German Research School for Simulation Sciences

Methods for performance evaluation and optimization on modern HPC systems

Felix Wolf

08-06-2011

JÜLICH FORSCHUNGSZENTRUM

RWTHAACHEN UNIVERSITY

Objectives

- Learn about basic performance measurement and analysis methods and techniques for HPC applications
- Get to know Scalasca, a scalable and portable performance analysis tool

Performance tuning: an old problem

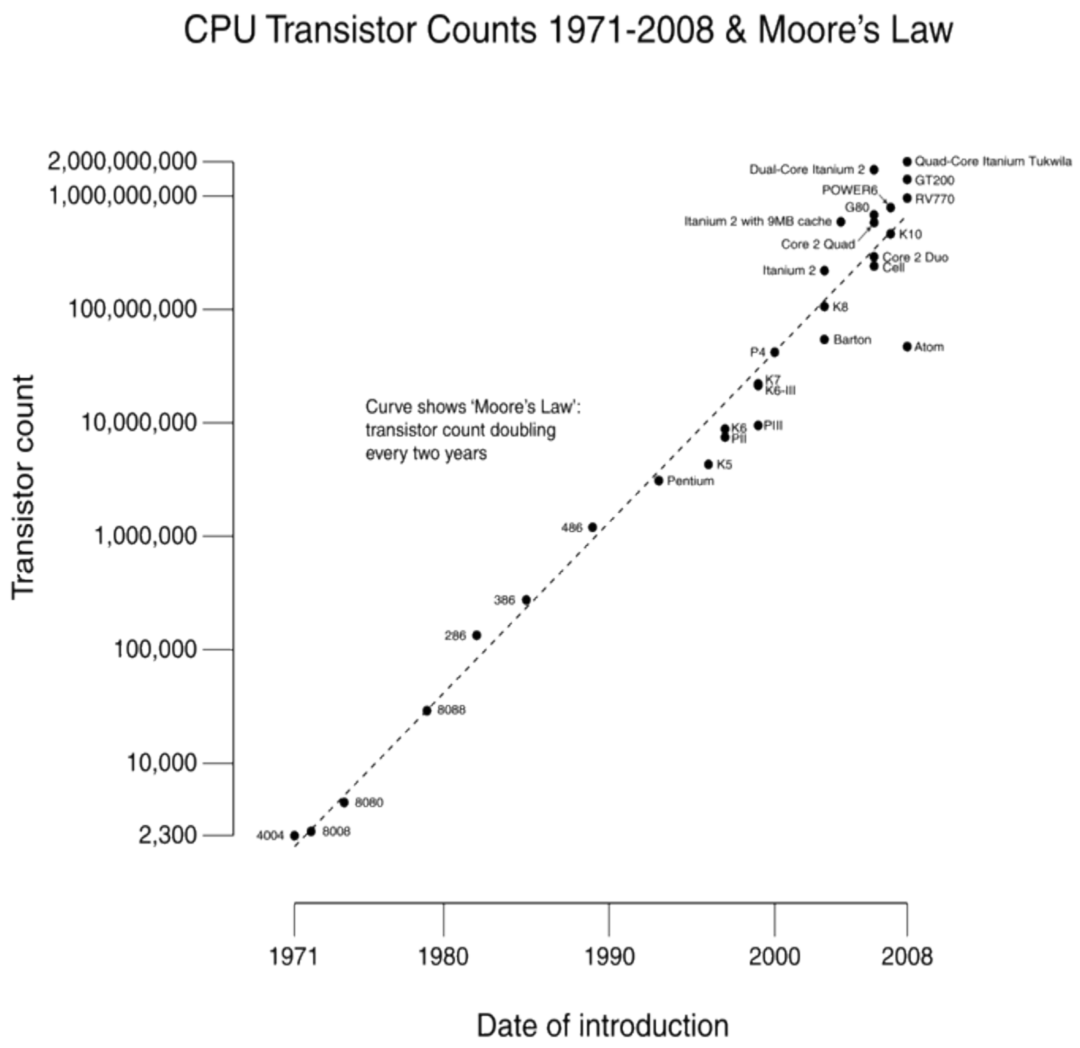
"The most constant difficulty in contriving the engine has arisen from the desire to reduce the time in which the calculations were executed to the shortest which is possible."

Charles Babbage
1791 - 1871

Outline

- Principles of parallel performance
- Performance analysis techniques
- Practical performance analysis using Scalasca

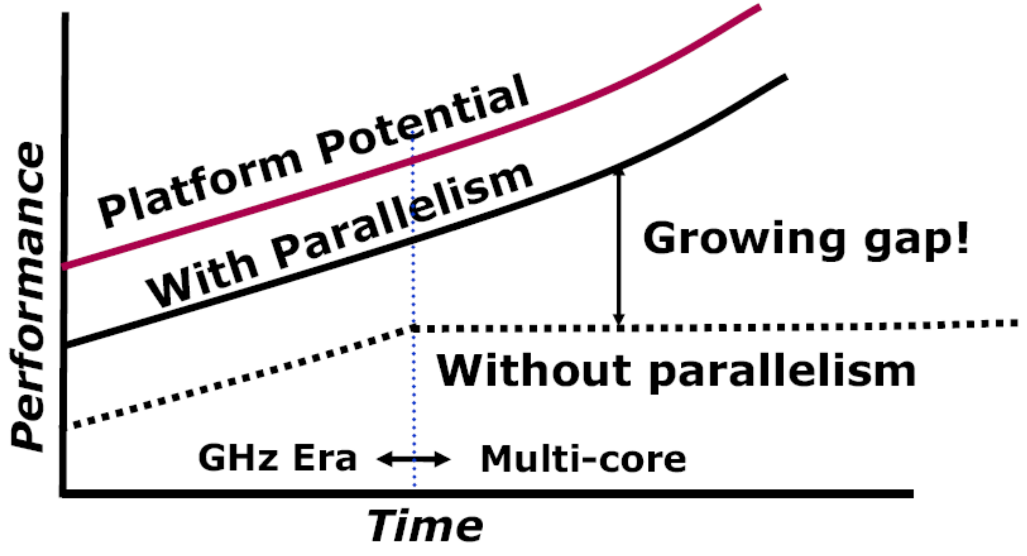
Why parallelism at all?
Moore's Law is still in charge…



Source: Wikipedia

Free lunch is over…

Need for Parallelism



Parallelism is crucial for optimal performance

Parallelism

- System/application level
  - Server throughput can be improved by spreading workload across multiple processors or disks
  - Ability to add memory, processors, and disks is called scalability
- Individual processor
  - Pipelining
  - Depends on the fact that many instructions do not depend on the results of their immediate predecessors
- Detailed digital design
  - Set-associative caches use multiple banks of memory
  - Carry-lookahead in modern ALUs

Amdahl's Law for parallelism

- Assumption – program can be parallelized on p processors except for a sequential fraction f with

$$0 \le f \le 1$$

$$Speedup(p) = \frac{1}{f + \frac{1-f}{p}} < \frac{1}{f}$$

- Speedup limited by sequential fraction

Available parallelism

- Overall speedup of 80 on 100 processors

$$80 = \frac{1}{f + \frac{1-f}{p}}$$

$$f = 0.0025$$

Law of Gustafson

- Amdahl's Law ignores increasing problem size
  - Parallelism often applied to calculate bigger problems instead of calculating a given problem faster
- Fraction of sequential part may be function of problem size
- Assumption
  - Sequential part has constant runtime τ_f
  - Parallel part has runtime $\tau_v(n,p)$
- Speedup

$$\text{Speedup}(n,p) = \frac{\tau_f + \tau_v(n,1)}{\tau_f + \tau_v(n,p)}$$

If parallel part can be perfectly parallelized

Parallel efficiency

$$\text{Efficiency}(p) = \frac{\text{Speedup}(p)}{p}$$

- Metric for cost of parallelization (e.g., communication)
- Without super-linear speedup

$$\text{Efficiency}(p) \leq 1$$

- Super-linear speedup possible
  - Critical data structures may fit into the aggregate cache

Scalability

- Weak scaling
  - Ability to solve a larger input problem by using more resources (here: processors)
  - Example: larger domain, more particles, higher resolution
- Strong scaling
  - Ability to solve the same input problem faster as more resources are used
  - Usually more challenging
  - Limited by Amdahl’s Law and communication demand

Serial vs. parallel performance

- Serial programs
  - Cache behavior and ILP
- Parallel programs
  - Amount of parallelism
  - Granularity of parallel tasks
  - Frequency and nature of inter-task communication
  - Frequency and nature of synchronization
    - Number of tasks that synchronize much higher → contention

Goals of performance analysis

- Compare alternatives
  - Which configurations are best under which conditions?
- Determine the impact of a feature
  - Before-and-after comparison
- System tuning
  - Find parameters that produce best overall performance
- Identify relative performance
  - Which program / algorithm is faster?
- Performance debugging
  - Search for bottlenecks
- Set expectations
  - Provide information for users

Analysis techniques (1)

- Analytical modeling
  - Mathematical description of the system
  - Quick change of parameters
  - Often requires restrictive assumptions rarely met in practice
    - Low accuracy
  - Rapid solution
  - Key insights
    - Validation of simulations / measurements
- Example
  - Memory delay $t_{avg} = ht_c + (1-h)t_m$
  - Parameters obtained from manufacturer or measurement

Analysis techniques (2)

- Simulation
  - Program written to model important features of the system being analyzed
  - Can be easily modified to study the impact of changes
  - Cost
    - Writing the program
    - Running the program
  - Impossible to model every small detail
    - Simulation refers to “ideal” system
    - Sometimes low accuracy
- Example
  - Cache simulator
  - Parameters: size, block size, associativity, relative cache and memory delays

Analysis techniques (3)

- Measurement
  - No simplifying assumptions
  - Highest credibility
  - Information only on specific system being measured
  - Harder to change system parameters in a real system
  - Difficult and time consuming
  - Need for software tools

- Should be used in conjunction with modeling
  - Can aid the development of performance models
  - Performance models set expectations against which measurements can be compared

Metrics of performance

- What can be measured?
  - A count of how many times an event occurs
    - E.g., Number of input / output requests
  - The duration of some time interval
    - E.g., duration of these requests
  - The size of some parameter
    - Number of bytes transmitted or stored

- Derived metrics
  - E.g., rates / throughput
  - Needed for normalization

Primary performance metrics

- Execution time, response time
  - Time between start and completion of a program or event
  - Only consistent and reliable measure of performance
  - Wall-clock time vs. CPU time
- Throughput
  - Total amount of work done in a given time
- Performance = $\dfrac{1}{\text{Execution time}}$
- Basic principle: reproducibility
- Problem: execution time is slightly non-deterministic
  - Use mean or minimum of several runs

Alternative performance metrics

- Clock rate
- Instructions executed per second
- FLOPS
  - Floating-point operations per second
- Benchmarks
  - Standard test program(s)
  - Standardized methodology
  - E.g., SPEC, Linpack
- QUIPS / HINT [Gustafson and Snell, 95]
  - Quality improvements per second
  - Quality of solution instead of effort to reach it

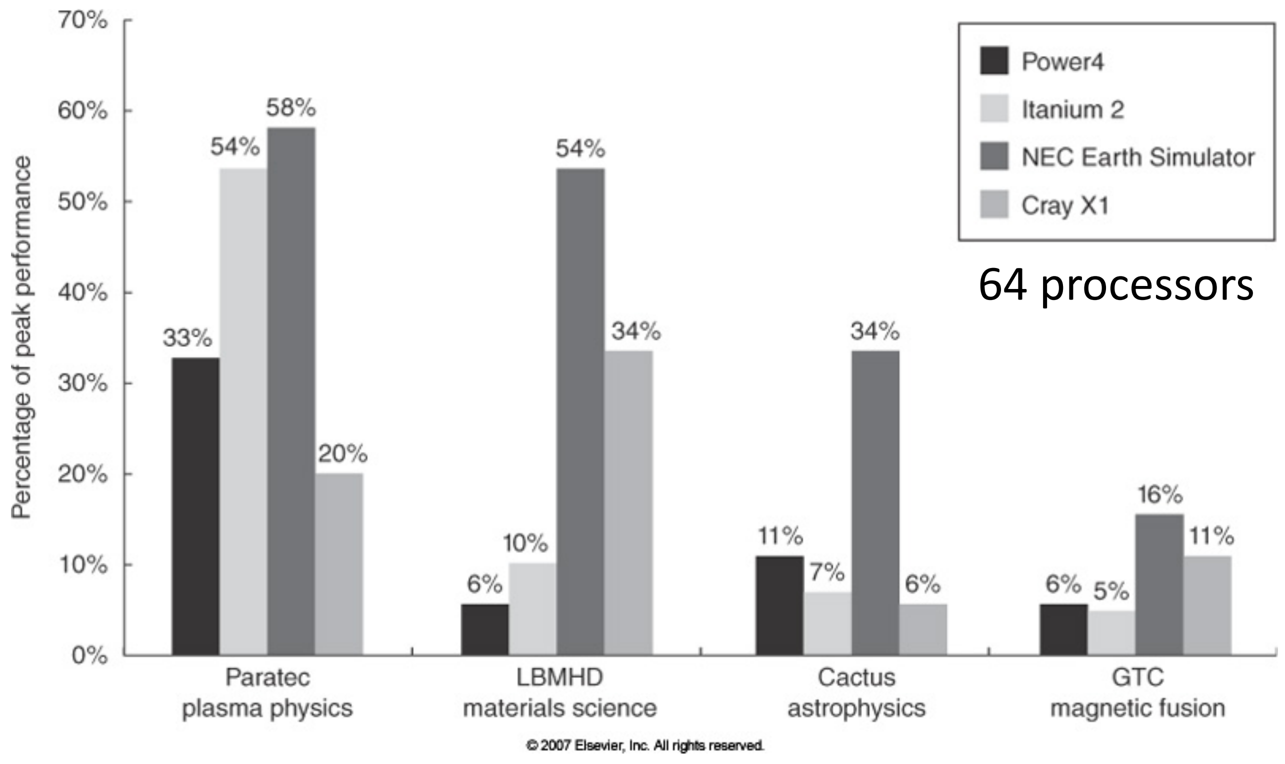
"Math" operations?
HW operations?
HW instructions?
Single or double precision?

