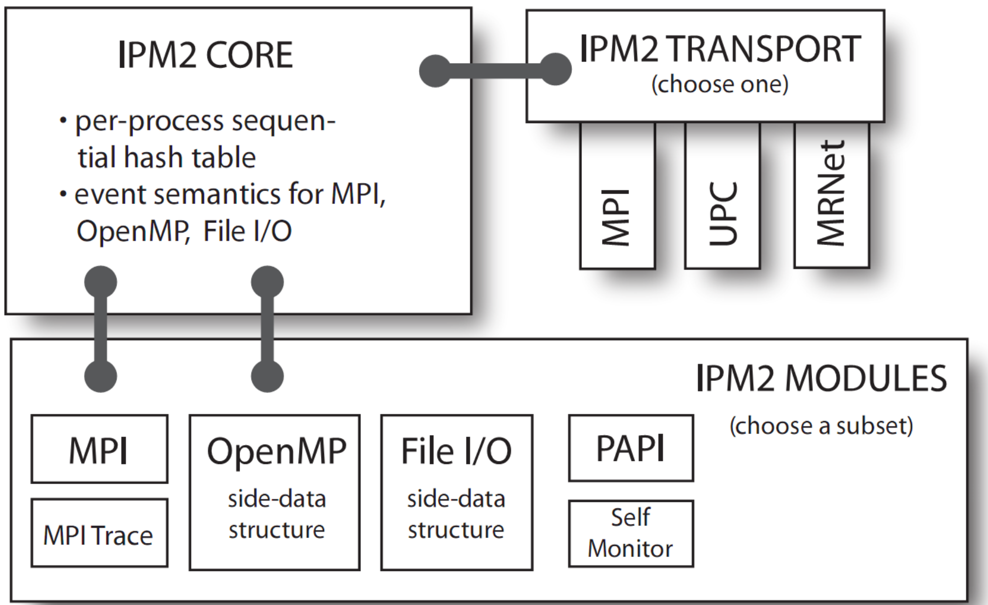

Pairwise com-munication volume (comm. topology)

CDF of time per MPI call over message sizes



# IPM2's Modularized Design

- IPM2 modules
  - **Core**, **Transport**, **Functional** modules
  - IPM2 modules are a compile-time adaptation and configuration mechanism, not runtime DLLs




- New OpenMP module
- New File-I/O module
- Thread-safe side data structures in OpenMP module, sequential hash table in core module
- Self-monitoring module reports on IPM2's time and resource consumption
- IPM2 can be built w/o any dependency on MPI, e.g., to monitor just File I/O or CUDA

## Outline

- IPM Overview and Philosophy
- IPM2 Design and Implementation
  - Event signatures and hashing
  - IPM2's modularized design
- Using IPM2
  - Banners, logs, and reports
  - Example: profiling GTC at scale
- IPM2 Efficiency and Scalability
  - Overheads, perturbation of IPM2
- New Areas of Research
  - IPM in the cloud
  - Execution flow graphs and trace recovery
  - Workload analysis

# Using IPM2

- Do "**module load ipm**", then run normally
  - Uses LD_PRELOAD
  - Re-linking required for static binaries
- Upon completion you get :

```
##IPM2############################################################
# command   : ./a.out
# start     : Sun Mar 14 16:55:39 2010   host      : nid01829
# stop      : Sun Mar 14 17:04:33 2010   wallclock : 533.12
# mpi_tasks : 2048 on 1024 nodes         %comm     : 29.41
# omp_thrds : 6                          %omp      : 50.63
# files     : 12                         %i/o      : 12.09
# mem [GB]  : 2774.44                    gflop/sec : 418.58
##################################################################
```

- Environment variables
  - IPM_HPM for PAPI counters
  - IPM_REPORT = **full** | **terse** | **none**
  - IPM_LOG = **full** | **terse** | **none**

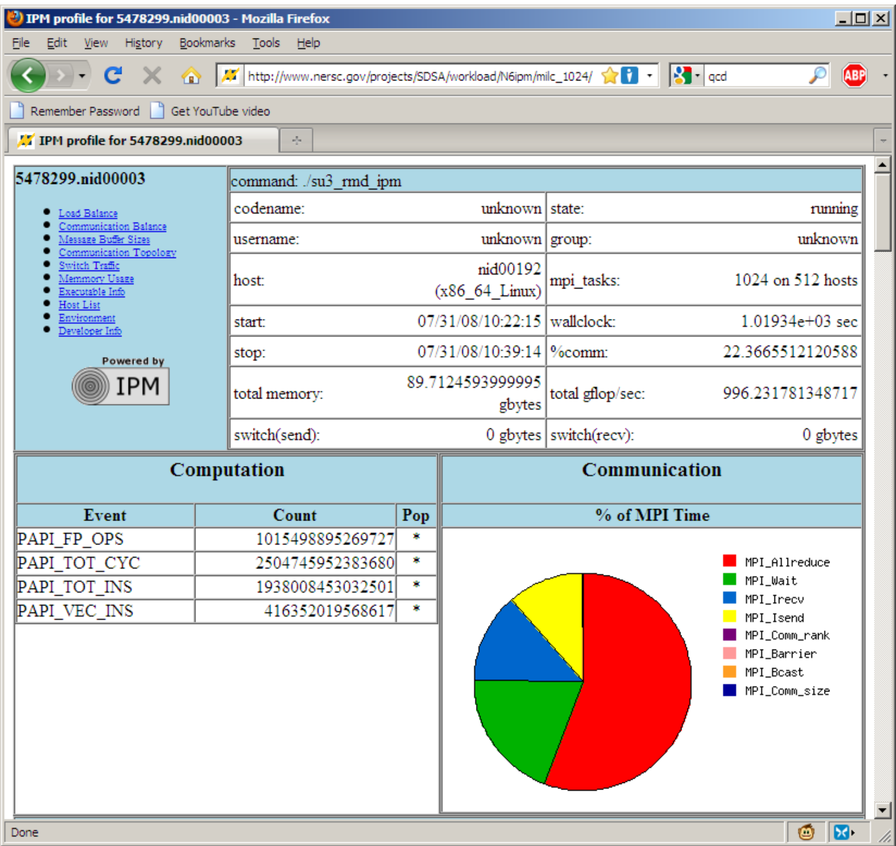
# More details with IPM_REPORT=full

```
##IPM2################################################################
# command   : ./a.out
# start     : Sun Mar 14 16:55:39 2010   host      : nid01829
# stop      : Sun Mar 14 17:04:33 2010   wallclock : 533.12
# mpi_tasks : 2048 on 1024 nodes         %comm     : 29.41
# omp_thrds : 6                          %omp      : 50.63
# files     : 12                         %i/o      : 12.09
# mem [GB]  : 2774.44                    gflop/sec : 418.58
#
#           :       [total]        <avg>         min          max
# wallclock :    1091671.57       533.04      532.99       533.12
# MPI       :     321034.43       156.76      109.03       239.23
# I/O       :     131947.08        64.43       11.83       113.87
# OMP       :     552665.28       269.86      205.07       305.36
# OMP idle  :      48262.98        23.57       21.30        27.40
# %wall     :
#   MPI     :                      29.41       20.45        44.88
#   OMP     :                      50.63       38.47        57.28
#   I/O     :                      12.09        2.22        21.36
# #calls    :
#   MPI     :      76235998        37224       37223        37320
# mem [GB]  :       2774.44         1.35        1.35         1.36
#
#                           [time]       [count]          <%wall>
# OMP_PARALLEL           552665.28     131439989           50.63
# MPI_Allreduce          247648.04      14438400           22.69
# fread                   69813.27       5488640            6.40
# ...
######################################################################
```

