Runtime analysis of parallel applications for industrial software development

Markus Geimer and Christian Feld, Forschungszentrum Jülich GmbH
Daniel Becker, Siemens AG

Outline

Application requirements

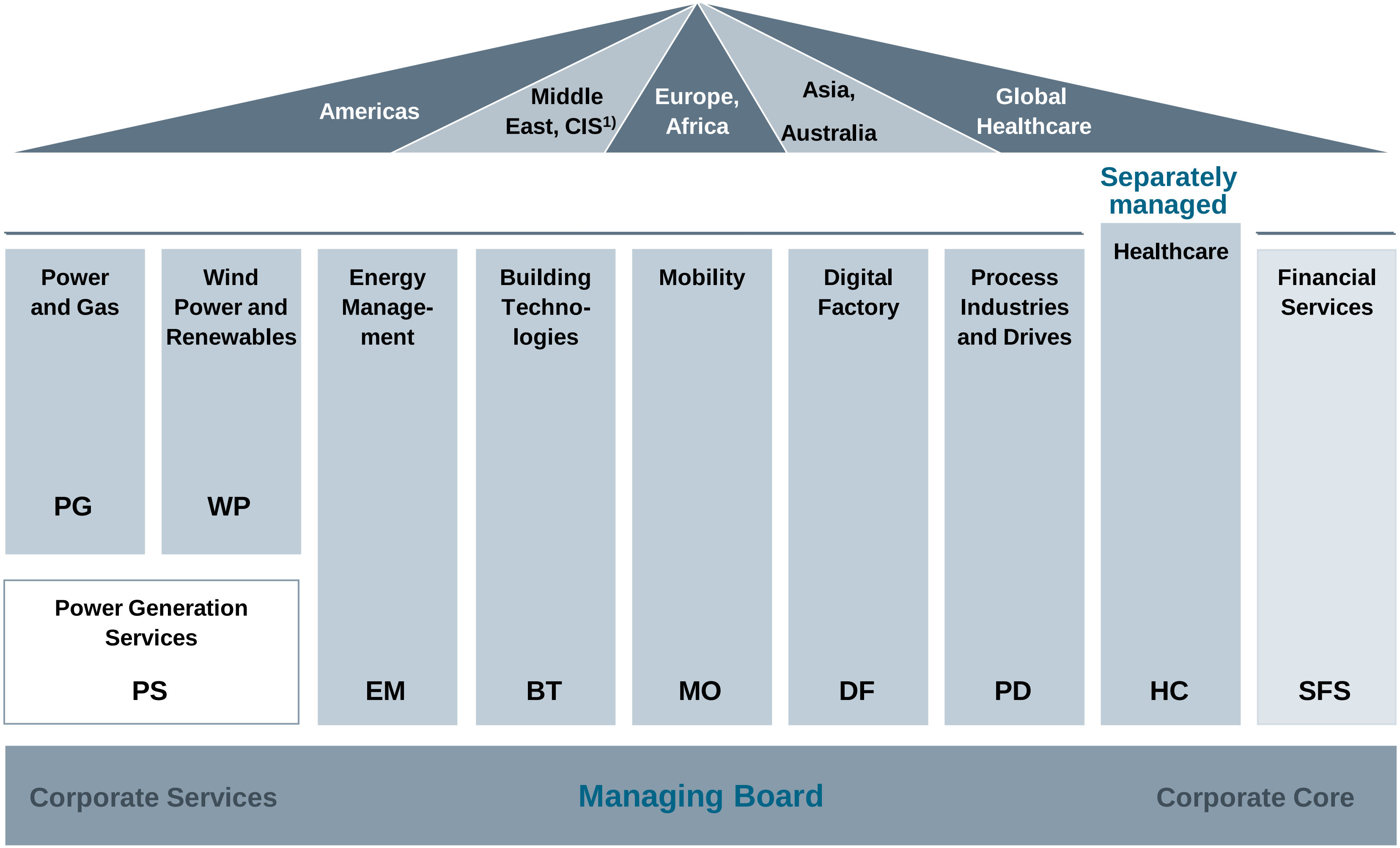
Multicore tool support

Runtime recording with Score-P

Supportive tool stack

Summary

Siemens AG



Americas

Middle East, CIS[1]