Comparison of analysis techniques

	Analytical modeling	Simulation	Measurement
Flexibility	High	High	Low
Cost	Low	Medium	High
Credibility	Low	Medium	High
Accuracy	Low	Medium	High

Peak performance

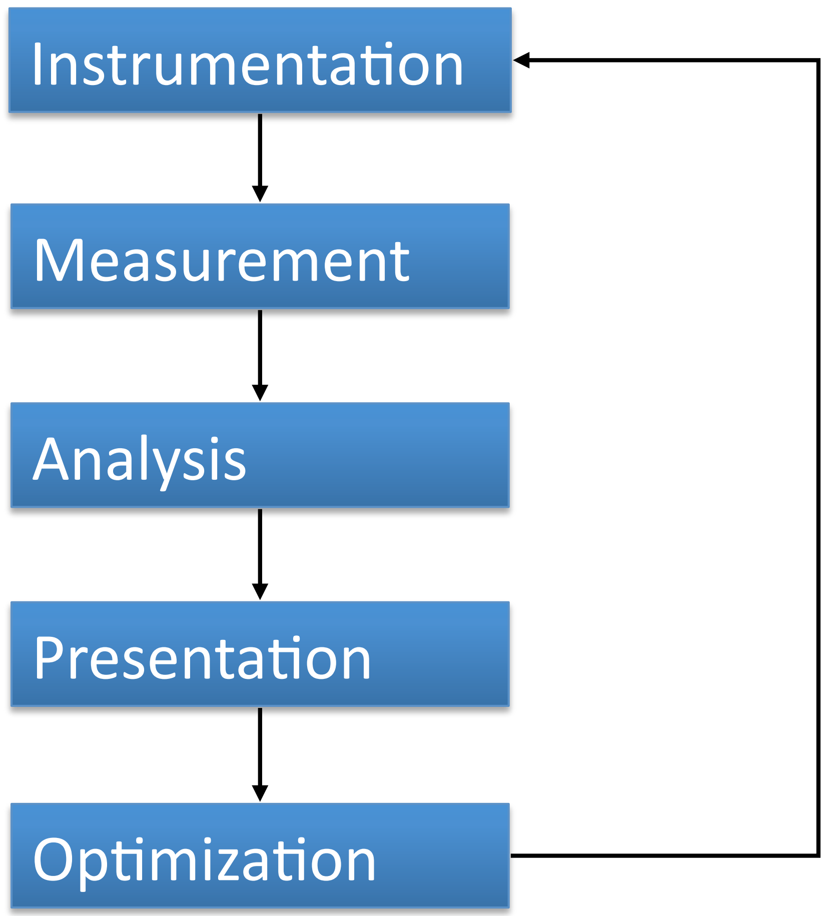
- Peak performance is the performance a computer is guaranteed not to exceed



64 processors

Source: Hennessy, Patterson: Computer Architecture, 4th edition, Morgan Kaufmann

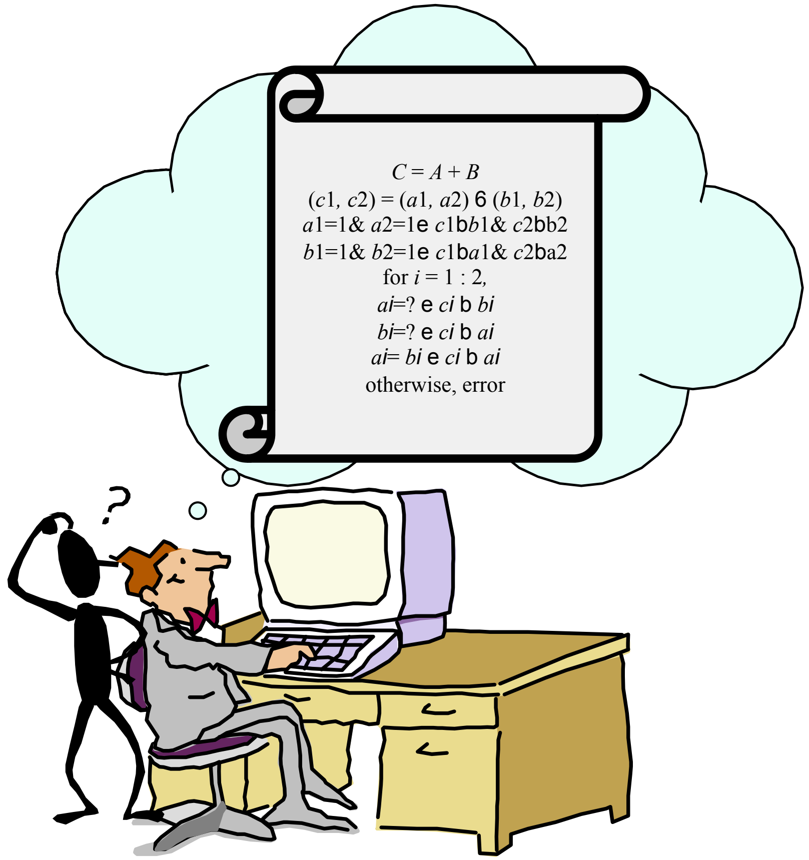
Performance tuning cycle



Performance measurement cycle (2)

- Instrumentation
  - Insertion of extra code (probes) into application
- Measurement
  - Collection of data relevant to performance analysis
- Analysis
  - Calculation of metrics
  - Identification of performance bottlenecks
- Presentation
  - Transformation of the results into a representation that can be easily understood by a human user
- Optimization
  - Elimination of bottlenecks

Semantic gap



```
C = A + B
(c1, c2) = (a1, a2) 6 (b1, b2)
a1=1& a2=1e c1bb1& c2bb2
b1=1& b2=1e c1ba1& c2ba2
for i = 1 : 2,
ai=? e ci b bi
bi=? e ci b ai
ai= bi e ci b ai
otherwise, error
```

```
...
v09,S    [a30,1],m00
a30      -26612:abcd
v12,S    [a31,1],m00
a30      a12+a30
a31      -26616:abcd
v10,S    [a30,1],m00
a16      -22516:abcd
a31      a12+a31
a30      a15+a16
v14,S    [a31,1],m00
a16      -32764:abcd
v11,S    v10-v14,m00
...
```

- Programmer’s mental model of the program does not match the executed version
  - Performance tools needed to bridge this semantic gap

Semantic performance mapping

- Instrumentation levels
  - Source code
  - Library
  - Runtime system
  - Object code
  - Operating system
  - Runtime image
  - Virtual machine
- Problem
  - Every level provides different information
  - Often instrumentation on multiple levels required
- Challenge
  - Mapping performance data onto application-level abstraction

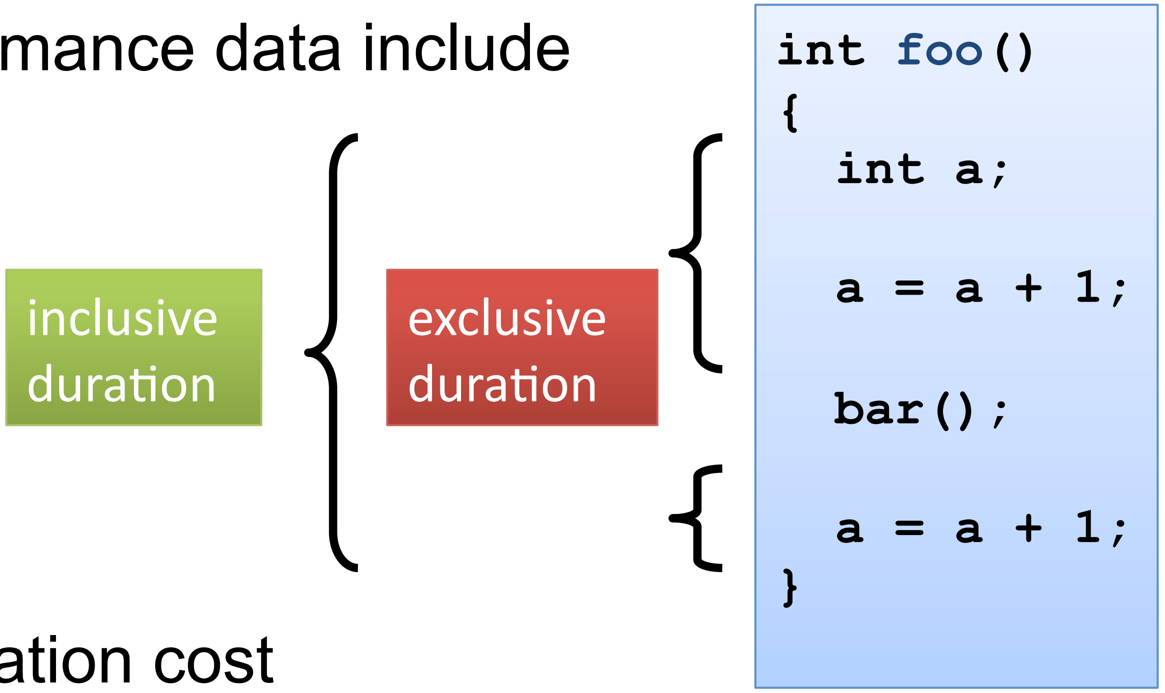
Instrumentation techniques

- Static instrumentation
  - Program is instrumented prior to execution
- Dynamic instrumentation
  - Program is instrumented at runtime
- Code is inserted
  - Manually
  - Automatically
    - By preprocessor
    - By compiler
    - By linking against preinstrumented (interposition) library
    - By binary-rewrite / dynamic instrumentation tool

Measurement

Typical performance data include

- Counts
- Durations



```
int foo()
{
  int a;

  a = a + 1;

  bar();

  a = a + 1;
}
```

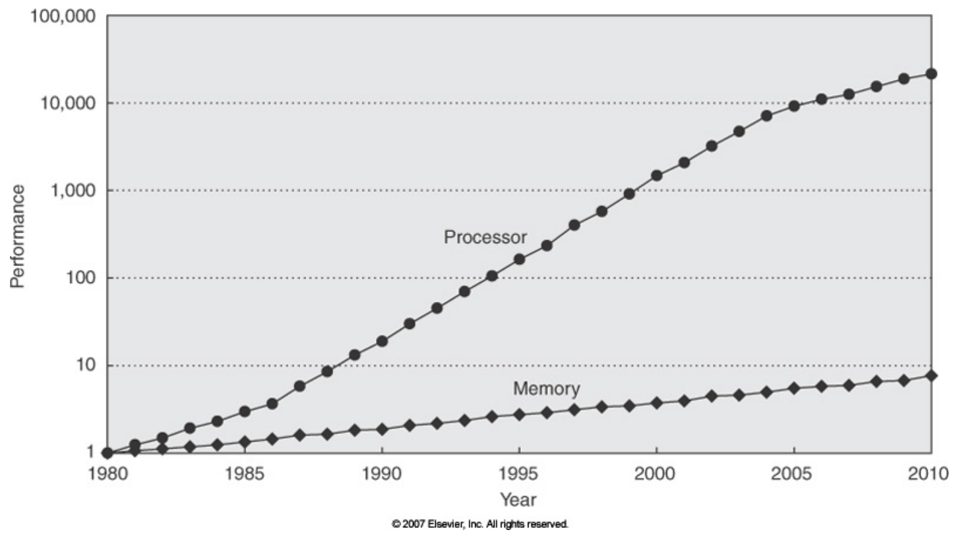
- Communication cost
- Synchronization cost
- IO accesses
- System calls
- Hardware events

Critical issues

- Accuracy
  - Perturbation
    - Measurement alters program behavior
    - E.g., memory access pattern
  - Intrusion overhead
    - Measurement itself needs time and thus lowers performance
  - Accuracy of timers, counters
- Granularity
  - How many measurements
    - Pitfall: short but frequently executed functions
  - How much information / work during each measurement
- Tradeoff
  - Accuracy ⇔ expressiveness of data

Single-node performance

- Huge gap between CPU and memory speed



Source: Hennessy, Patterson: Computer Architecture, 4th edition, Morgan Kaufmann

- Internal operation of a microprocessor potentially complex
  - Pipelining
  - Out-of-order instruction issuing
  - Branch prediction
  - Non-blocking caches

Hardware counters

- Small set of registers that count events
- Events are signals related to the processor’s internal function
- Original purpose: design verification and performance debugging for microprocessors
- Idea: use this information to analyze the performance behavior of an application as opposed to a CPU

Typical hardware counters

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

Profiling

- Mapping of aggregated information
  - Time
  - Counts
    - Calls
    - Hardware counters
- Onto program and system entities
  - Functions, loops, call paths
  - Processes, threads

- Methods to create a profile
  - PC sampling (statistical approach)
  - Interval timer / direct measurement (deterministic approach)

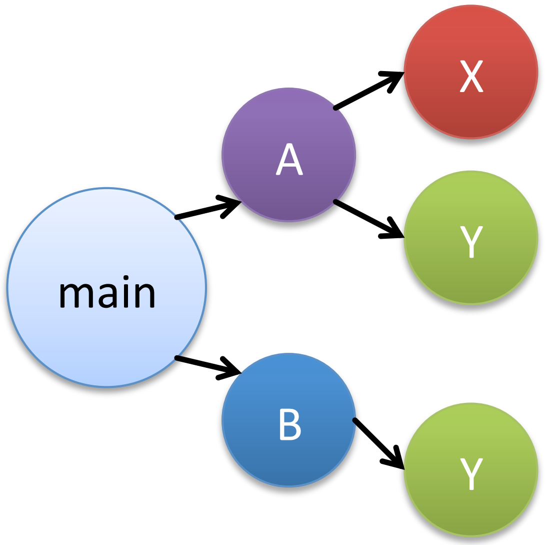
Profiling (2)

- Sampling
  - Statistical measurement technique
    - Based on the assumption that a subset of a population being examined is representative for the whole population
    - Requires long-running programs
  - Periodic operating system signal interrupts the running program
  - Interrupt service routine examines return-address stack to find address of instruction being executed when interrupt occurred
  - Using symbol-table information this address is mapped onto specific subroutine
- Interval timing
  - Time measurement at beginning and end of a code region
  - Requires high-resolution / low-overhead clock