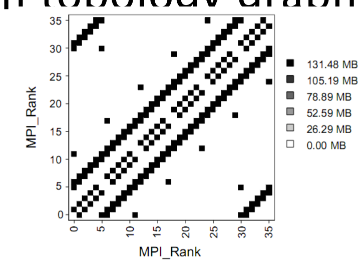
- Statistics of high level metrics across tasks
- Details of the contribution of individual events

# IPM HTML Profiling Report

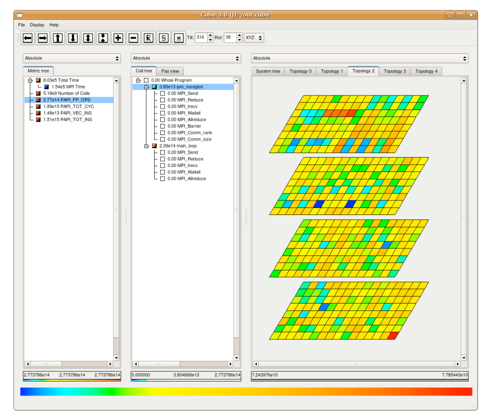
- ipm_parse generates HTML profiling report



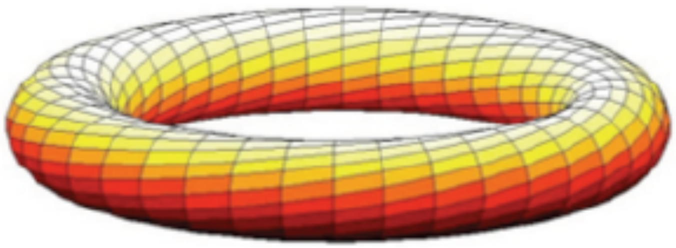

- Contents of the webpage:
  - Banner
  - Communication time breakdown
  - Load balance by task graph
  - Communication balance by task graph
  - Communication topology graph

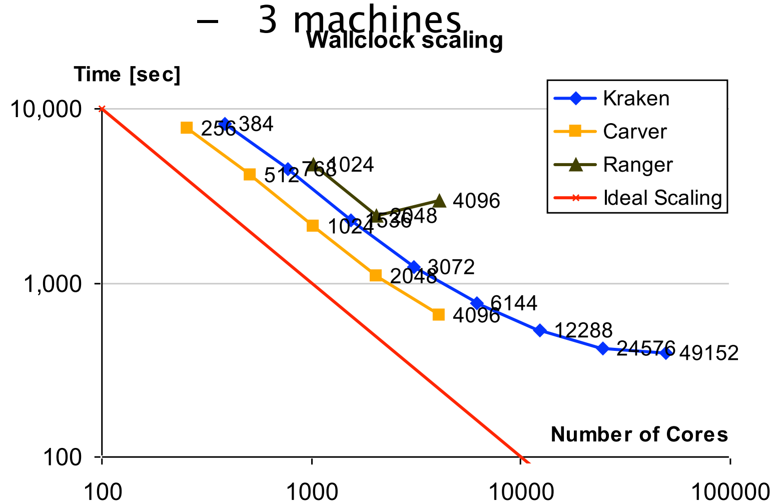

- IPM to CUBE converter

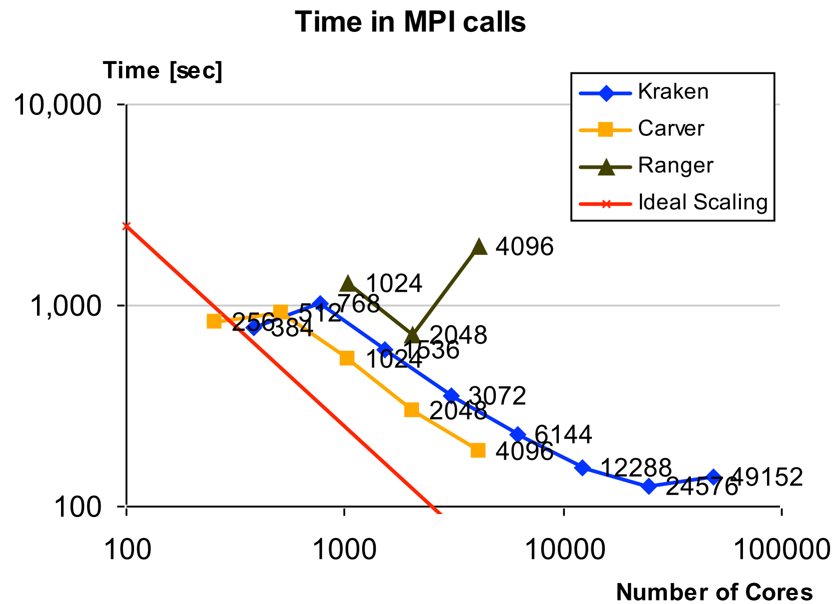


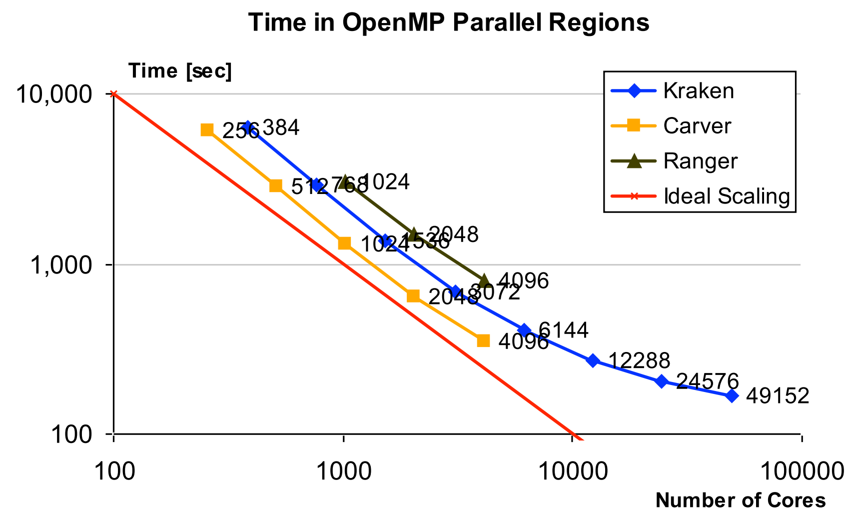
# Using IPM2 at Scale – GTC (1)



