Scalable performance analysis of large-scale parallel applications on Cray XT systems with Scalasca

Brian J. N. Wylie, **David Böhme**, **Wolfgang Frings**, **Markus Geimer**, **Bernd Mohr**, **Zoltán Szebenyi**
Jülich Supercomputing Centre, Forschungszentrum Jülich, Jülich, Germany
Daniel Becker, **Marc-André Hermanns**, **Felix Wolf**
German Research School for Simulation Sciences, Aachen, Germany

ABSTRACT: *The open-source Scalasca toolset (available from www.scalasca.org) supports integrated runtime summarization and automated trace analysis on a diverse range of HPC computer systems. An HPC-Europa2 visit to EPCC in 2009 resulted in significantly enhanced support for Cray XT systems, particularly the auxilliary programming environments and hybrid OpenMP/MPI. Combined with its previously demonstrated extreme scalability and portable performance analyses comparison capabilities, Scalasca has been used to analyse and tune numerous key applications (and benchmarks) on Cray XT and other PRACE prototype systems, from which experience with a representative selection is reviewed.*

KEYWORDS: OpenMP/MPI, parallel/distributed systems, performance measurement and analysis tools, scalability

1 Introduction

Scalasca is an open-source toolset for analysing the execution behaviour of applications based on the MPI and/or OpenMP parallel programming interfaces supporting a wide range of current HPC platforms [5]. It combines compact runtime summaries, that are particularly suited to obtaining an overview of execution performance, with in-depth analyses of concurrency inefficiencies via event tracing and parallel replay. With its highly scalable design, Scalasca has facilitated performance analysis and tuning of applications consisting of unprecedented numbers of processes [13].

The predecessor of the Scalasca toolset, KOJAK, supported measurement and automatic trace analysis on Cray T3E, X1, XD1 and XT systems, and Scalasca inherited those capabilities, however, increasing usage and demand warranted targeted enhancement of support for the latest Cray XT systems. With the support of the HPC-Europa2 Transnational Access programme of the European Union, a visit to EPCC in 2009 with access to the *HECToR* system was arranged, allowing face-to-face interaction with local performance analysts and application developers to determine specific requirements and desires, as well as an opportunity to assist them in exploiting performance analysis and optimisation opportunities. [12]

This paper provides an overview of the Scalasca toolset and its usage on Cray XT systems, covers the recent enhancements (many of which also apply to other systems), and reviews experience using Scalasca with a variety of applications.

2 Scalasca overview

Scalasca supports measurement and analysis of MPI applications written in C, C++ and Fortran on a wide range of current HPC platforms [11, 14]. Hybrid codes making use of basic OpenMP features in addition to message passing are also supported. Figure 1 shows the Scalasca workflow for instrumentation, measurement, analysis and presentation.

Before performance data can be collected, the target application must be instrumented and linked to the measurement library. The instrumenter for this purpose is used as a prefix to the usual compile and link commands, offering a variety of manual and automatic instrumentation options. MPI operations are captured simply via re-linking, whereas a source preprocessor is used to instrument OpenMP parallel regions. Often compilers can be directed to automatically instrument the entry and exits of user-level source routines, or the PDToolkit source-code instrumenter can be used for more selective instrumentation of routines [4]. Finally, programmers can manually add custom instrumentation annotations into the source code for important regions via macros or pragmas which are ignored when instrumentation is not activated.

The Scalasca measurement and analysis nexus configures and manages collection of execution performance experiments, which is similarly used as a prefix to the parallel execution launch command of the instrumented application executable (i.e., `aprun` on Cray XT systems) and results in the generation of a unique experiment archive directory containing measurement and analysis artifacts, including log files and configuration information.

Users can choose between generating a summary analysis report ('profile') with aggregate performance metrics for each function callpath and/or generating event traces recording runtime events from which a profile or time-line visualization can later be produced. Summarization is particularly useful to obtain an overview of the performance behaviour and for local metrics such as those derived from hardware counters. Since

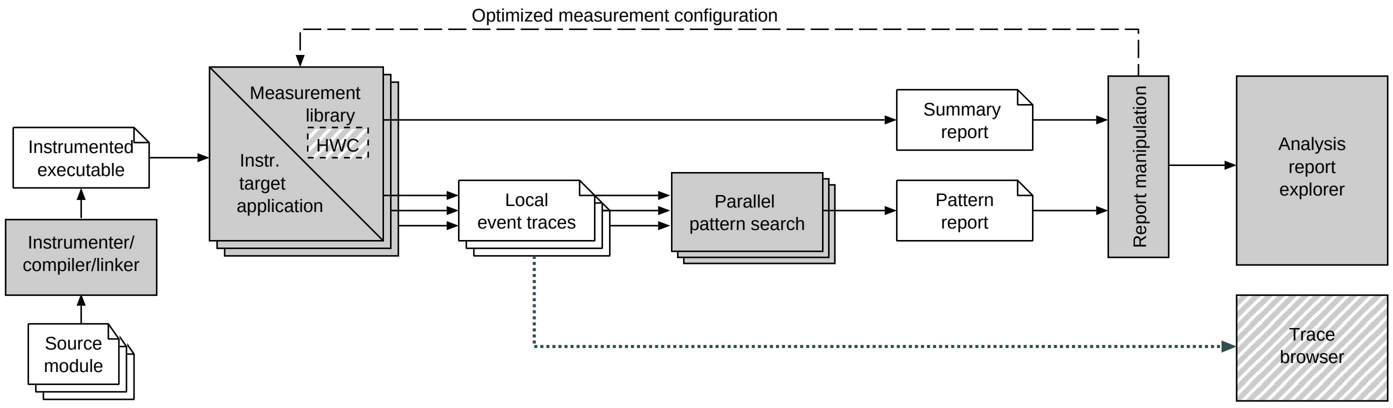


Figure 1: Schematic overview of Scalasca instrumentation, measurement, analysis and presentation.