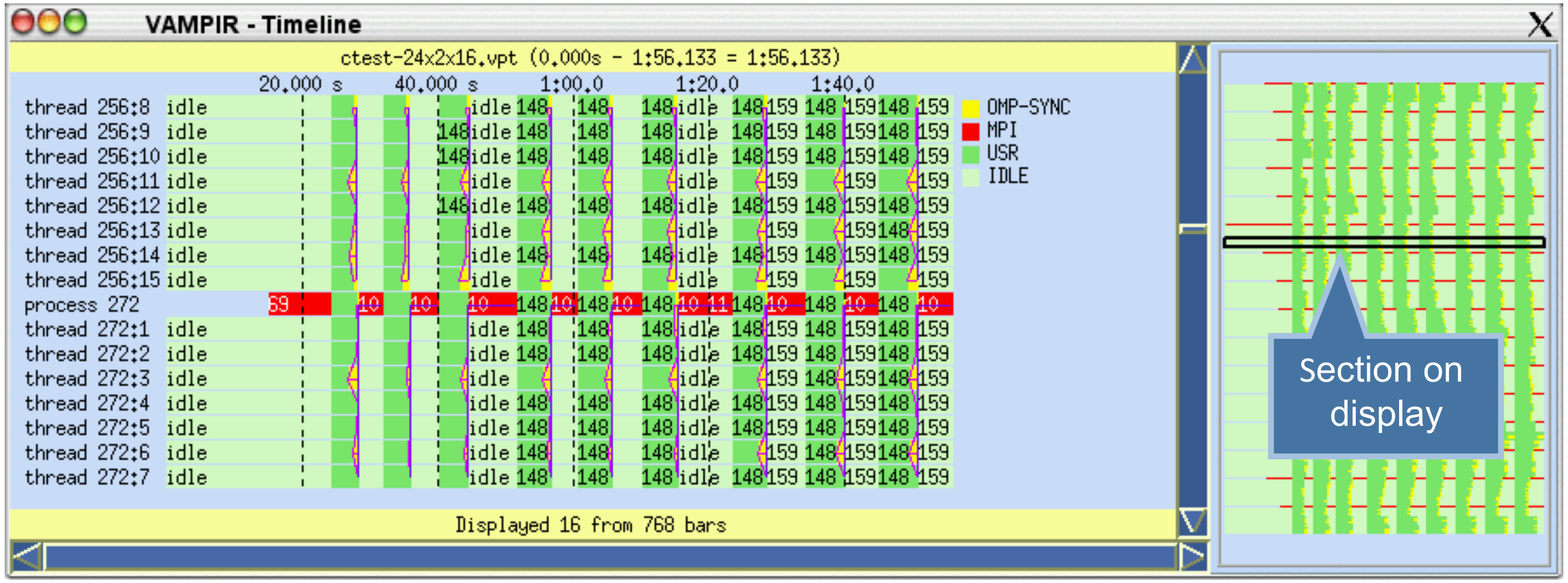
Call-path profiling

- Behavior of a function may depend on caller (i.e., parameters)
- Flat function profile often not sufficient
- How to determine call path at runtime?
  - Runtime stack walk
  - Maintain shadow stack
    - Requires tracking of function calls

```
main()
{
  A( );
  B( );
}


A( )      B( )
{         {
  X();      Y();
  Y();    }
}
```



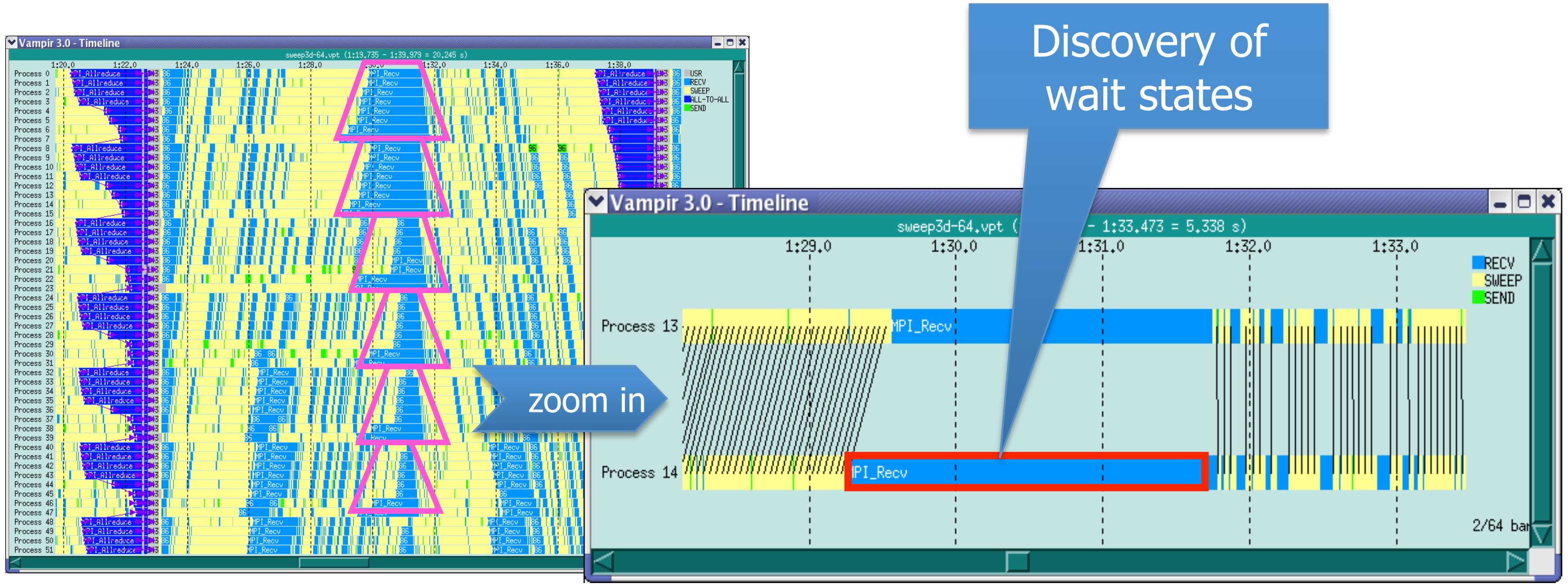
Event tracing



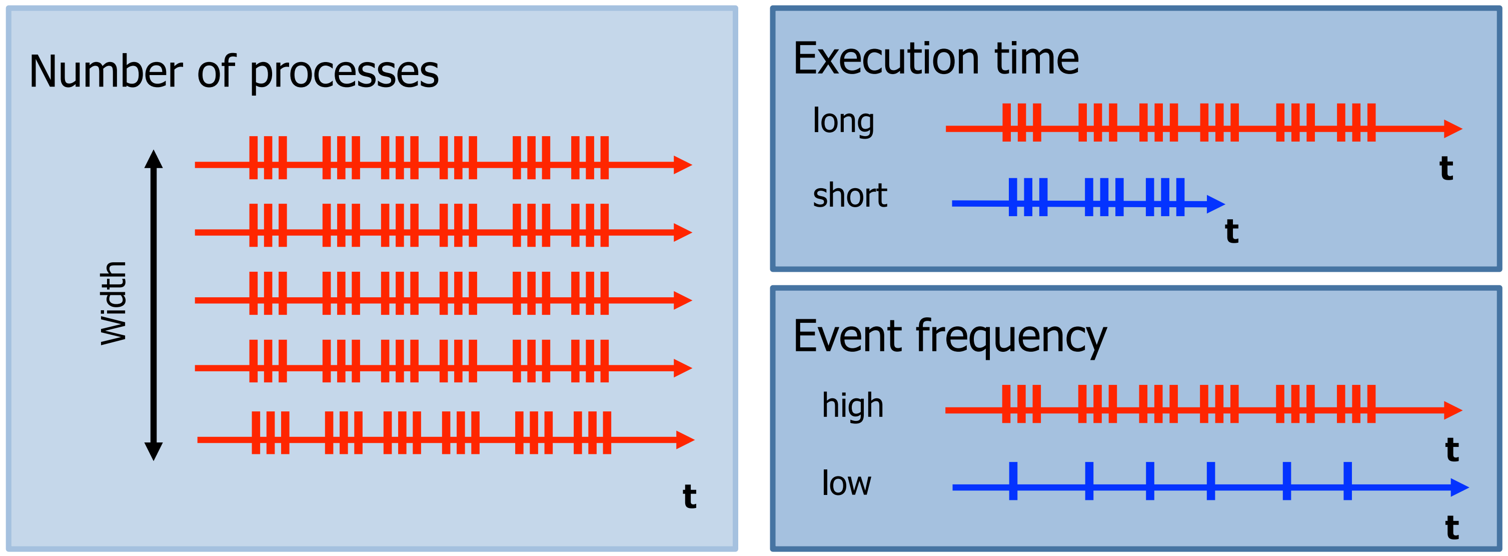
- Typical events
  - Entering and leaving a function
  - Sending and receiving a message

Why tracing?

- High level of detail
- Allows in-depth post-mortem analysis of program behavior
  - Time-line visualization
  - Automatic pattern search
- Identification of wait states



Obstacle: trace size



- Problem: width and length of event trace

Tracing vs. profiling

- Advantages of tracing
  - Event traces preserve the temporal and spatial relationships among individual events
  - Allows reconstruction of dynamic behavior of application on any required abstraction level
  - Most general measurement technique
    - Profile data can be constructed from event traces
- Disadvantages
  - Traces can become very large
  - Writing events to a file at runtime can cause perturbation
  - Writing tracing software is complicated
    - Event buffering, clock synchronization, …

scalasca

- Scalable performance-analysis toolset for parallel codes
  - Focus on communication & synchronization
- Integrated performance analysis process
  - Performance overview on call-path level via call-path profiling
  - In-depth study of application behavior via event tracing
- Supported programming models
  - MPI-1, MPI-2 one-sided communication
  - OpenMP (basic features)
- Available for all major HPC platforms

Joint project of

JÜLICH
FORSCHUNGSZENTRUM

German Research School
for Simulation Sciences

The team

www.scalasca.org

Scalasca

http://www.scalasca.org/

LEO Dictionary.com SPIEGEL Handelsblatt Wikipedia (DE) Wikipedia (EN) Google Scholar Google Maps

Home | Imprint

scalasca

Search...

About Download Team Publications Projects News Contact

Scalasca

Scalasca is a software tool that supports the performance optimization of parallel programs by measuring and analyzing their runtime behavior. The analysis identifies potential performance bottlenecks – in particular those concerning communication and synchronization – and offers guidance in exploring their causes.

more...

News

7th VI-HPS Tuning Workshop

HLRS, Stuttgart/Germany, March 28-30, 2011 Three-day hands-on workshop covering the... more...

Scalasca at SC'10

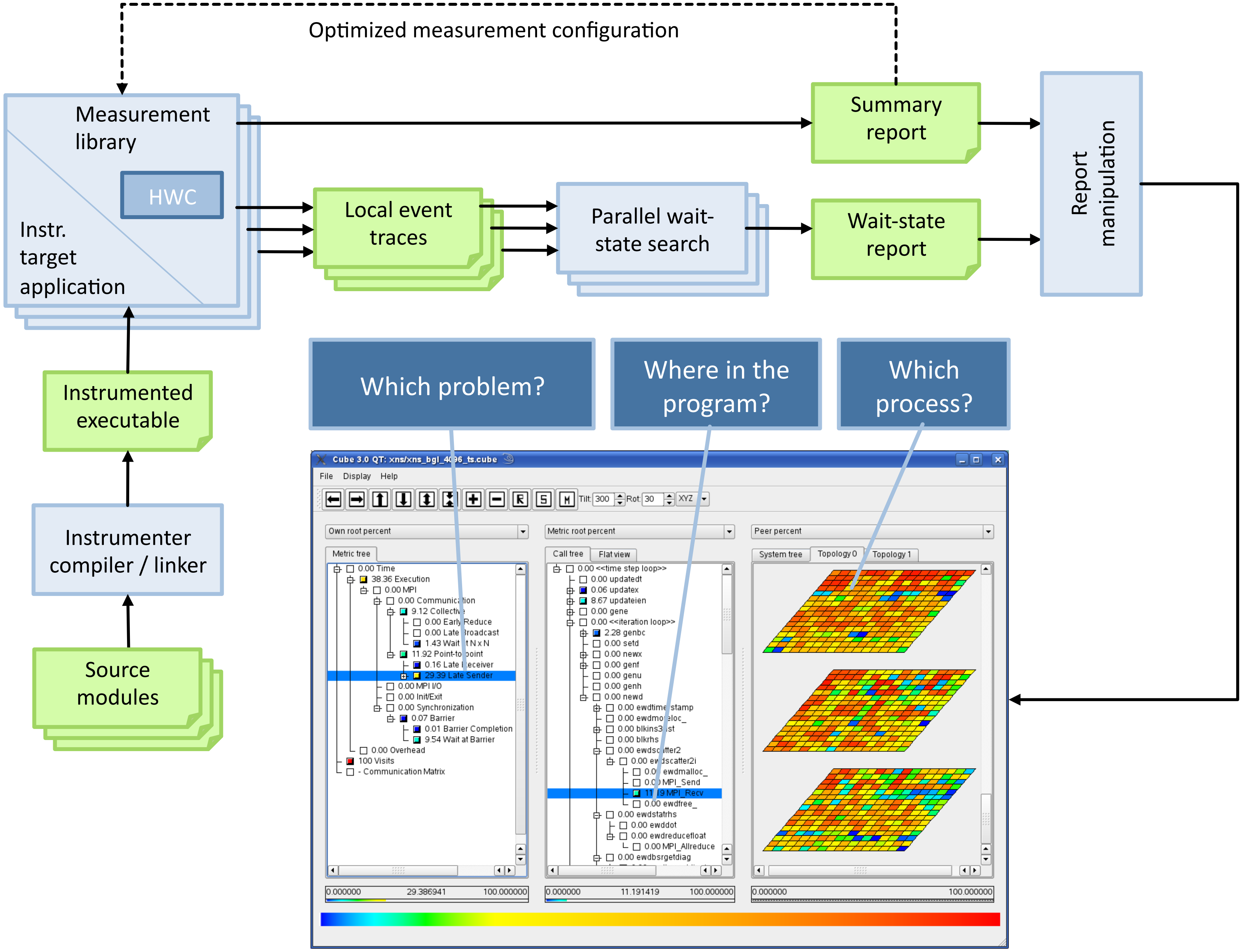
November 13-19, 2010: Join us at SC'10 in New Orleans, LA, USA. Scalasca team... more...