Europe, Africa

Asia, Australia

Global Healthcare

Separately managed

Power and Gas PG

Wind Power and Renewables WP

Power Generation Services PS

Energy Management EM

Building Technologies BT

Mobility MO

Digital Factory DF

Process Industries and Drives PD

Healthcare HC

Financial Services SFS

Corporate Services

Managing Board

Corporate Core

[1] Commonwealth of Independent States

Typical multicore migration

Let us assume that an application has already multiple threads…

Concurrency bugs

- Data races
- Deadlocks/livelocks
- Nondeterminism
- Memory model
- …

Poor performance

- Synchronization & communication
- Contention on shared resources
- …

High effort

- Explicit thread management and synchronization
- Limited portability
- …

Multicore migration is extremely challenging

Outline

Application requirements

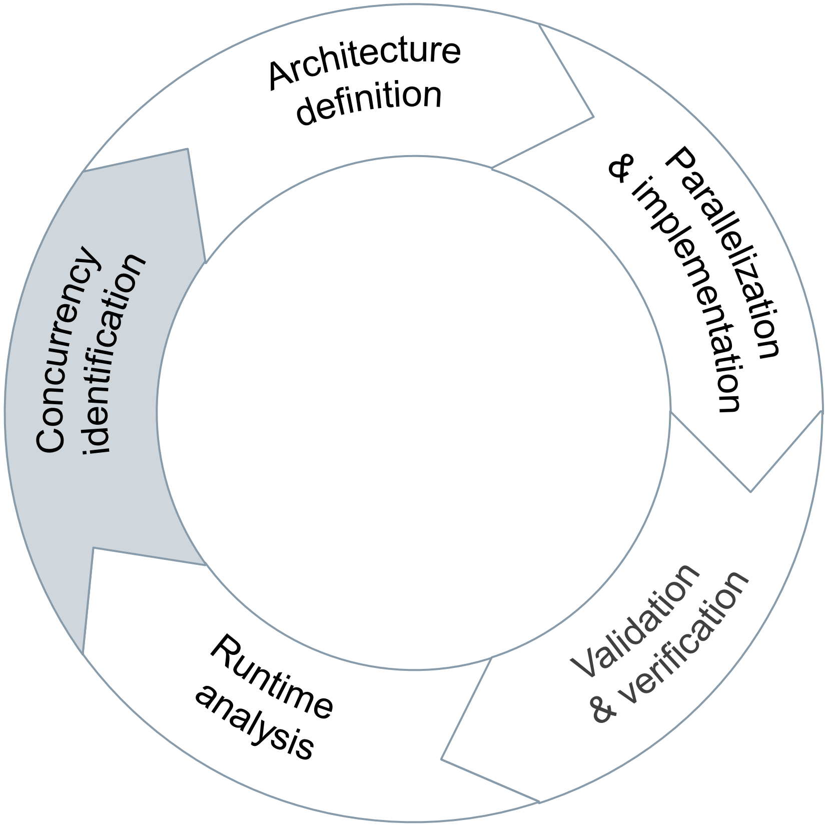
Multicore tool support

Runtime recording with Score-P

Supportive tool stack

Summary

Design process towards multicore software



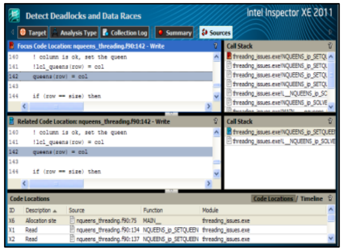
Architecture definition

Parallelization & implementation

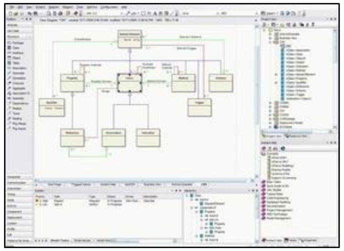
Validation & verification

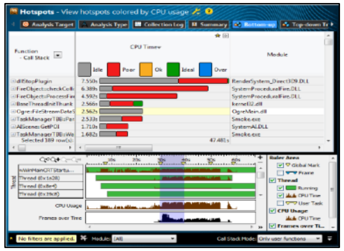
Runtime analysis

Concurrency identification



Vampir - Trace View - /Vampir/Large/wrf.otf
Function Summary
Context View
Communication Matrix View





Tool support is essential for an effective and efficient parallelization

Multicore migration scenario

Multicore migration scenario

- Legacy code is code where nobody understands the details
- Runtime behavior is also complex and hard to follow
- Manual extraction of information is cumbersome
- Profiling and tracing tools are essential
  - To understand and debug
  - To engineer and optimize the runtime

Requirements for profiling & tracing systems

- Focus on understanding the application and its parallel aspects
  - Threads & processes
  - Locks & messages
- Portable to Windows, Linux
- HW independent (x86, ARM, and ppc)
- Heterogeneous system support (e.g., Intel Xeon Phi, CUDA)
- Formats enabling interoperability and custom analysis types