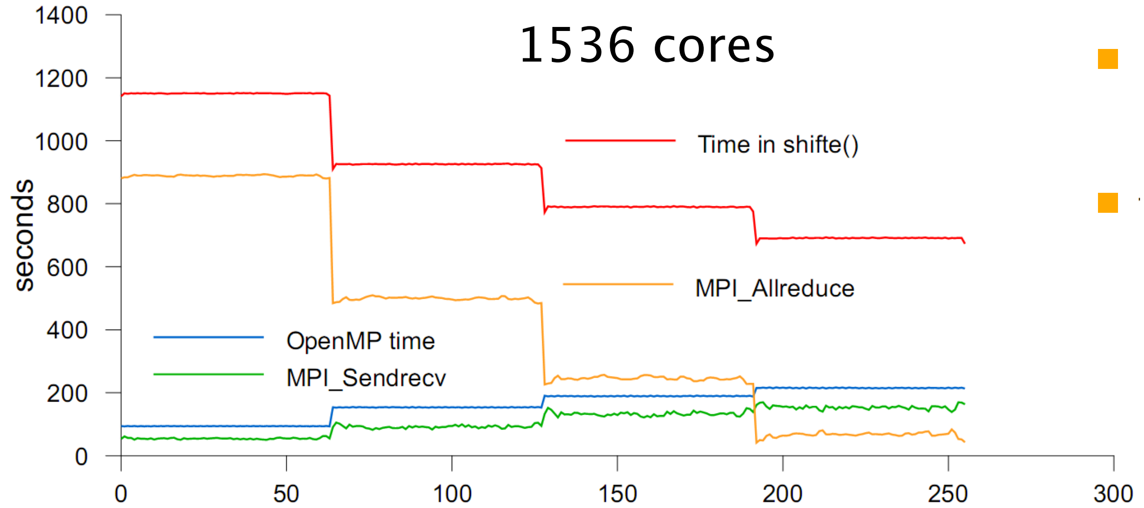
- Gyrokinetic Toroidal Code (fusion simulation)
  - OpenMP enabled 4/6 threads
  - Scaling up to 49152 cores
  - 3 machines

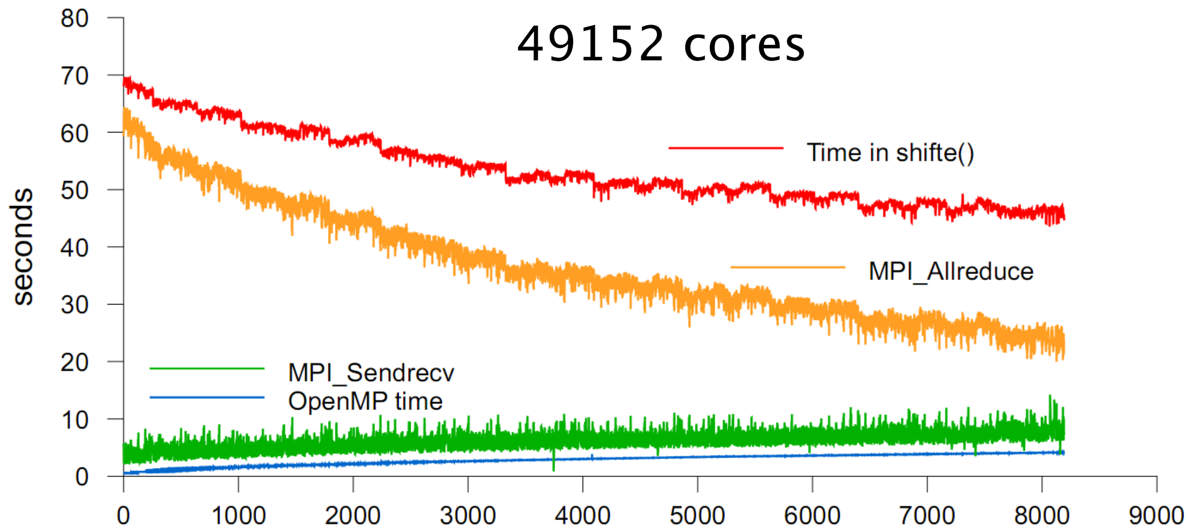


# Using IPM2 at Scale – GTC (2)




- Severe imbalance due to particle distribution among tasks
- Data from the XML profiling log

# Outline

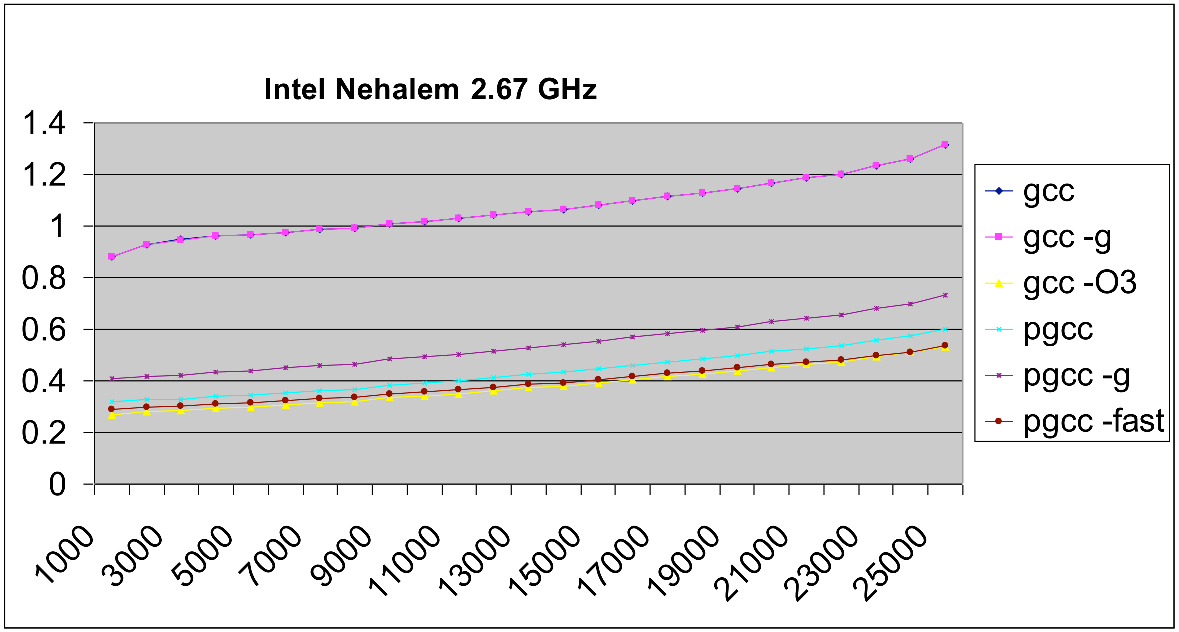
- IPM Overview and Philosophy
- IPM2 Design and Implementation
  - Event signatures and hashing
  - IPM2's modularized design
- Using IPM2
  - Banners, logs, and reports
  - Example: profiling GTC at scale
- IPM2 Efficiency and Scalability
  - Overheads, perturbation of IPM2
- New Areas of Research
  - IPM in the cloud
  - Execution flow graphs and trace recovery
  - Workload analysis

## IPM2 Efficiency Considerations

- Two areas of interest:
  - Application perturbation/dilatation at runtime
  - IPM2 scalabiliy as the core count increases
- Perturbation
  - Performance of raw hash table operations
  - Overheads of wrappers and timing
  - IPM2 overheads at the application level overheads vs. system runtime variation
- Scalabiliy
  - Parallel logfile writing