JÜLICH FORSCHUNGSZENTRUM

German Research School for Simulation Sciences

Installations and users

- Companies
  - Bull (France)
  - Dassault Aviation (France)
  - Efield Solutions (Sweden)
  - GNS (Germany)
  - INTES (Germany)
  - MAGMA (Germany)
  - RECOM (Germany)
  - SciLab (France)
  - Shell (Netherlands)
  - Sun Microsystems (USA, Singapore, India)
  - Qontix (UK)
- Research/supercomputing centers
  - ANL (USA)BSC (Spain)
  - CASPUR (Italy)
  - CEA (France)
  - CERFACS (France)
  - CINECA (Italy)
  - CSC (Finland)
  - CSCS (Switzerland)
  - DLR (Germany)
  - DKRZ (Germany)
  - EPCC (UK)
  - FZJ (Germany)
  - HLRN (Germany)
  - HLRS (Germany)
  - ICHEC (Ireland)
  - IDRIS (France)
  - KIT (Germany)
  - LLNL (USA)
- Research/supercomputing centers (cont.)
  - LRZ (Germany)
  - MCH (Switzerland)
  - NCAR (USA)
  - NCSA (USA)
  - ORNL (USA)
  - PIK (Germany)
  - PSC (USA)
  - RZG (Germany)
  - SARA (Netherlands)
  - SAITC (Bulgaria)
  - TACC (USA)
- Universities
  - Lund University (Sweden)
  - MSU (Russia)
  - RPI (USA)
  - RWTH (Germany)
  - TUD (Germany)
  - UOregon (USA)
  - UTK (USA)
- DoD/MoD computing centers
  - AFRL DSRC (USA)
  - ARL DSRC (USA)
  - ARSC DSRC (USA)
  - AWE (UK)
  - ERDC DSRC (USA)
  - Navy DSRC (USA)
  - MHPCC DSRC (USA)
  - SSC-Pacific (USA)
  - MetOffice (UK)



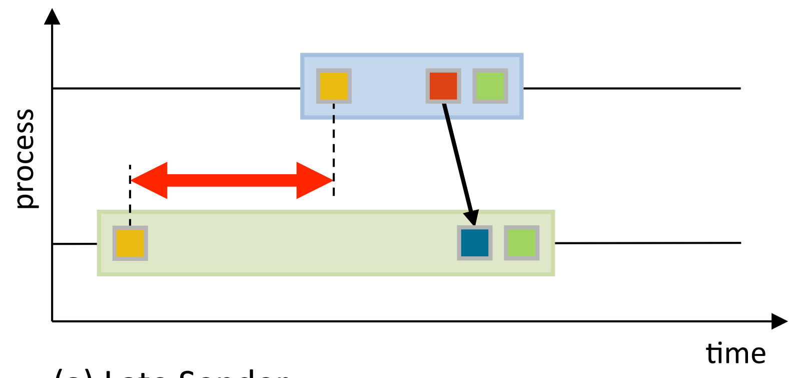
```
0.00 Time
  38.36 Execution
    0.00 MPI
      0.00 Communication
        9.12 Collective
          0.00 Early Reduce
          0.00 Late Broadcast
          1.43 Wait at N x N
        11.92 Point-to-point
          0.16 Late Receiver
          29.39 Late Sender
      0.00 MPI I/O
      0.00 Init/Exit
      0.00 Synchronization
        0.07 Barrier
          0.01 Barrier Completion
          9.54 Wait at Barrier
    0.00 Overhead
  100 Visits
  - Communication Matrix
```

```
0.00 <<time step loop>>
  0.00 updatedt
  0.06 updatex
  8.67 updateien
  0.00 gene
  0.00 <<iteration loop>>
    2.28 genbc
    0.00 setd
    0.00 newx
    0.00 genf
    0.00 genu
    0.00 genh
    0.00 newd
      0.00 ewdtimestamp
      0.00 ewdmoveloc_
      0.00 blkins3dst
      0.00 blkrhs
      0.00 ewdscatter2
        0.00 ewdscatter2i
          0.00 ewdmalloc_
          0.00 MPI_Send
          11.19 MPI_Recv
          0.00 ewdfree_
      0.00 ewdstatrhs
        0.00 ewddot
        0.00 ewdreducefloat
          0.00 MPI_Allreduce
      0.00 ewdbsrgetdiag
```

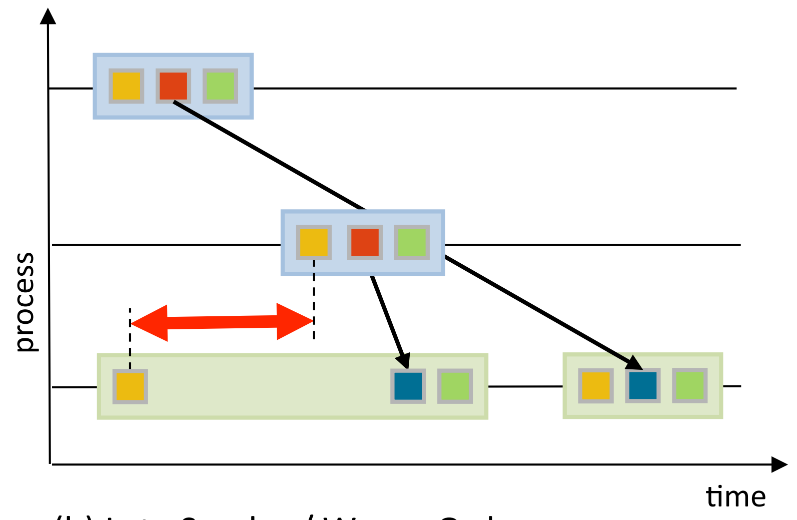
```
0.000000        29.386941     100.000000
0.000000        11.191419     100.000000
0.000000                      100.000000
```

Wait-state analysis

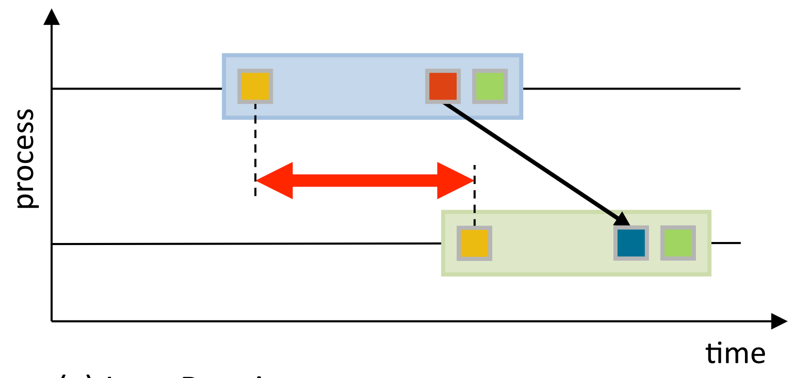
- Classification
- Quantification



(a) Late Sender



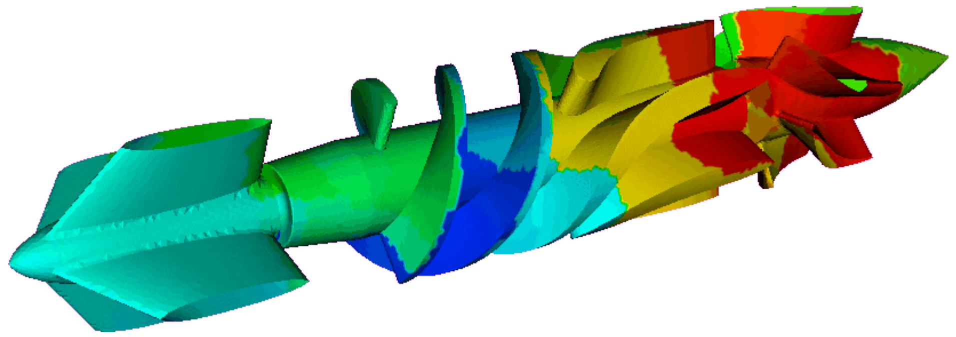
(b) Late Sender / Wrong Order



(c) Late Receiver

XNS CFD simulation application

- Computational fluid dynamics code
  - Developed by Chair for Computational Analysis of Technical Systems, RWTH Aachen University
  - Finite-element method on unstructured 3D meshes
  - Parallel implementation based on message passing
  - >40,000 lines of Fortran & C
  - DeBakey blood pump test case
    - Scalability of original version limited <1024 CPUs

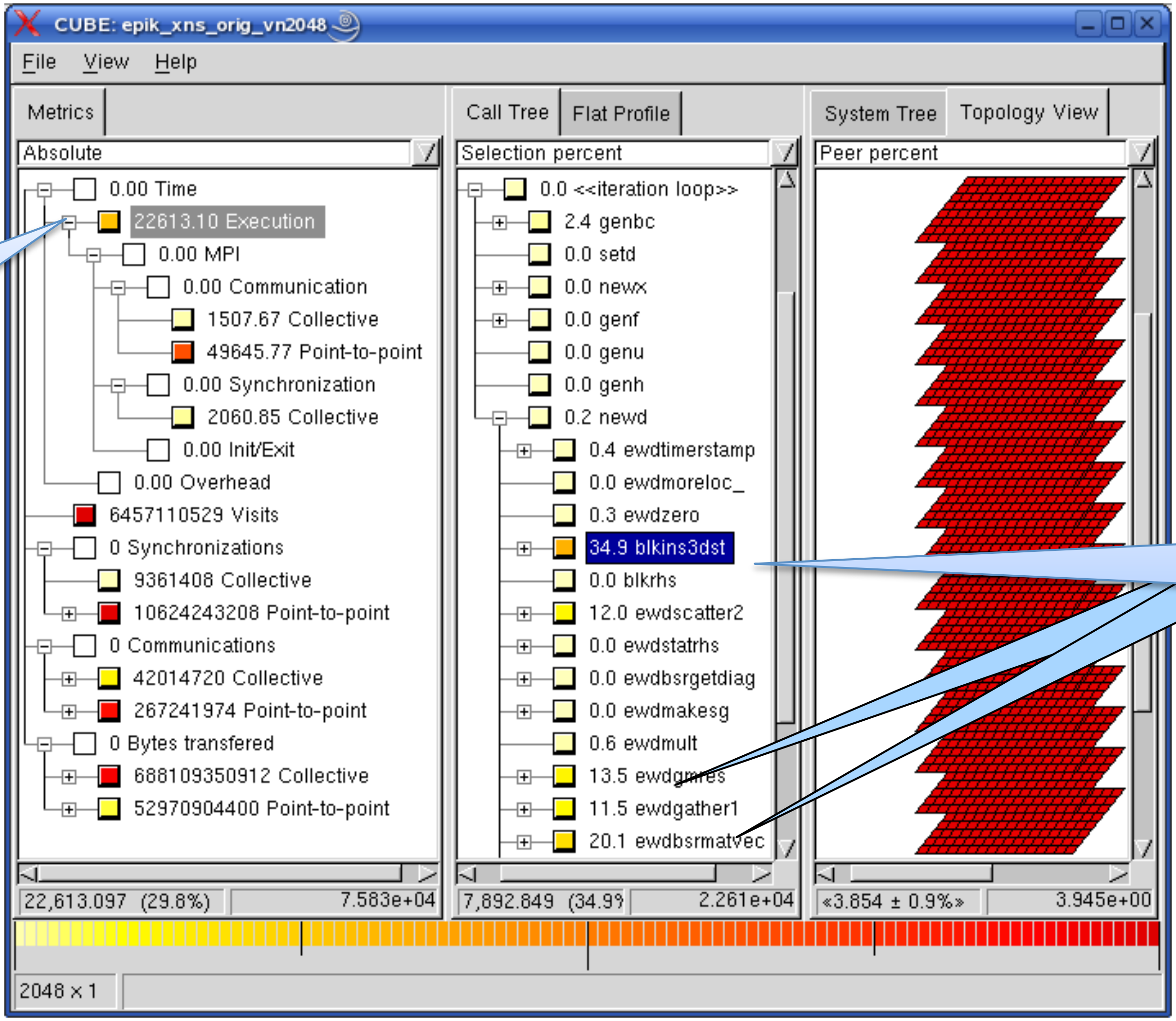

Partitioned finite-element mesh

Call-path profile: Computation

Execution time excl. MPI comm

Just 30% of simulation

Widely spread in code



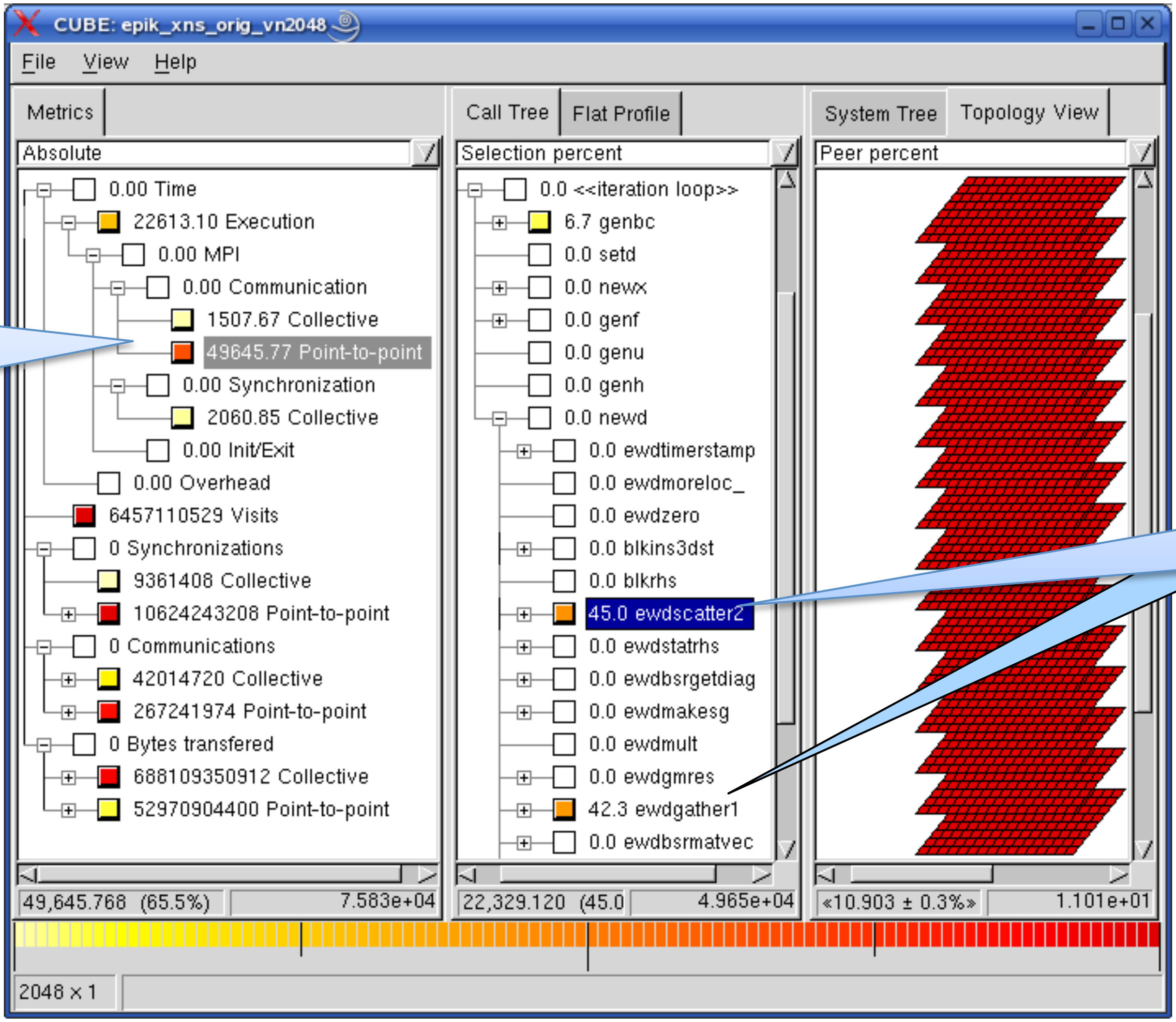
```
0.00 Time                          0.0 <<iteration loop>>
  22613.10 Execution                 2.4 genbc
    0.00 MPI                         0.0 setd
      0.00 Communication             0.0 newx
        1507.67 Collective           0.0 genf
        49645.77 Point-to-point      0.0 genu
      0.00 Synchronization           0.0 genh
        2060.85 Collective           0.2 newd
      0.00 Init/Exit                   0.4 ewdtimerstamp
  0.00 Overhead                        0.0 ewdmoreloc_
6457110529 Visits                      0.3 ewdzero
0 Synchronizations                     34.9 blkins3dst
  9361408 Collective                   0.0 blkrhs
  10624243208 Point-to-point           12.0 ewdscatter2
0 Communications                       0.0 ewdstatrhs
  42014720 Collective                  0.0 ewdbsrgetdiag
  267241974 Point-to-point             0.0 ewdmakesg
0 Bytes transfered                     0.6 ewdmult
  688109350912 Collective              13.5 ewdgmres
  52970904400 Point-to-point           11.5 ewdgather1
                                       20.1 ewdbsrmatvec

22,613.097 (29.8%)   7.583e+04 | 7,892.849 (34.9%)   2.261e+04 | «3.854 ± 0.9%»   3.945e+00

2048 x 1
```