Outline

Application requirements

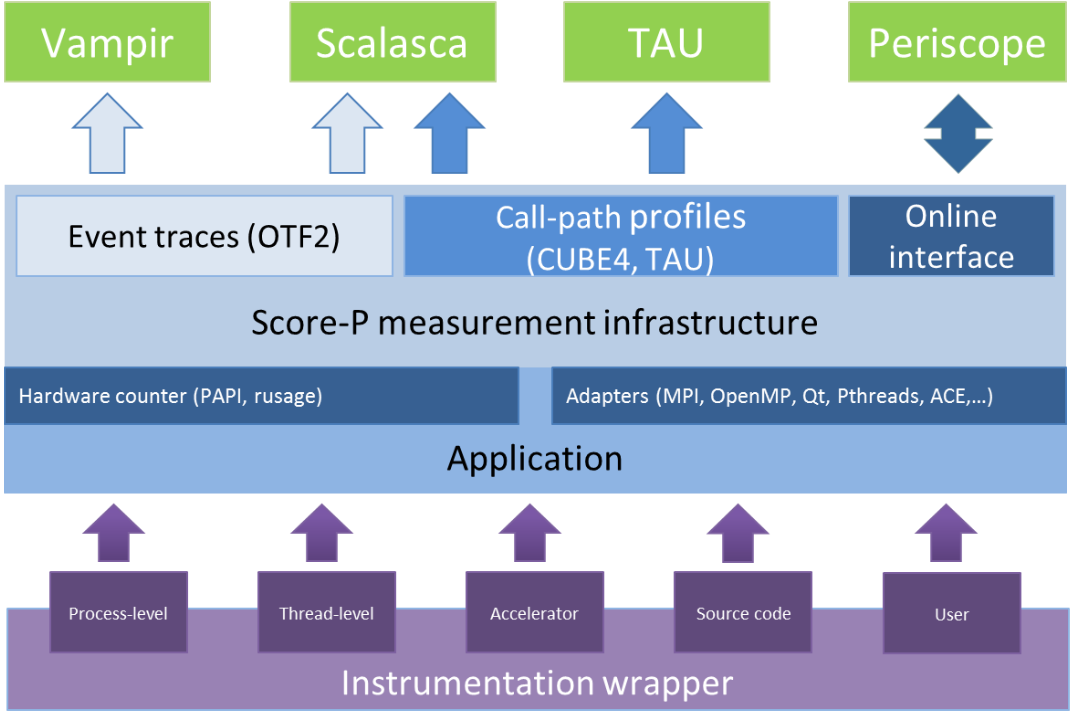
Multicore tool support

Runtime recording with Score-P

Supportive tool stack

Summary

Runtime recording with Score-P



- Open source community
- Linux (& Windows)
- HW independent (x86, ARM, PPC, …)
- Heterogeneous systems (e.g., Intel Xeon Phi, CUDA)
- Open formats enabling interoperability and custom analysis types
- Extremely scalable

Bundesministerium für Bildung und Forschung
U.S. DEPARTMENT OF ENERGY | Office of Science

JÜLICH FORSCHUNGSZENTRUM
TECHNISCHE UNIVERSITÄT DRESDEN
RWTH AACHEN UNIVERSITY
TUM
German Research School for Simulation Sciences
UNIVERSITY OF OREGON