measurement overheads can be prohibitive for small routines that are executed often and traces tend to rapidly become very large, optimizing the instrumentation and measurement configuration based on the summary report is usually recommended. When tracing is enabled, each process generates a trace containing records for its process-local events: by default separate files are created for each MPI rank, or SIONlib can be used to improve file handling by transparently mapping task-local files into a smaller number of physical files [3]. After program termination (and with the same partition of processors), the Scalasca nexus automatically loads the trace files into main memory and analyzes them in parallel using as many processes as have been used for the target application itself. During trace analysis, Scalasca searches for wait states and related performance properties, classifies detected instances by category, and quantifies their significance. The result is a wait-state report similar in structure to the summary report but enriched with higher-level communication and synchronization inefficiency metrics.

Both summary and wait-state reports contain performance metrics for every measured function callpath and process/thread which can be interactively examined in the provided analysis report explorer. Prior to initial presentation, raw measurement reports are processed to derive additional metrics and structure the resulting set of metrics in hierarchies. Additional processing to combine reports or extract sections produces new reports with the same format. Scalasca event traces may also be examined directly (or after conversion if necessary) by various third-party trace visualization and analysis tools.

3 Scalasca enhancements for Cray XT

Initial Scalasca support for Cray XT3 systems was limited to the PGI programming environment and dual-core Opteron compute nodes running Catamount, but otherwise supported the full range of Scalasca instrumentation, measurement and analysis functionality at the time. This included appropriate configuration of runtime libraries and components that run on the compute nodes as well as instrumentation, post-processing and presentation components for login and service nodes. Since Cray compute nodes don't have globally synchronized high-resolution clocks, timestamps on events in traces need to be adjusted for logical event order consistency during analysis [2].

Capturing the mappings of application processes to processors in the Cray XT physical torus was incorporated relatively early, allowing investigation of associations between performance and the placement of processes. This additional 'topology' information is inserted in Scalasca analysis reports during their post-processing using node mappings available from various XT utilities. Since an application typically uses only a subset of the entire XT system, and the partition allocated often consists of disjoint processors, the GUI for interactive analysis report exploration can trim unused sections of hardware from its presentation of the physical topology for a more compact view. As XT systems with quad-core and six-core compute nodes have become available, this approach has required only minor extensions.

Migration from Catamount to Compute Node Linux (CNL), and associated changes of MPI library, also involved only small changes to the Scalasca measurement libraries and how these are linked with instrumented application executables.

A more significant recent development has been the provision of multiple programming environments (PrgEnvs) on Cray XT systems, combined with support for 'hybrid' programming with OpenMP within the MPI processes running on SMP compute nodes. In addition to the default PGI environment, GNU, Intel, Pathscale and Cray's own CCE have made different compilers available. Since Scalasca already supported these compilers (apart from CCE), including OpenMP and hybrid OpenMP/MPI measurement, on a variety of systems, adaption for the Cray XT programming environments was straightforward. In this process, Scalasca support for PGI compilers was also enhanced to support runtime filtering of events for instrumented user-level source routines, thereby reducing measurement overhead and improving overall Scalasca usability on Cray XT systems. (Since newer versions of the PGI compilers support multiple instrumentation interfaces, this was also addressed.)

Configuring support for Scalasca source instrumentation with

PDToolkit and measurement including access to hardware counters with PAPI is the same for all programming environments. While a limited amount of interoperability is possible between code compiled and instrumented with different compilers, differences in OpenMP runtime systems require distinct configurations and installations of Scalasca. (Since MPI offers only source-level portability, multiple MPI library families, such as MPICH and OpenMPI, also require separate Scalasca installations.) The Cray XT programming environment modules abstract compile commands (such as `ftn` for Fortran compilation) as a convenience for users, however, this required careful dissociation during Scalasca installation, so that an appropriate configuration was built. A user switching from the default PGI programming environment to another therefore also needs to ensure that a corresponding version of Scalasca is used when instrumenting their application.

Typically that same version of Scalasca would also be used for measurement and analysis of the instrumented application execution. Since the Scalasca trace analyzer is also a parallel application with dependencies on the shared libraries of the programming environment, and it is automatically launched after measurement to analyse traces, it will generally fail to run in a different programming environment. (Switching environment to analyse already collected traces is also possible.)

Scalasca can be configured and installed along with its analysis report explorer GUI, CUBE3, however, this is generally undesirable when there are multiple configurations, as with the Cray programming environments. Since the GUI has no MPI or OpenMP dependencies, yet requires the (same) Qt4 toolkit libraries, each installation would be redundant at best. Since each programming environment configures different libraries or versions of libraries (e.g., for libgcc.so), these dependencies become problematic when building Scalasca as the GUI may require a different version from that of the parallel components. Fortunately, CUBE3 is also distributed separately and can be configured using generic library versions, such that a single installation can be used by Scalasca with all programming environments. Furthermore, since remote usage of GUI applications can be troublesome, due to performance and security issues, often it is preferable to install CUBE3 on a local system, where it can more conveniently be used to examine Scalasca experiment archives transferred from remote supercomputers.

4 Scalasca measurement & analysis case studies

Scalasca has been used to analyse and tune the execution of a range of parallel applications on Cray XT systems over the years, from which illustrative examples will now be reviewed.

