German Research School
for Simulation Sciences

Performance Engineering in HPC Application Development

Felix Wolf
27-06-2012

JÜLICH
FORSCHUNGSZENTRUM

RWTH AACHEN UNIVERSITY

German Research School for Simulation Sciences

- Joint venture of
  - Forschungszentrum Jülich
  - RWTH Aachen University
- Four research laboratories
  - Computational biophysics
  - Computational engineering
  - Computational materials science
  - Parallel programming
- Education
  - M.Sc. in Simulation Sciences
  - Ph.D. program
- About 50 scientific staff members

Aachen

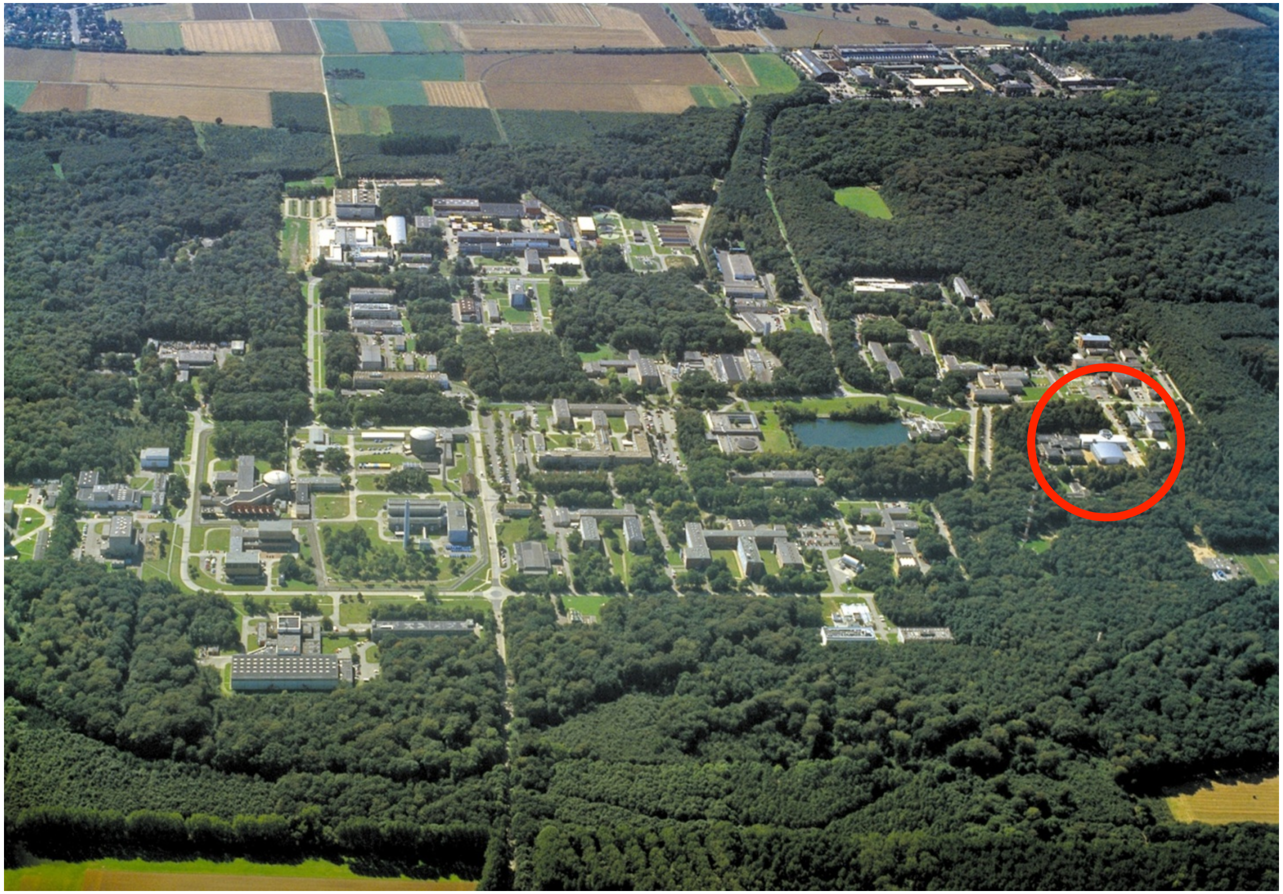
Jülich

Forschungszentrum Jülich

Helmholtz Center with ca. 4400 employees

Application areas
- Health
- Energy
- Environment
- Information

Key competencies
- Physics
- Supercomputing



Rheinisch-Westfälische Technische Hochschule Aachen

- 260 institutes in nine faculties
- Strong focus on engineering
- > 200 M€ third-party funding per year
- Around 31,000 students are enrolled in over 100 academic programs
- More than 5,000 are international students from 120 different countries
- Cooperates with Jülich within the Jülich Aachen Research Alliance (JARA)

University main building

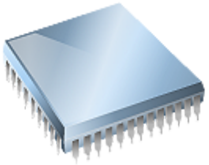
Euro-Par 2013 in Aachen

- International conference series
  - Dedicated to parallel and distributed computing
- Wide spectrum of topics
  - Algorithms and theory
  - Software technology
  - Hardware-related issues

Euro-Par
Conference Series

Performance

$$\textbf{Performance} \sim \frac{1}{\text{Resources to solution}}$$



Hardware

Time

Energy

… and ultimately

Money

Performance optimization pays off

Example: HPC Service RWTH Aachen
~300 TFlops Bull/Intel cluster

Total cost of ownership (TCO) per year: **5.5** M€

Resource type	Fraction of TCO
Hardware	1/2
Energy	1/4
Staff	1/8
Others	1/8

Tuning the
workload by 1%
will "save"
55k€ per year
~ 1 FTE

Source: Bischof, an Mey, Iwainsky: Brainware for green HPC, Computer Science-Research and Development, Springer

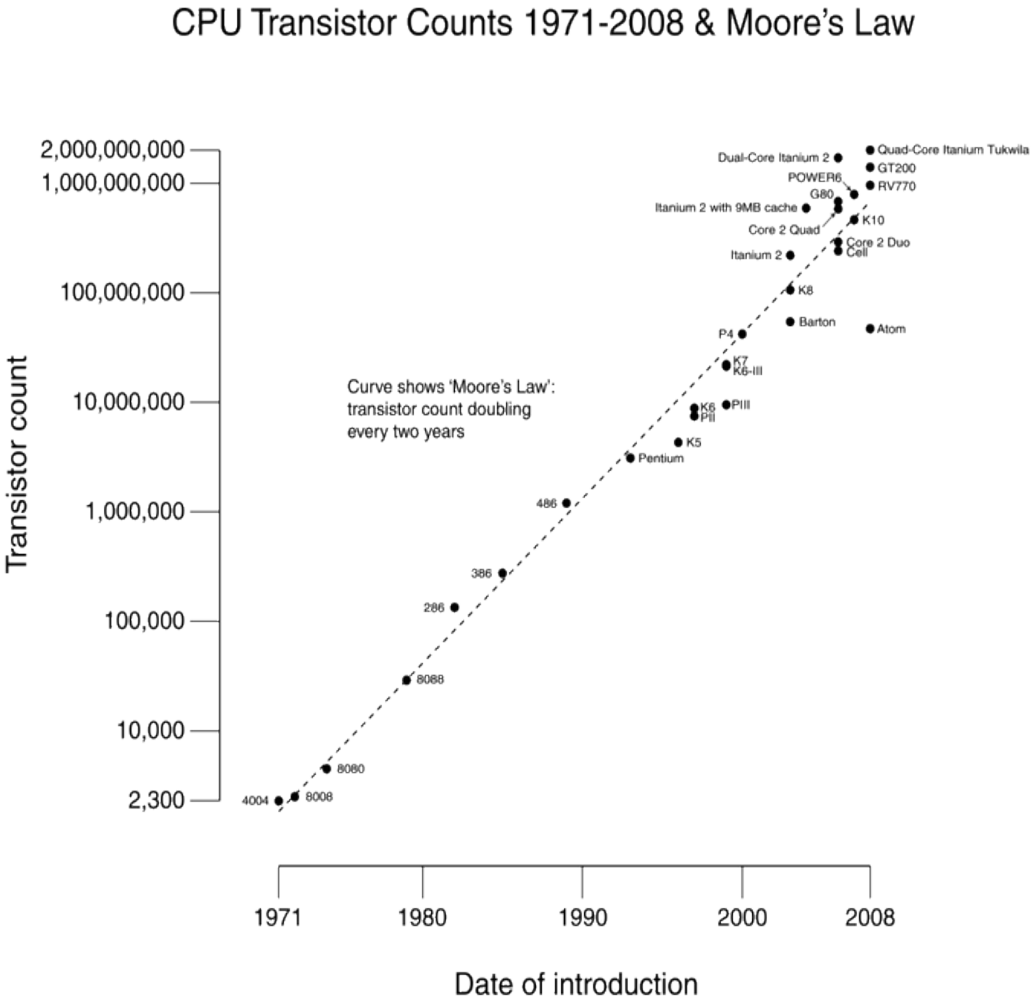
Objectives

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

Outline

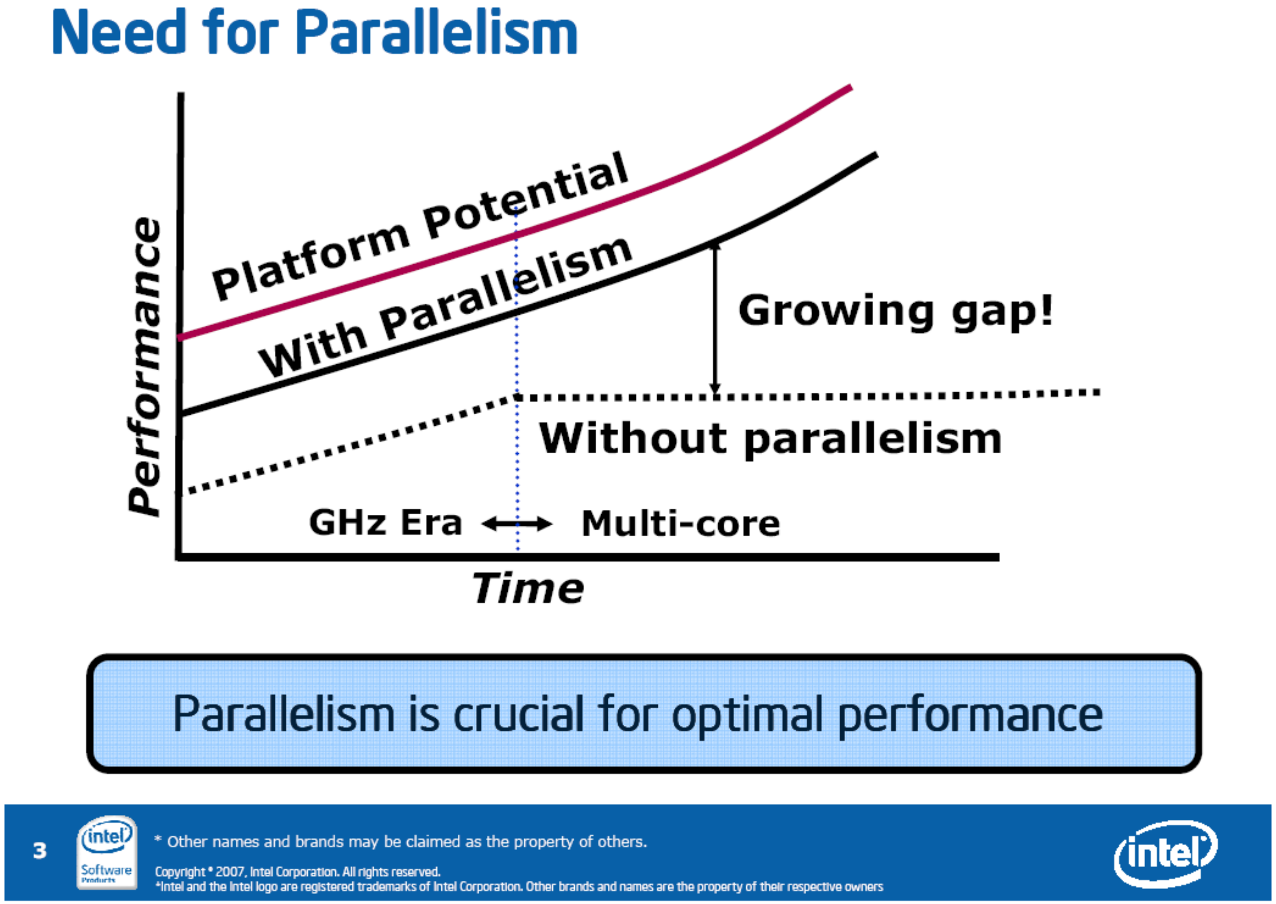
- Principles of parallel performance
- Performance analysis techniques
- Practical performance analysis using Scalasca

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

$$0 \leq f \leq 1$$

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

- 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 $\tau_\mathrm{v}(\mathrm{n},\mathrm{p})$
- Speedup

$$\mathrm{Speedup}(\mathrm{n},\mathrm{p}) = \frac{\tau_\mathrm{f} + \tau_\mathrm{v}(\mathrm{n},1)}{\tau_\mathrm{f} + \tau_\mathrm{v}(\mathrm{n},\mathrm{p})}$$

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

Comparison of analysis techniques

Measurement – Simulation – Performance model

Number of parameters

Model error

Based on SC'11 paper from Torsten Hoefler et al.

