Mitglied der Helmholtz-Gemeinschaft

JÜLICH FORSCHUNGSZENTRUM

Scalasca Parallel Performance Analyses of PEPC

Oct 17 2008

Zoltán Szebenyi, Brian Wylie, Felix Wolf

JÜLICH FORSCHUNGSZENTRUM

Research Group
Performance Analysis of Parallel Programs

- Goal – make the optimization of parallel applications on large-scale systems both more effective and more efficient
- Located at the Jülich Supercomputing Centre

JÜLICH FORSCHUNGSZENTRUM

Motivation

- Variations of performance behavior along space & time
  - Gradual development of performance problems
  - Appropriate visualization techniques

Outline

- The Scalasca performance-analysis toolset
- Measuring the temporal and spatial evolution of application behavior with Scalasca
- Case study PEPC

JÜLICH FORSCHUNGSZENTRUM

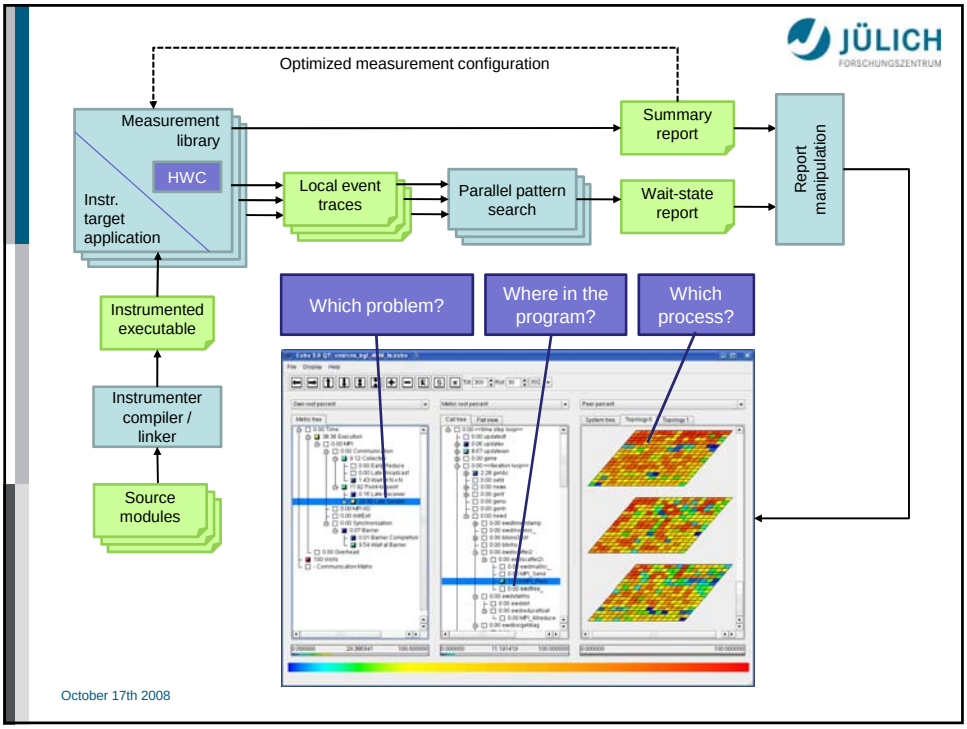
scalasca

- Scalable performance-analysis toolset for parallel codes
  - MPI, OpenMP and hybrid in progress
  - Emphasis on analysis of wait states
- Designed for large-scale systems such as IBM Blue Gene or Cray XT
  - Demonstrated for up to 65 K cores
- Funded by Helmholtz Association
- Developed in cooperation with the University of Tennessee
- http://www.scalasca.org/