4.1 SMG2000

Immediately after the integration of the formerly separate XT3 and XT4 systems at Oak Ridge National Laboratory in mid-2007, the *Jaguar* system consisted of 11,508 dual-core compute nodes running a Catamount microkernel. The ASC SMG2000 semi-coarsening multigrid solver benchmark [1] uses a com-

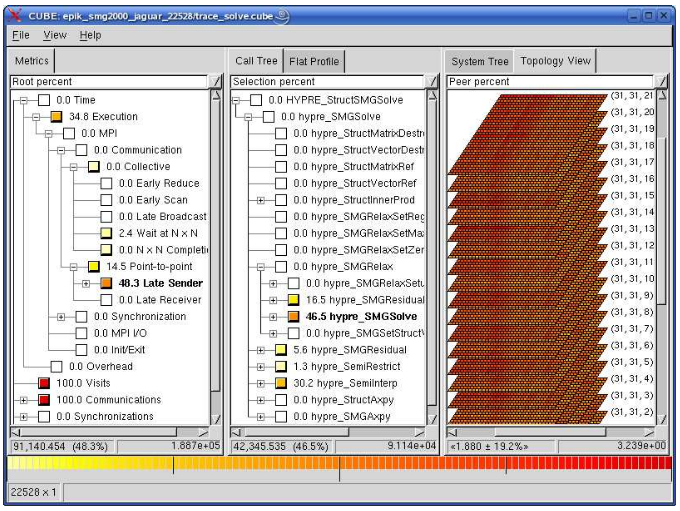


Figure 2: Scalasca analysis report explorer presentation of an SMG2000 trace experiment with 22,528 processes on the *Jaguar* Cray XT3/4 showing percentages of "Late Sender" time in a recursive `hypre_SMGSolve` callpath distributed by process on the application's three-dimensional grid. (Metric values in the trees and topology views are colour-coded according to the scale at the bottom of the window.)

plex communication pattern with a lot of non-nearest neighbour MPI point-to-point communication in a three-dimensional problem decomposition, and is considered a tough case for trace-based performance analysis tools. With an early version of Scalasca, source code routine instrumentation was generated by the PGI compiler, and the instrumented executable run with a $64 \times 64 \times 32$ problem size per process on 22,528 (22k) processes arranged in a $32 \times 32 \times 22$ grid. [14]

At this time, runtime filtering of compiler-instrumented routines was not implemented for the PGI compiler adapter in the Scalasca measurement library, such that up to 328 MB of trace event data was collected for some processes and 5 TB in total. At the end of measurement, writing these trace files and associated metadata took 20 minutes followed by an additional 16 minutes for the trace analysis on the same processors.

From the trace analysis, half of the total execution time was determined to be waiting time of early receivers blocked on senders yet to initiate message transfers, i.e., "Late Sender" time, as shown in Figure 2. (This metric is explained more fully in the next example.) Although not distinguishable in the figure due to the large number of processes, the distribution of waiting time per process indicated that certain processes in the interior of the grid were most responsible.

This was the largest experiment collected and analysed with Scalasca at the time which demonstrated the general viability of the toolset and its approach, however, it also helped identify scalability and performance issues that needed to be addressed. For example, targeted optimization of definition identifier unification immediately resulted in a six-fold improvement.

4.2 Sweep3D

As HPC systems have grown, scalability to what was formerly considered extreme scales has become essential both for applications and associated performance measurement and analysis tools. Recently experiments with the latest version of Scalasca on *Jaguar* with Sweep3D were undertaken to investigate measurement and analysis scalability.

The extensively studied ASCI Sweep3D benchmark code [7] solves a 1-group time-independent discrete ordinates neutron transport problem, calculating the flux of neutrons through each cell of a three-dimensional grid (i, j, k) along several directions (angles) of travel. Angles are split into eight octants, corresponding to one of the eight directed diagonals of the grid. It uses an explicit two-dimensional decomposition (i, j) of the three-dimensional computation domain, resulting in point-to-point communication of grid-points between neighbouring processes, and reflective boundary conditions. A wavefront process is employed in the i and j directions, combined with pipelining of blocks of k-planes and octants to expose parallelism.

To investigate scaling behaviour of Sweep3D for a large range of scales, the benchmark input was configured with a fixed-size $32 \times 32 \times 512$ subgrid for each process. The benchmark performs 12 iterations, with flux corrections (referred to as 'fixups') applied after 7 iterations. Built on *Jaguar* with the default PrgEnv-pgi 10.2 Fortran compiler using the `-O3` optimization flag, the executables were run using all six available cores per processor.

Execution times reported for the timed Sweep3D kernel for a range of process counts up to 196,608 (192k) processes are shown in Figure 3 with diamonds, showing a progressive slowdown which is not uncommon when weak-scaling applications over such a large range. To understand the execution performance behaviour, the Scalasca toolset was employed using automatic instrumentation of source routines by the compiler. Since the elapsed times reported for the benchmark kernel of the uninstrumented version were within 3% of those when Scalasca measurements were made, instrumentation and measurement dilation were acceptable and refinement was not needed.

From the runtime summary profiles, it was found that the computation time (i.e., execution time excluding time in MPI operations) was 18 seconds, independent of scale, but the MPI communication time in the sweep kernel grew to over 100 seconds. (MPI communication time is not shown in Figure 3 as it is indistinguishable from MPI waiting time on the logarithmic scale.)