Call-path profile: P2P messaging

MPI point-to-point communic-ation time

P2P comm 66% of simulation

Primarily in scatter & gather



```
0.00 Time
  22613.10 Execution
    0.00 MPI
      0.00 Communication
        1507.67 Collective
        49645.77 Point-to-point
      0.00 Synchronization
        2060.85 Collective
      0.00 Init/Exit
  0.00 Overhead
6457110529 Visits
0 Synchronizations
  9361408 Collective
  10624243208 Point-to-point
0 Communications
  42014720 Collective
  267241974 Point-to-point
0 Bytes transfered
  688109350912 Collective
  52970904400 Point-to-point

49,645.768 (65.5%)    7.583e+04
```

```
0.0 <<iteration loop>>
  6.7 genbc
  0.0 setd
  0.0 newx
  0.0 genf
  0.0 genu
  0.0 genh
  0.0 newd
    0.0 ewdtimerstamp
    0.0 ewdmoreloc_
    0.0 ewdzero
    0.0 blkins3dst
    0.0 blkrhs
    45.0 ewdscatter2
    0.0 ewdstatrhs
    0.0 ewdbsrgetdiag
    0.0 ewdmakesg
    0.0 ewdmult
    0.0 ewdgmres
    42.3 ewdgather1
    0.0 ewdbsrmatvec

22,329.120 (45.0    4.965e+04
```

```
«10.903 ± 0.3%»    1.101e+01
```

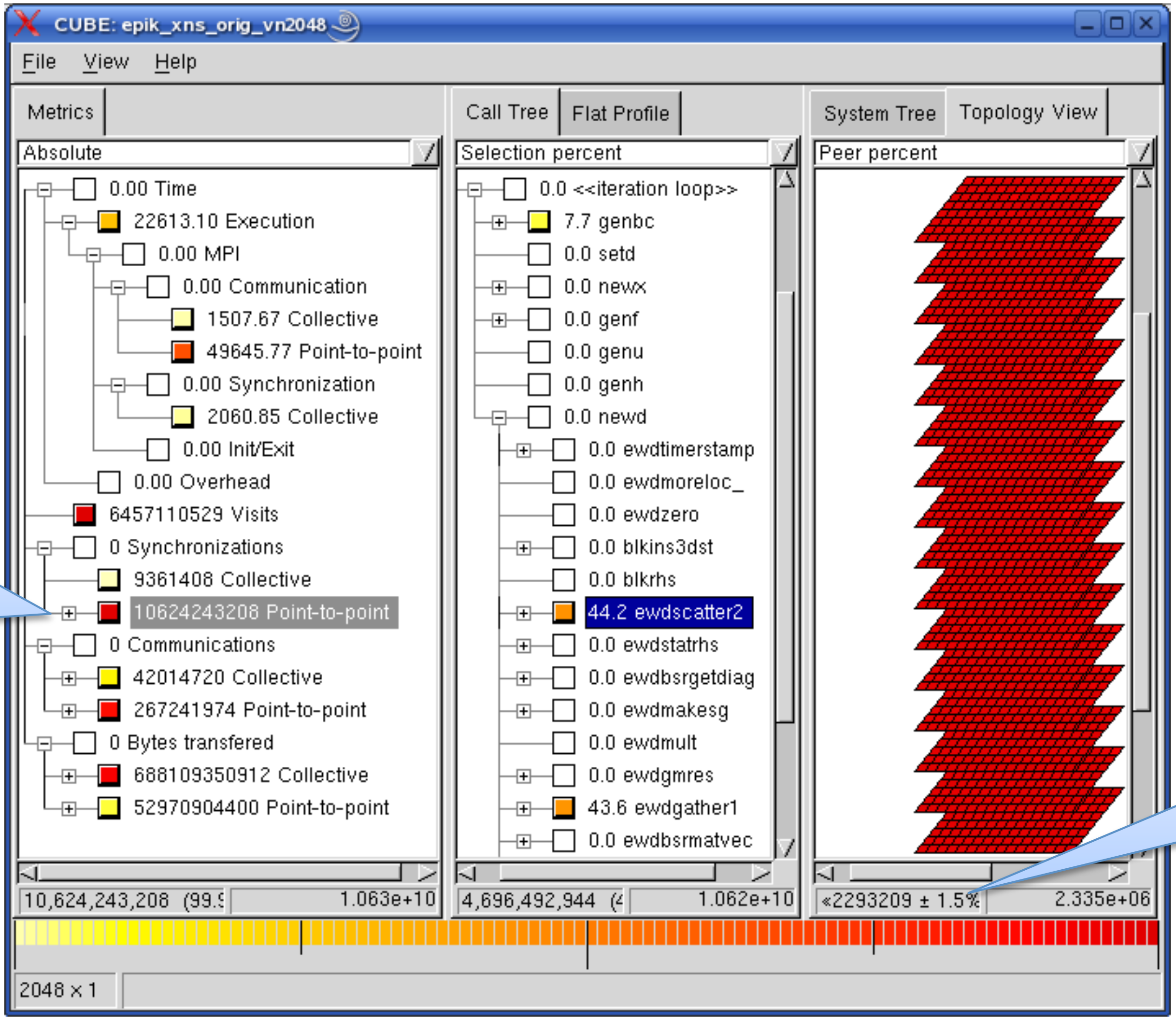
```
2048 x 1
```

Call-path profile: P2P sync. ops.

Point-to-point msgs w/o data

Masses of P2P sync. operations

Processes all equally responsible



CUBE: epik_xns_orig_vn2048

File View Help

Metrics | Call Tree | Flat Profile | System Tree | Topology View

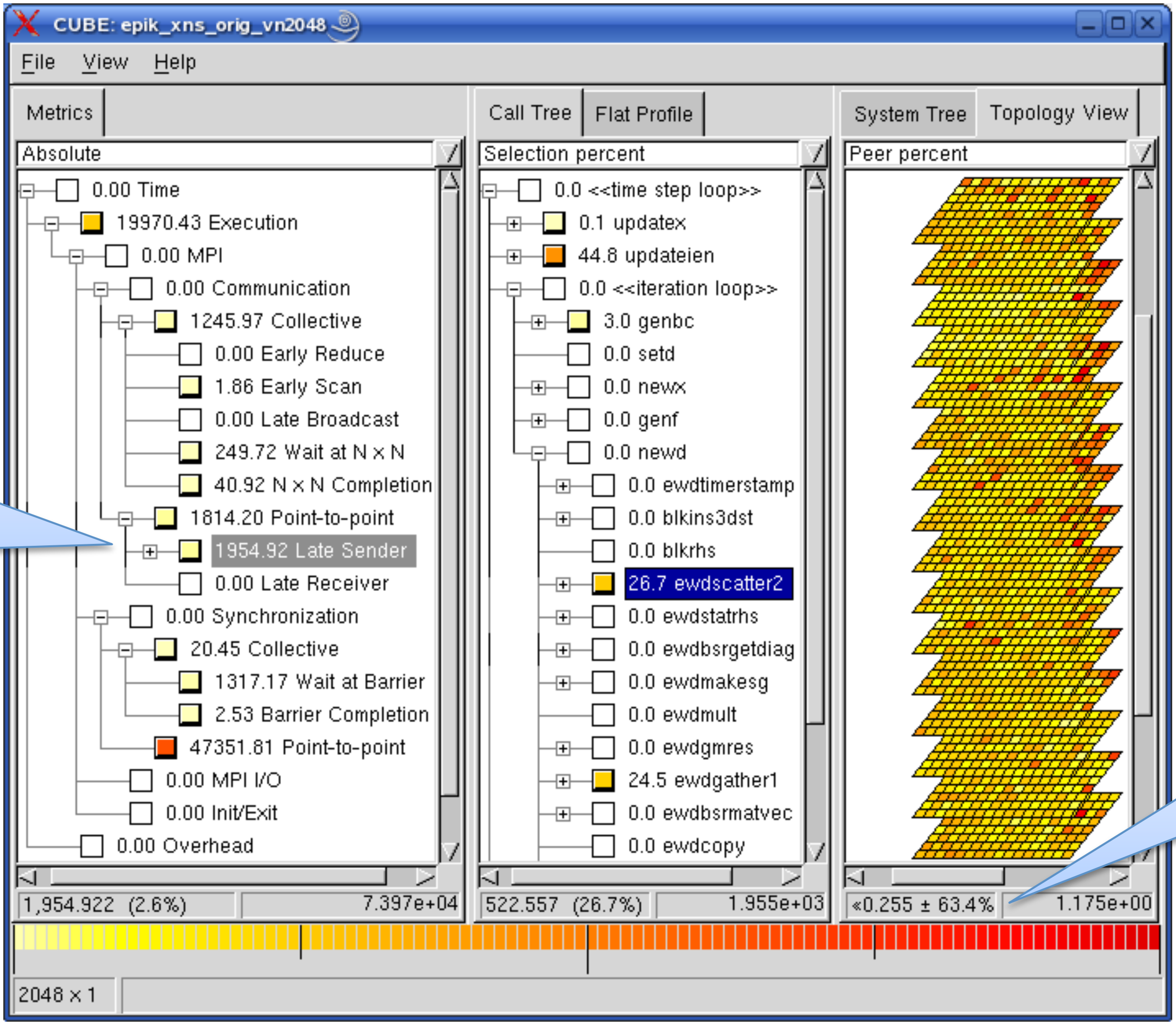
```
Absolute                              Selection percent               Peer percent
0.00 Time                             0.0 <<iteration loop>>
  22613.10 Execution                    7.7 genbc
    0.00 MPI                            0.0 setd
      0.00 Communication                0.0 newx
        1507.67 Collective              0.0 genf
        49645.77 Point-to-point         0.0 genu
      0.00 Synchronization              0.0 genh
        2060.85 Collective              0.0 newd
      0.00 Init/Exit                      0.0 ewdtimerstamp
  0.00 Overhead                           0.0 ewdmoreloc_
6457110529 Visits                         0.0 ewdzero
0 Synchronizations                        0.0 blkins3dst
  9361408 Collective                      0.0 blkrhs
  10624243208 Point-to-point              44.2 ewdscatter2
0 Communications                          0.0 ewdstatrhs
  42014720 Collective                     0.0 ewdbsrgetdiag
  267241974 Point-to-point                0.0 ewdmakesg
0 Bytes transfered                        0.0 ewdmult
  688109350912 Collective                 0.0 ewdgmres
  52970904400 Point-to-point              43.6 ewdgather1
                                          0.0 ewdbsrmatvec

10,624,243,208 (99.9  1.063e+10   4,696,492,944 (4  1.062e+10   «2293209 ± 1.5%  2.335e+06

2048 x 1
```

Trace analysis: Late sender

Wait time of receivers blocked for late sender

Half of the send time is waiting

Significant process imbalance



CUBE: epik_xns_orig_vn2048

File View Help

Metrics | Absolute

```
0.00 Time
  19970.43 Execution
    0.00 MPI
      0.00 Communication
        1245.97 Collective
          0.00 Early Reduce
          1.86 Early Scan
          0.00 Late Broadcast
          249.72 Wait at N x N
          40.92 N x N Completion
        1814.20 Point-to-point
          1954.92 Late Sender
          0.00 Late Receiver
      0.00 Synchronization
        20.45 Collective
          1317.17 Wait at Barrier
          2.53 Barrier Completion
        47351.81 Point-to-point
      0.00 MPI I/O
      0.00 Init/Exit
  0.00 Overhead
```

1,954.922 (2.6%) 7.397e+04

Call Tree | Flat Profile | Selection percent

```
0.0 <<time step loop>>
  0.1 updatex
  44.8 updateien
  0.0 <<iteration loop>>
    3.0 genbc
    0.0 setd
    0.0 newx
    0.0 genf
    0.0 newd
      0.0 ewdtimerstamp
      0.0 blkins3dst
      0.0 blkrhs
      26.7 ewdscatter2
      0.0 ewdstatrhs
      0.0 ewdbsrgetdiag
      0.0 ewdmakesg
      0.0 ewdmult
      0.0 ewdgmres
      24.5 ewdgather1
      0.0 ewdbsrmatvec
      0.0 ewdcopy
```