HELMHOLTZ ASSOCIATION

THE UNIVERSITY of TENNESSEE

October 17th 2008



Phase-instrumentation

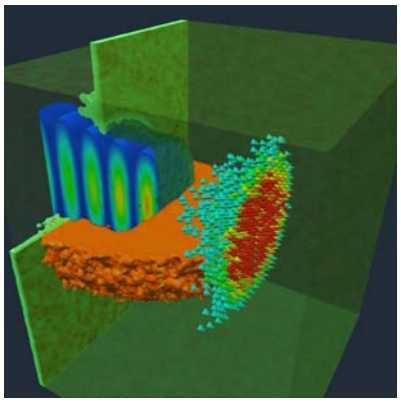
- How can I see how things evolve over time?
  - Is there a way to analyze time-dependent behavior?
- Phase-instrumentation
  - Identify the main loop of the application
  - Place markers at the beginning and the end of the loop
  - Scalasca will now measure every iteration separately
- Corresponds to dynamic phases in TAU

```
#include "epik_user.h"

int main() {
  int t;
  int epik_phase_time;
  EPIK_PHASE_REGISTER(epik_phase_time, "TIME");
  initialize();
  for(t=0; t<100; t++) {
    EPIK_PHASE_START(epik_phase_time);
    do_timestep();
    EPIK_PHASE_END(epik_phase_time);
  }
  finalize();
}
```

Pretty Efficient Parallel Coulomb-solver (PEPC)

- Multi-purpose parallel tree code
  - Molecular dynamics
  - Laser-plasma interactions
- Developed at the Jülich Supercomputing Centre
- Runs on IBM Power and Blue Gene machines
- One data set caused it to crash on BG/L



Jülich Blue Gene/P (JUGENE)

16 Racks Blue Gene/P

- 16384 Compute Nodes (4 processors, 2GB memory)
- 65536 CPUs, 32 TB memory
- 223 TFlop/s peak performance
- 167 TFlop/s Linpack
- 560 kW power consumption

Measurement overhead

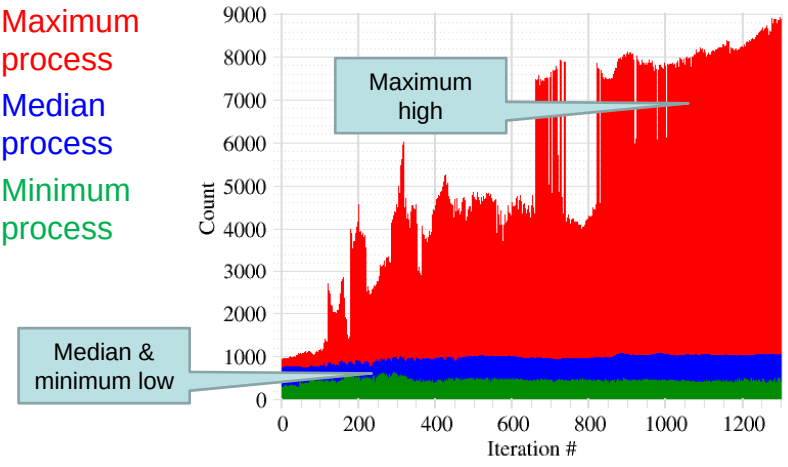
- 1300 PEPC time steps on JUGENE with 1024 processes

Instrumentation method	Dilation [%]
No instrumentation	0
MPI summary	1
Dynamic filtering of user functions	74
Dynamic & static filtering of user functions	5

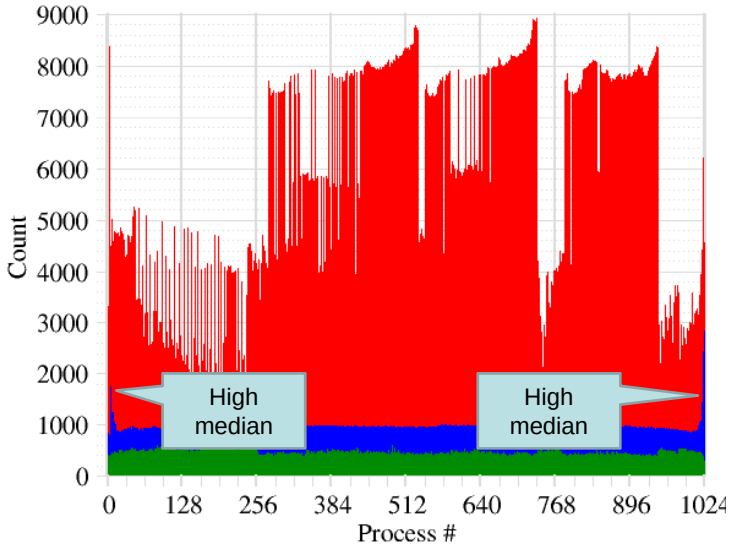
- Filtering of short but frequently executed functions