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

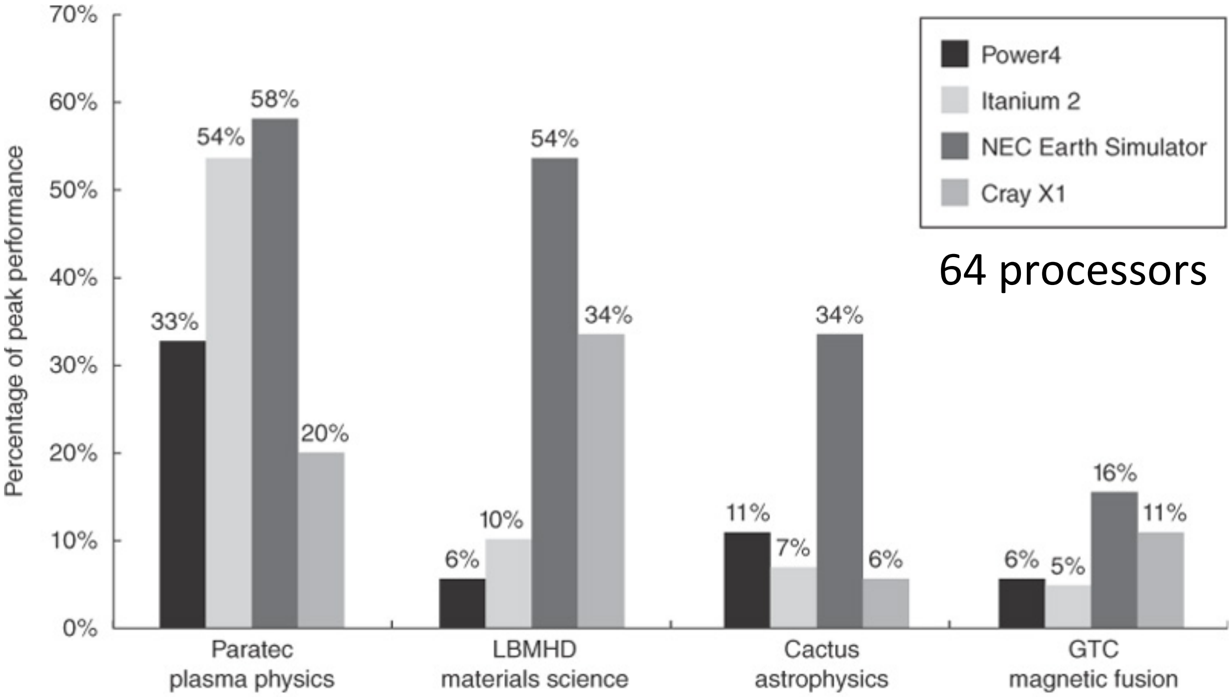
- Clock rate
- Instructions executed per second
- FLOPS
  - Floating-point operations per second

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

- 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

Peak performance

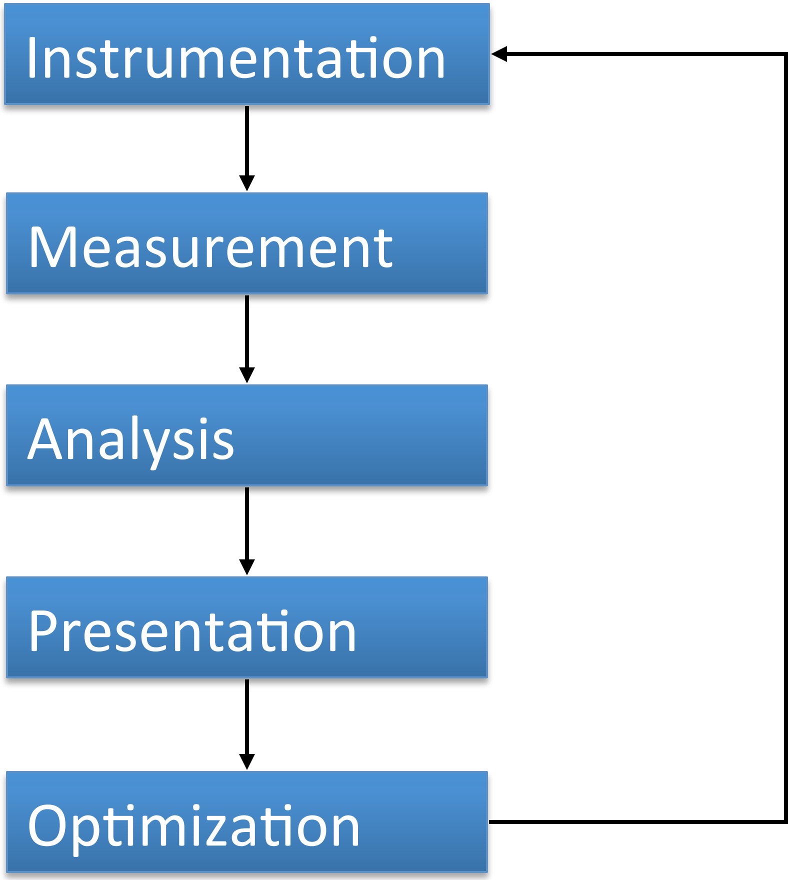
- Peak performance is the performance a computer is guaranteed not to exceed



64 processors

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

Performance tuning cycle


Instrumentation

Measurement

Analysis

Presentation

Optimization

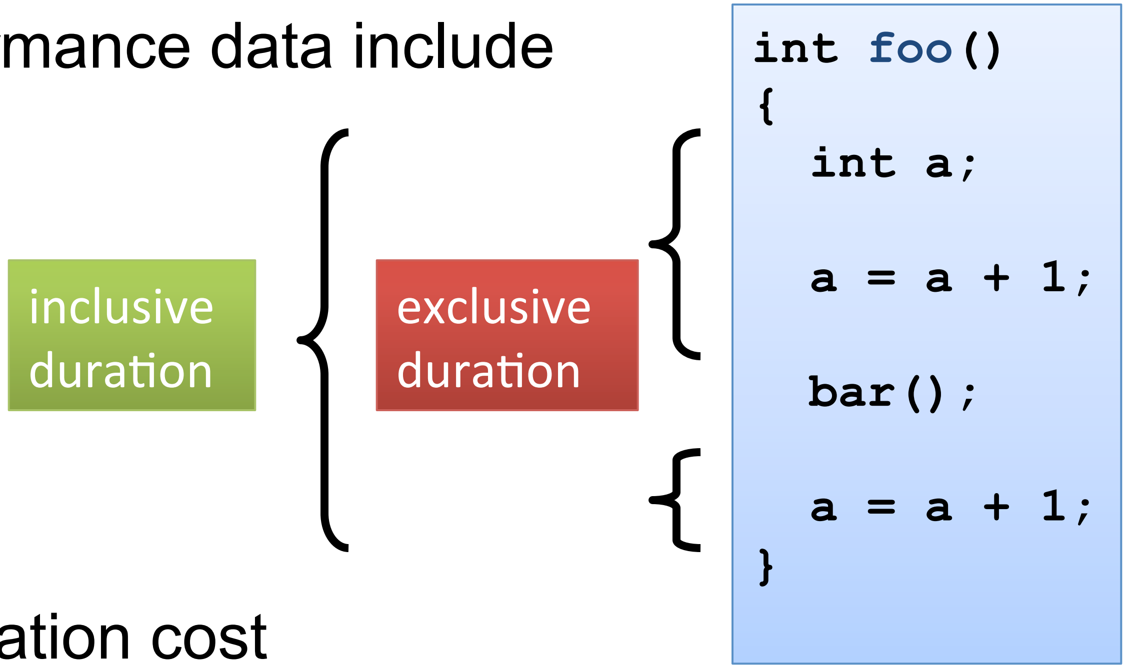
Instrumentation techniques

- Direct instrumentation
  - Measurement code is inserted at certain points in the program
    - Example: function entry/exit, dispatch or receipt of messages
  - Can be done manually or automatically
  - Advantage: captures all instrumented events
  - Disadvantage: overhead more difficult to control
- Sampling (statistical approach)
  - Based on the assumption that a subset of a population being examined is representative for the whole population
  - Measurement performed only in certain intervals - usually implemented with timer interrupt
  - Advantage: overhead can be easily controlled
  - Disadvantage: incomplete information, harder to access program state

Measurement

Typical performance data include

- Counts
- Durations



inclusive duration

exclusive duration

```
int foo()
{
   int a;

   a = a + 1;

   bar();

   a = a + 1;
}
```

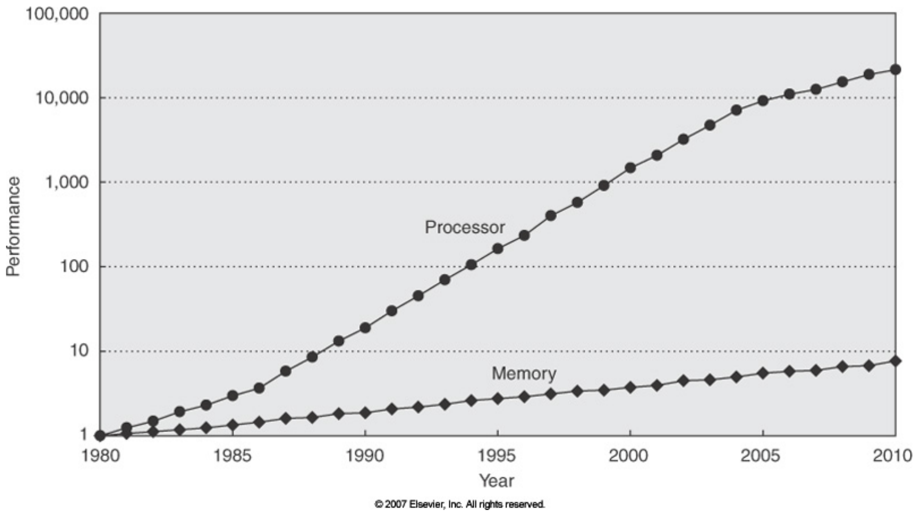
- Communication cost
- Synchronization cost
- IO accesses
- System calls
- Hardware events