# IPM2 Efficiency – Raw Hash Table Operations

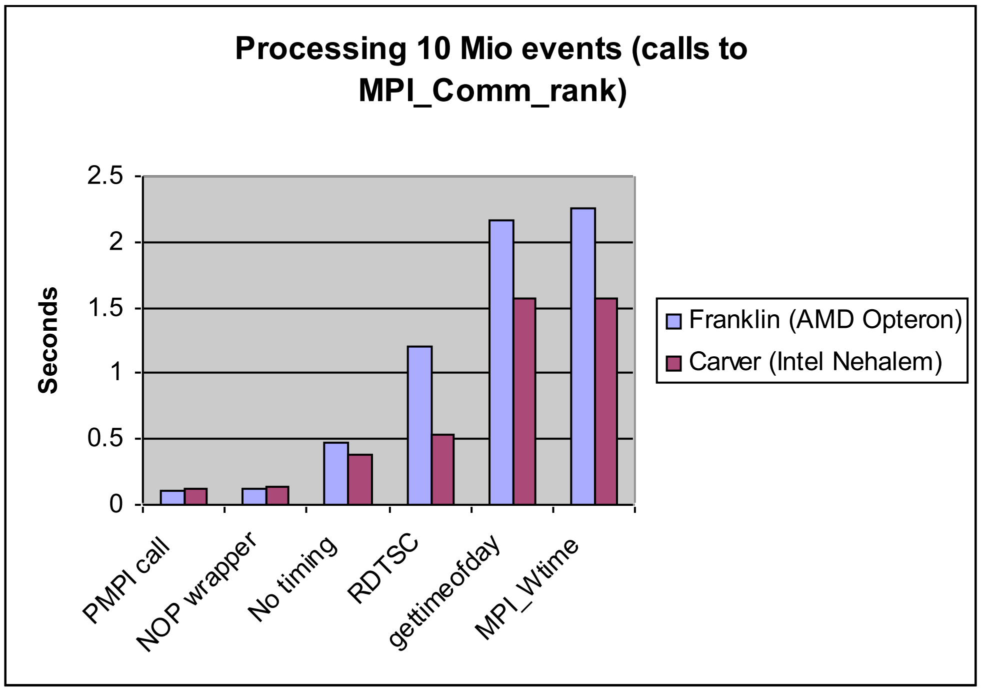
- Streaming analysis of 10 Mio events (random keys) in a hash table of size ~32k entries
  - Includes updating counts and times




- Typically we can handle ~10 Mio events or more per second
- Compiler optimization level is important
- This isn't optimized on the instruction level yet...

# IPM2 Efficiency – Timestamp Collectoin

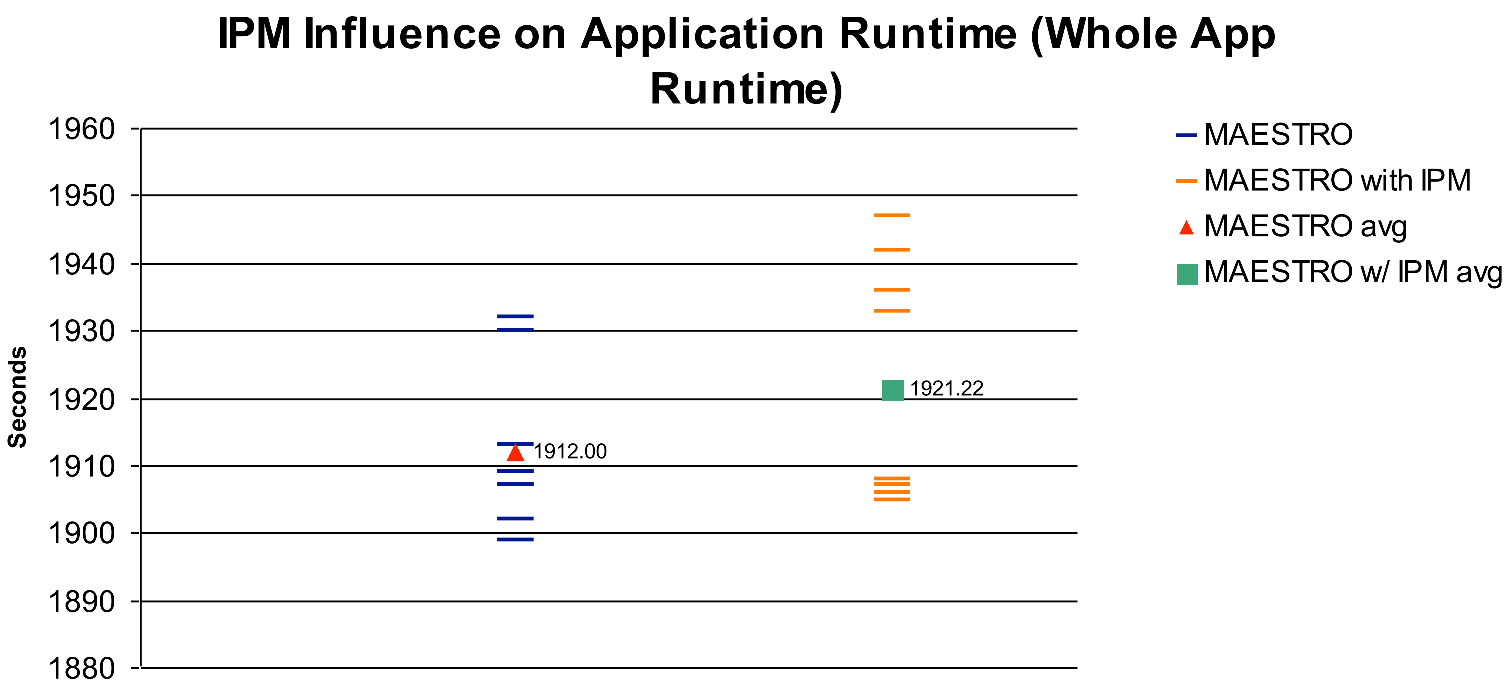
- 10 Mio events (calls to MPI_Comm_rank())
  - What's the overhead added by IPM wrappers?
  - Various timing options




- Again, ~10Mio or more events per second

# IPM2 Efficiency – Application Level

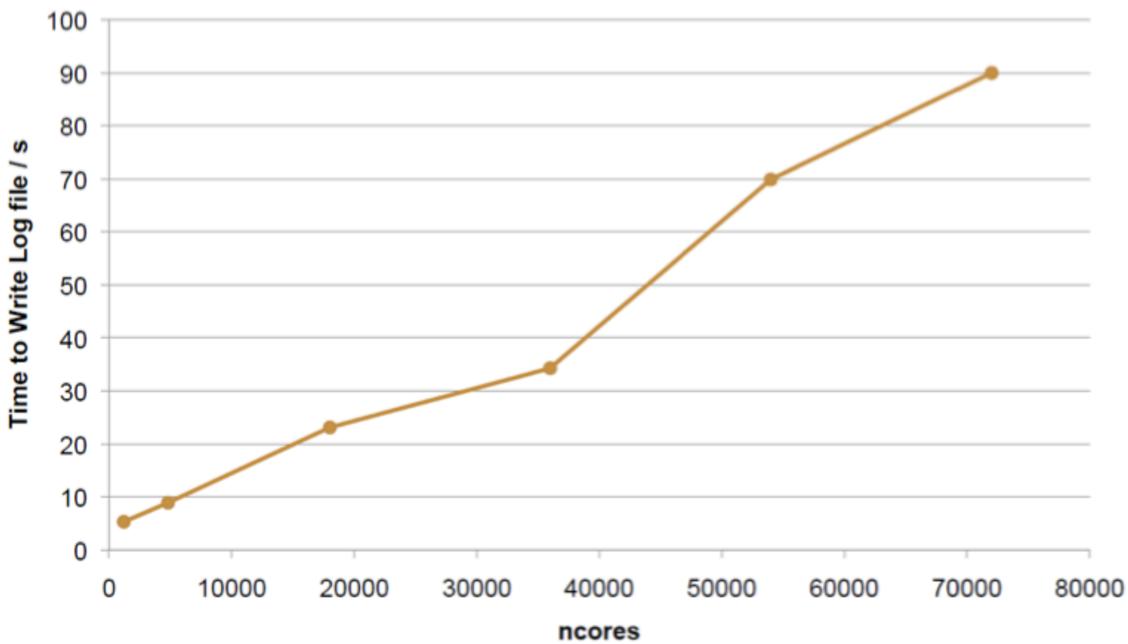
- MAESTRO
  - Astrophysics AMR code, 256 MPI tasks
  - Ensemble study: 9 runs over the course of a couple of days
  - Look at runtime variation with IPM2 and w/o IPM2