Point-to-point communication count per iteration

- Maximum process
- Median process
- Minimum process

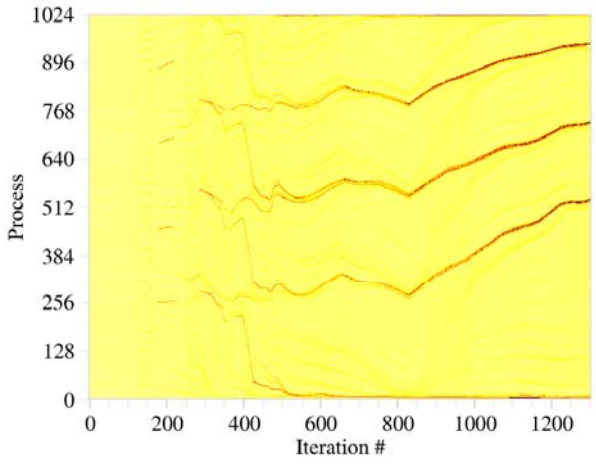


Point-to-point communication count per process



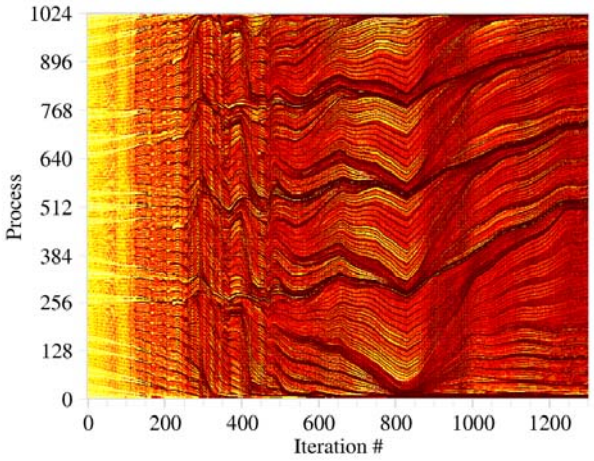
Linear-scale heat map of point-to-point communication count

- Hot spots moving to neighbors
- Severity of hot spots increasing
- What about the fine details?



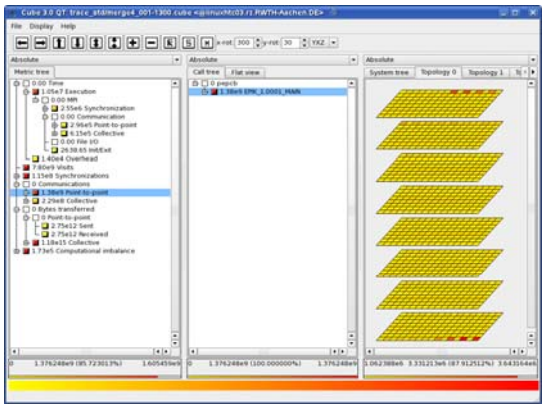
Histogram-equalized heat map of point-to-point communication count

- Equal number of pixels at each brightness level
- Reveals the details but loses scale information
- Hot spots just the tip of the iceberg