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

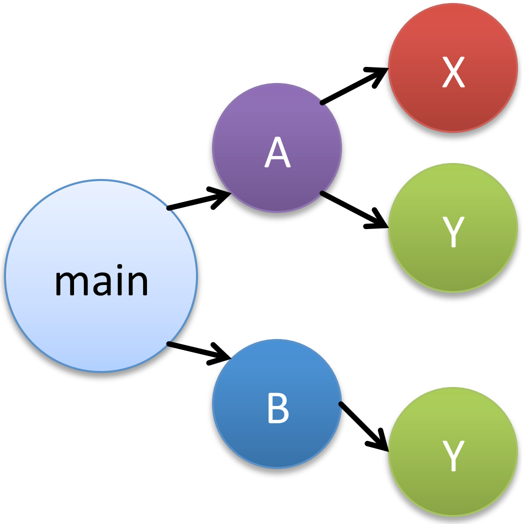
Profiling

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

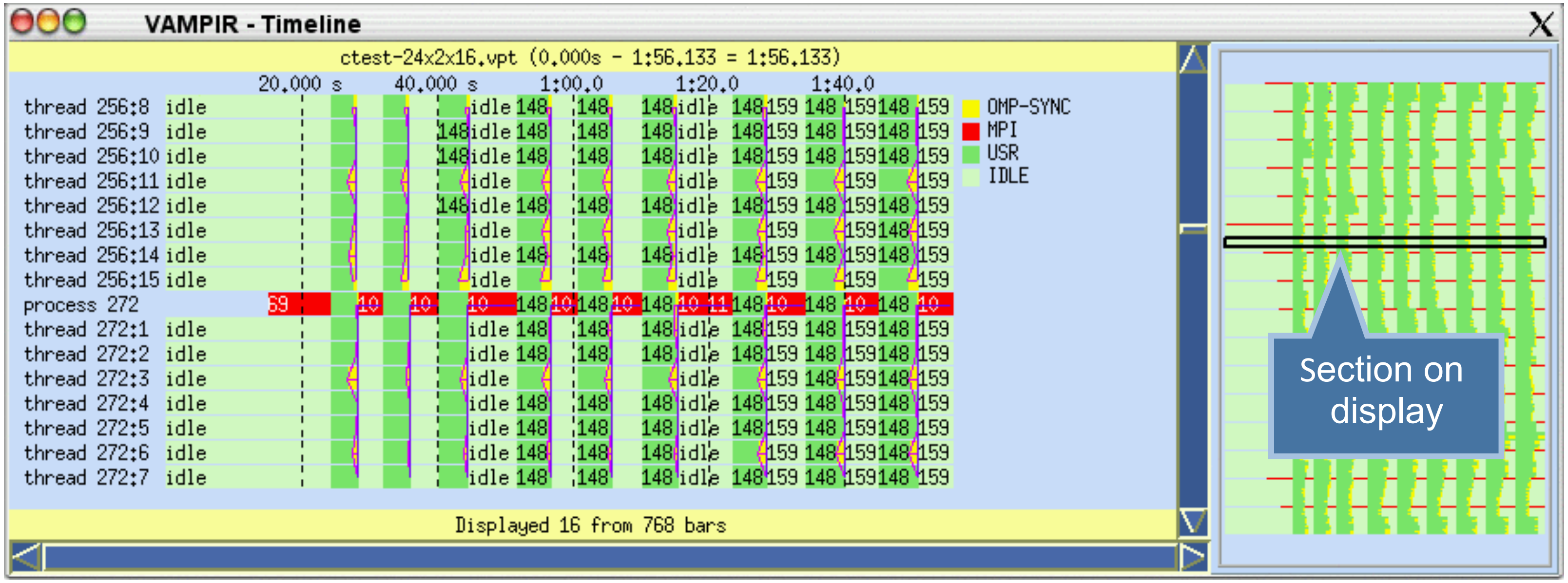
Call-path profiling

- 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

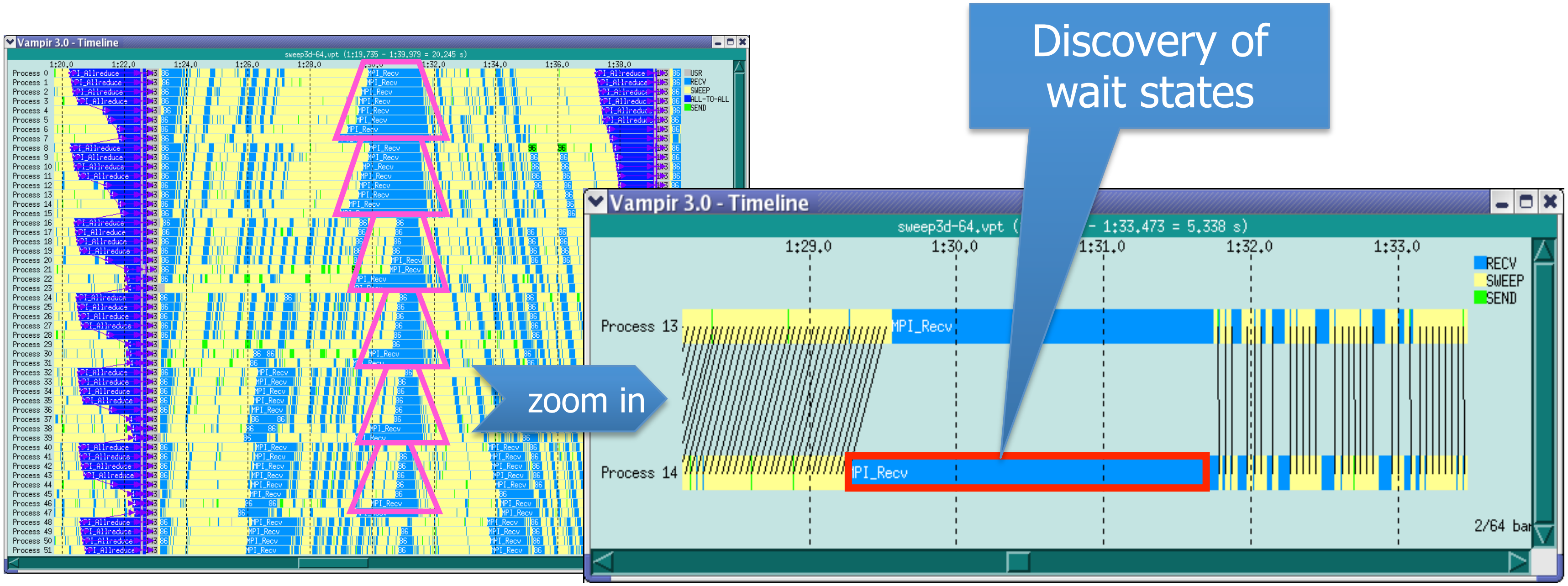


Section on display

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

Why tracing?

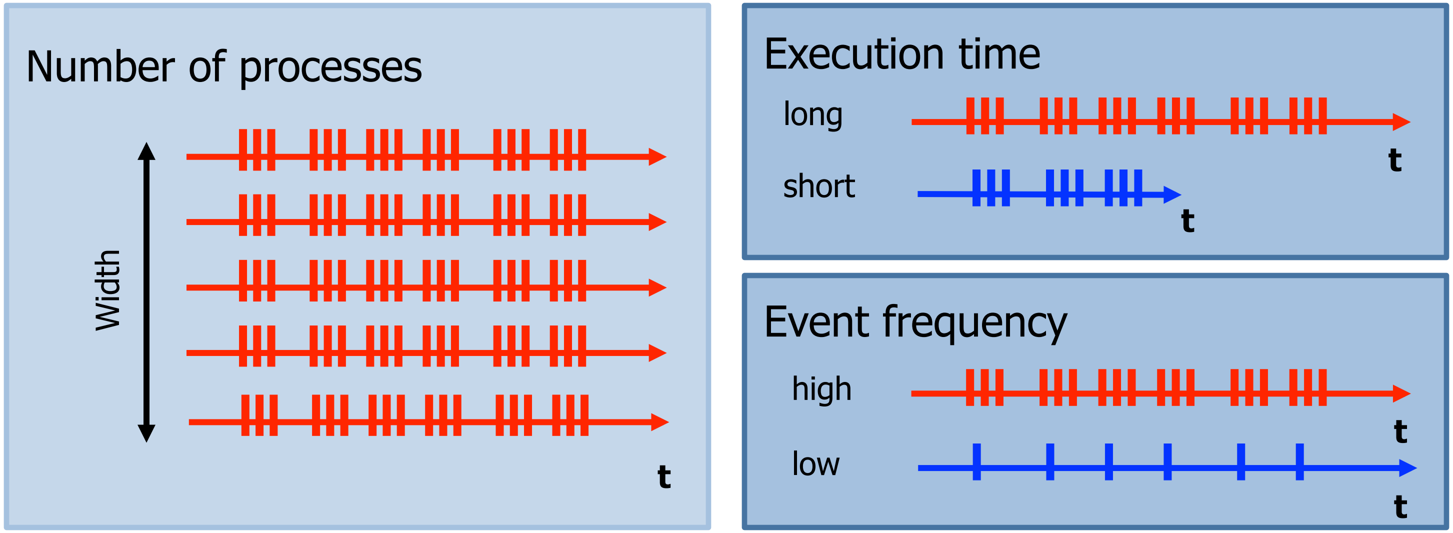
- High level of detail
- Allows in-depth post-mortem analysis of program behavior
  - Time-line visualization
  - Automatic pattern search
- Identification of wait states



Discovery of wait states

zoom in

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

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

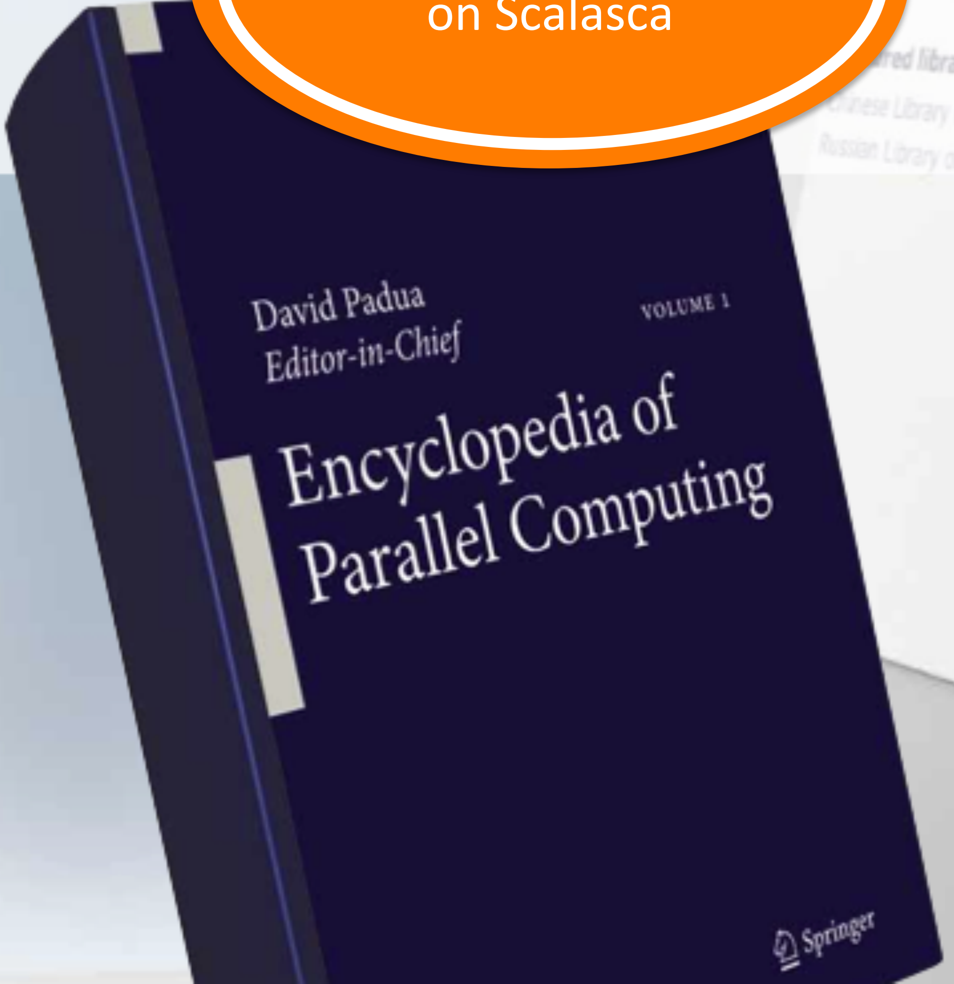
November 13-19, 2010: Join us at SC'10 in New Orleans, LA, USA. Scalasca team... more...

Encyclopedia of Parallel Computing

Edited by **David Padua**

SPRINGER
REFERENCE

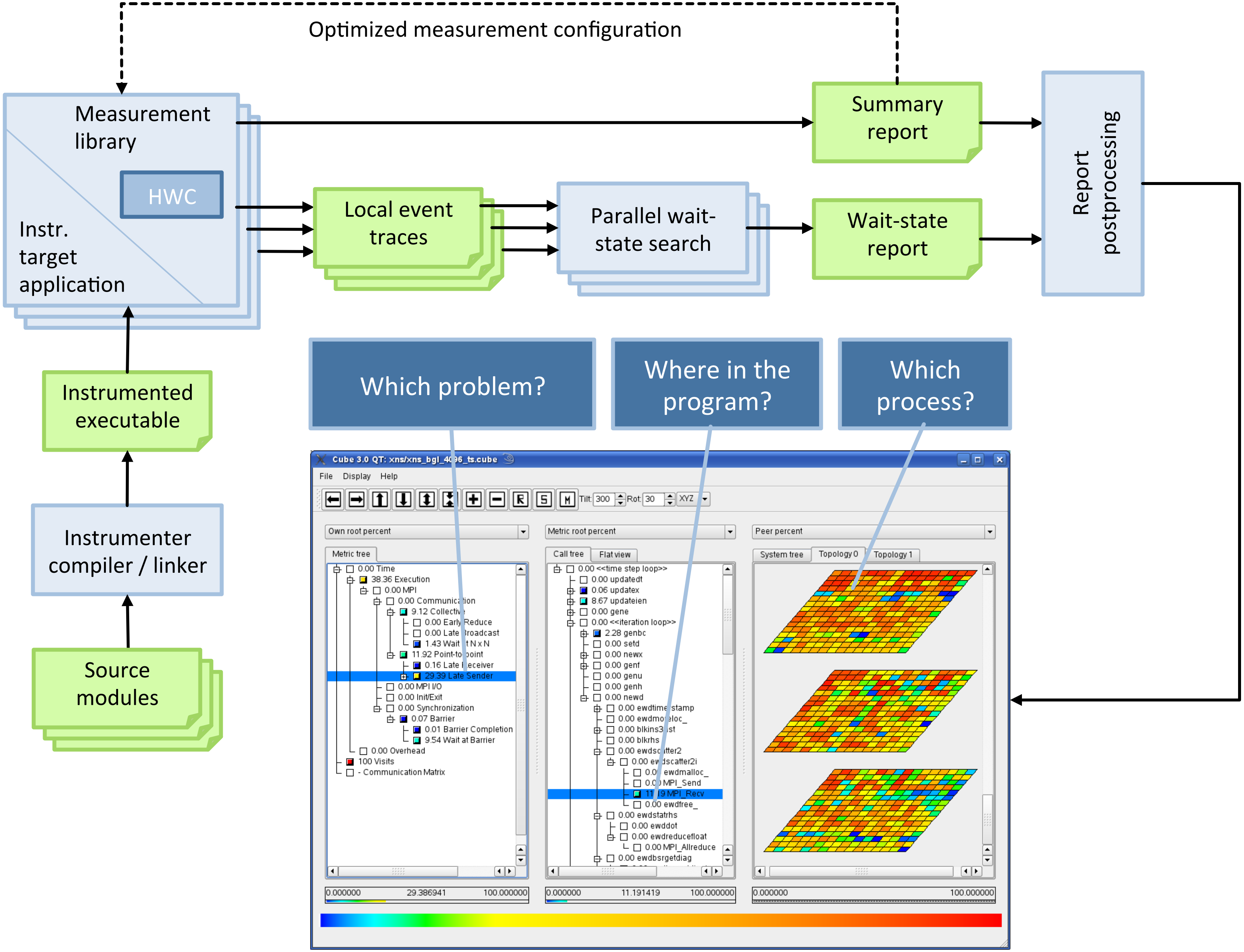
- The comprehensive source of information in the field
- Published as a fully searchable and hyperlinked eReference and in hardcover