- **Less than 0.25%** overheads for application kernel
- **Less than 0.5%** overheads for whole application
  (includes writing XML log file, timing done in batch script)

## IPM2 Scaling

- All IPM2 operations at runtime are local, except
  - MPI_Init wrapper – figure out MPI task placement
  - MPI_Finalize wrapper – write log file
- Writing profiling logs at scale example
  - Kraken XT5@NICS
  - Write a full hashtable from every MPI process
  - Uses parallel MPI-IO strategy



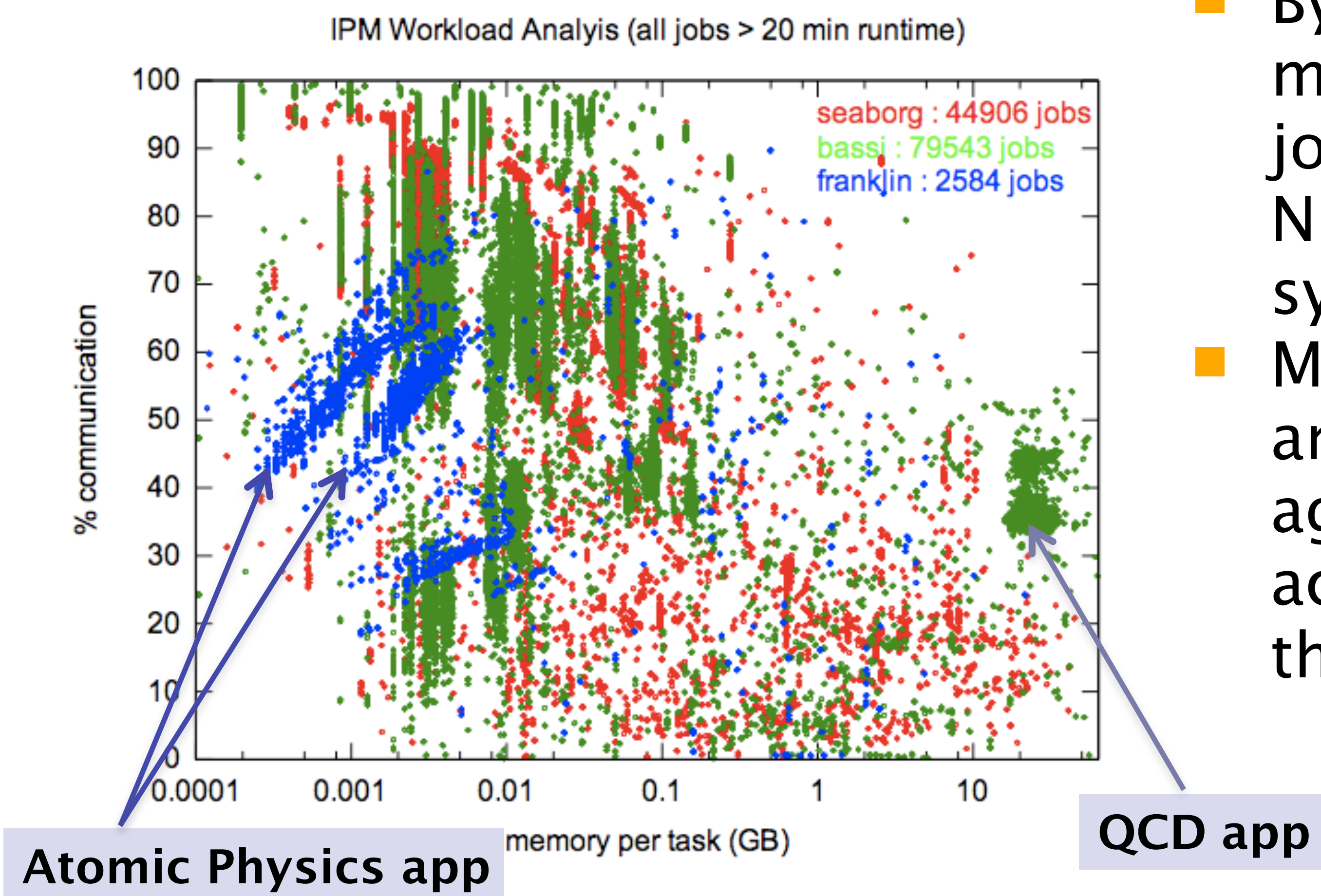

- Application example (GTC): about 30 seconds at 49,152 mpi tasks

# Outline

- IPM Overview and Philosophy
- IPM2 Design and Implementation
  - Event signatures and hashing
  - IPM2's modularized design
- Using IPM2
  - Banners, logs, and reports
  - Example: profiling GTC at scale
- IPM2 Efficiency and Scalability
  - Overheads, perturbation of IPM2
- Workload Analysis
  - Job and task level analysis
  - IPM in the cloud
  - Execution flow graphs and trace recovery

# Workload Analysis of HPC Jobs

- We have collected over 300k IPM profiles
  - Jobs running longer than 20min
  - Covers a period of 6 years

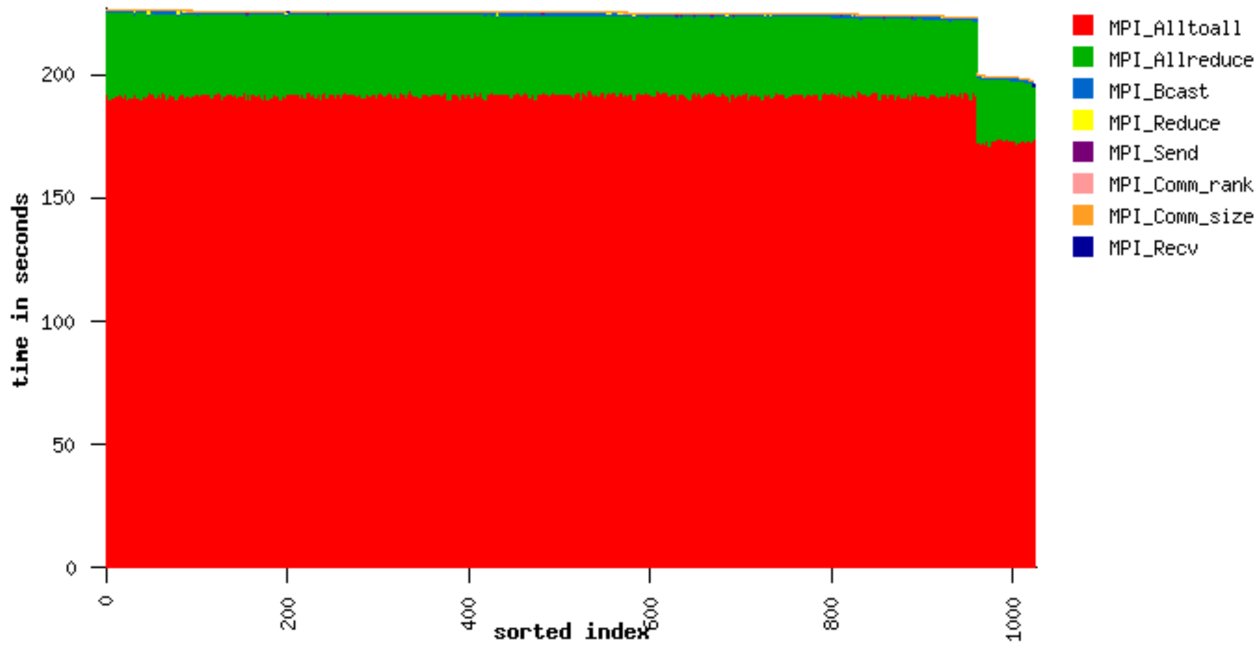



- By-default monitoring of all jobs on one of NERSC's cluster system soon
- Metrics are rates and sizes aggregated across tasks/ threads