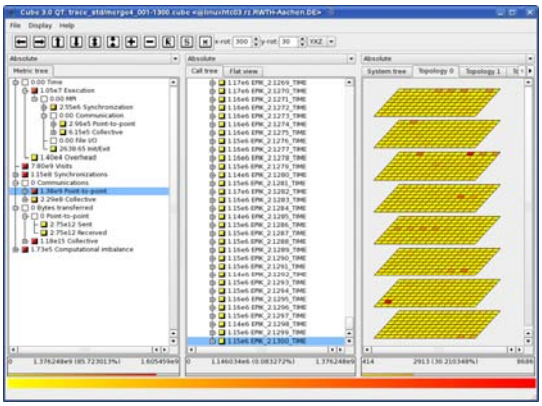
Summarized data without phases

- Only temporary invariant hot spots visible



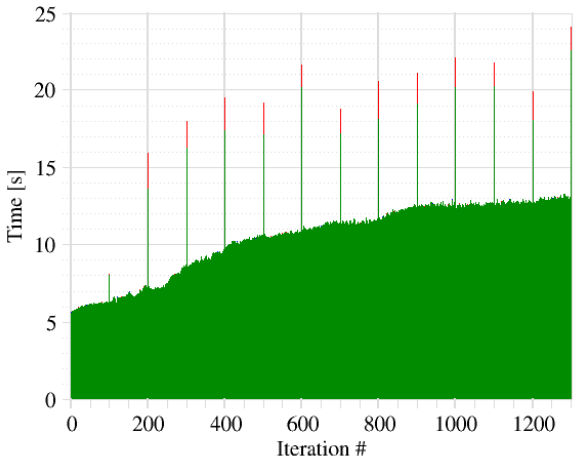
Data divided by phases

- Individual phases visible
- Stepping through time steps creates animation



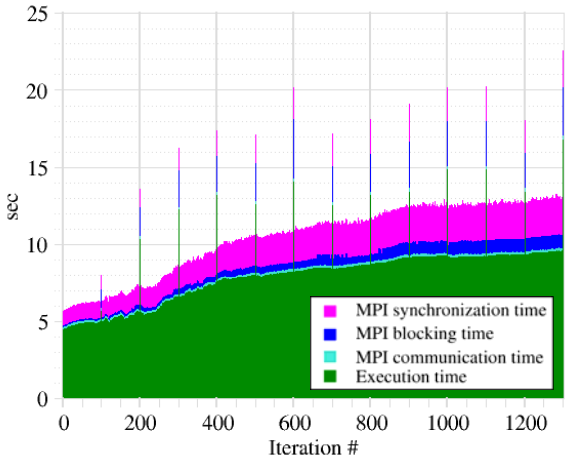
Execution time per phase

- The most basic graph
- Shows the evolution execution time per iteration
- More than doubles after 1300 iterations
- Checkpoints every 100th iteration



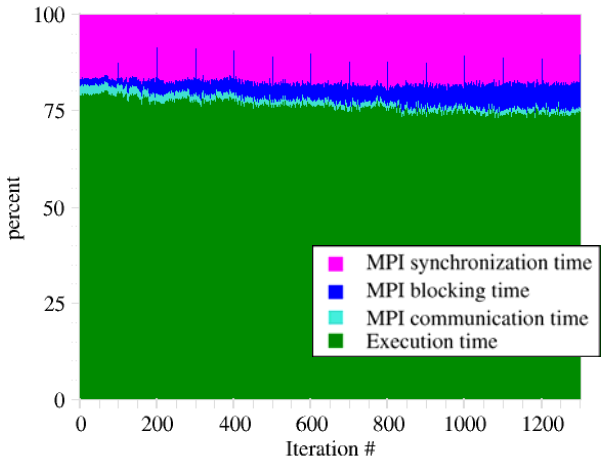
Execution time breakdown

- MPI blocking time is grows considerably (0.1->0.8s / iteration)
- Synchronization time also grows (1->3s / iteration)
- Most of the slowdown is due to more time spent in computation
- So the workload itself must be growing over time



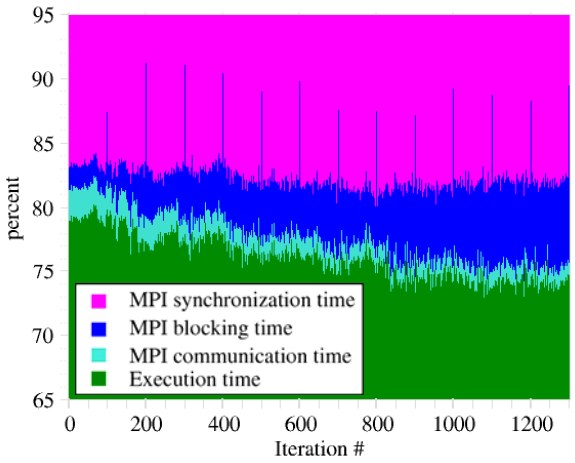
Normalized execution time breakdown

- Proportion of the times spent in different modes
- Fraction of pure execution time shrinks