www.score-p.org

SIEMENS

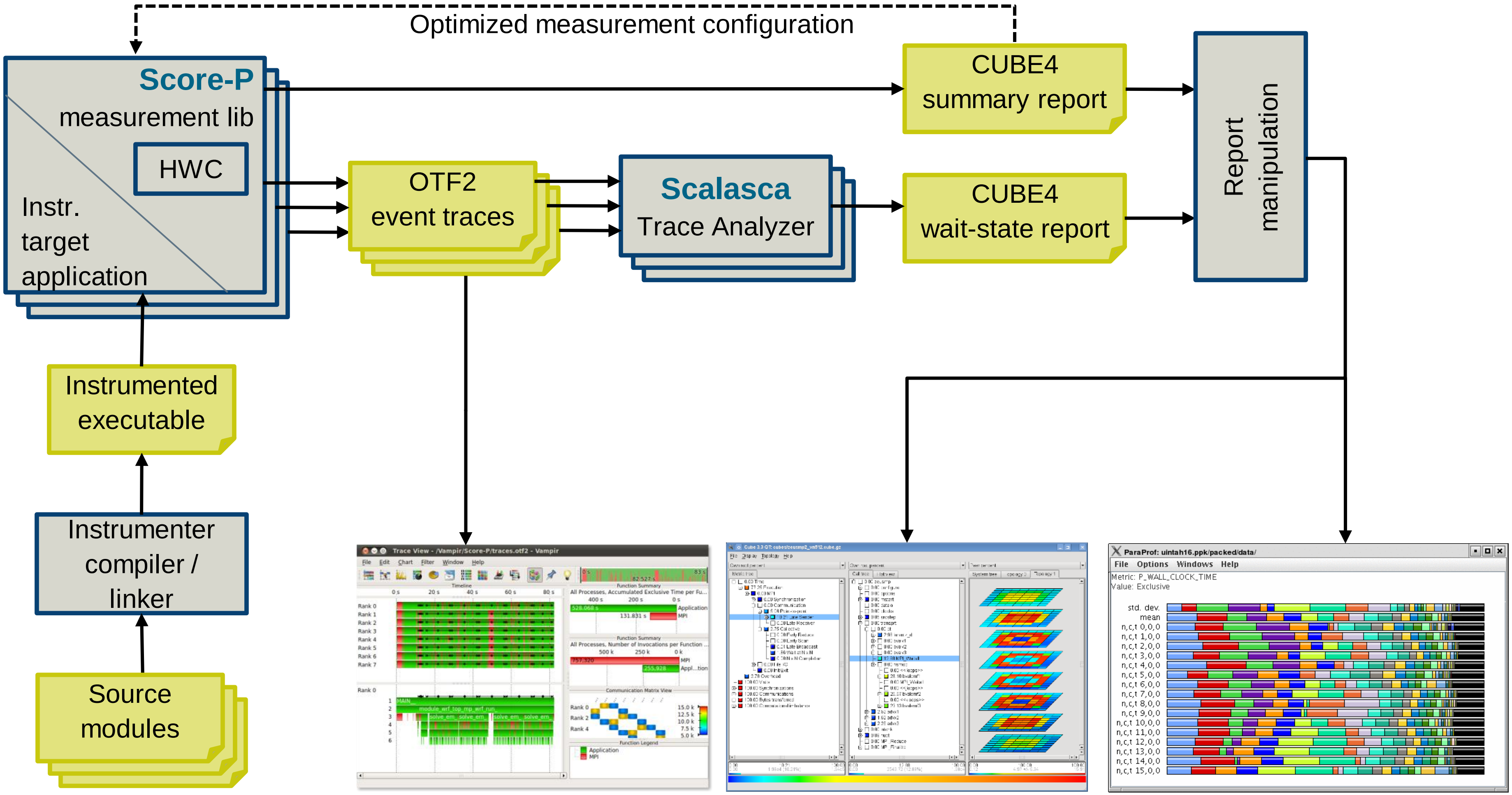
Score-P 1.3

Key features

- Provides typical functionality for HPC performance tools
  - Support for process-level parallelism using MPI/SHMEM
  - Support for thread-level parallelism using OpenMP/Pthreads
  - Support for accelerator-level parallelism using CUDA
- Based on instrumentation
  - Supports various techniques
  - Extensive runtime filtering & selective recording capabilities
- Flexible measurement with single re-compilation
  - Basic and advanced profile generation
  - Event trace recording
  - Online access to profiling data
- Scalability: Petascale
- Portability: Supports all major HPC platforms, incl.
  IBM Blue Gene, Cray XT/XE/XK/XC, Fujitsu FX10 & K computer,
  SGI Altix, Power/AIX, Linux-based clusters (x86, ARM, Power)
- Open source: 3-clause BSD license

JUQUEEN
28 rack IBM Blue Gene/Q
28,672 nodes (458,752 cores, 4-way SMT)
448 TB RAM, 5.9 Petaflop/s peak

Score-P workflow



Optimized measurement configuration

Score-P measurement lib

HWC

Instr. target application

Instrumented executable

Instrumenter compiler / linker

Source modules

OTF2 event traces

Scalasca Trace Analyzer

CUBE4 summary report

CUBE4 wait-state report

Report manipulation

Vampir

Cube

TAU ParaProf

Score-P instrumentation options

Manual instrumentation

- Extensive API (C/C++/Fortran), supporting
  - Program phases
  - Functions
  - Arbitrary code regions
  - Measurement control

Automatic instrumentation

- By using the compiler
  (GCC, Intel, PGI, IBM xl, Cray, Fujitsu)
- By using source-to-source translation
  (Opari2, PDToolkit)
- By linking against a pre-instrumented library
  (MPI, SHMEM)
- By using GNU linker symbol renaming
  (POSIX threads, SHMEM)

Automatic binary instrumentation using Pin

- Pin: dynamic binary instrumentation tool from Intel
- Flexible instrumentation at image-load time
  - No recompilation necessary
  - But only supports x86 architectures

- **Current prototype supports**
  - Function wrapping
  - Incl. pre-runtime filtering (i.e., file level, region level, shared-object level)
  - Replacement of threading API routines
    (by calls to Score-P measurement system)
    - POSIX threads
    - Windows threads
    - Qt thread API
    - ACE threads