Multi-page article on Scalasca

Installations and users

- Companies
  - Bull (France)
  - Dassault Aviation (France)
  - EDF (France)
  - Efield Solutions (Sweden)
  - GNS (Germany)
  - IBM (France, Germany)
  - INTES (Germany)
  - MAGMA (Germany)
  - RECOM (Germany)
  - SciLab (France)
  - Shell (Netherlands)
  - SiCortex (USA)
  - Sun Microsystems (USA, Singapore, India)
  - Qontix (UK)
- Research / supercomputing centers
  - Argonne National Laboratory (USA)
  - Barcelona Supercomputing Center (Spain)
  - Bulgarian Supercomputing Centre (Bulgaria)
  - CERFACS (France)
  - Centre Informatique National de l'Enseignement Supérieur (France)
  - Commissariat à l'énergie atomique (France)
  - Computation-based Science and Technology Research Center (Cyprus)
  - CASPUR (Italy)
  - CINECA (Italy)
  - Deutsches Klimarechenzentrum (Germany)
  - Deutsches Zentrum für Luft- und Raumfahrt (Germany)
  - Edinburgh Parallel Computing Centre (UK)
  - Federal Office of Meteorology and Climatology (Switzerland)
  - Flanders ExaScience Lab (Belgium)
  - Forschungszentrum Jülich (Germany)
  - IT Center for Science (Finland)
  - High Performance Computing Center Stuttgart (Germany)
  - Irish Centre for High-End Computing (Ireland)
  - Institut du développement et des ressources en informatique scientifique (France)
  - Karlsruher Institut für Technologie (Germany)
  - Lawrence Livermore National Laboratory (USA)
  - Leibniz-Rechenzentrum (Germany)
  - National Authority For Remote Sensing & Space Science (Egypt)
  - National Center for Atmospheric Research (USA)
- Research/supercomputing centers (cont.)
  - National Center for Supercomputing Applications (USA)
  - National Laboratory for High Performance Computing (Chile)
  - Norddeutscher Verbund zur Förderung des Hoch- und Höchstleistungsrechnens (Germany)
  - Oak Ridge National Laboratory (USA)
  - PDC Center for High Performance Computing (Sweden)
  - Pittsburgh Supercomputing Center (USA)
  - Potsdam-Institut für Klimafolgenforschung (Germany)
  - Rechenzentrum Garching (Germany)
  - SARA Computing and Networking Services (Netherlands)
  - Shanghai Supercomputer Center (China)
  - Swiss National Supercomputing Center (Switzerland)
  - Texas Advanced Computing Center (USA)
  - Texas A&M Supercomputing Facility (USA)
  - Très Grand Centre de calcul (France)
- Universities
  - École Centrale Paris (France)
  - École Polytechnique Fédérale de Lausanne (Switzerland)
  - Institut polytechnique de Grenoble (France)
  - King Abdullah University of Science and Technology (Saudi Arabia)
  - Lund University (Sweden)
  - Lomonosov Moscow State University (Russia)
  - Michigan State University (USA)
  - Norwegian University of Science & Technology (Norway)
  - Politechnico di Milano (Italy)
  - Rensselaer Polytechnic Institute (USA)
  - Rheinisch-Westfälische Technische Hochschule Aachen (Germany)
  - Technische Universität Dresden (Germany)
  - Università degli Studi di Genova (Italy)
  - Universität Basel (Switzwerland)
  - Universitat Autònoma de Barcelona (Spain)
  - Université de Versailles St-Quentin-en-Yvelines (France)
  - University of Graz (Austria)
  - University of Oregon (USA)
  - University of Oslo (Norway)
  - University of Paderborn (Germany)
  - University of Tennessee (USA)
  - University of Tsukuba (Japan)
  - University of Warsaw (Poland)
- 9 defense-related computing centers



Optimized measurement configuration

Measurement library

HWC

Instr. target application

Summary report

Local event traces

Parallel wait-state search

Wait-state report

Report postprocessing

Instrumented executable

Instrumenter compiler / linker

Source modules

Which problem?

Where in the program?

Which process?

```
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
```

0.000000 29.386941 100.000000

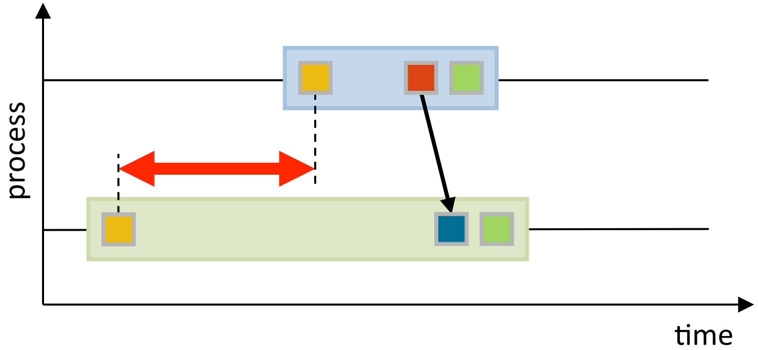
```
0.00 <<time step loop>>
  0.00 updatedt
  0.06 updatex
  8.67 updateien
  0.00 gene
  0.00 <<iteration loop>>
    2.28 genbc
    0.00 setd
    0.00 newx
    0.00 genf
    0.00 genu
    0.00 genh
    0.00 newd
      0.00 ewdtime stamp
      0.00 ewdmo eloc_
      0.00 blkins3 st
      0.00 blkrhs
      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 11.191419 100.000000

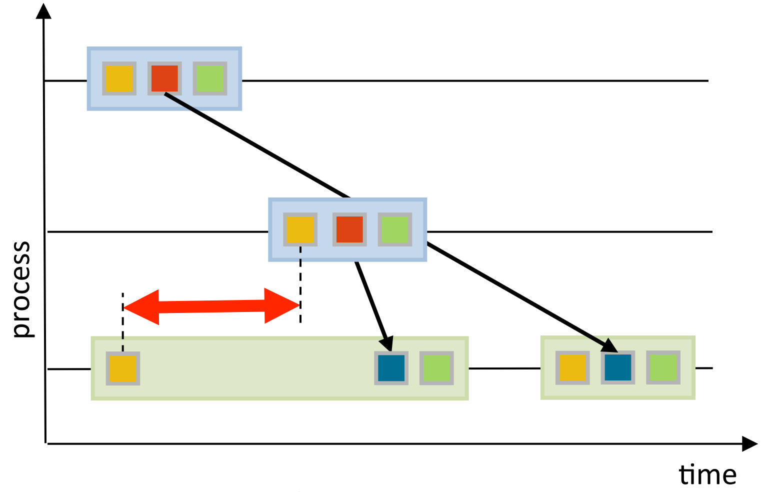
0.000000 100.000000

Wait-state analysis

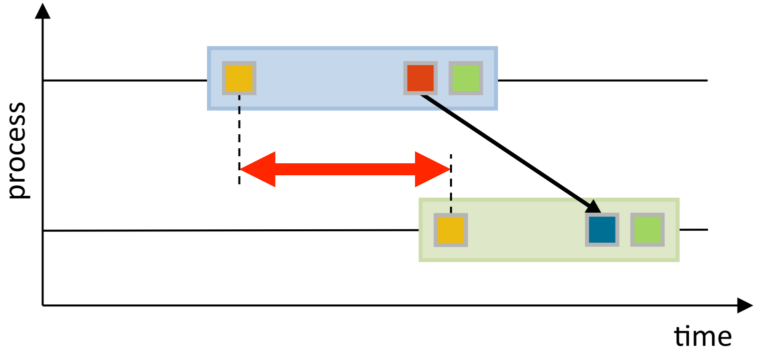
- Classification
- Quantification



(a) Late Sender



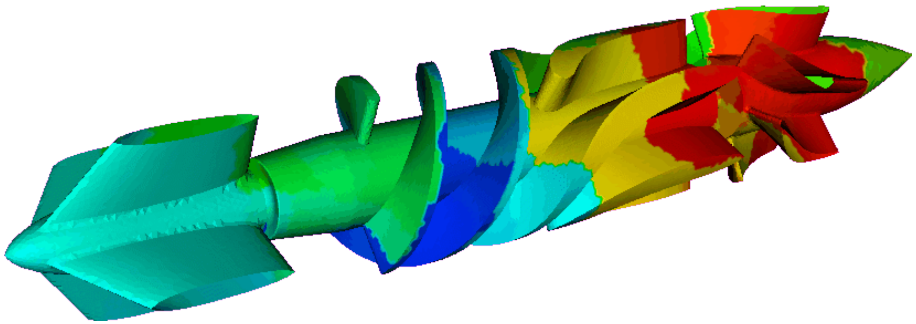
(b) Late Sender / Wrong Order



(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

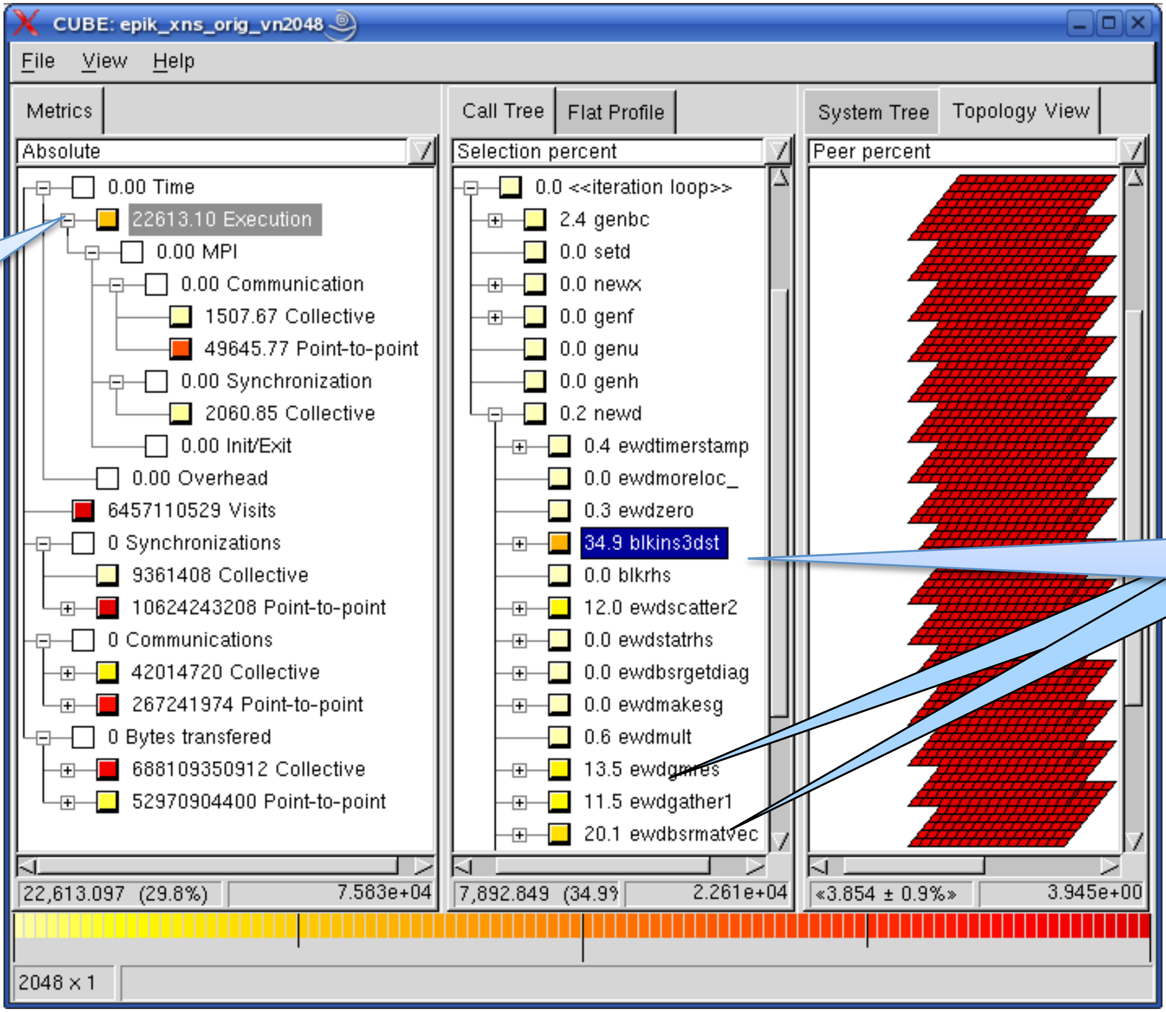


Partitioned finite-element mesh

Call-path profile: Computation

Execution time excl. MPI comm

Just 30% of simulation