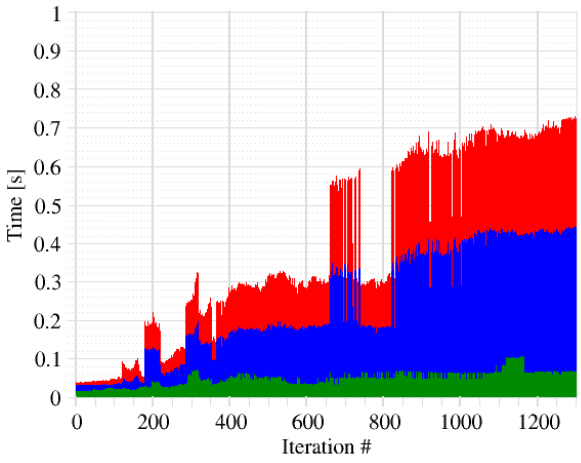
A closer look

- Sync. time grows from 17% to 19%
- MPI blocking time grows from 2% to 6%
- Where are they coming from?



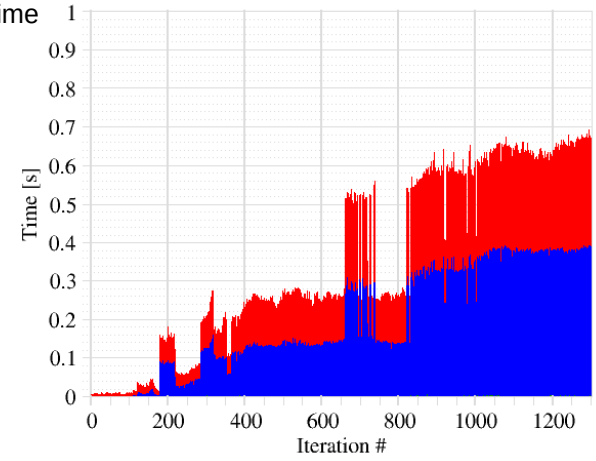
Point-to-point communication time

- Large imbalance



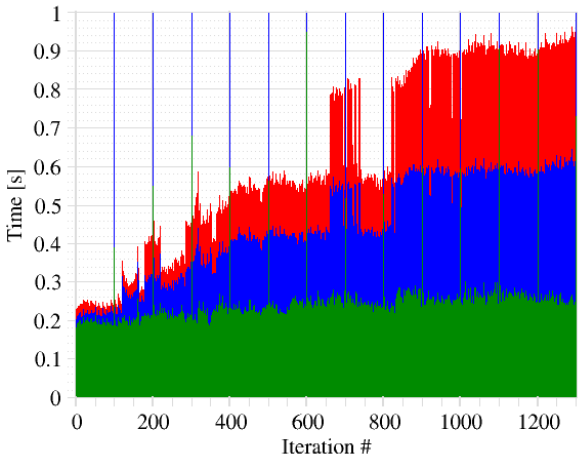
Point-to-point late sender time

- Most of the P2P time is spent in late sender situations



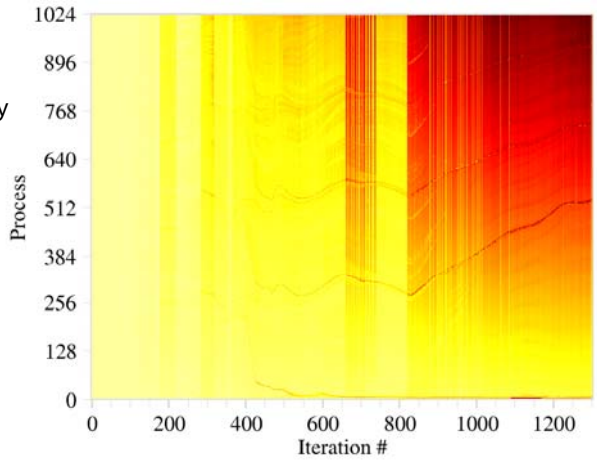
Collective communication time phase graph

- Collective communication suspiciously alike
- Same amount of imbalance
- Why?