Even with all user routines instrumented and events for all MPI operations, scoring of the summary profile analysis report determined that the size of trace buffer required for each process was only 2.75 MB. Since this is less than the Scalasca default value of 10 MB, and the majority of this space was for MPI events, trace collection and analysis required no special configuration of trace buffers or filters. Storing trace event data in a separate file for each process, Scalasca trace analysis proceeds automatically after measurement is complete using the same

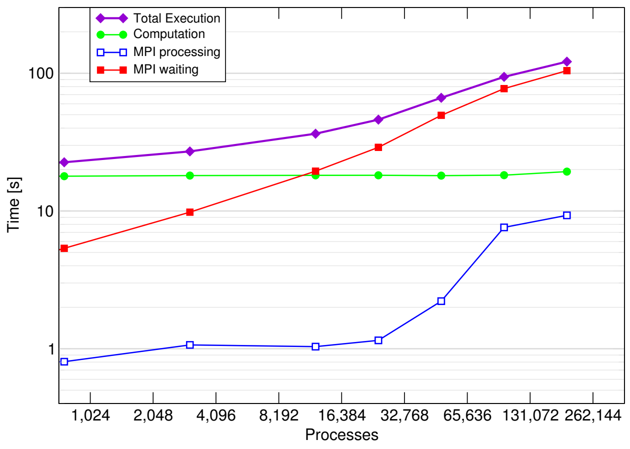


Figure 3: Scaling of Sweep3D execution time on *Jaguar* Cray XT5 with breakdown of computation and message-passing costs from Scalasca summary and trace analyses.

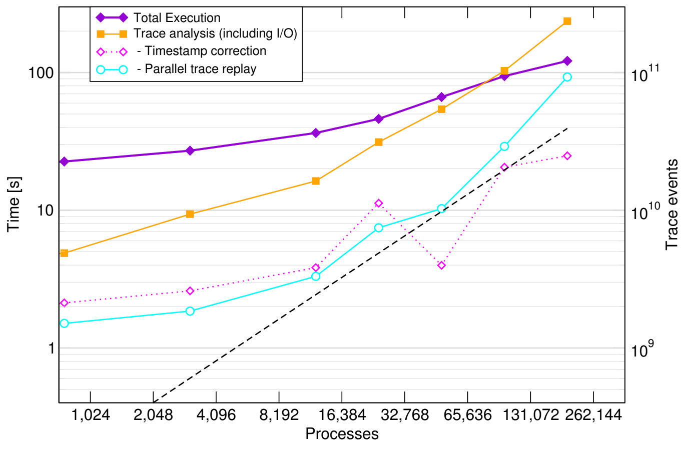
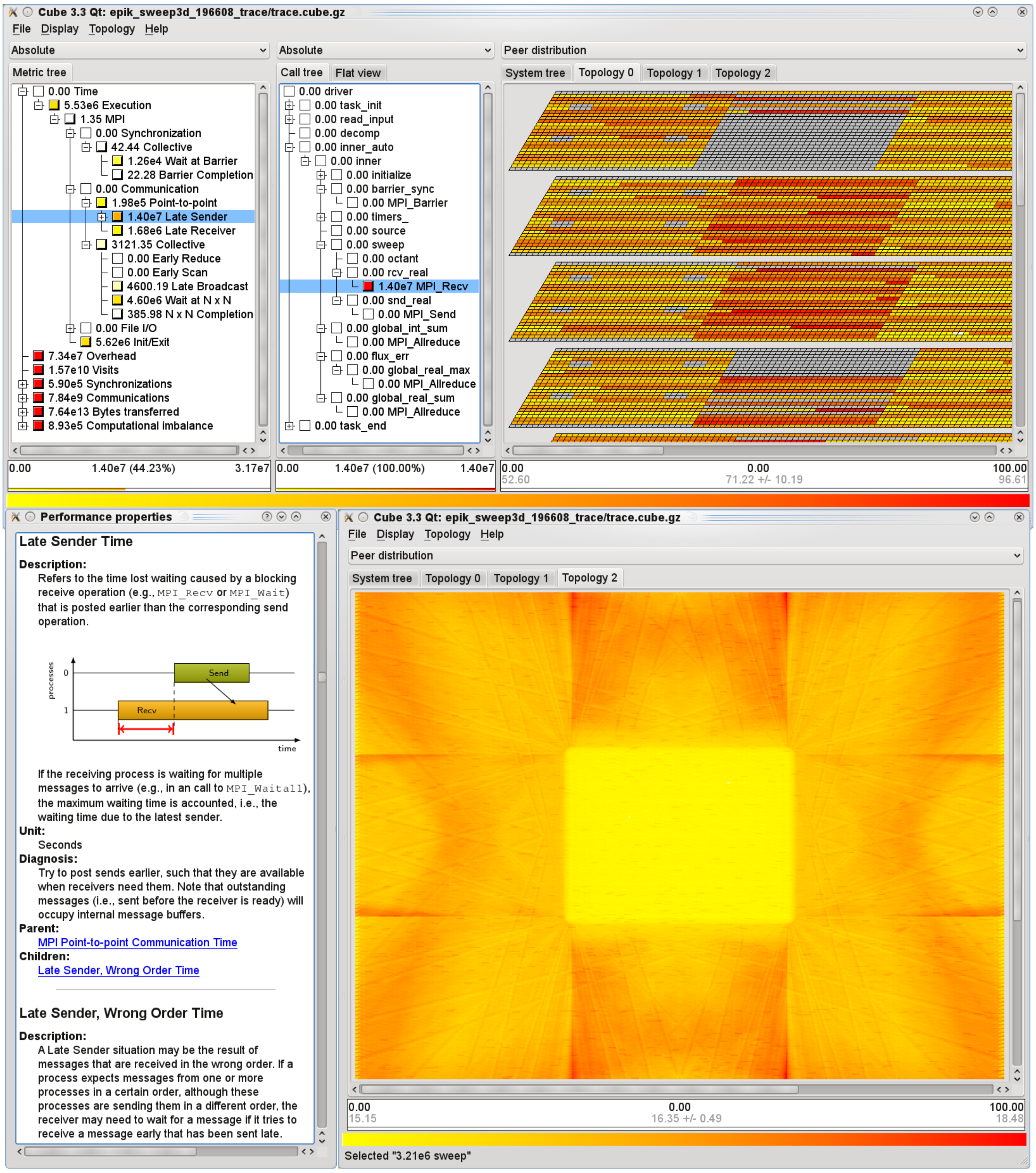