```
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
```

```
0.0 <<iteration loop>>
  2.4 genbc
  0.0 setd
  0.0 newx
  0.0 genf
  0.0 genu
  0.0 genh
  0.2 newd
    0.4 ewdtimerstamp
    0.0 ewdmoreloc_
    0.3 ewdzero
    34.9 blkins3dst
    0.0 blkrhs
    12.0 ewdscatter2
    0.0 ewdstatrhs
    0.0 ewdbsrgetdiag
    0.0 ewdmakesg
    0.6 ewdmult
    13.5 ewdgmres
    11.5 ewdgather1
    20.1 ewdbsrmatvec
```

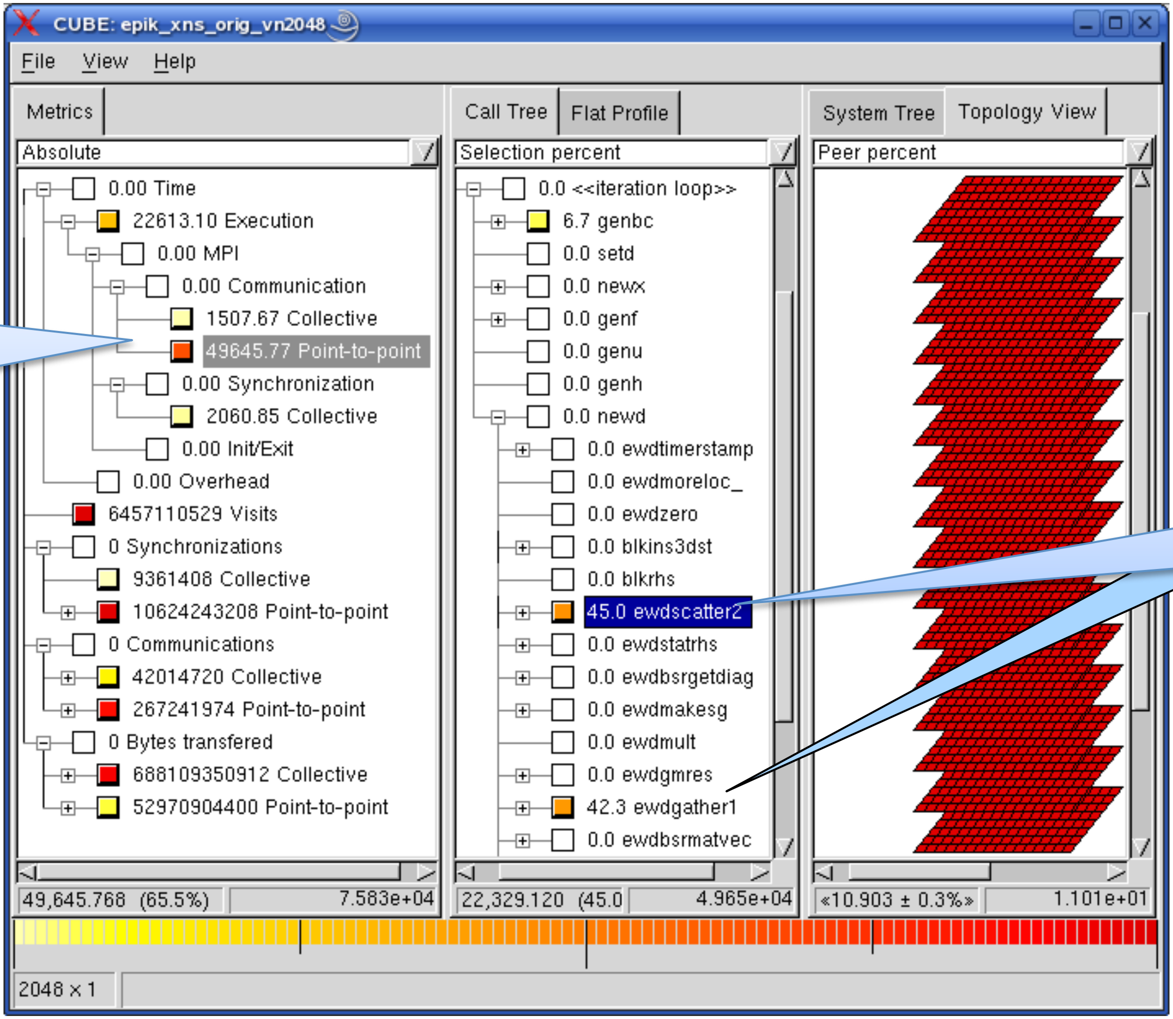
```
22,613.097  (29.8%)        7.583e+04
7,892.849  (34.9%        2.261e+04
«3.854 ± 0.9%»        3.945e+00
```

Widely spread in code

Call-path profile: P2P messaging

MPI point-to-point communication time

P2P comm 66% of simulation



Absolute

- 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

Selection percent

- 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

Peer percent

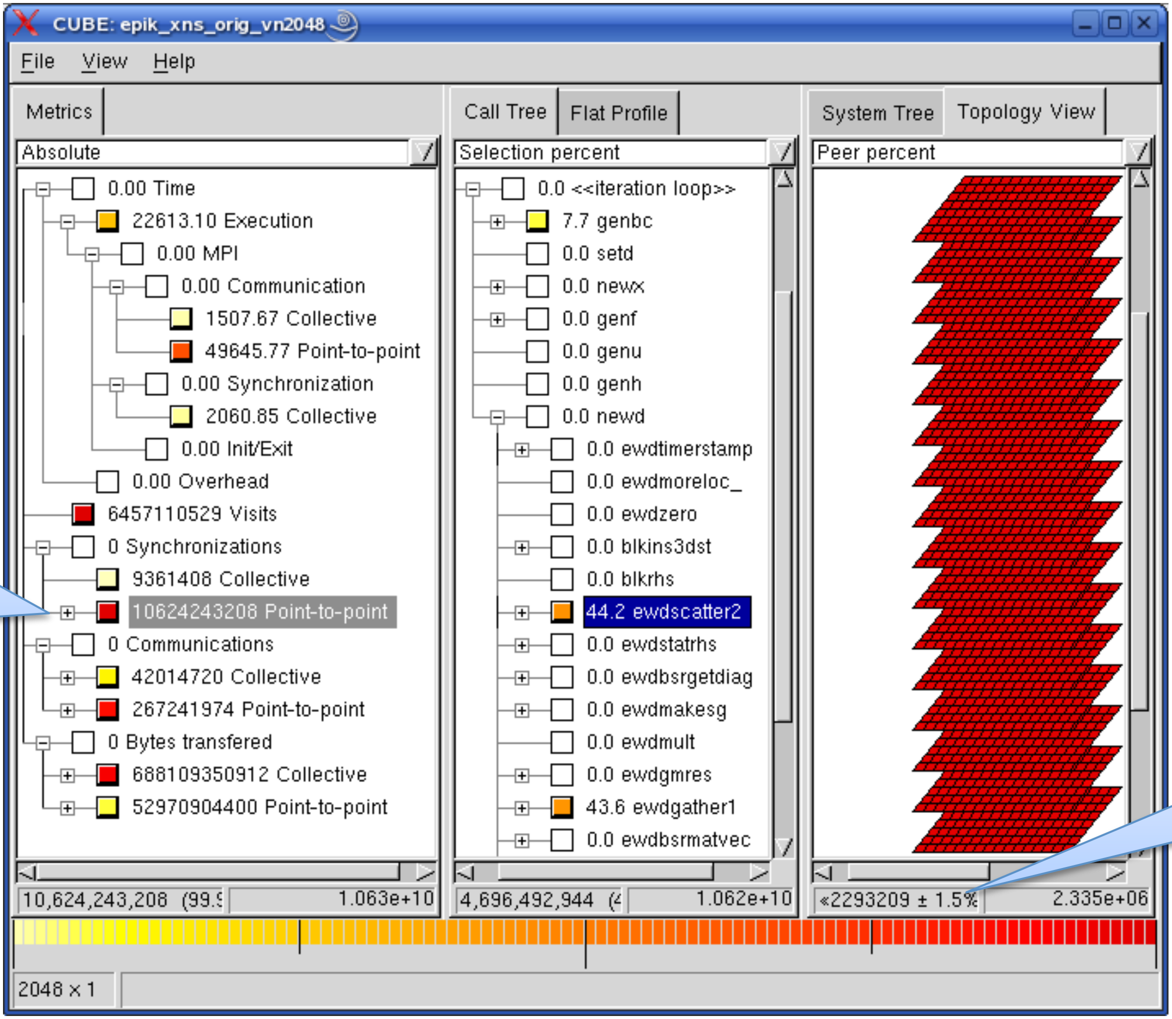
«10.903 ± 0.3%» 1.101e+01

2048 x 1

Primarily in scatter & gather

Call-path profile: P2P sync. ops.

Point-to-point msgs w/o data



```
Absolute
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
10,624,243,208  (99.5        1.063e+10
```

```
Selection percent
0.0 <<iteration loop>>
  7.7 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
    44.2 ewdscatter2
    0.0 ewdstatrhs
    0.0 ewdbsrgetdiag
    0.0 ewdmakesg
    0.0 ewdmult
    0.0 ewdgmres
    43.6 ewdgather1
    0.0 ewdbsrmatvec
4,696,492,944  (4        1.062e+10
```