# Workload Analysis of HPC Jobs (by task/ thread)

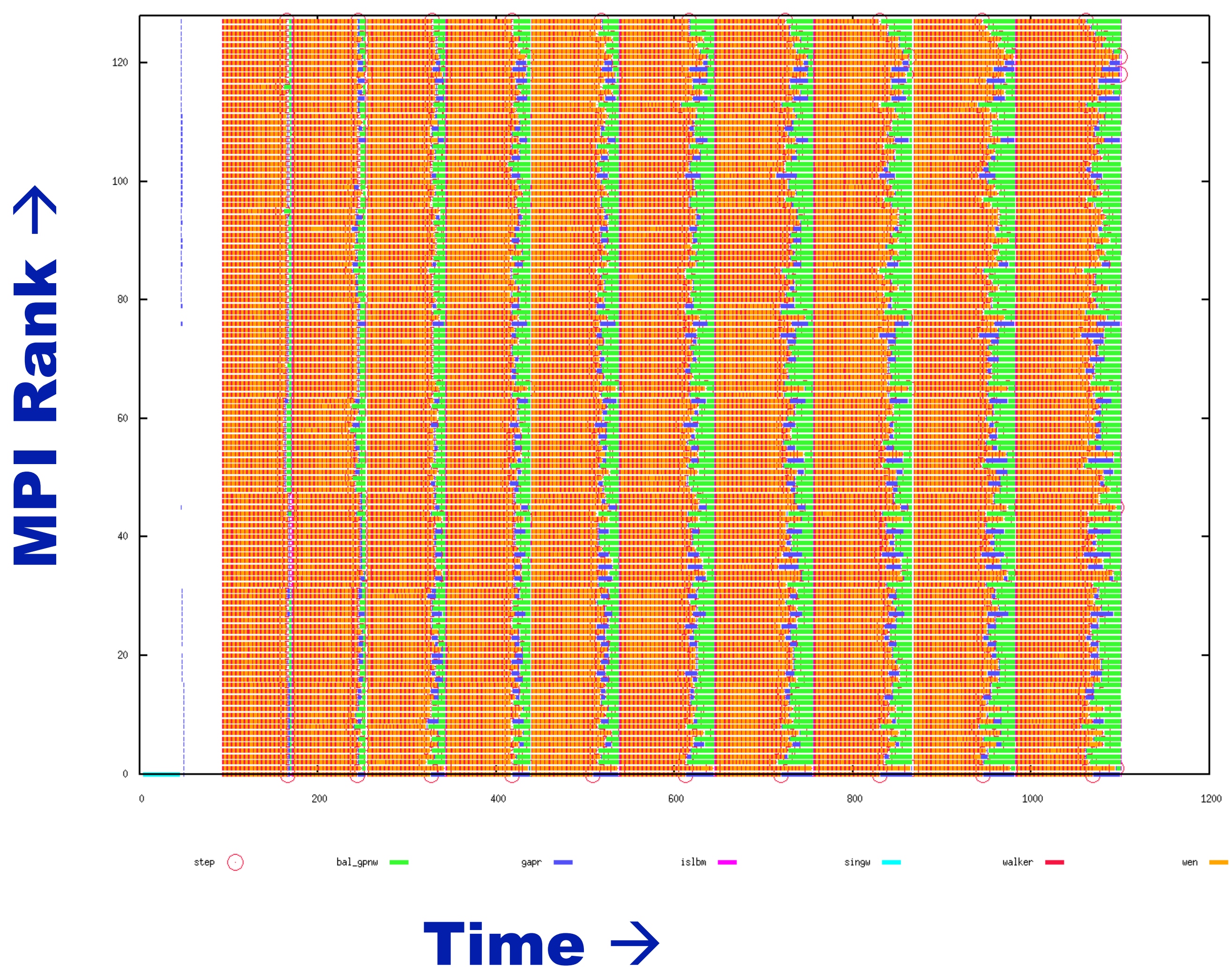
- Each dot contains N (#tasks x #threads) sets of metrics

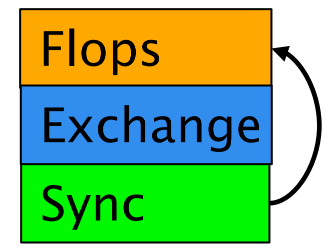
- In production computing settings load imbalance is c




- Luckily we can detect and correct through simple high

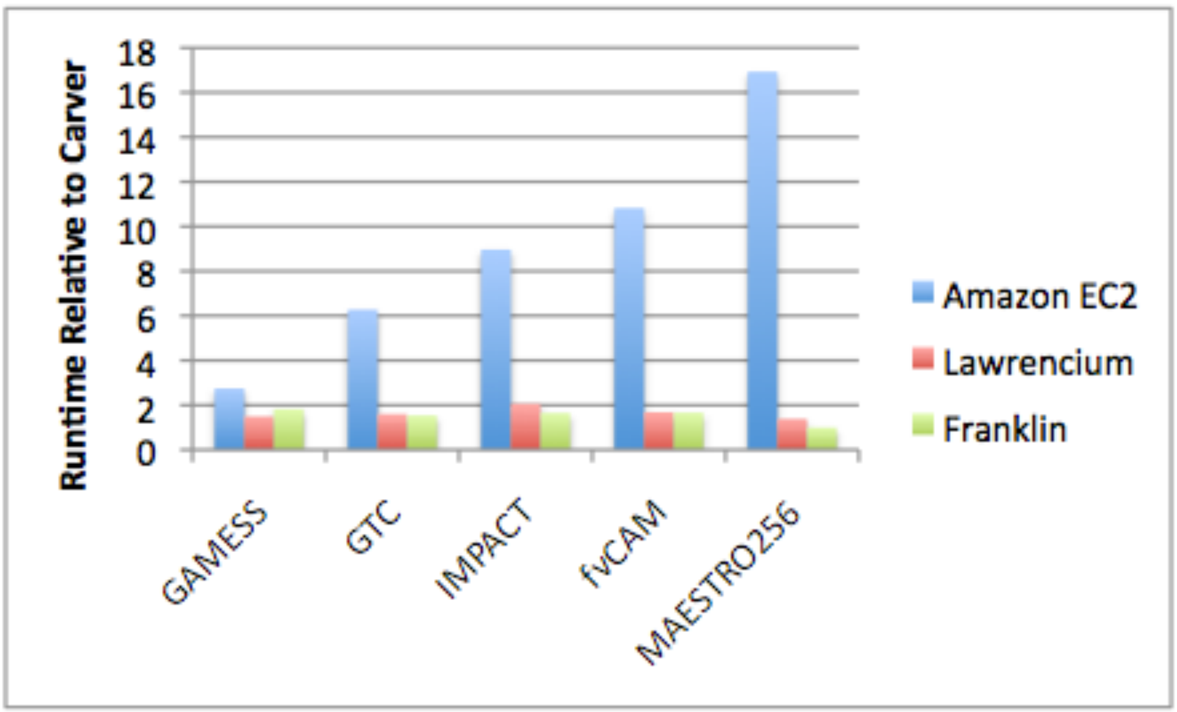
# Dynamically Disordered Load Balance: What we miss with