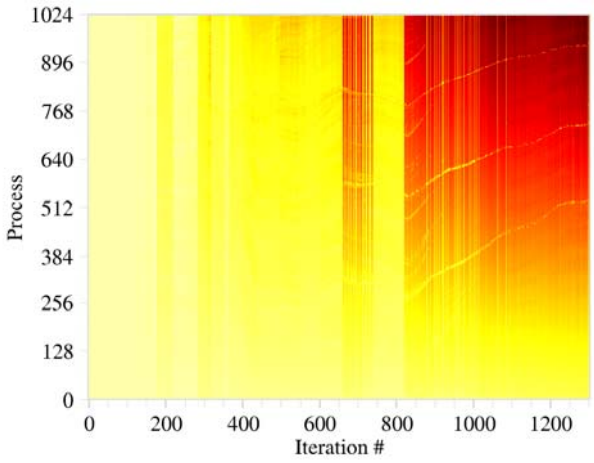
Point-to-point communication time (linear-scale)

- The heat map reveals the secret
- The processes get somehow serialized by P2P communication
- Do you see the dark diagonal lines?
- Let's take a look at late sender time!



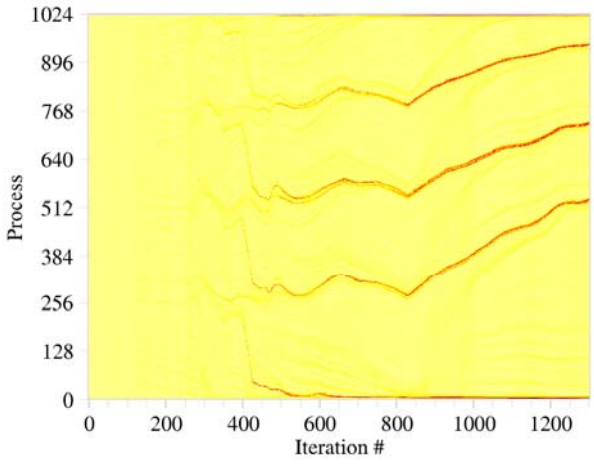
Point-to-point late sender time (linear-scale)

- The dark diagonal lines just disappeared!
- So the others are actually waiting for the hot spots



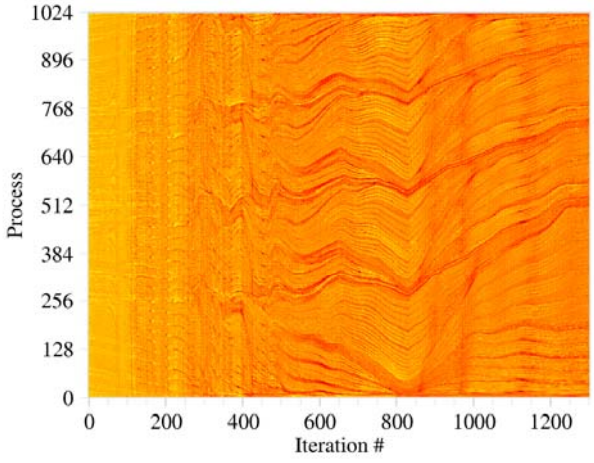
Number of bytes sent (linear-scale)

- It seems they are sending a lot of data...



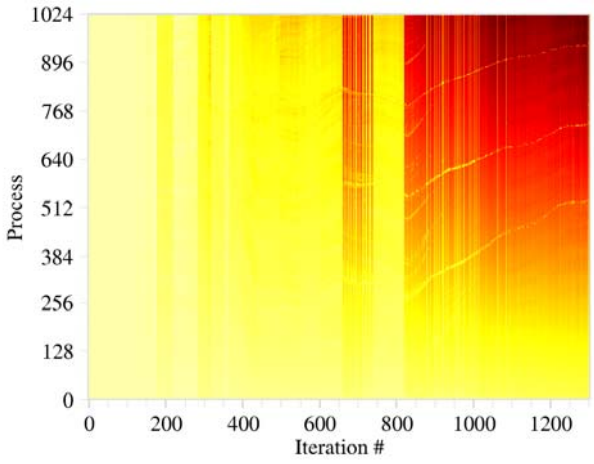
Number of bytes received (absolute value)