Future work: Support for MTAPI (Multicore Association), Intel TBB

Outline

Application requirements

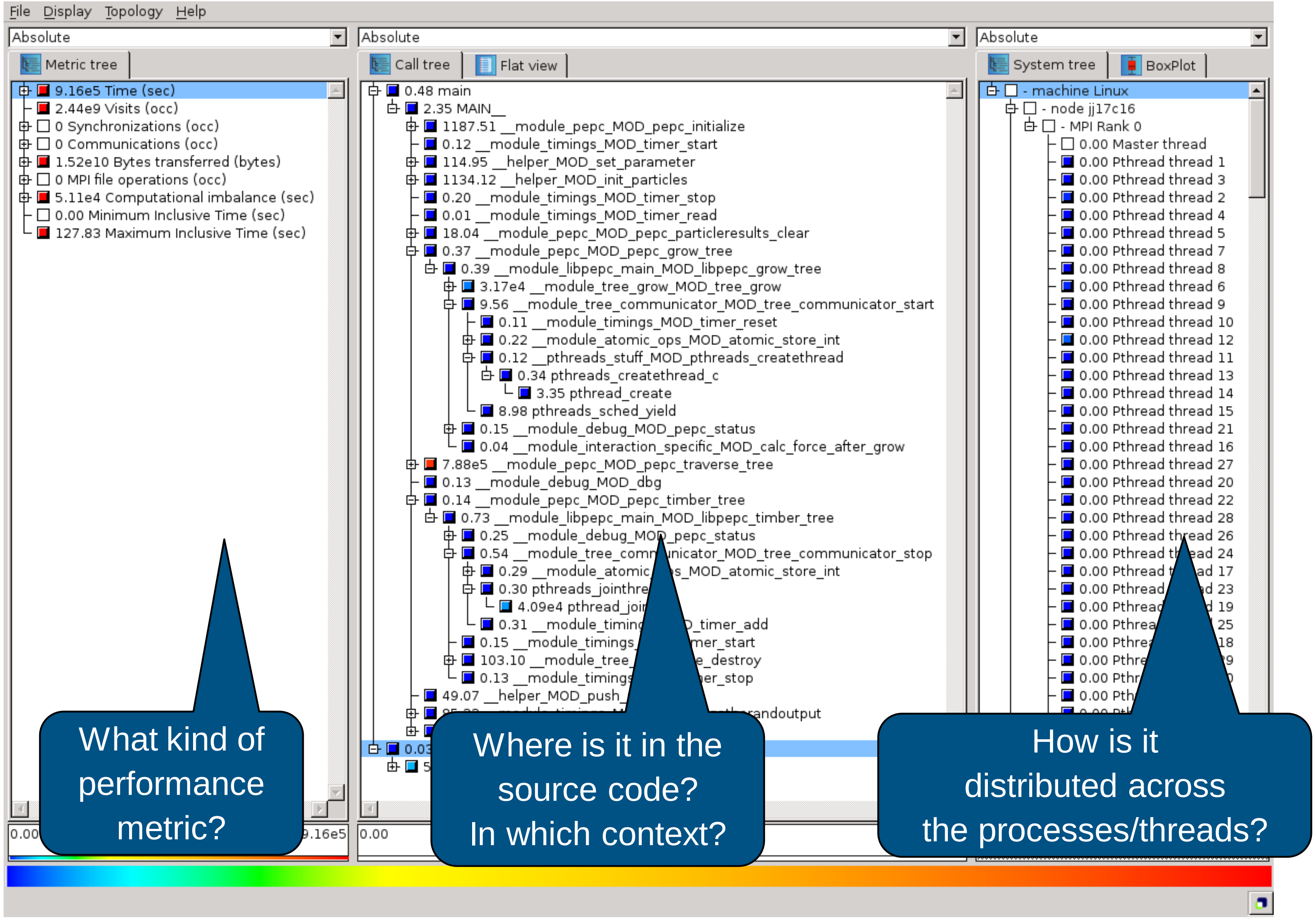
Multicore tool support

Runtime recording with Score-P

Supportive tool stack

Summary

Cube profile viewer



```
9.16e5 Time (sec)
2.44e9 Visits (occ)
0 Synchronizations (occ)
0 Communications (occ)
1.52e10 Bytes transferred (bytes)
0 MPI file operations (occ)
5.11e4 Computational imbalance (sec)
0.00 Minimum Inclusive Time (sec)
127.83 Maximum Inclusive Time (sec)
```

```
0.48 main
  2.35 MAIN__
    1187.51 __module_pepc_MOD_pepc_initialize
    0.12 __module_timings_MOD_timer_start
    114.95 __helper_MOD_set_parameter
    1134.12 __helper_MOD_init_particles
    0.20 __module_timings_MOD_timer_stop
    0.01 __module_timings_MOD_timer_read
    18.04 __module_pepc_MOD_pepc_particleresults_clear
    0.37 __module_pepc_MOD_pepc_grow_tree
      0.39 __module_libpepc_main_MOD_libpepc_grow_tree
        3.17e4 __module_tree_grow_MOD_tree_grow
        9.56 __module_tree_communicator_MOD_tree_communicator_start
          0.11 __module_timings_MOD_timer_reset
          0.22 __module_atomic_ops_MOD_atomic_store_int
          0.12 __pthreads_stuff_MOD_pthreads_createthread
            0.34 pthreads_createthread_c
              3.35 pthread_create
          8.98 pthreads_sched_yield
        0.15 __module_debug_MOD_pepc_status
        0.04 __module_interaction_specific_MOD_calc_force_after_grow
    7.88e5 __module_pepc_MOD_pepc_traverse_tree
    0.13 __module_debug_MOD_dbg
    0.14 __module_pepc_MOD_pepc_timber_tree
      0.73 __module_libpepc_main_MOD_libpepc_timber_tree
        0.25 __module_debug_MOD_pepc_status
        0.54 __module_tree_comm unicator_MOD_tree_communicator_stop
          0.29 __module_atomic s_MOD_atomic_store_int
          0.30 pthreads_jointhre
            4.09e4 pthread_joi
          0.31 __module_timing _timer_add
        0.15 __module_timings_ mer_start
        103.10 __module_tree_ e_destroy
        0.13 __module_timings er_stop
    49.07 __helper_MOD_push
    85.23 __module_timings_M randoutput
0.03
  5
```

```
- machine Linux
  - node jj17c16
    - MPI Rank 0
      0.00 Master thread
      0.00 Pthread thread 1
      0.00 Pthread thread 3
      0.00 Pthread thread 2
      0.00 Pthread thread 4
      0.00 Pthread thread 5
      0.00 Pthread thread 7
      0.00 Pthread thread 8
      0.00 Pthread thread 6
      0.00 Pthread thread 9
      0.00 Pthread thread 10
      0.00 Pthread thread 12
      0.00 Pthread thread 11
      0.00 Pthread thread 13
      0.00 Pthread thread 14
      0.00 Pthread thread 15
      0.00 Pthread thread 21
      0.00 Pthread thread 16
      0.00 Pthread thread 27
      0.00 Pthread thread 20
      0.00 Pthread thread 22
      0.00 Pthread thread 28
      0.00 Pthread thread 26
      0.00 Pthread th ead 24
      0.00 Pthread t ad 17
      0.00 Pthread d 23
      0.00 Pthread d 19
      0.00 Pthrea 25
      0.00 Pthrea 18
      0.00 Pthre 9
      0.00 Pthr 0
      0.00 Pth
      0.00 Pth
```