```
Peer percent
«2293209 ± 1.5%        2.335e+06
2048 x 1
```

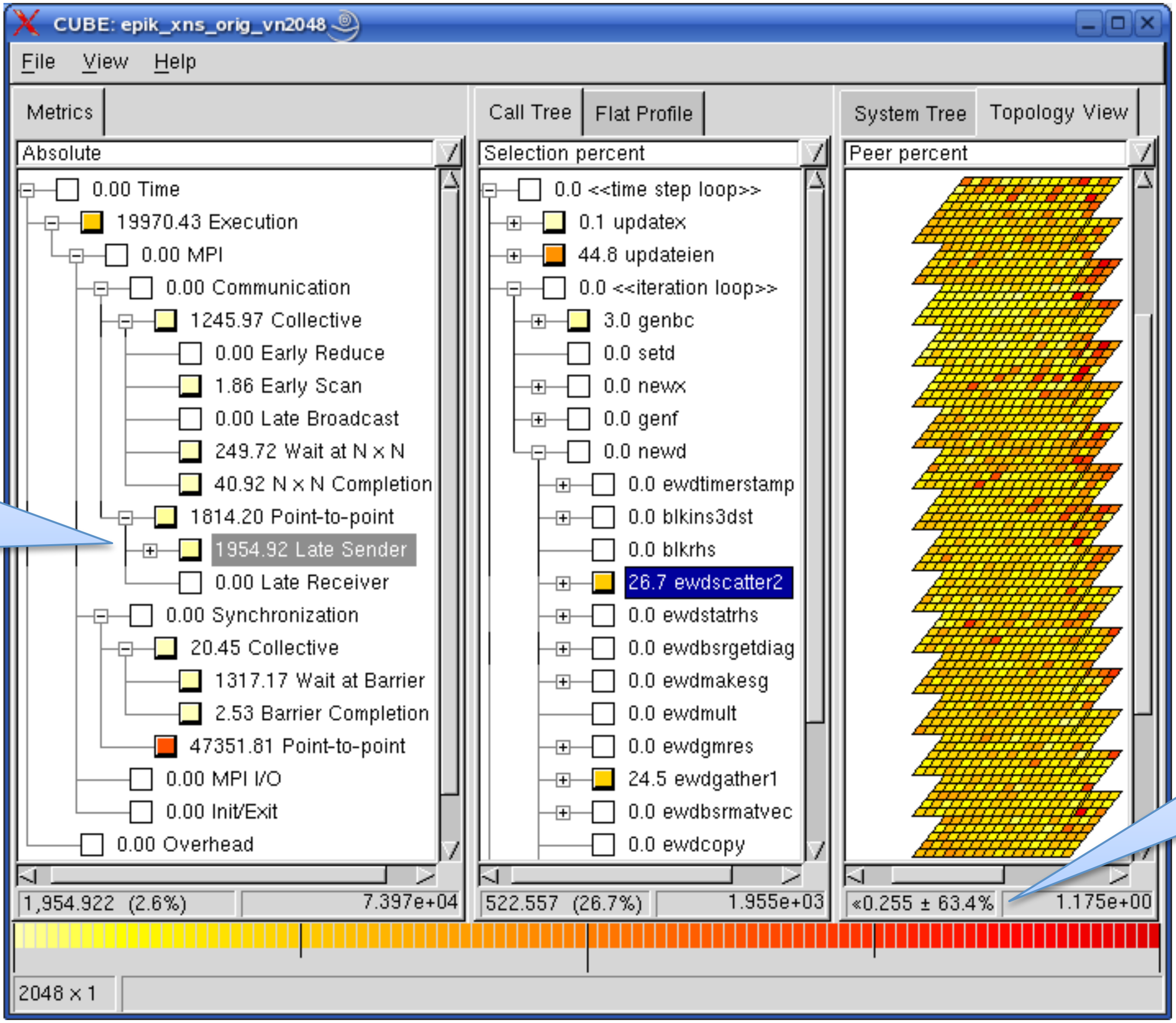
Masses of P2P sync. operations

Processes all equally responsible

Trace analysis: Late sender

Wait time of receivers blocked for late sender

Half of the send time is waiting



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

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

Peer percent

«0.255 ± 63.4% 1.175e+00

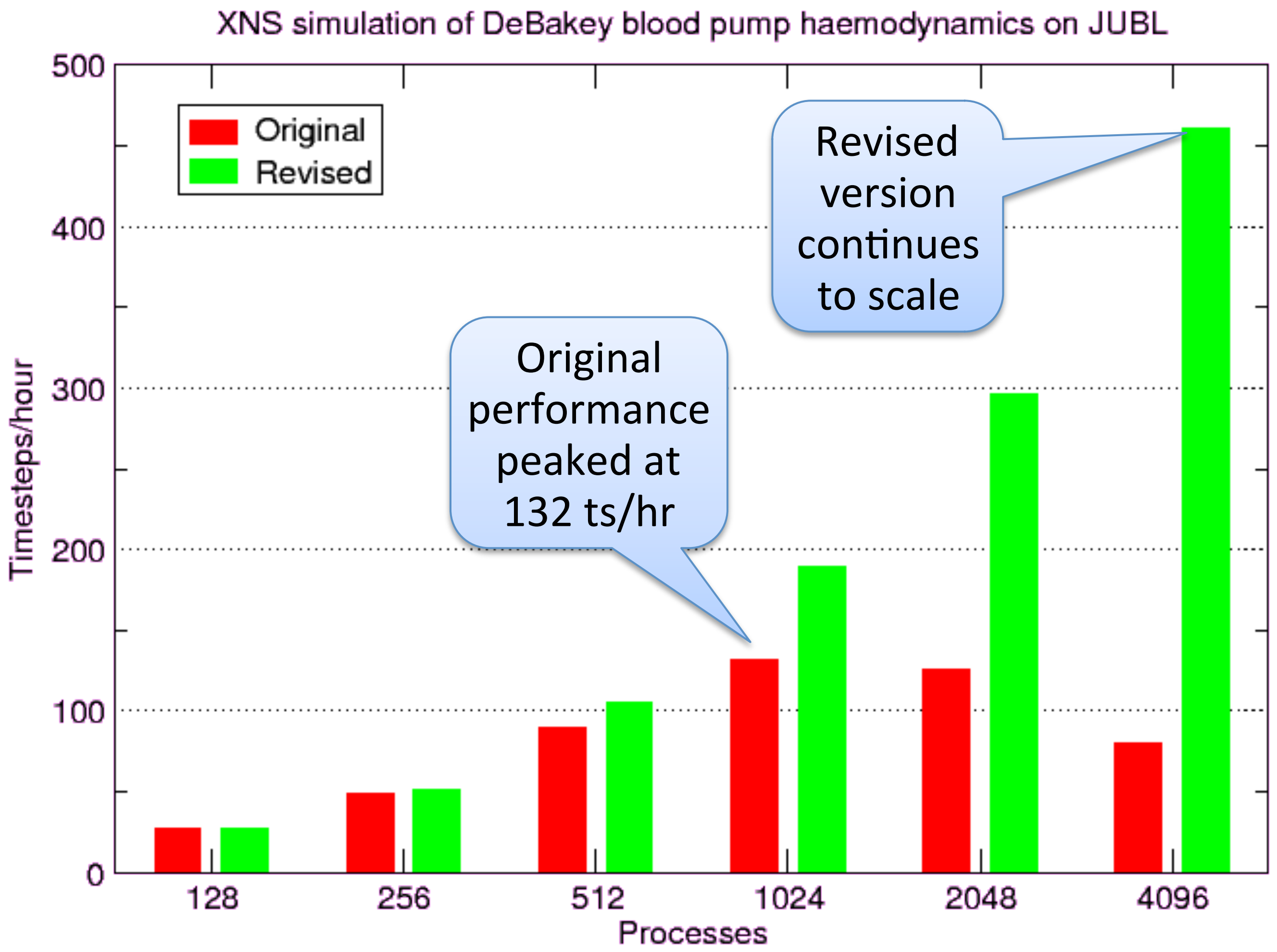
2048 x 1

Significant process imbalance

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)

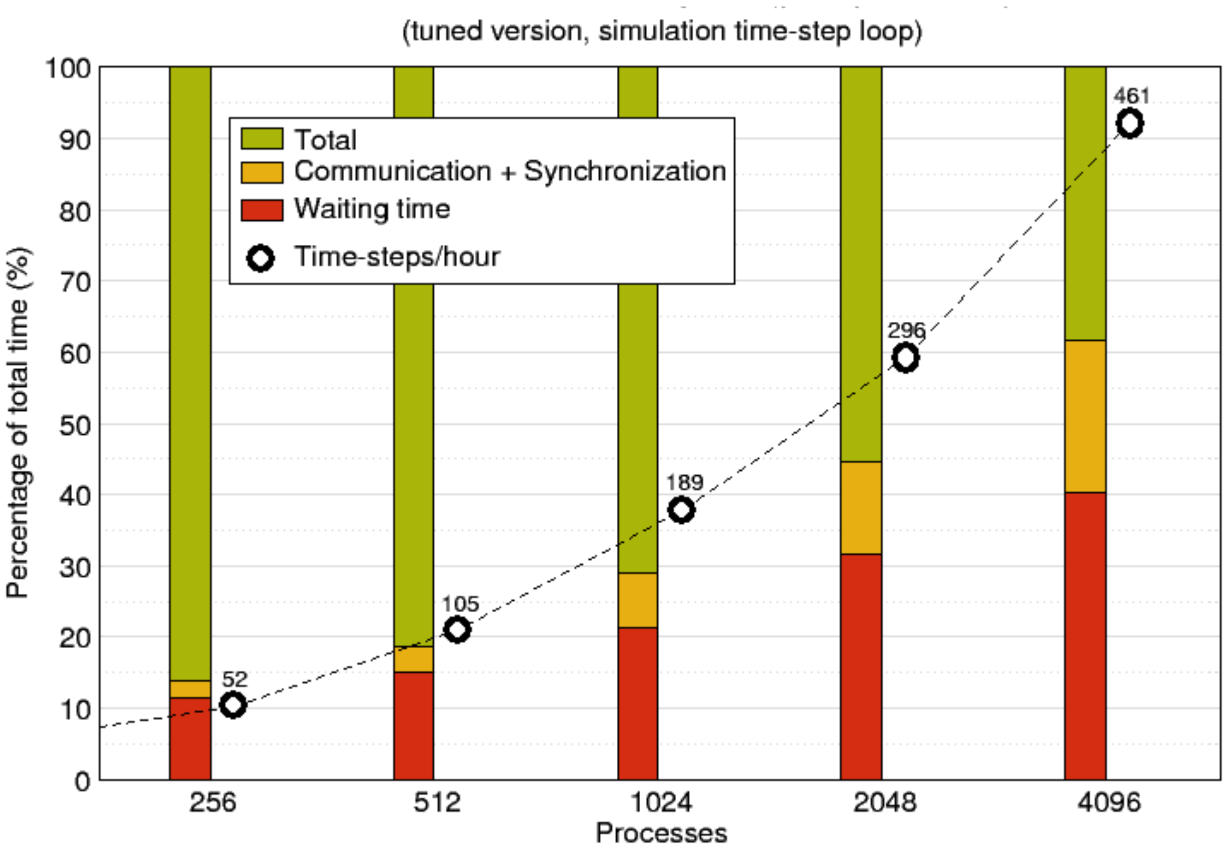
After removal of redundant messages



Original performance peaked at 132 ts/hr

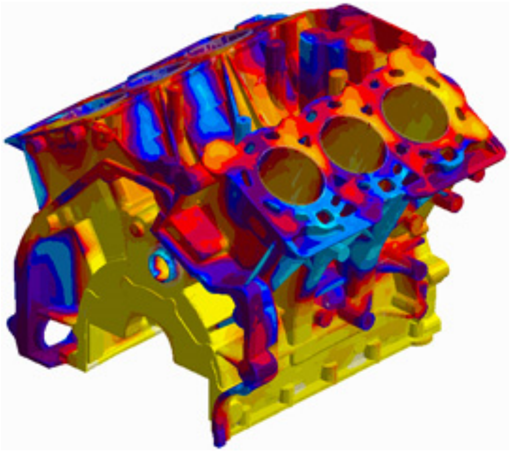
Revised version continues to scale

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



- 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...

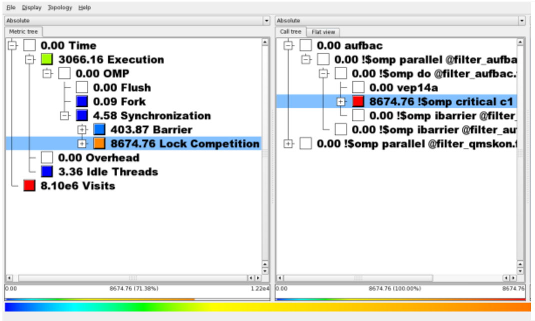
0.0 -3191.0069253191.006925

- 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

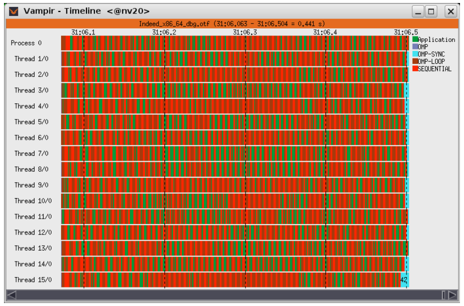
INDEED by GNS® mbh

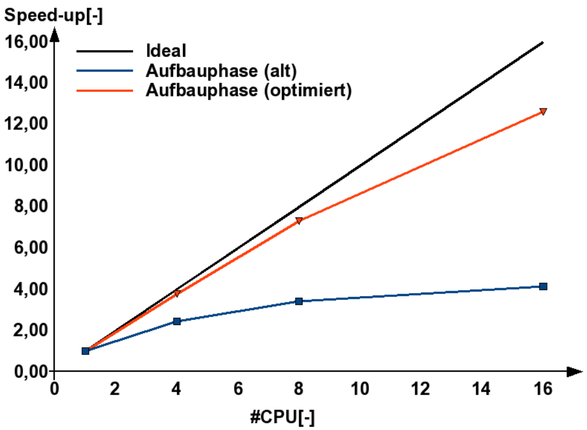
- Finite-element code for the simulation of material-forming processes
  - Focus on creation of element-stiffness matrix
- Tool workflow
  - Scalasca identified serialization in critical section as bottleneck
  - In-depth analysis using Vampir
- Speedup of 30-40% after optimization