- ... to all the others



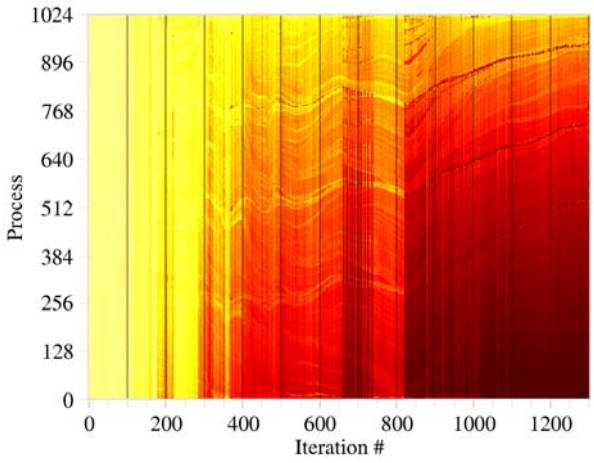
Point-to-point late sender time (linear-scale)

- And this causes serious waiting times on higher ranks
- What else does it cause?



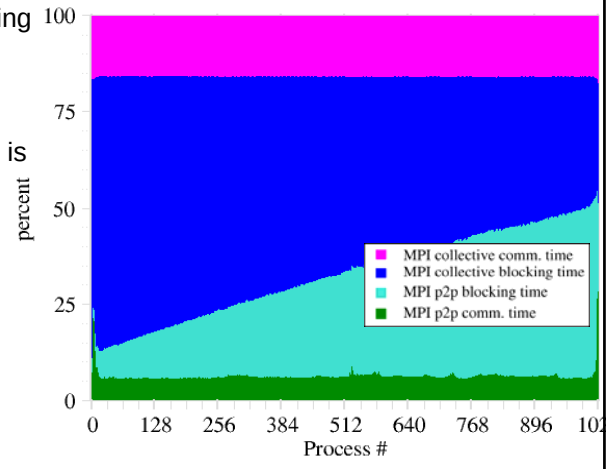
Collective communication time (histogram-equalized)

- Just the inverse of point-to-point time
- The lower ranks are waiting longer
- Imbalance of point-to-point is being evened out here
- Is there a way to support this theory?



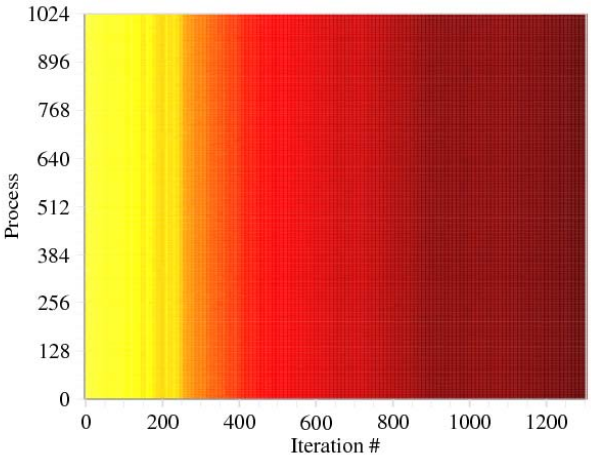
Normalized communication time breakdown

- The collective blocking time evens out the P2P blocking time
- About 70% of all communication time is spent waiting
- Why did we have to wait for the hot-spot processes then?