What kind of performance metric?

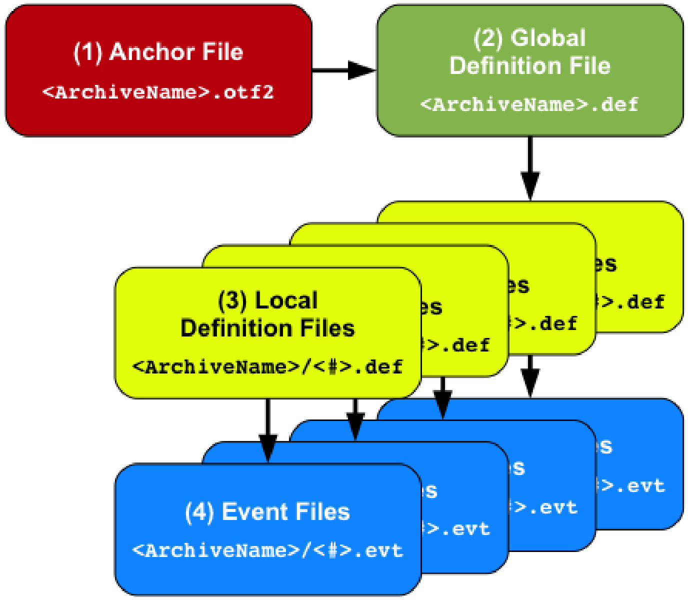
Where is it in the source code? In which context?

How is it distributed across the processes/threads?

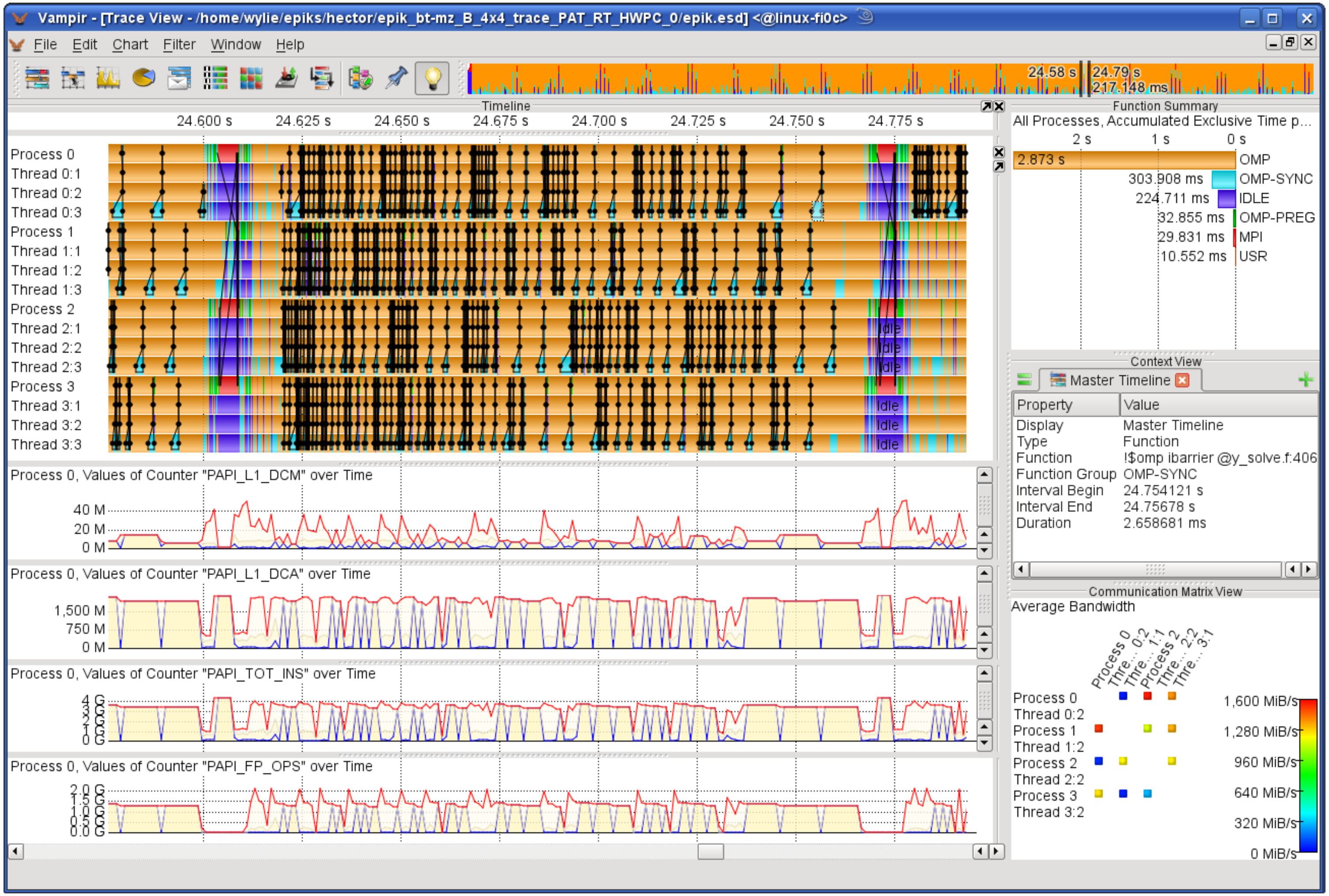
Open Trace Format 2 (OTF2)