522.557 (26.7%) 1.955e+03

System Tree | Topology View | Peer percent

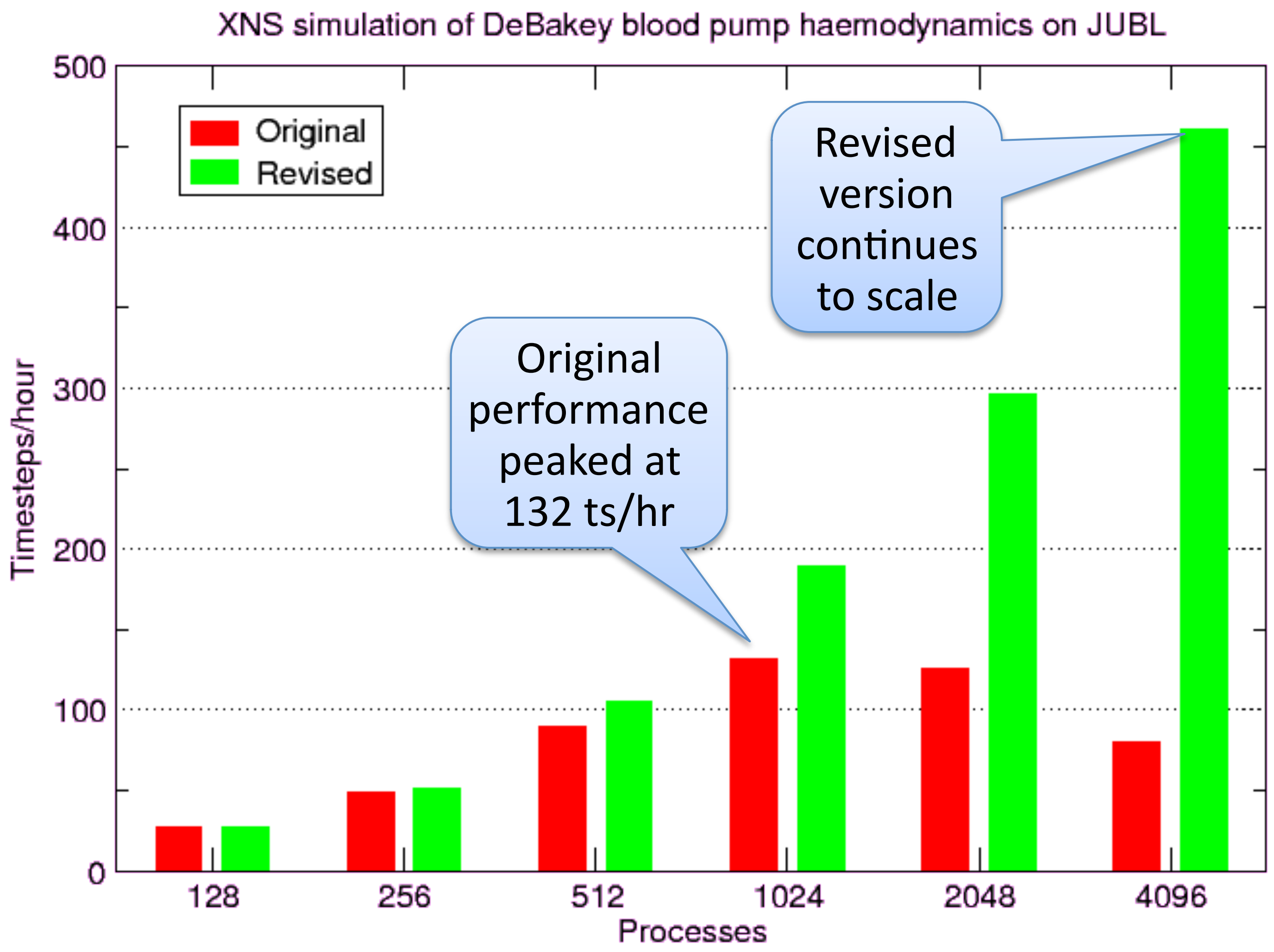
«0.255 ± 63.4% 1.175e+00

2048 x 1

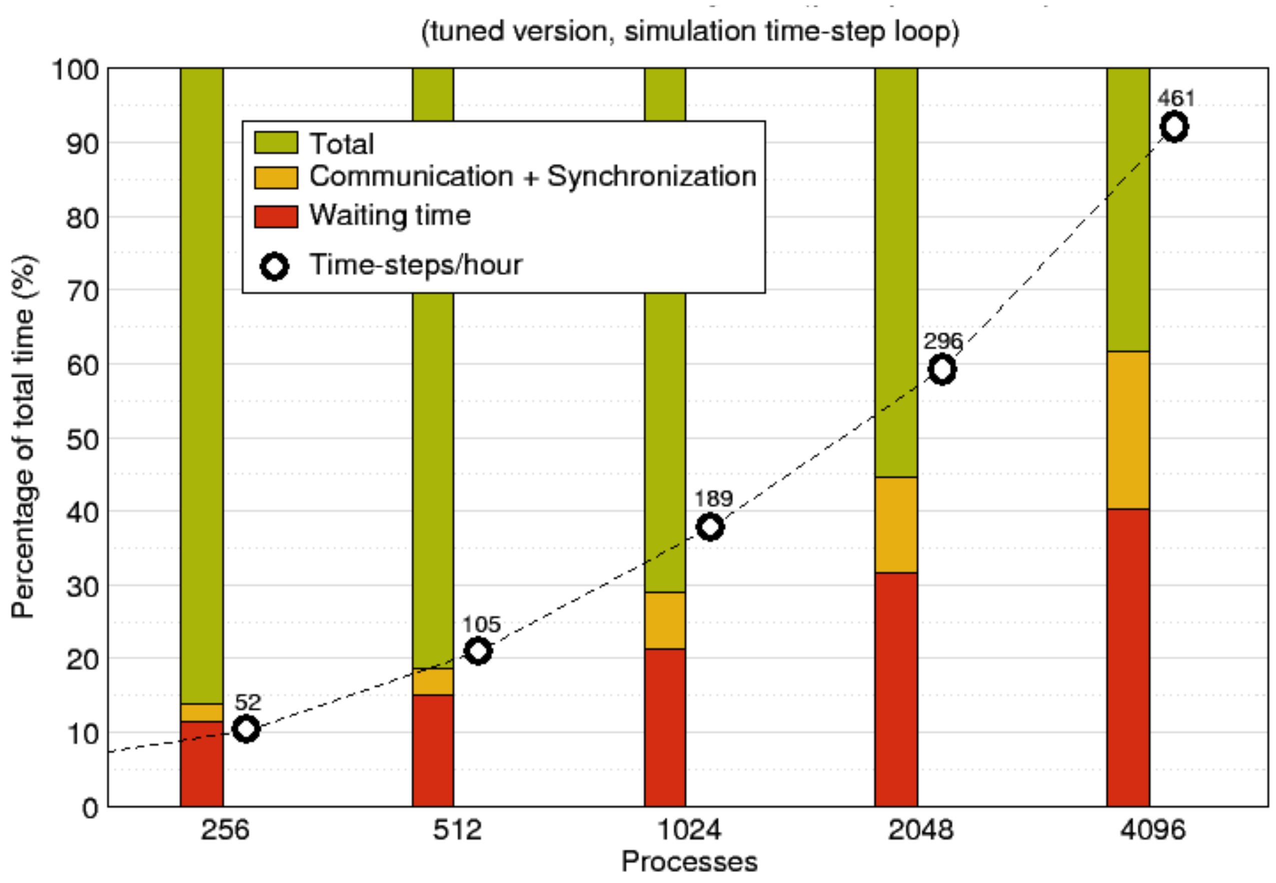
XNS scalability remediation

- Review of original XNS
  - Computation is well balanced
  - Real communication is very imbalanced
  - Huge amounts of P2P synchronisations
    - Grow exponentially with number of processes
- Elimination of redundant messages
  - Relevant neighbor partitions known in advance from static mesh partitioning
  - Most transfers still required at small scale while connectivity is relatively dense
  - Growing benefits at larger scales (>512)

After removal of redundant messages

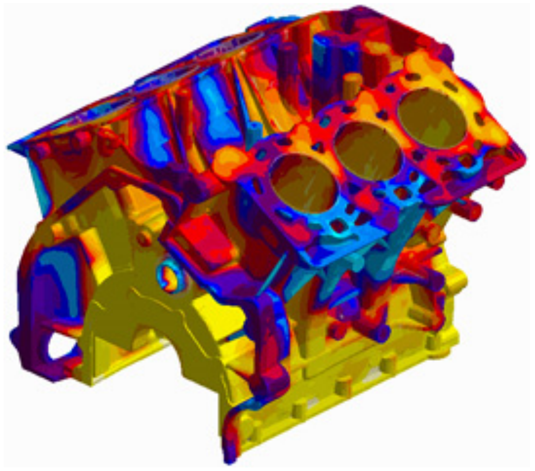


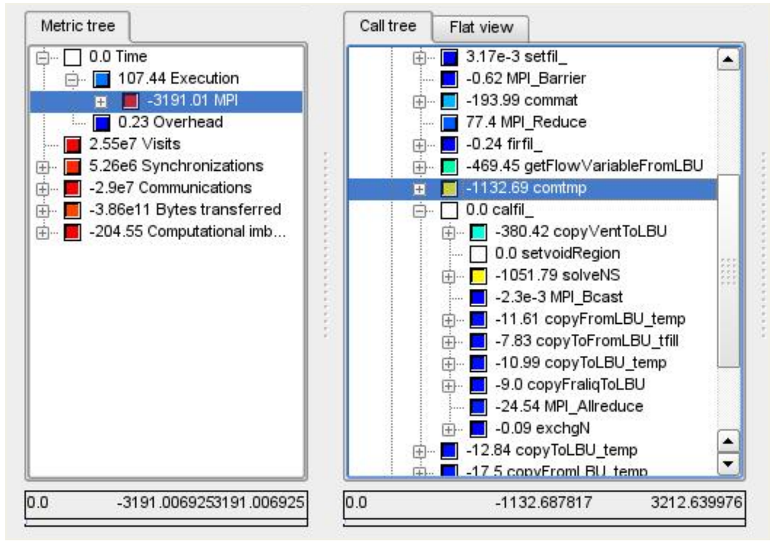
XNS wait-state analysis of tuned version



MAGMAfill by MAGMASOFT® GmbH

- Simulates mold-filling in casting processes
- Scalasca used
  - To identify communication bottleneck
  - To compare alternatives using performance algebra utility
- 23% overall runtime improvement



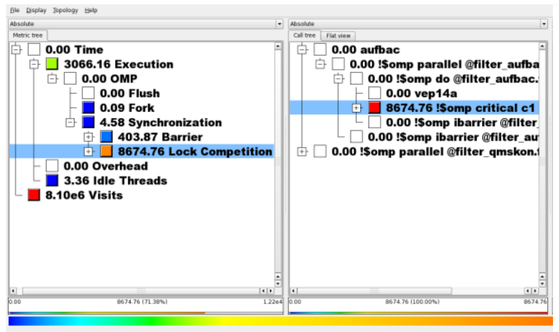


```
Metric tree
0.0 Time
  107.44 Execution
    -3191.01 MPI
  0.23 Overhead
2.55e7 Visits
5.26e6 Synchronizations
-2.9e7 Communications
-3.86e11 Bytes transferred
-204.55 Computational imb...

0.0        -3191.0069253191.006925
```

```
Call tree   Flat view
3.17e-3 setfil_
-0.62 MPI_Barrier
-193.99 commat
77.4 MPI_Reduce
-0.24 firfil_
-469.45 getFlowVariableFromLBU
-1132.69 comtmp
0.0 calfil_
  -380.42 copyVentToLBU
  0.0 setvoidRegion
  -1051.79 solveNS
  -2.3e-3 MPI_Bcast
  -11.61 copyFromLBU_temp
  -7.83 copyToFromLBU_tfill
  -10.99 copyToLBU_temp
  -9.0 copyFraliqToLBU
  -24.54 MPI_Allreduce
  -0.09 exchgN
-12.84 copyToLBU_temp
-17.5 copyFromLBU_temp

0.0        -1132.687817        3212.639976
```

INDEED by GNS® mbh

- Finite-element code for the simulation of material-forming processes
  - Focus on creation of element-stiffness matrix
- Tool workflow
  - Scalasca identified serialization in critical section as bottleneck
  - In-depth analysis using Vampir
- Speedup of 30-40% after optimization

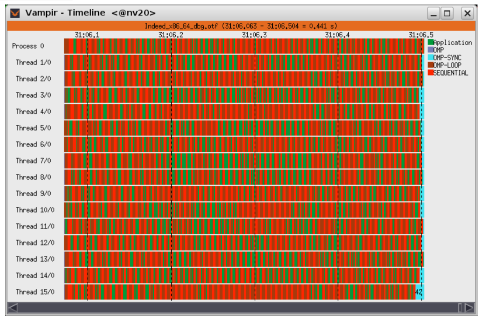


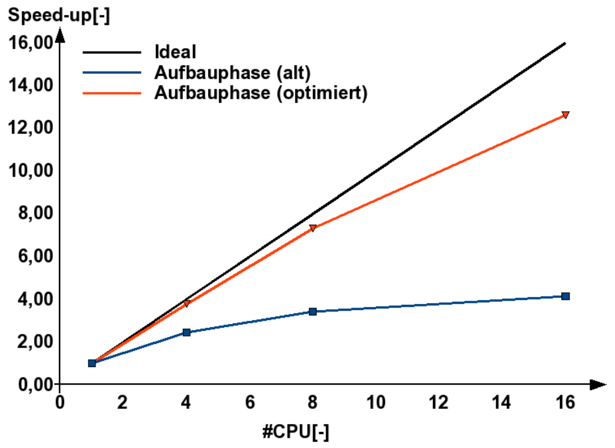
```
0.00 Time
  3066.16 Execution
    0.00 OMP
      0.00 Flush
      0.09 Fork
      4.58 Synchronization
        403.87 Barrier
        8674.76 Lock Competition
  0.00 Overhead
  3.36 Idle Threads
8.10e6 Visits
```

```
0.00 aufbac
  0.00 !$omp parallel @filter_aufba
    0.00 !$omp do @filter_aufbac.
      0.00 vep14a
      8674.76 !$omp critical c1
      0.00 !$omp ibarrier @filter_
    0.00 !$omp ibarrier @filter_au
0.00 !$omp parallel @filter_qmskon.
```

0.00 8674.76 (71.38%) 1.22e4

0.00 8674.76 (100.00%) 8674.76

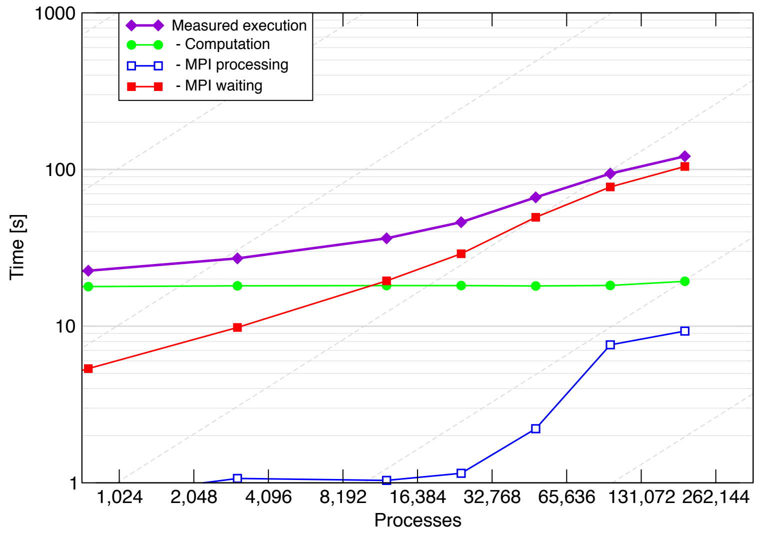


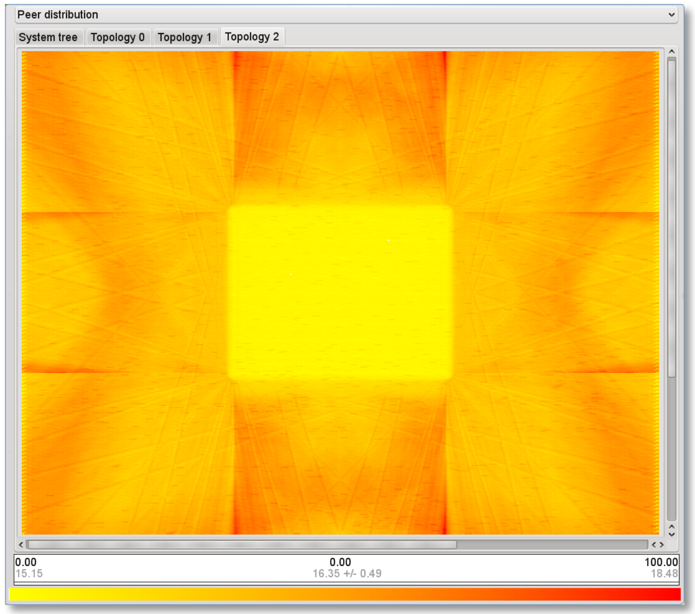


Scalability in terms of the number of cores

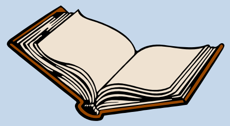
- Application study of ASCI Sweep3D benchmark
- Identified MPI waiting time correlating with computational imbalance
- Measurements & analyses demonstrated on
  - Jaguar with up to 192k cores
  - Jugene with up to 288k cores