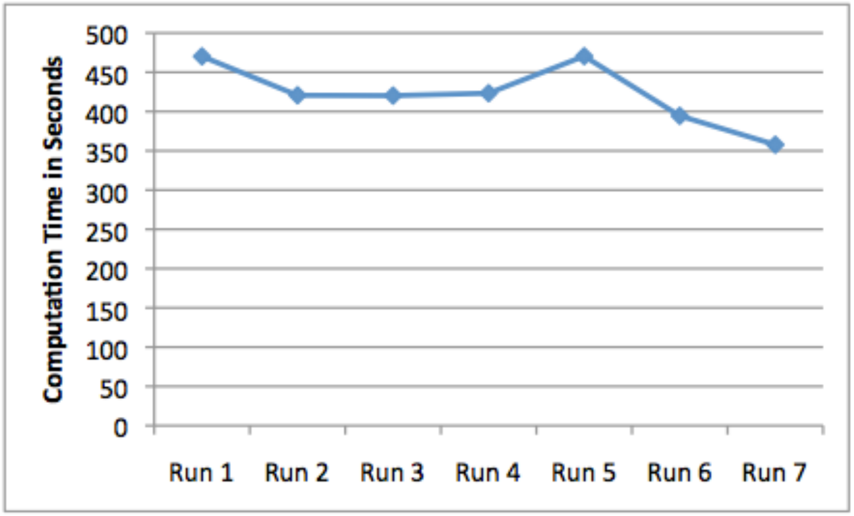
## Cloud Workloads

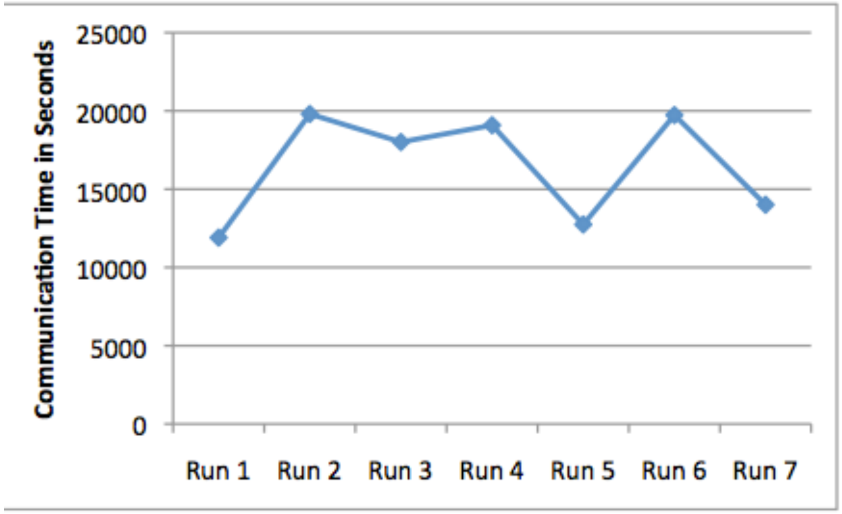
- IPM in the Cloud
  - Building timings into machine images is trivial
  - Getting at performance counters (or I/O counters) is another matter
- Some preliminary job level profiles



## Cloud Workloads

- What's behind those numbers?

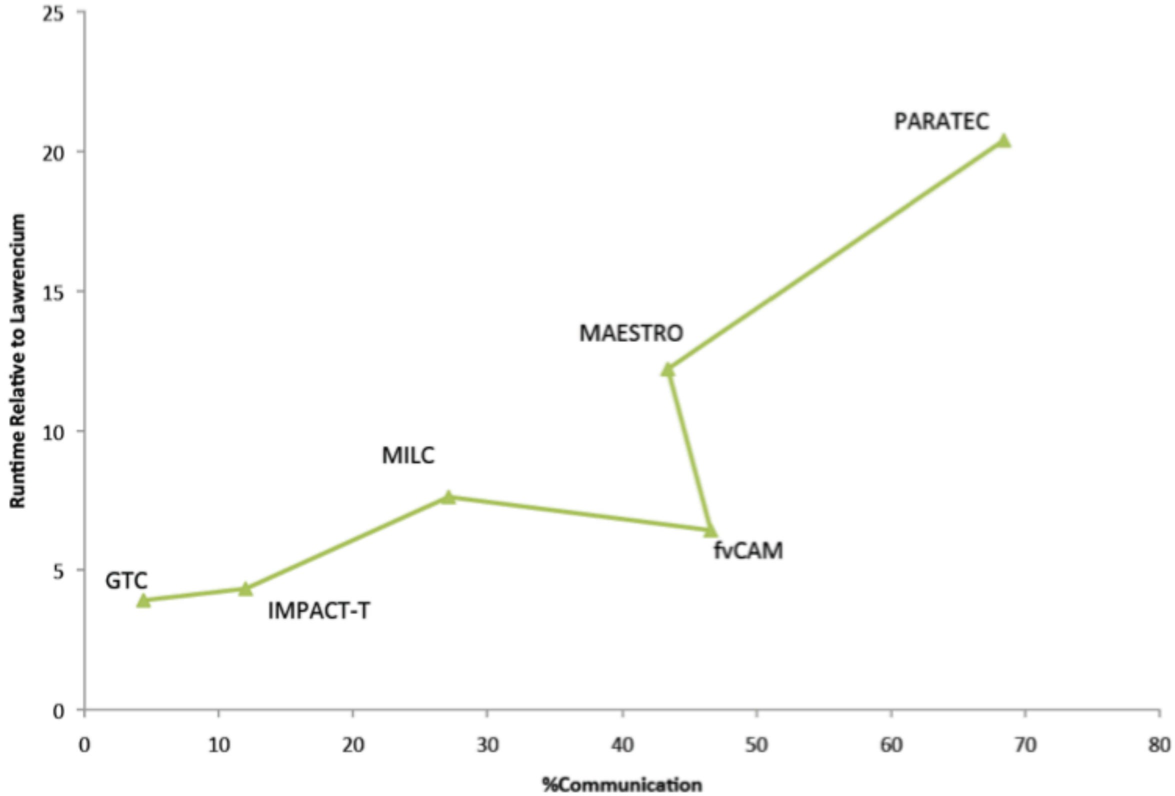



- Large variability
- The bulk synchronous nature of many apps remains to be addressed in cloud computing offerings

# Cloud Workloads

- IPM in the Cloud




Communication percentage vs. slowdown on the cloud

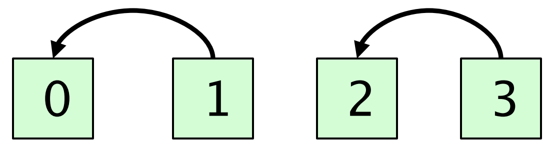
Keith R. Jackson, et al. “Performance Analysis of High Performance Computing Applications on the Amazon Web Services Cloud”, Submitted for Supercomputing 2010