Figure 4: Scaling of Sweep3D trace analysis time on *Jaguar* Cray XT5 with breakdown of times for timestamp synchronization and parallel event replay. (Dashed line is the total number of trace event records.)

configuration of processes to replay the traced events in a scalable fashion. Figure 4 shows that trace analysis times (squares) remain modest, even though total sizes of traces increase linearly with the number of processes to 526 GB for 39e10 traced events (dashed line). The time required for timestamp correction varies considerably according to the number and type of logical consistency violation encountered, such that the trace from 24k processes which had many violations required more processing than that from 48k which was fortunate to have none at all.

Scalasca trace analysis distinguishes various MPI blocking and waiting conditions from basic message processing (as shown in the upper left of Figure 5). While the basic MPI processing time charted in Figure 3 is found to increase from 1 to almost 10 seconds, MPI communication time is dominated by increasingly onerous waiting time (filled squares) that gov-


Cube 3.3 Qt: epik_sweep3d_196608_trace/trace.cube.gz

File Display Topology Help

Absolute | Absolute | Peer distribution

Metric tree | Call tree | Flat view | System tree | Topology 0 | Topology 1 | Topology 2

```
0.00 Time
  5.53e6 Execution
    1.35 MPI
      0.00 Synchronization
        42.44 Collective
          1.26e4 Wait at Barrier
          22.28 Barrier Completion
      0.00 Communication
        1.98e5 Point-to-point
          1.40e7 Late Sender
          1.68e6 Late Receiver
        3121.35 Collective
          0.00 Early Reduce
          0.00 Early Scan
          4600.19 Late Broadcast
          4.60e6 Wait at N x N
          385.98 N x N Completion
      0.00 File I/O
    5.62e6 Init/Exit
  7.34e7 Overhead
  1.57e10 Visits
  5.90e5 Synchronizations
  7.84e9 Communications
  7.64e13 Bytes transferred
  8.93e5 Computational imbalance
```

```
0.00 driver
  0.00 task_init
  0.00 read_input
  0.00 decomp
  0.00 inner_auto
    0.00 inner
      0.00 initialize
      0.00 barrier_sync
        0.00 MPI_Barrier
      0.00 timers_
      0.00 source
      0.00 sweep
        0.00 octant
        0.00 rcv_real
          1.40e7 MPI_Recv
        0.00 snd_real
          0.00 MPI_Send
      0.00 global_int_sum
        0.00 MPI_Allreduce
      0.00 flux_err
        0.00 global_real_max
          0.00 MPI_Allreduce
      0.00 global_real_sum
        0.00 MPI_Allreduce
  0.00 task_end
```

0.00 1.40e7 (44.23%) 3.17e7 | 0.00 1.40e7 (100.00%) 1.40e7 | 0.00 52.60 0.00 71.22 +/- 10.19 100.00 96.61

Performance properties

Late Sender Time

Description:
Refers to the time lost waiting caused by a blocking receive operation (e.g., `MPI_Recv` or `MPI_Wait`) that is posted earlier than the corresponding send operation.

If the receiving process is waiting for multiple messages to arrive (e.g., in an call to `MPI_Waitall`), the maximum waiting time is accounted, i.e., the waiting time due to the latest sender.

Unit:
Seconds

Diagnosis:
Try to post sends earlier, such that they are available when receivers need them. Note that outstanding messages (i.e., sent before the receiver is ready) will occupy internal message buffers.

Parent:
MPI Point-to-point Communication Time

Children:
Late Sender, Wrong Order Time

Late Sender, Wrong Order Time

Description:
A Late Sender situation may be the result of messages that are received in the wrong order. If a process expects messages from one or more processes in a certain order, although these processes are sending them in a different order, the receiver may need to wait for a message if it tries to receive a message early that has been sent late.