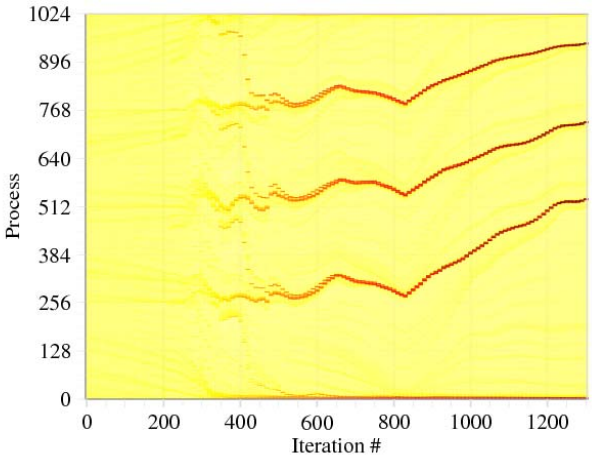
Workload heat map

- A metric called “workload” is balanced after every iteration
- Application specific metric based on log files calculated every 10th iteration

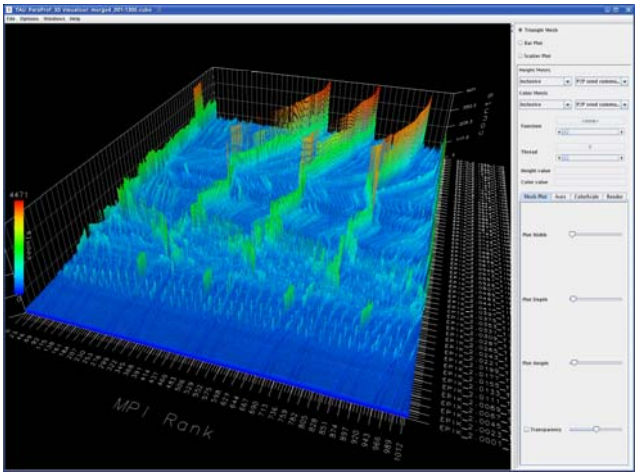


Particle number heat map

- Load-balancing causes the hot-spot processes to have many more particles than others
- Seems to be the root cause of a huge part of both P2P and collective waiting times
- Caused the application to crash on BG/L

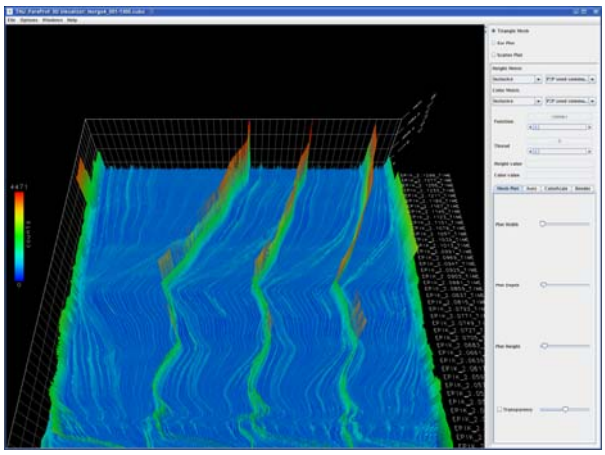


TAU Paraprof 3D visualization



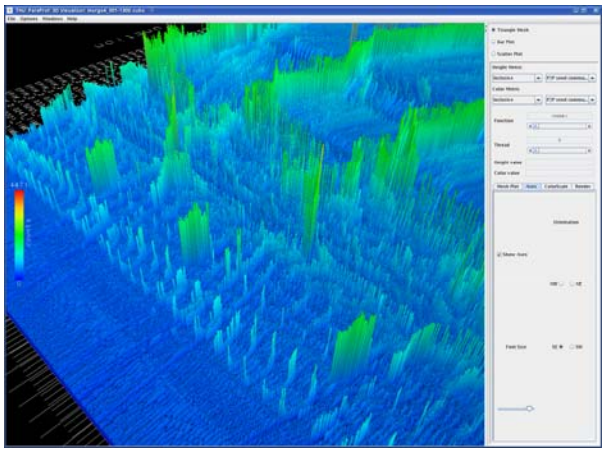
TAU Paraprof 3D visualization (2)

- 3D visualization is a very powerful tool
- A great way to get an overview…



TAU Paraprof 3D visualization (3)

- … or to zoom in to the finest details



Conclusion

- Spatial & temporal behavior can be highly unstable
- Visualization at larger scales is not straightforward
- Choosing the right visualization techniques can be decisive in finding the right answers
- Improved load balancing in PEPC in progress

Future work

- Make both 2D and 3D visualization more scalable
- Provide better support for application-specific metrics