Jaguar, MK = 10 (default)





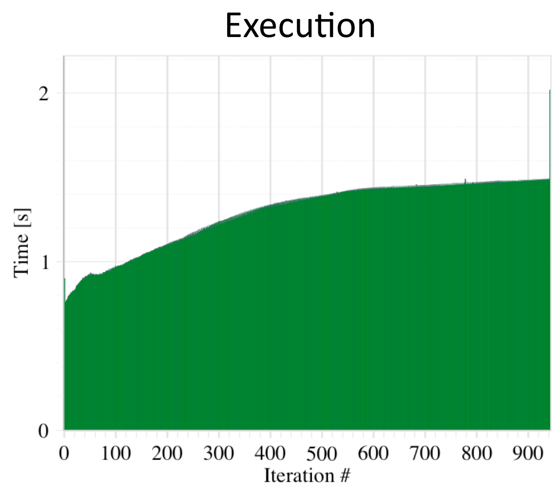
Computation

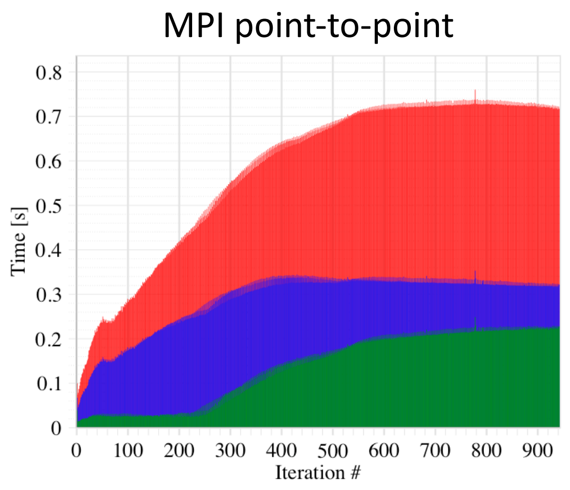


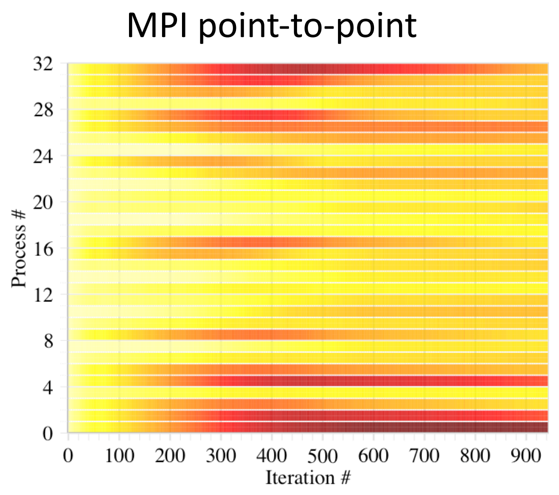
Brian J.N. Wylie et al.: Large-scale performance analysis of Sweep3D with the Scalasca toolset. Parallel Processing Letters, 20(4):397-414, December 2010.

Performance dynamics

- Most simulation codes work iteratively
- Growing complexity of codes makes performance behavior more dynamic – even in the absence of failures
  - Periodic extra activities
  - Adaptation to changing state of computation
- External influence (e.g., dynamic reconfiguration)

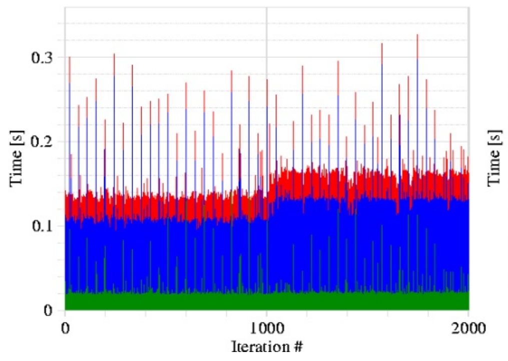




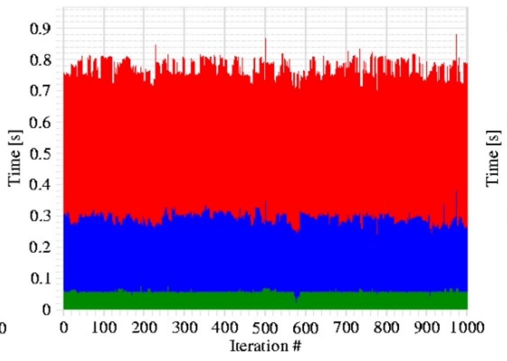


129.tera_tf

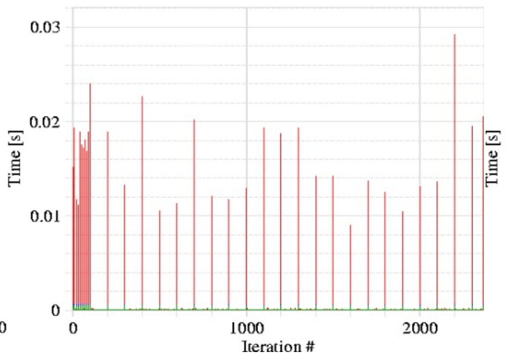
P2P communication in SPEC MPI 2007 suite



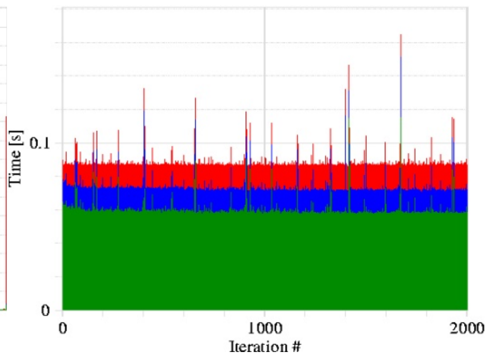
107.leslie3d



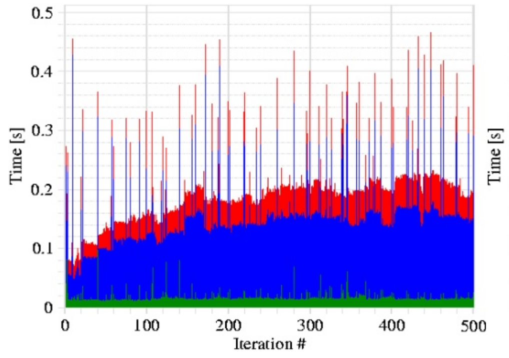
113.GemsFDTD



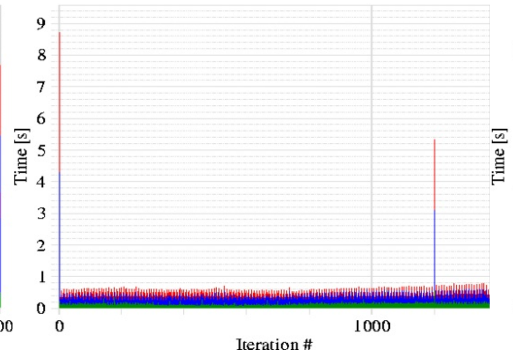
115.fds4



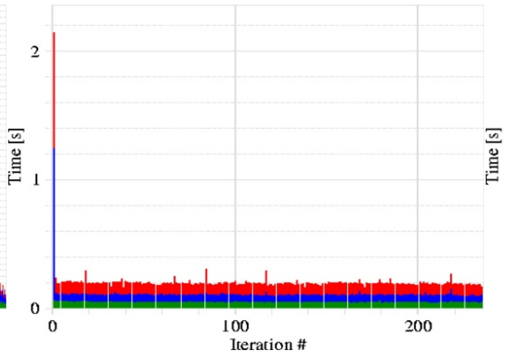
121.pop2



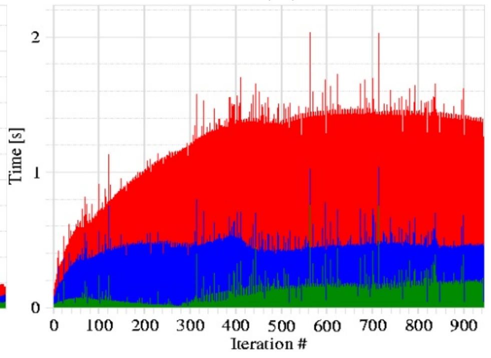
126.leslie3d



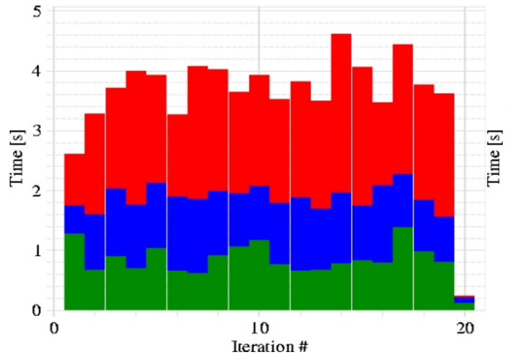
127.wrf2



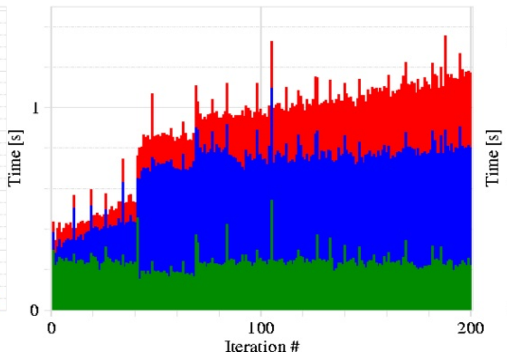
128.GAPgeofem



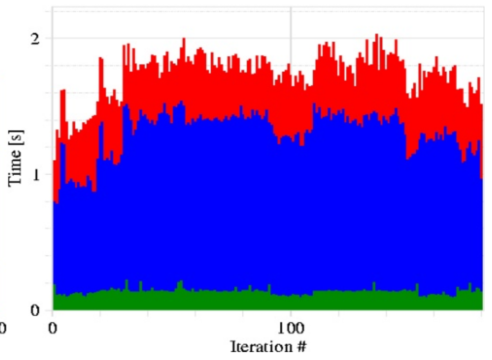
129.tera_tf



130.socorro



132.zeusmp2



137.lu

Scalasca’s approach to performance dynamics

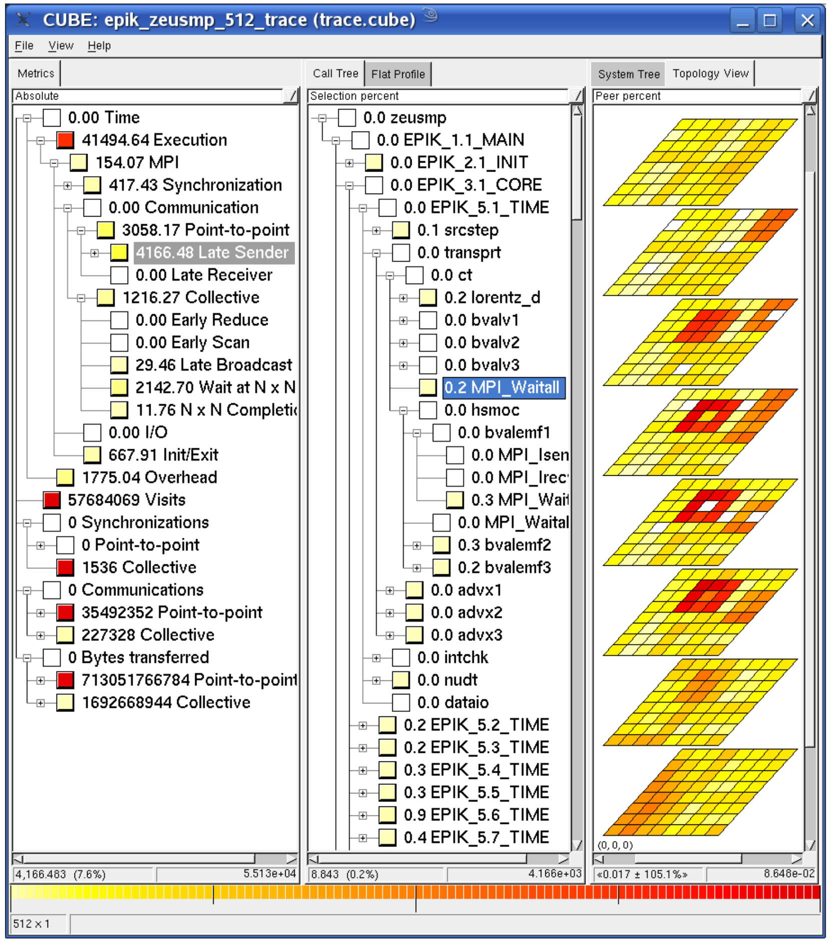
- Capture overview of performance dynamics via time-series profiling
  - Time and count-based metrics
- Identify pivotal iterations
  - If reproducible
- In-depth analysis of these iterations via tracing
  - Analysis of wait-state formation including root cause analysis
  - Tracing restricted to iterations of interest

New

Scalable time-series call-path profiling

- Instrumentation of the main loop to distinguish individual iterations
- Complete call-tree recorded for each iteration
  - With multiple metrics collected for every call-path
- Low-overhead online compression of iteration profiles
  - Reduces memory requirements

Zoltán Szebenyi et al.: Space-Efficient Time-Series Call-Path Profiling of Parallel Applications. In Proc. of the SC09 Conference, Portland, Oregon, ACM, November 2009.