Cube 3.3 Qt: epik_sweep3d_196608_trace/trace.cube.gz

File Display Topology Help

Peer distribution

System tree | Topology 0 | Topology 1 | Topology 2

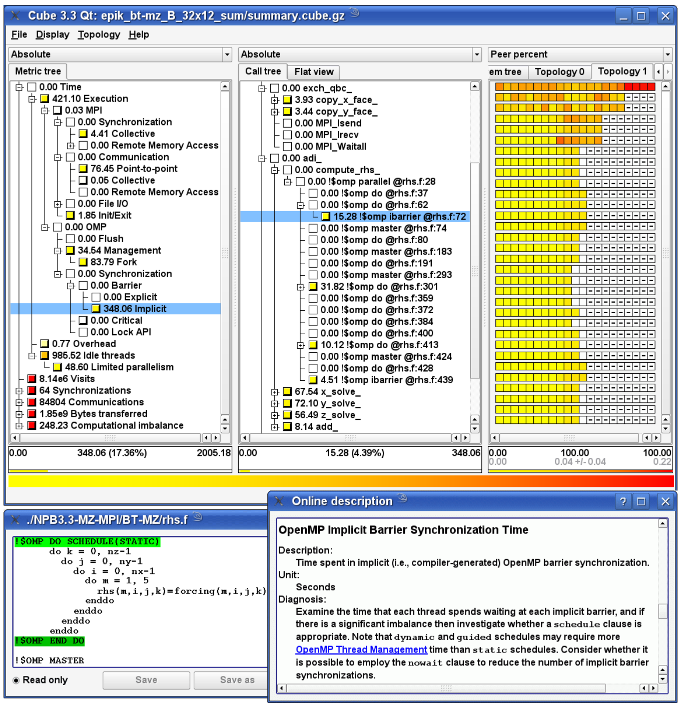
0.00 15.15 0.00 16.35 +/- 0.49 100.00 18.48

Selected "3.21e6 sweep"

Figure 5: Scalasca analysis report explorer presentations of a Sweep3D trace experiment with 196,608 processes on the *Jaguar* Cray XT5 system. On top “Late Sender” time in `MPI_Recv` and its distribution by process shown with the machine physical topology (where unallocated nodes are shown grey), and below it an additional view of the exclusive “Execution” time corresponding to local computation in the `sweep` routine shown with the application’s 512 × 384 virtual topology which is the primary origin of the imbalance.

ern the performance of Sweep3D at larger scales. Most waiting time is found to be "Late Sender" conditions, where a receive is early and must block until its associated send operation is initiated. (Hyperlinked explanatory descriptions and diagnosis tips for metrics are available in a separate performance properties window.) In the case of the 196,608-process experiment shown in Figure 5, "Late Sender" time is 44% of the total execution time. The distribution of "Late Sender" time according to process location in the Cray XT physical topology shows no clear pattern (upper right), though the location of service nodes and unallocated compute nodes can be clearly distinguished. Using the application's two-dimensional virtual grid topology (lower right), however, reveals several pronounced characteristics which can be tracked back to a significant imbalance in the sweep computation when applying flux corrections: foremost is a central rectangular region with less time for local computation and correspondingly higher waiting time, surrounded by an intricate pattern of sharp oblique lines radiating from the central region to the edges. In combination with the wavefront nature of the diagonal sweeps, the otherwise relatively minor computational imbalance (with a range of values that is 22% of the mean, and standard deviation of 3%) amplifies message waiting times at larger scales.

While the same computational imbalance exists at smaller scales (and on other platforms), Scalasca measurement and analysis at large scale proved insightful for its characterization and understanding of its impact on overall execution performance [13]. Furthermore, Scalasca has thereby been able to demonstrate its operation at previously unprecedented scale.



```
0.00 Time
  421.10 Execution
    0.03 MPI
      0.00 Synchronization
        4.41 Collective
        0.00 Remote Memory Access
      0.00 Communication
        76.45 Point-to-point
        0.05 Collective
        0.00 Remote Memory Access
      0.00 File I/O
      1.85 Init/Exit
    0.00 OMP
      0.00 Flush
      34.54 Management
        83.79 Fork
      0.00 Synchronization
        0.00 Barrier
          0.00 Explicit
          348.06 Implicit
        0.00 Critical
        0.00 Lock API
    0.77 Overhead
  985.52 Idle threads
    48.60 Limited parallelism
8.14e6 Visits
64 Synchronizations
84804 Communications
1.85e9 Bytes transferred
248.23 Computational imbalance

0.00          348.06 (17.36%)          2005.18
```

```
0.00 exch_qbc_
  3.93 copy_x_face_
  3.44 copy_y_face_
  0.00 MPI_Isend
  0.00 MPI_Irecv
  0.00 MPI_Waitall
0.00 adi_
  0.00 compute_rhs_
    0.00 !$omp parallel @rhs.f:28
      0.00 !$omp do @rhs.f:37
      0.00 !$omp do @rhs.f:62
        15.28 !$omp ibarrier @rhs.f:72
      0.00 !$omp master @rhs.f:74
      0.00 !$omp do @rhs.f:80
      0.00 !$omp master @rhs.f:183
      0.00 !$omp do @rhs.f:191
      0.00 !$omp master @rhs.f:293
      31.82 !$omp do @rhs.f:301
      0.00 !$omp do @rhs.f:359
      0.00 !$omp do @rhs.f:372
      0.00 !$omp do @rhs.f:384
      0.00 !$omp do @rhs.f:400
      10.12 !$omp do @rhs.f:413
      0.00 !$omp master @rhs.f:424
      0.00 !$omp do @rhs.f:428
      4.51 !$omp ibarrier @rhs.f:439
  67.54 x_solve_
  72.10 y_solve_
  56.49 z_solve_
  8.14 add_

0.00          15.28 (4.39%)          348.06
```

```
0.00          100.00          100.00
0.00       0.04 +/- 0.04        0.22
```

```
!$OMP DO SCHEDULE(STATIC)
      do k = 0, nz-1
        do j = 0, ny-1
          do i = 0, nx-1
            do m = 1, 5
              rhs(m,i,j,k)=forcing(m,i,j,k)
            enddo
          enddo
        enddo
      enddo
!$OMP END DO

!$OMP MASTER
```

OpenMP Implicit Barrier Synchronization Time

Description:
Time spent in implicit (i.e., compiler-generated) OpenMP barrier synchronization.
Unit:
Seconds
Diagnosis:
Examine the time that each thread spends waiting at each implicit barrier, and if there is a significant imbalance then investigate whether a `schedule` clause is appropriate. Note that `dynamic` and `guided` schedules may require more OpenMP Thread Management time than `static` schedules. Consider whether it is possible to employ the `nowait` clause to reduce the number of implicit barrier synchronizations.