# Event Flow Graphs

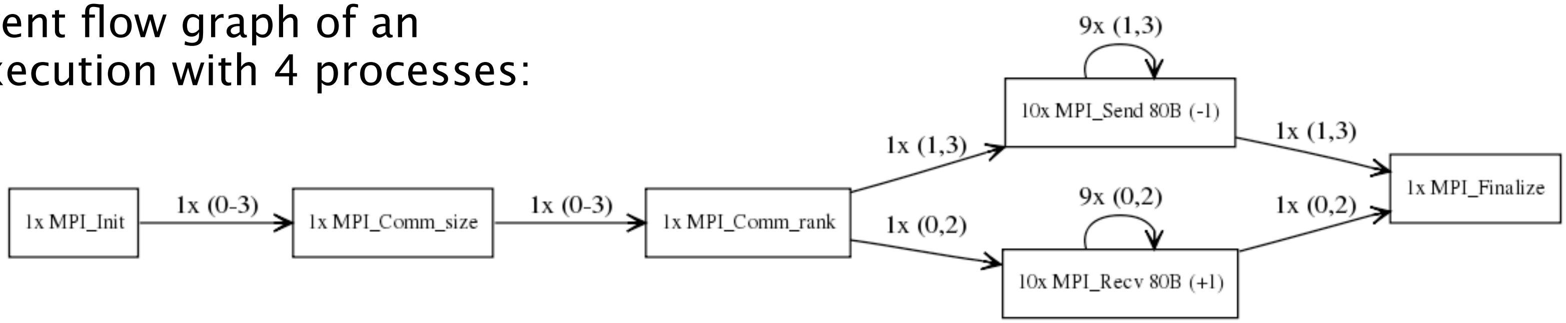
```
void main(int argc, char* argv[]) {
  MPI_Init(...);
  MPI_Comm_size(...);
  MPI_Comm_rank(..., &myrank);

  for(i=0; i<10; i++) {
    if(myrank is odd)
      MPI_Send(10 doubles to rank -1);
    else
      MPI_Recv(10 doubles from rank +1);
  }
  MPI_Finalize();
}
```

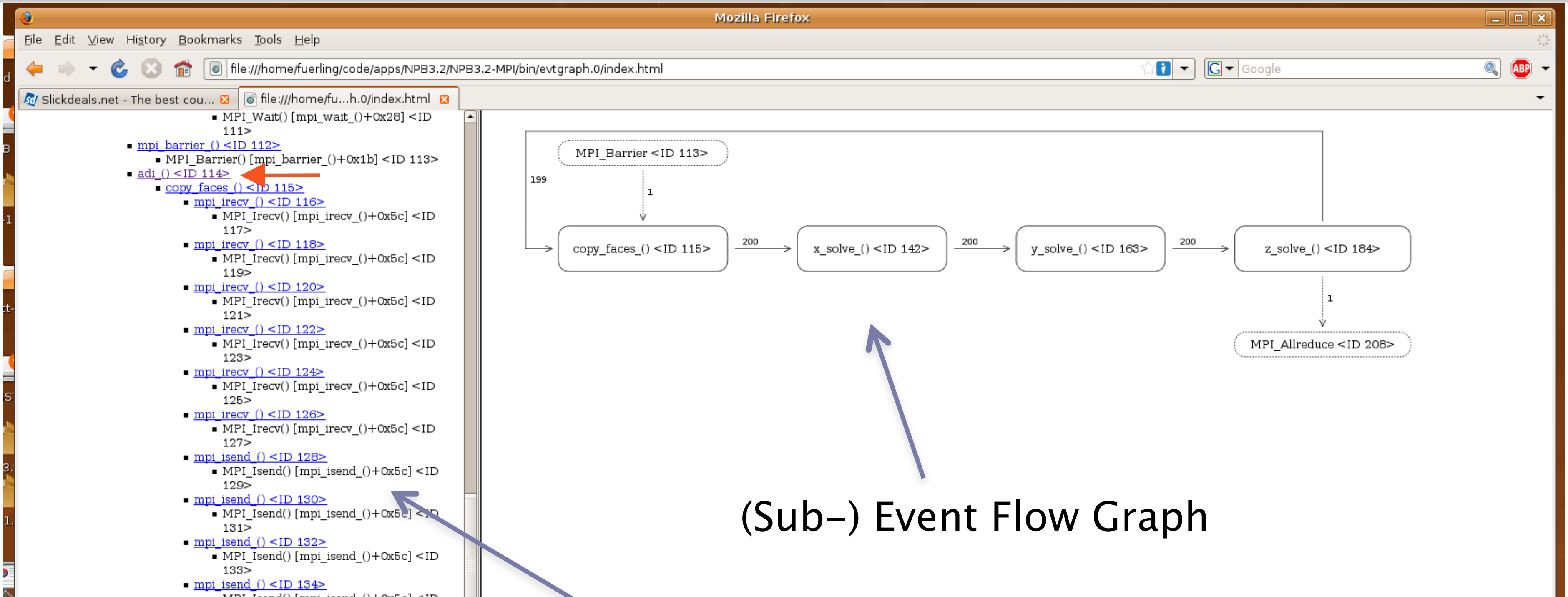
10 Iterations
Message size 80 bytes
Send to „left“ neighbor
Receive from „right“ neighbor




Event flow graph of an execution with 4 processes:



# Event Flow Graphs – Interactive Exploration



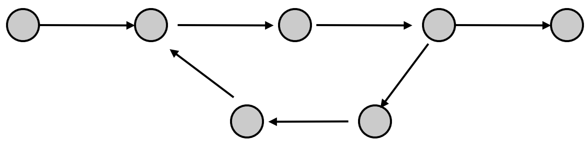

(Sub-) Event Flow Graph

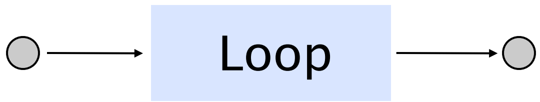
Call-graph Selection

Karl Fürlinger and David Skinner. Capturing and Visualizing Event Flow Graphs of MPI Applications. In Workshop on Productivity and Performance (PROPER 2009) in conjunction with Euro-Par 2009, August 2009.

# Event Flow Graphs – Trace Reconstruction

- Analyze graphs for loops and other control constructs and try to reconstruct traces, applying heuristics when neccessary






- Very early prototype can reconstruct traces of all NAS parallel benchmarks (some issues with MG and LU, which we think we can fix)
- Relationship to ScalaTrace work

# Conclusion

- IPM2 is a re-architected implementation of the IPM ideas
  - Same basic concept, modularized, monitoring coverage expanded
  - File-I/O, OpenMP, Cuda modules
  - IPM(1) is here: http://**ipm-hpc**.sourceforge.net/

- Scalability of IPM2
  - Scales well and is efficient to high concurrencies
  - Biggest remaining problems: file size and data analysis
    - Binary file representation
    - Clustering in the MPI rank space
    - Better load imbalance quantification
    - Analytics in MPI_Finalize()

- Workload analysis is underway
  - Instrumenting VM images
  - PAPI in virtual machines? PAPI for hypervisors?
  - Network counters
  - …

# Acknowledgements


- The IPM2 Team – http://**ipm-hpc**.sourceforge.net/

  David Skinner, NERSC
  Nick Wright, NERSC
  Andrew Usleton, NERSC
  Sascha Hunold, ICSI, UCB
  Michael Driscoll, UCB
  Kathy Yelick, UCB & NERSC
  Allan Snavely, SDSC


Thank you for your attention!

Reference for most of the topics covered in this talk:


Karl Fuerlinger, David Skinner, Nicholas J. Wright. Effective Performance Measurement at Petascale Using IPM2, Under review Cluster 2010, Heraklion, Greece.



- DOE
- NSF SDCI award
- Bavaria-California Technology Center (BaCaTec)