CUBE: epik_zeusmp_512_trace (trace.cube)

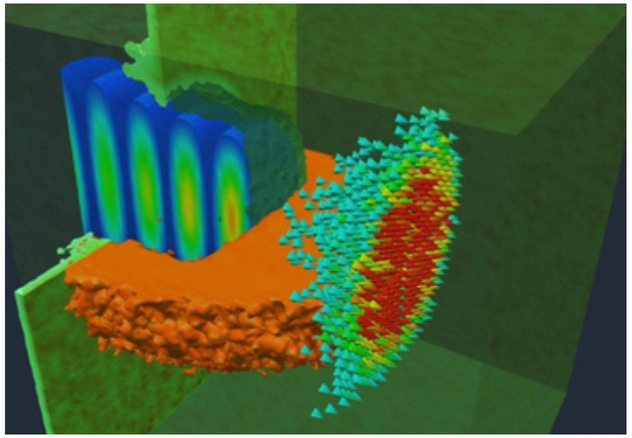
```
0.00 Time
  41494.64 Execution
    154.07 MPI
      417.43 Synchronization
      0.00 Communication
        3058.17 Point-to-point
          4166.48 Late Sender
          0.00 Late Receiver
        1216.27 Collective
          0.00 Early Reduce
          0.00 Early Scan
          29.46 Late Broadcast
          2142.70 Wait at N x N
          11.76 N x N Completi
      0.00 I/O
      667.91 Init/Exit
  1775.04 Overhead
57684069 Visits
0 Synchronizations
  0 Point-to-point
  1536 Collective
0 Communications
  35492352 Point-to-point
  227328 Collective
0 Bytes transferred
  713051766784 Point-to-point
  1692668944 Collective
```

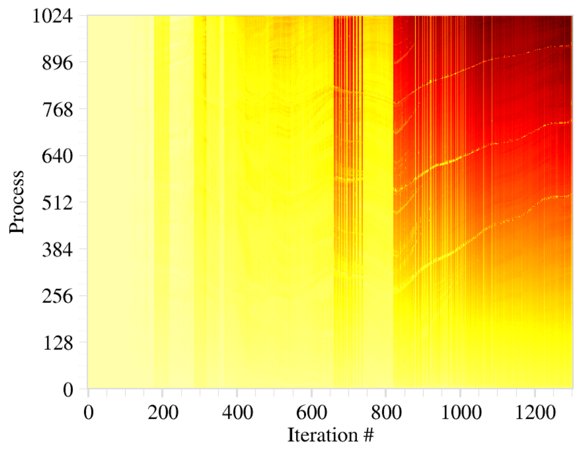
```
0.0 zeusmp
  0.0 EPIK_1.1_MAIN
    0.0 EPIK_2.1_INIT
    0.0 EPIK_3.1_CORE
      0.0 EPIK_5.1_TIME
        0.1 srcstep
        0.0 transprt
          0.0 ct
            0.2 lorentz_d
            0.0 bvalv1
            0.0 bvalv2
            0.0 bvalv3
            0.2 MPI_Waitall
            0.0 hsmoc
              0.0 bvalemf1
                0.0 MPI_Isen
                0.0 MPI_Irec
                0.3 MPI_Wai
              0.0 MPI_Waital
              0.3 bvalemf2
              0.2 bvalemf3
          0.0 advx1
          0.0 advx2
          0.0 advx3
        0.0 intchk
        0.0 nudt
        0.0 dataio
      0.2 EPIK_5.2_TIME
      0.2 EPIK_5.3_TIME
      0.3 EPIK_5.4_TIME
      0.3 EPIK_5.5_TIME
      0.9 EPIK_5.6_TIME
      0.4 EPIK_5.7_TIME
```

```
4,166.483  (7.6%)        5.513e+04
8.843  (0.2%)            4.166e+03
«0.017 ± 105.1%»         8.648e-02
512 x 1
```

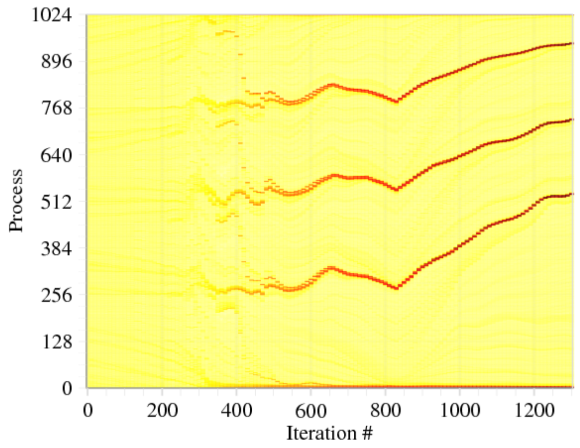
Pretty Efficient Parallel Coulomb-solver (PEPC)

- Multi-purpose parallel tree code
  - Molecular dynamics
  - Laser-plasma interactions
- Developed at JSC





Late Sender



particles owned by a process

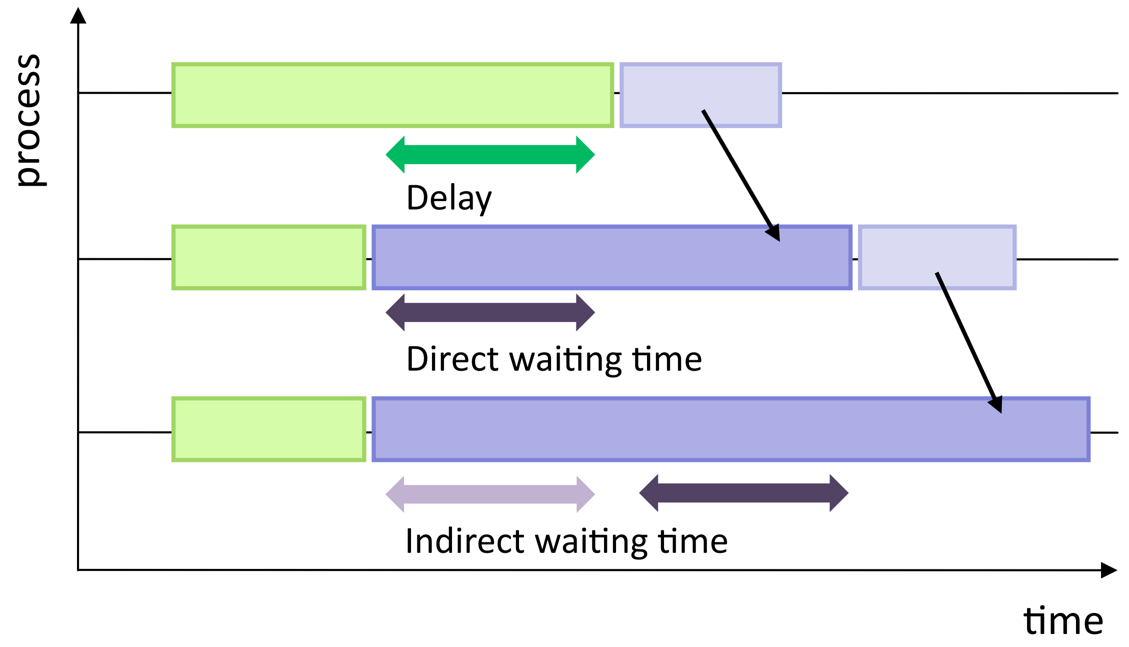
Reconciling sampling and direct instrumentation

- Semantic compression needs direct instrumentation to capture communication metrics and to track the call path
- Direct instrumentation may result in excessive overhead
- New hybrid approach
  - Applies low-overhead sampling to user code
  - Intercepts MPI calls via direct instrumentation
  - Relies on efficient stack unwinding
  - Integrates measurements in statistically sound manner

Joint work with Lawrence Livermore National Laboratory

Zoltan Szebenyi et al.: Reconciling sampling and direct instrumentation for unintrusive call-path profiling of MPI programs. In Proc. of IPDPS, Anchorage, AK, USA. IEEE Computer Society, May 2011. (to appear)

Delay analysis

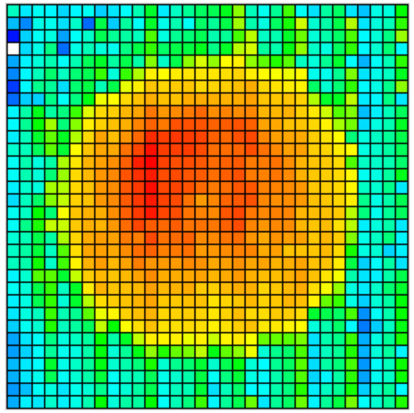


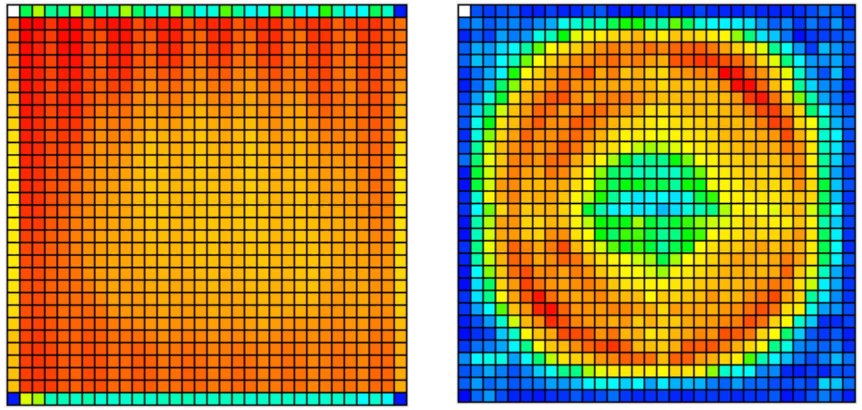
David Böhme et al.: Identifying the root causes of wait states in large-scale parallel applications. In Proc. of ICPP, San Diego, CA, IEEE Computer Society, September 2010.
Best Paper Award

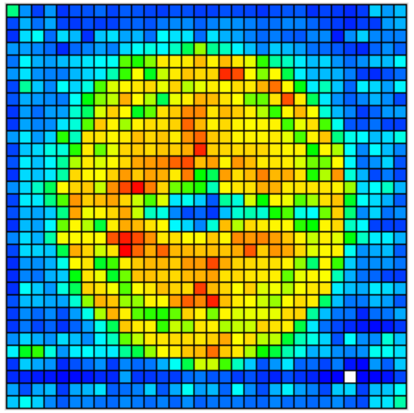
- Classification of waiting times into
  - Direct vs. indirect
  - Propagating vs. terminal
- Attributes costs of wait states to delay intervals
  - Scalable through parallel forward and backward replay of traces

Delay analysis of code Illumination

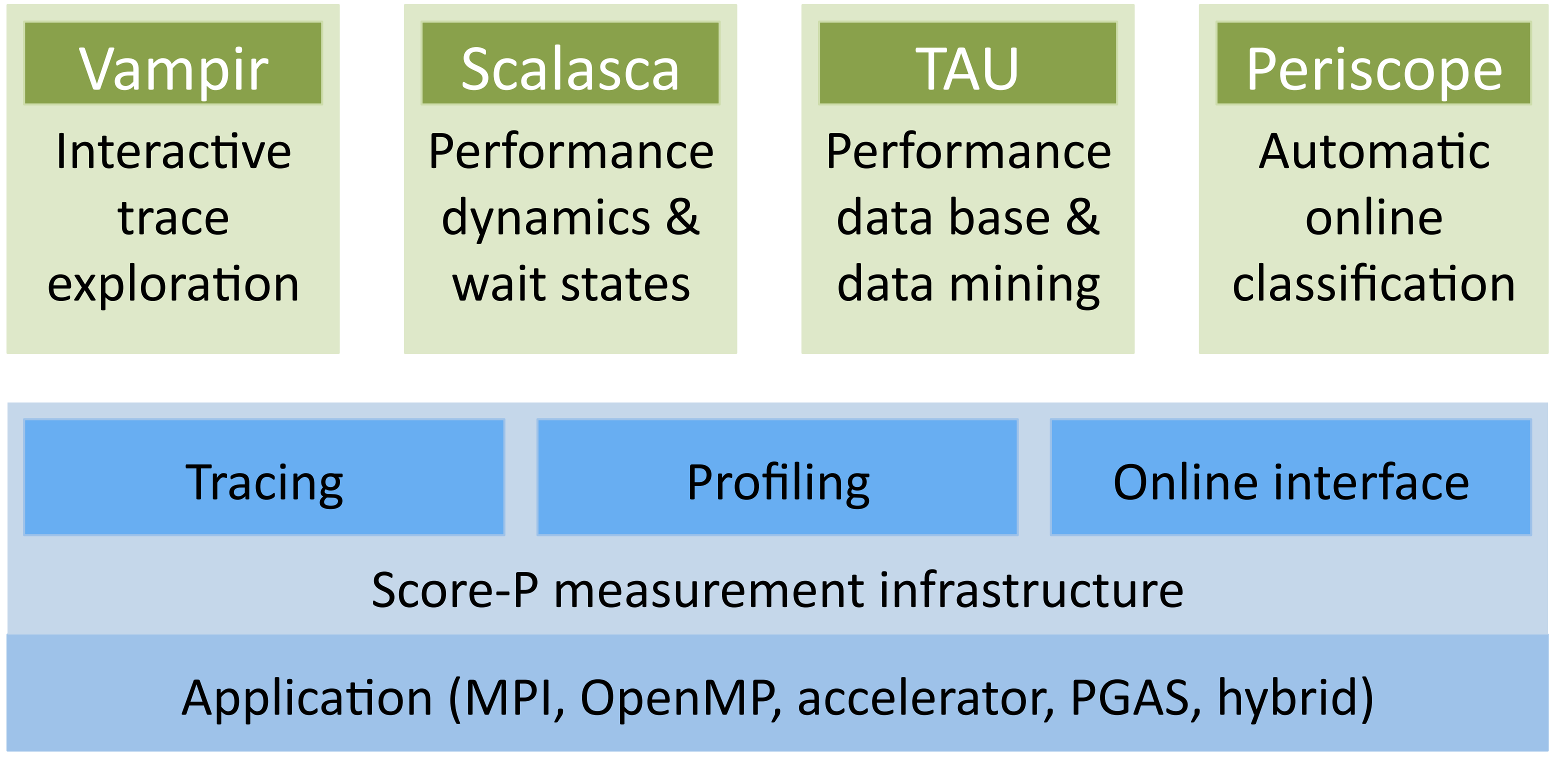
- Particle physics code (laser-plasma interaction)
- Delay analysis identified inefficient communication behavior as cause of wait states


Computation


Propagating wait states:
Original vs. optimized code


Costs of direct delay
in optimized code

Score-P measurement system


Vampir
Interactive trace exploration
Scalasca
Performance dynamics & wait states
TAU
Performance data base & data mining
Periscope
Automatic online classification
Tracing
Profiling
Online interface
Score-P measurement infrastructure
Application (MPI, OpenMP, accelerator, PGAS, hybrid)

JÜLICH FORSCHUNGSZENTRUM

German Research School for Simulation Sciences

gns

TUM Technische Universität München

RWTH AACHEN UNIVERSITY

TECHNISCHE UNIVERSITÄT DRESDEN

UNIVERSITY OF OREGON

Future work

- Further scalability improvements
- Emerging architectures and programming models
  - PGAS languages
  - Accelerator architectures
- Interoperability with 3rd-party tools
  - Common measurement library for several performance tools

The virtual institute in a…

- Partnership to develop advanced programming tools for complex simulation codes
- Goals
  - Improve code quality
  - Speed up development
- Activities
  - Tool development and integration
  - **Training**
  - Support
  - Academic workshops
- www.vi-hps.org

Thank you!

HELMHOLTZ ASSOCIATION

Deutsche Forschungsgemeinschaft DFG

Bundesministerium für Bildung und Forschung

DEPARTMENT OF ENERGY • UNITED STATES OF AMERICA