Key facts

- Successor to OTF (Vampir) and EPILOG (Scalasca)
- Very flexible, scalable, and space-efficient multi-file format
- Supports many event types
  - Enter/Exit regions/phases/…
  - Point-to-point, collective, one-sided communication, and synchronization
  - Fork/join and create/wait threading
  - HW + SW counters
  - Extensible event attributes
- Well-defined read/write C API



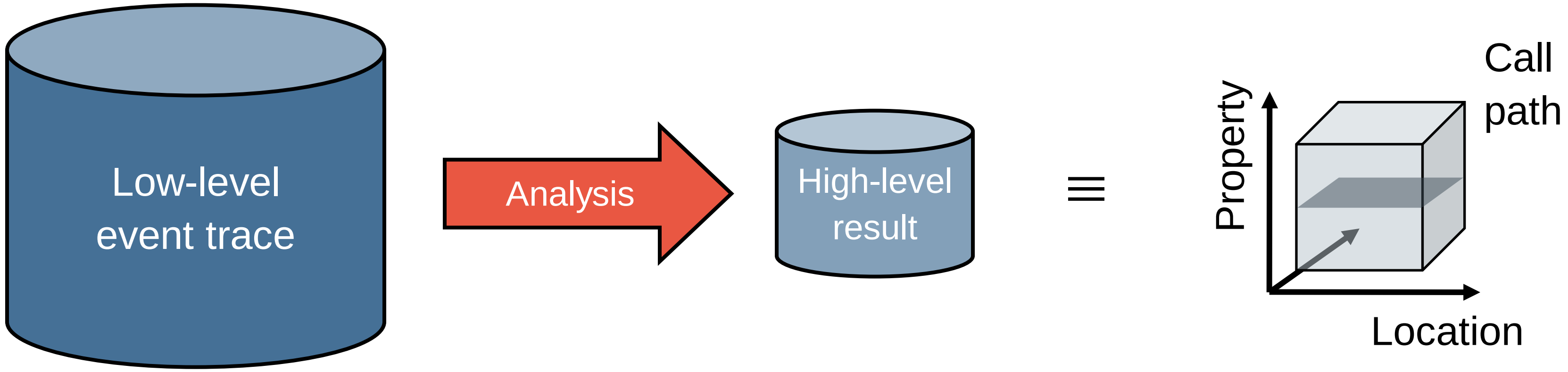
Time-line visualization in Vampir (TU Dresden)



Property	Value
Display	Master Timeline
Type	Function
Function	!$omp ibarrier @y_solve.f:406
Function Group	OMP-SYNC
Interval Begin	24.754121 s
Interval End	24.75678 s
Duration	2.658681 ms