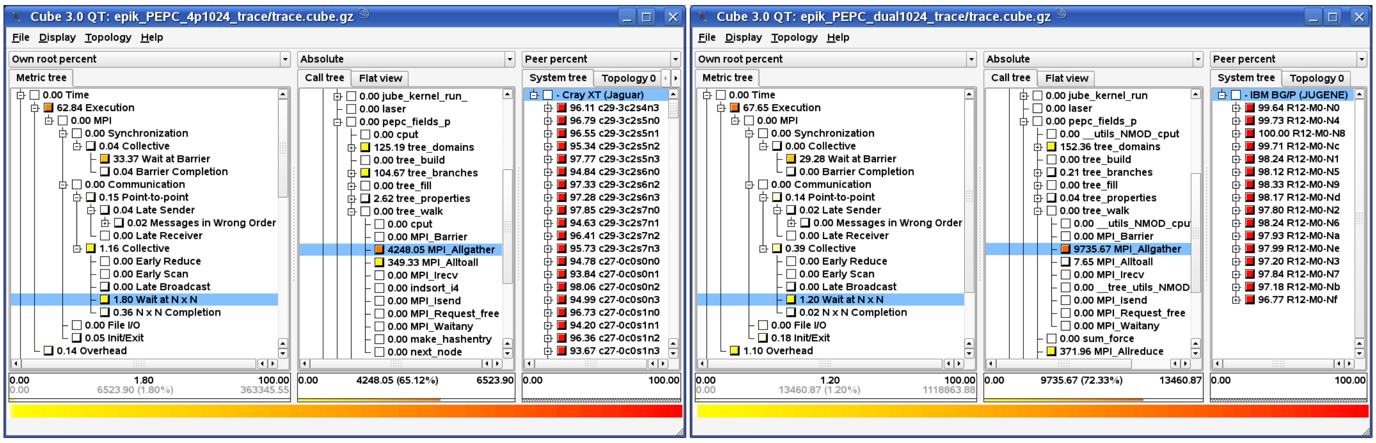
Figure 6: Scalasca summary analysis report explorer display of a hybrid OpenMP/MPI NPB3.3 BT-MZ benchmark Class B execution on 32 Cray XT5 twin six-core compute nodes of *Kraken*, showing OpenMP "Implicit Barrier Synchronization" time for a parallel loop in the `compute_rhs` routine (from lines 62 to 72 of file `rhs.f` in the source viewer) broken down by thread. Void thread locations in the topology pane are displayed in gray or with dashes.

4.3 NPB-MZ-MPI BT

Although message passing is still the predominant programming paradigm used in HPC, increasingly applications leverage OpenMP to exploit more fine-grained process-local parallelism, while communicating between processes using MPI. Scalasca extended runtime summarization and trace analysis support OpenMP/MPI hybrid applications by producing callpath profiles for each thread, calculating OpenMP-specific metrics and presenting these alongside serial and MPI metrics in integrated analysis reports.

While Scalasca trace analysis currently remains restricted to fixed-size teams of OpenMP threads, runtime summarization identifies threads that have not been used during parallel regions. The associated time within the parallel region is distinguished as a "Limited parallelism" metric from the "Idle threads" time which includes time outside OpenMP parallel regions when only the master thread executes. This matches the typical usage of dedicated HPC resources such as Cray XT which are allocated for the duration of the parallel job, or threads which busy-wait occupying compute resources in shared environments. The number of OpenMP threads included in the measurement can be explicitly specified, defaulting to the number of threads for an unqualified parallel region when measurement commences: a warning is provided if subsequent `omp_set_num_threads` calls or `num_threads` clauses result in additional threads not being included in the measurement experiment.