```
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                8674.76 (71.38%)                1.22e4
```

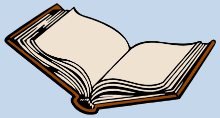
```
0.00 aufbac
  0.00 !$omp parallel @filter_aufba
    0.00 !$omp do @filter_aufbac.
      0.00 vep14a
      8674.76 !$omp critical c1
      0.00 !$omp ibarrier @filter_
    0.00 !$omp ibarrier @filter_au
0.00 !$omp parallel @filter_qmskon.
0.00                8674.76 (100.00%)                8674.76
```





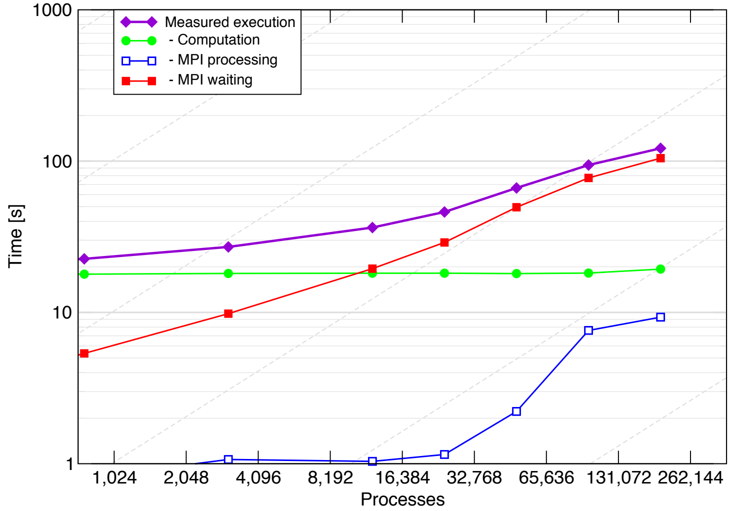
Scalability in terms of the number of cores

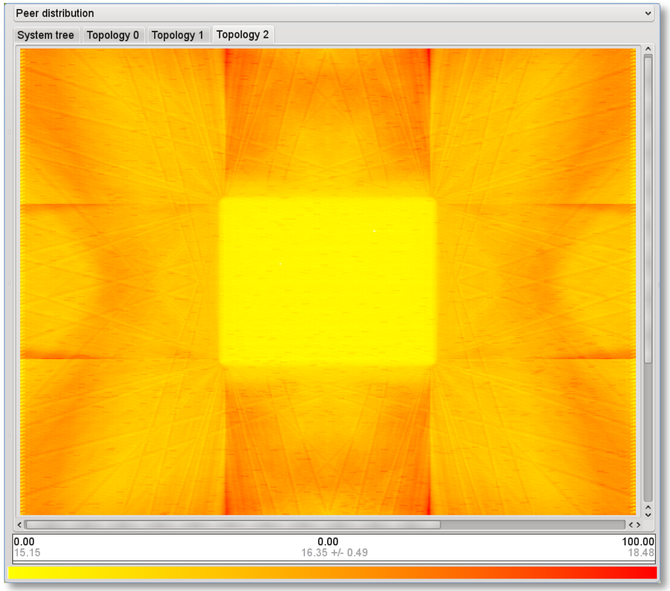
- Application study of ASCI Sweep3D benchmark
- Identified MPI waiting time correlating with computational imbalance
- Measurements & analyses demonstrated on
  - Jaguar with up to 192k cores
  - Jugene with up to 288k cores



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.

Jaguar, MK = 10 (default)

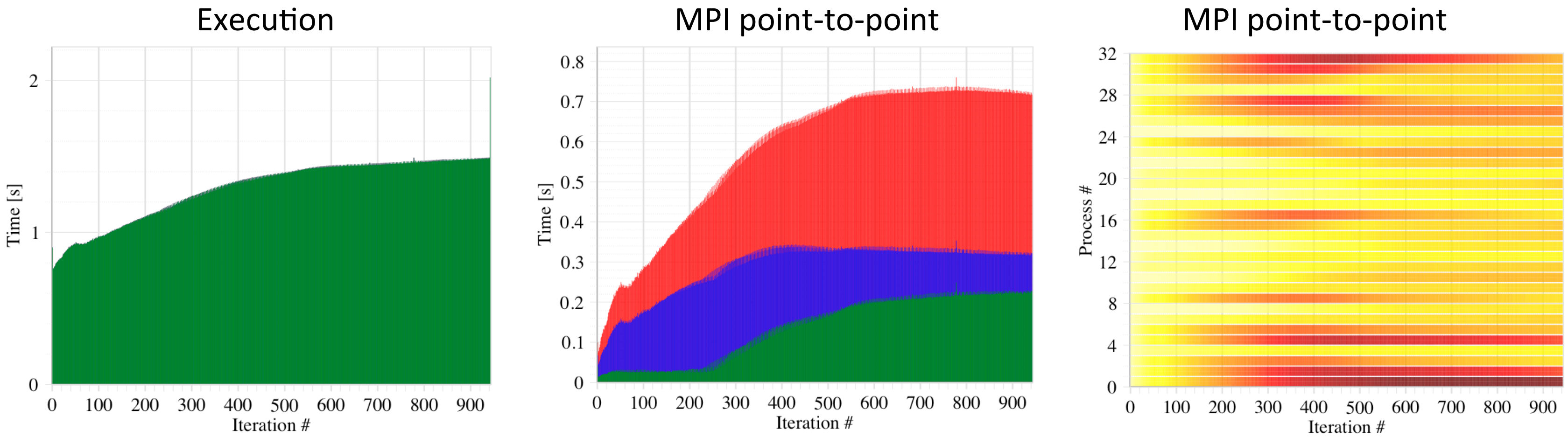




Computation

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)

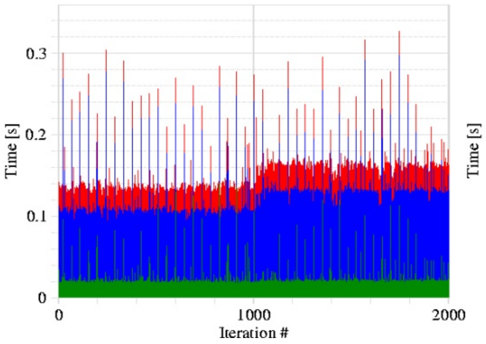

Execution

MPI point-to-point

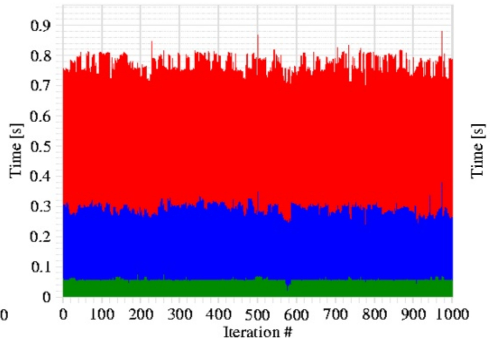
MPI point-to-point

129.tera_tf

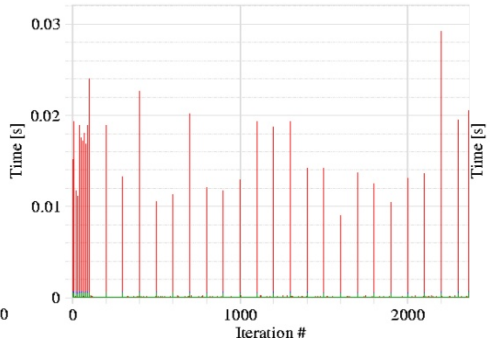
P2P communication in SPEC MPI 2007 suite



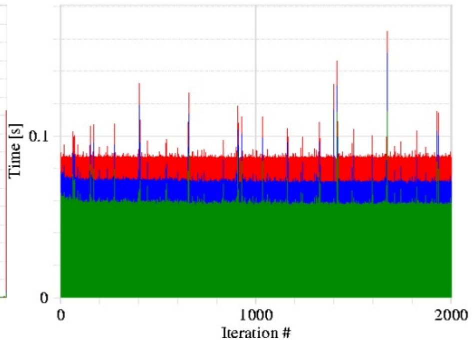
107.leslie3d



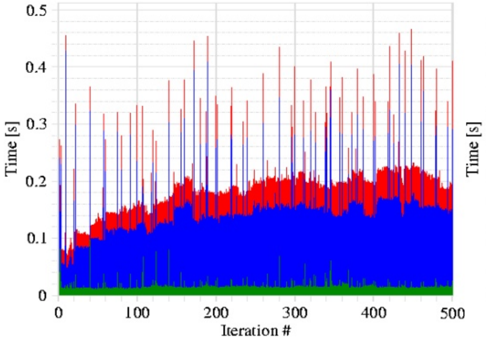
113.GemsFDTD



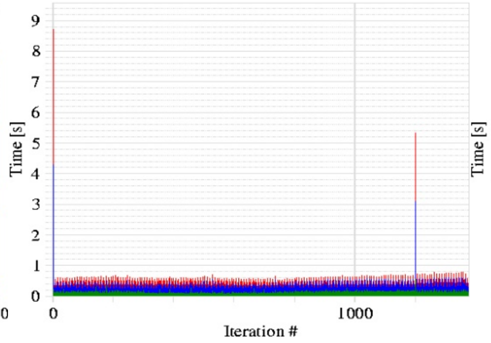
115.fds4



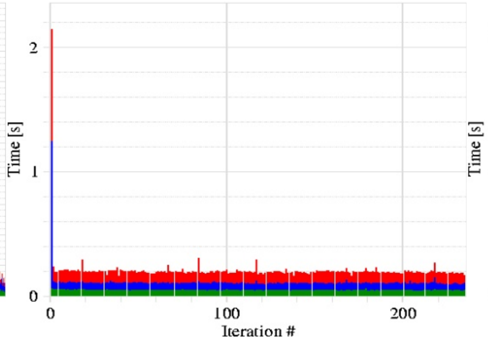
121.pop2



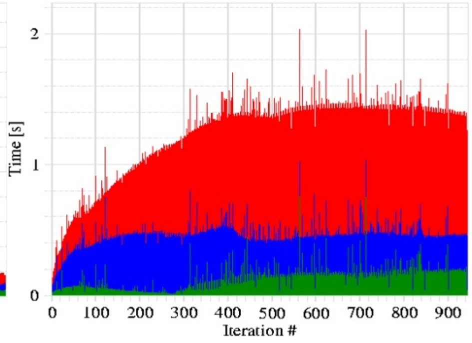
126.leslie3d



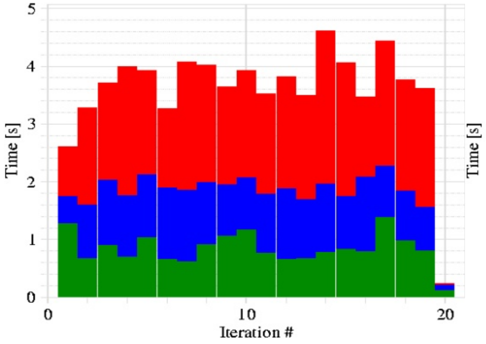
127.wrf2



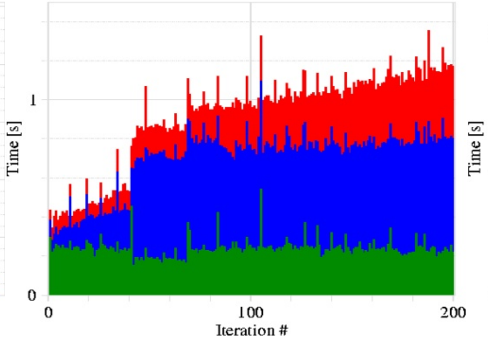
128.GAPgeofem



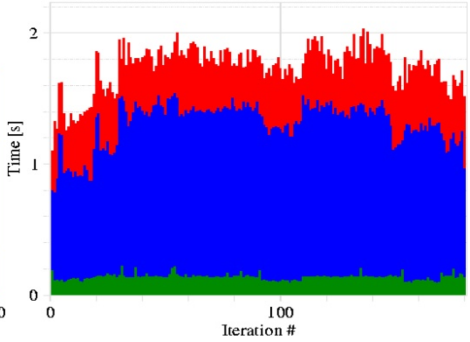
129.tera_tf



130.socorro



132.zeusmp2



137.lu

Scalasca's approach to performance dynamics

Overview

- Capture overview of performance dynamics via time-series profiling
    - Time and count-based metrics

Focus

- Identify pivotal iterations - if reproducible

In-depth analysis

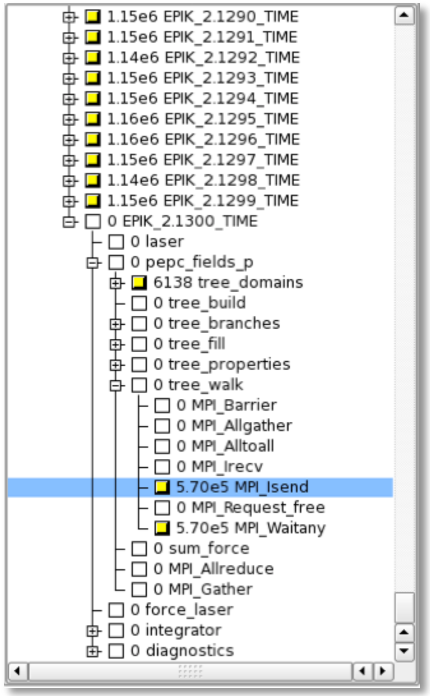
- In-depth analysis of these iterations via tracing
    - Analysis of wait-state formation
    - Critical-path analysis
    - Tracing restricted to iterations of interest

New

Time-series call-path profiling

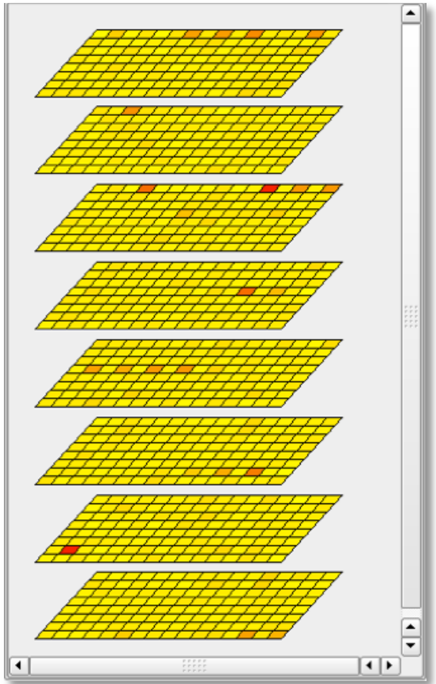
- Instrumentation of the main loop to distinguish individual iterations
  - Complete call tree with multiple metrics recorded for each iteration
  - Challenge: storage requirements proportional to #iterations

```
#include "epik_user.h"

void initialize() {}
void read_input() {}
void do_work() {}
void do_additional_work() {}
void finish iteration() {}
void write_output() {}

int main() {
  int iter;
  PHASE_REGISTER(iter,”ITER”);
  int t;
  initialize();
  read_input();
  for(t=0; t<5; t++) {
    PHASE_START(iter);
    do_work();
    do_additional_work();
    finish_iteration();
    PHASE_END(iter);
  }
  write_output();