Automatic trace analysis with Scalasca

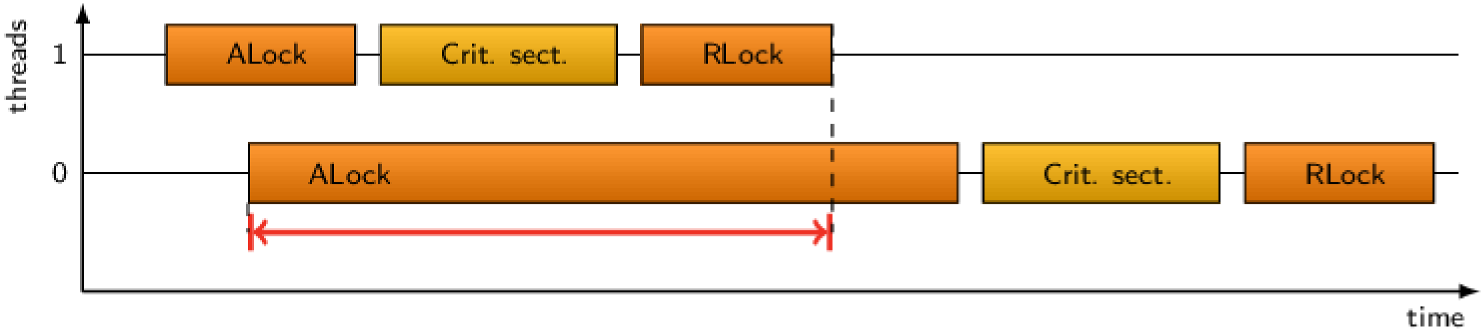
- **Idea**
  - Automatic search for patterns of inefficient behavior
  - Classification of behavior & quantification of significance



- **Advantages**
  - Guaranteed to cover the entire event trace
  - Quicker than manual/visual trace analysis
  - Parallel replay analysis exploits available memory & processors to deliver scalability

SIEMENS

Example: Lock contention analysis



- Frequent and severe problem during multicore migration
- Automatic determination of waiting times in acquire lock operations
- Easy identification of blocking threads in different call paths

- Currently supports
  - OpenMP critical sections & lock API
  - Pthread mutex & condition variable APIs

- Future work
  - Determine root-cause of waiting time

Outline

Application requirements

Multicore tool support

Runtime recording with Score-P

Supportive tool stack

Summary

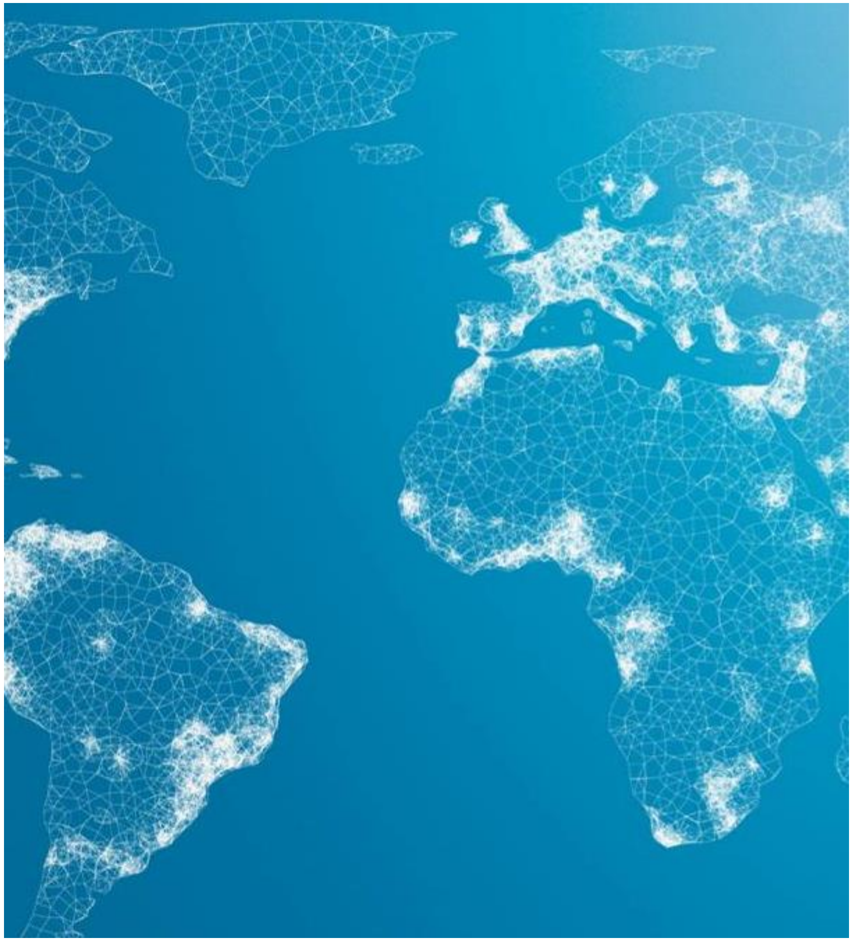
Summary

- Parallelizing legacy software is tedious
  - Developers have to understand all parts of the software
  - Getting synchronization right *and* efficient is challenging

- Profiling and tracing tools
  - Enable developers to understand, debug, engineer, and optimize their application
  - Have to be portable and extendable
  - Should allow to focus only on relevant parts of the application

- Score-P fulfills these requirements and comes with an supportive tool stack enabling effective and efficient multicore migrations

More information and contacts



Multicore Expert Center, Siemens AG

- Dr. Daniel Becker
- becker.daniel@siemens.com

Score-P & OTF2

- http://www.score-p.org
- support@scorep.org

Scalasca & Cube

- http://www.scalasca.org
- scalasca@fz-juelich.de