  return 0;
}
```



```
⊞ ■ 1.15e6 EPIK_2.1290_TIME
⊞ ■ 1.15e6 EPIK_2.1291_TIME
⊞ ■ 1.14e6 EPIK_2.1292_TIME
⊞ ■ 1.15e6 EPIK_2.1293_TIME
⊞ ■ 1.15e6 EPIK_2.1294_TIME
⊞ ■ 1.16e6 EPIK_2.1295_TIME
⊞ ■ 1.16e6 EPIK_2.1296_TIME
⊞ ■ 1.15e6 EPIK_2.1297_TIME
⊞ ■ 1.14e6 EPIK_2.1298_TIME
⊞ ■ 1.15e6 EPIK_2.1299_TIME
⊟ ☐ 0 EPIK_2.1300_TIME
    ☐ 0 laser
  ⊟ ☐ 0 pepc_fields_p
    ⊞ ■ 6138 tree_domains
      ☐ 0 tree_build
    ⊞ ☐ 0 tree_branches
    ⊞ ☐ 0 tree_fill
    ⊞ ☐ 0 tree_properties
    ⊟ ☐ 0 tree_walk
        ☐ 0 MPI_Barrier
        ☐ 0 MPI_Allgather
        ☐ 0 MPI_Alltoall
        ☐ 0 MPI_Irecv
        ■ 5.70e5 MPI_Isend
        ☐ 0 MPI_Request_free
        ■ 5.70e5 MPI_Waitany
      ☐ 0 sum_force
      ☐ 0 MPI_Allreduce
      ☐ 0 MPI_Gather
    ☐ 0 force_laser
  ⊞ ☐ 0 integrator
  ⊞ ☐ 0 diagnostics
```

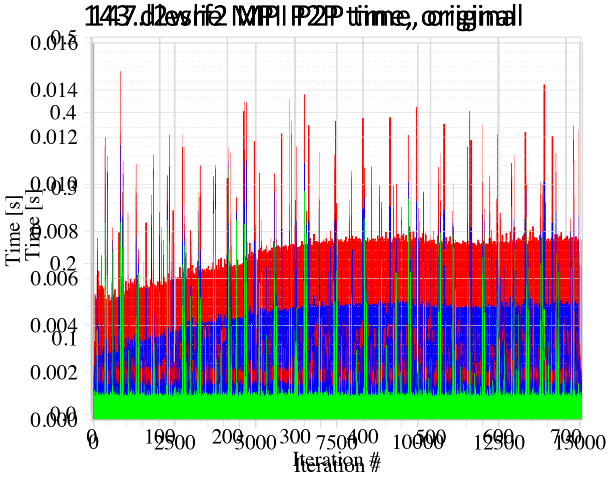
Call tree

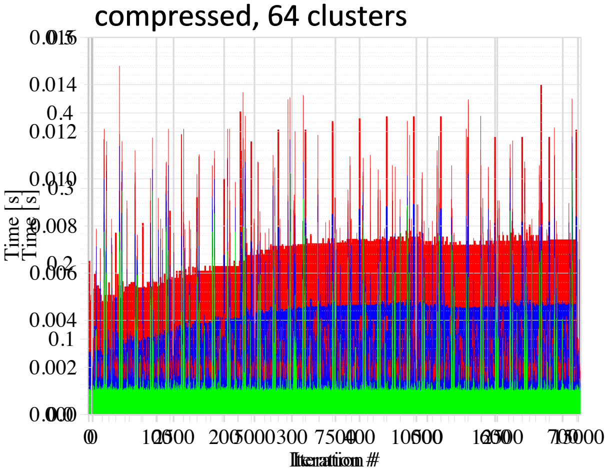


Process topology

Online compression

- Exploits similarities between iterations
  - Summarizes similar iterations in a single iteration via clustering and structural comparisons
- On-line to save memory at run-time
- Process-local to
  - Avoid communication
  - Adjust to local temporal patterns
- The number of clusters never exceeds a predefined maximum
  - Merging of the two closest ones

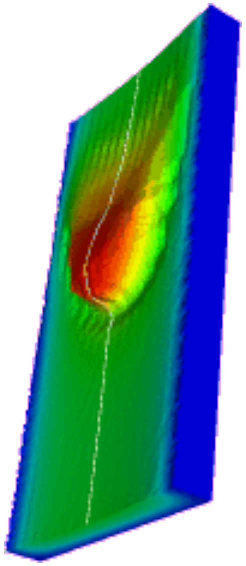



compressed, 64 clusters


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.

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

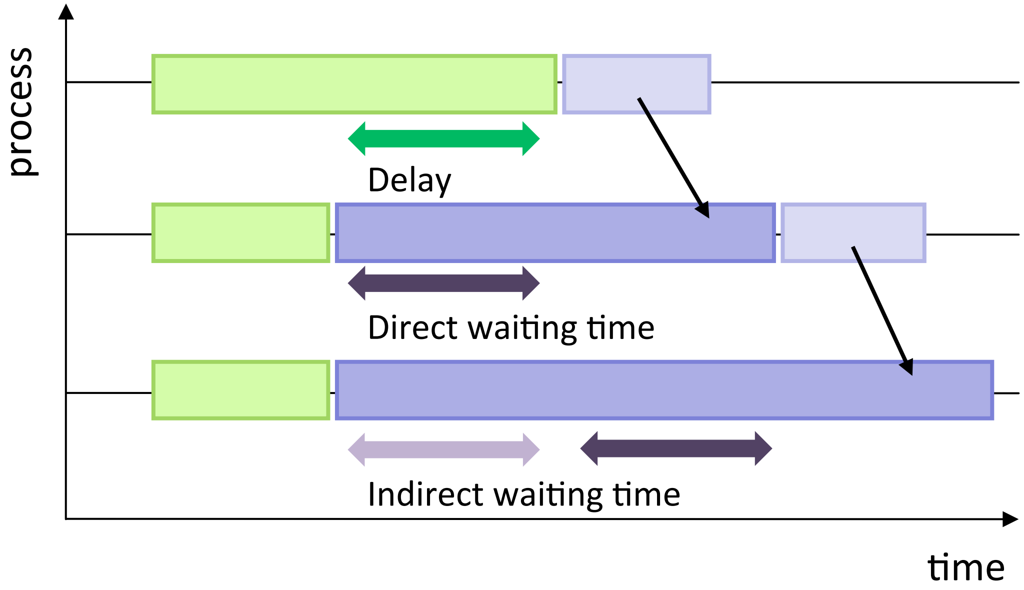


DROPS
IGPM & SC, RWTH

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.

Delay analysis

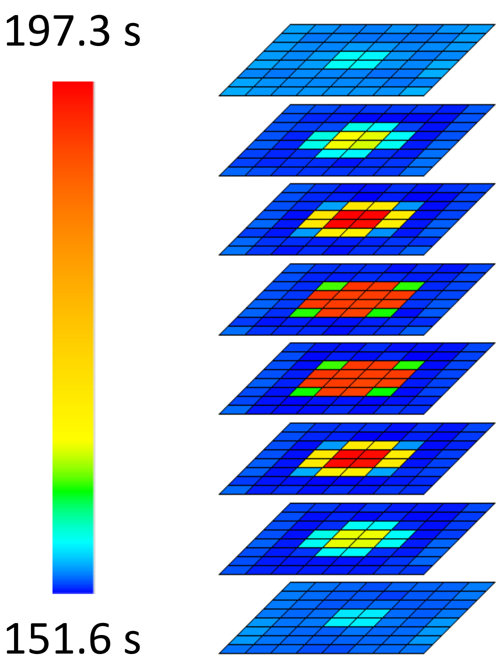


- Classification of waiting times into
  - Direct vs. indirect
  - Propagating vs. terminal
- Attributes costs of wait states to delay intervals
  - Scalable through parallel forward and backward replay of traces

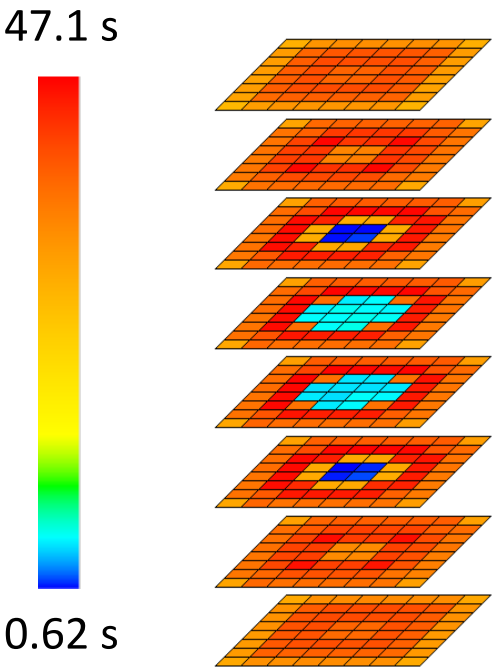
David Böhme et al.: Identifying the root causes of wait states in large-scale parallel applications. In Proc. of ICPP, San Diego, CA, IEEE Computer Society, September 2010.
Best Paper Award

Zeus-MP/2

- Performance solving 3-D magnetohydrodynamic blast wave problem on 512 processes



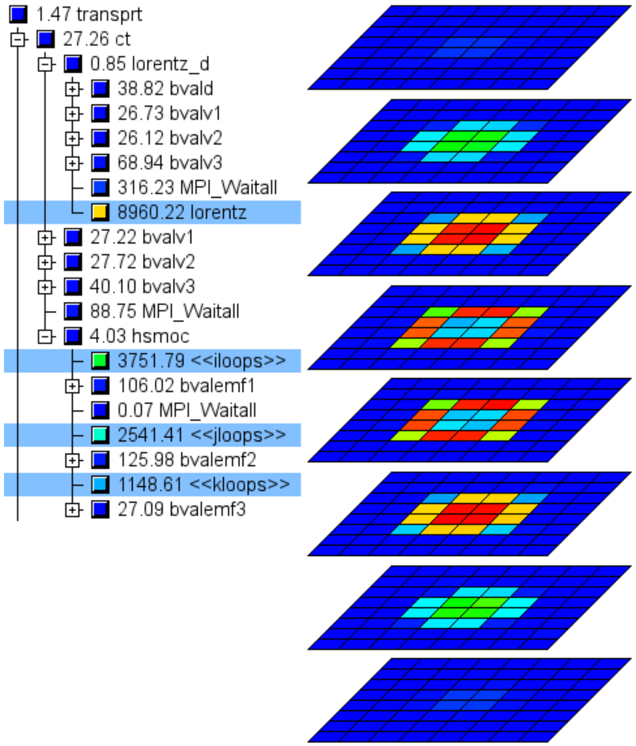
Computation



Late-sender wait states

Zeus-MP/2 delay analysis

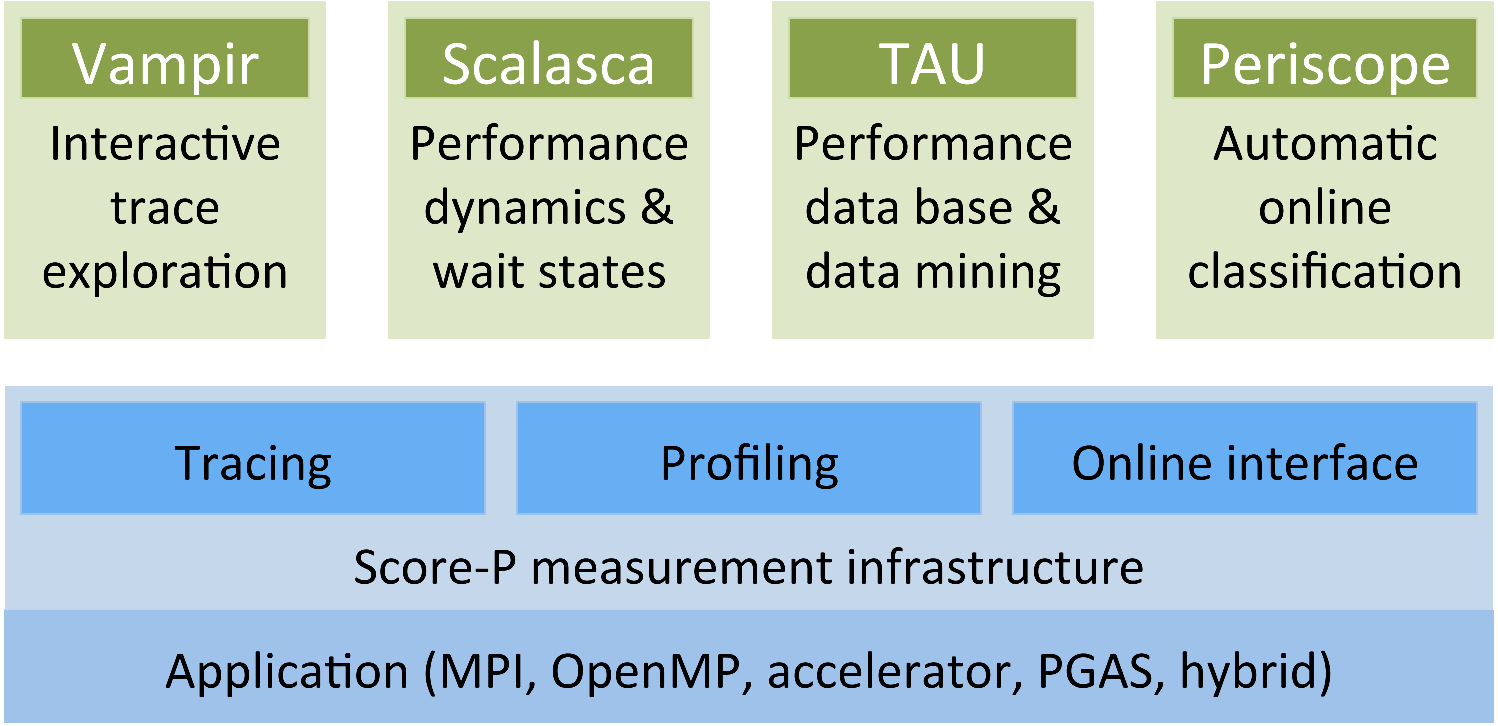
- Subroutine “lorentz” has highest delay costs
- Delay originates from border of central region
- Cost distribution:
  - 15.9 % short-term
  - 84.1 % long-term



```
1.47 transprt
27.26 ct
    0.85 lorentz_d
        38.82 bvald
        26.73 bvalv1
        26.12 bvalv2
        68.94 bvalv3
        316.23 MPI_Waitall
        8960.22 lorentz
    27.22 bvalv1
    27.72 bvalv2
    40.10 bvalv3
    88.75 MPI_Waitall
    4.03 hsmoc
        3751.79 <<iloops>>
        106.02 bvalemf1
        0.07 MPI_Waitall
        2541.41 <<jloops>>
        125.98 bvalemf2
        1148.61 <<kloops>>
        27.09 bvalemf3
```

Delay cost distribution across process topology

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

- Integrate into production version
  - Time-series compression
  - Hybrid measurement technique
  - Delay & critical-path analysis
- Further scalability improvements
- Emerging architectures and programming models
  - Accelerators
- Interoperability with 3rd-party tools
  - Common measurement library for several performance tools
- Support for performance modeling
  - Performance extrapolation
  - Multi-experiment analysis

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