Figure 6 shows a Scalasca summary analysis report from a hybrid OpenMP/MPI NAS NPB3.3 Block Triangular Multi-Zone (BT–MZ) benchmark [10] Class B execution in a *Kraken* Cray XT5 partition consisting of 32 compute nodes, each with two six-core Opteron processors. An instrumented executable was built using PrgEnv-gnu GCC compilers, and measurement done with one MPI process started on each of the compute nodes and OpenMP threads run within each SMP node. In an unsuccessful attempt at load balancing by the application, more than 12 OpenMP threads were created by the first 6 MPI ranks (shown at the top of the topological presentation in the right pane), and 20 of the remaining ranks used fewer than 12 OpenMP threads. While the 49 seconds of "Limited parallelism" time for the unused cores represent only 2% of the allocated compute resources, half of the total time is wasted by "Idle threads" while each process executes serially, including MPI operations done outside of parallel regions by the master thread of each process. Although the exclusive "Execution" time in local computation is relatively well balanced on each OpenMP thread, the over-subscription of the first 6 compute nodes mani-



```
Cube 3.0 QT: epik_PEPC_4p1024_trace/trace.cube.gz
File  Display  Topology  Help
Own root percent                      Absolute                          Peer percent
Metric tree                           Call tree | Flat view              System tree | Topology 0
0.00 Time                              0.00 jube_kernel_run_             - Cray XT (Jaguar)
  62.84 Execution                      0.00 laser                          96.11 c29-3c2s4n3
    0.00 MPI                           0.00 pepc_fields_p                  96.79 c29-3c2s5n0
      0.00 Synchronization               0.00 cputt                        96.55 c29-3c2s5n1
        0.04 Collective                  125.19 tree_domains               95.34 c29-3c2s5n2
          33.37 Wait at Barrier          0.00 tree_build                   97.77 c29-3c2s5n3
          0.04 Barrier Completion        104.67 tree_branches              94.84 c29-3c2s6n0
      0.00 Communication                 0.00 tree_fill                    97.33 c29-3c2s6n2
        0.15 Point-to-point              2.62 tree_properties              97.28 c29-3c2s6n3
          0.04 Late Sender               0.00 tree_walk                    97.85 c29-3c2s7n0
            0.02 Messages in Wrong Order   0.00 cput                       94.63 c29-3c2s7n1
          0.00 Late Receiver               0.00 MPI_Barrier                96.41 c29-3c2s7n2
        1.16 Collective                    4248.05 MPI_Allgather           95.73 c29-3c2s7n3
          0.00 Early Reduce                349.33 MPI_Alltoall             94.78 c27-0c0s0n0
          0.00 Early Scan                  0.00 MPI_Irecv                  93.84 c27-0c0s0n1
          0.00 Late Broadcast              0.00 indsort_i4                 98.06 c27-0c0s0n2
          1.80 Wait at N x N               0.00 MPI_Isend                  94.99 c27-0c0s0n3
          0.36 N x N Completion            0.00 MPI_Request_free           96.73 c27-0c0s1n0
      0.00 File I/O                        0.00 MPI_Waitany                94.20 c27-0c0s1n1
      0.05 Init/Exit                       0.00 make_hashentry             96.36 c27-0c0s1n2
  0.14 Overhead                            0.00 next_node                  93.67 c27-0c0s1n3
0.00          1.80          100.00    0.00   4248.05 (65.12%)   6523.90   0.00          100.00
0.00  6523.90 (1.80%)    363345.55
```

```
Cube 3.0 QT: epik_PEPC_dual1024_trace/trace.cube.gz
File  Display  Topology  Help
Own root percent                      Absolute                          Peer percent
Metric tree                           Call tree | Flat view              System tree | Topology 0
0.00 Time                              0.00 jube_kernel_run              - IBM BG/P (JUGENE)
  67.65 Execution                      0.00 laser                          99.64 R12-M0-N0
    0.00 MPI                           0.00 pepc_fields_p                  99.73 R12-M0-N4
      0.00 Synchronization               0.00 __utils_NMOD_cput            100.00 R12-M0-N8
        0.00 Collective                  152.36 tree_domains               99.71 R12-M0-Nc
          29.28 Wait at Barrier          0.00 tree_build                   98.24 R12-M0-N1
          0.00 Barrier Completion        0.21 tree_branches                98.12 R12-M0-N5
      0.00 Communication                 0.00 tree_fill                    98.33 R12-M0-N9
        0.14 Point-to-point              0.04 tree_properties              98.17 R12-M0-Nd
          0.02 Late Sender               0.00 tree_walk                    97.80 R12-M0-N2
            0.00 Messages in Wrong Order   0.00 __utils_NMOD_cpu           98.24 R12-M0-N6
          0.00 Late Receiver               0.00 MPI_Barrier                97.93 R12-M0-Na
        0.39 Collective                    9735.67 MPI_Allgather           97.99 R12-M0-Ne
          0.00 Early Reduce                7.65 MPI_Alltoall               97.20 R12-M0-N3
          0.00 Early Scan                  0.00 MPI_Irecv                  97.84 R12-M0-N7
          0.00 Late Broadcast              0.00 __tree_utils_NMOD          97.18 R12-M0-Nb
          1.20 Wait at N x N               0.00 MPI_Isend                  96.77 R12-M0-Nf
          0.02 N x N Completion            0.00 MPI_Request_free
      0.00 File I/O                        0.00 MPI_Waitany
      0.18 Init/Exit                     0.00 sum_force
  1.10 Overhead                          371.96 MPI_Allreduce
0.00          1.20          100.00    0.00   9735.67 (72.33%)   13460.87   0.00          100.00
0.00  13460.87 (1.20%)   1118863.88
```

Figure 7: Scalasca analysis report explorer presentations of PEPC trace experiments for 1,024 processes on the quad-core *Jaguar* Cray XT4 (left) and *Jugene* IBM Blue Gene/P (right), comparing the percentage of time due to "Wait at N x N" waiting for the last rank to enter `MPI_Allgather` collective communication for the tree walk used updating fields. (The system tree for Cray XT shows individual compute nodes, whereas Blue Gene/P nodeboards consist of 32 quad-core processors).

fests as excessive "Implicit Barrier Synchronization" time at the end of parallel regions (as well as additional OpenMP "Thread Management" overhead), and higher "MPI Point-to-point Communication" time on the other processes is then a consequence of this. When over-subscription of cores is avoided, benchmark execution time is reduced by one third (with "MPI" time reduced by 52%, "OMP" time reduced by 20% and time for "Idle threads" reduced by 55%).

As hybrid OpenMP/MPI applications become more prevalent, it can be expected that performance analysis tools that integrate and correlate performance characteristics of both programming models will become essential in optimizing execution performance. Current Scalasca limitations and inefficiencies (particularly with respect to handling of trace files) for such applications will also need to be addressed accordingly.

4.4 PEPC

PEPC [6] is a three-dimensional particle simulation code developed by Jülich Supercomputing Centre employing a parallel tree-code for rapid computation of long-range Coulomb forces for large ensembles of charged particles that is used for various applications in plasma physics and astrophysics. As part of DEISA and PRACE benchmarking activities, a benchmark version of PEPC was run and its performance analysed on Cray XT, IBM Blue Gene/P and other HPC systems.

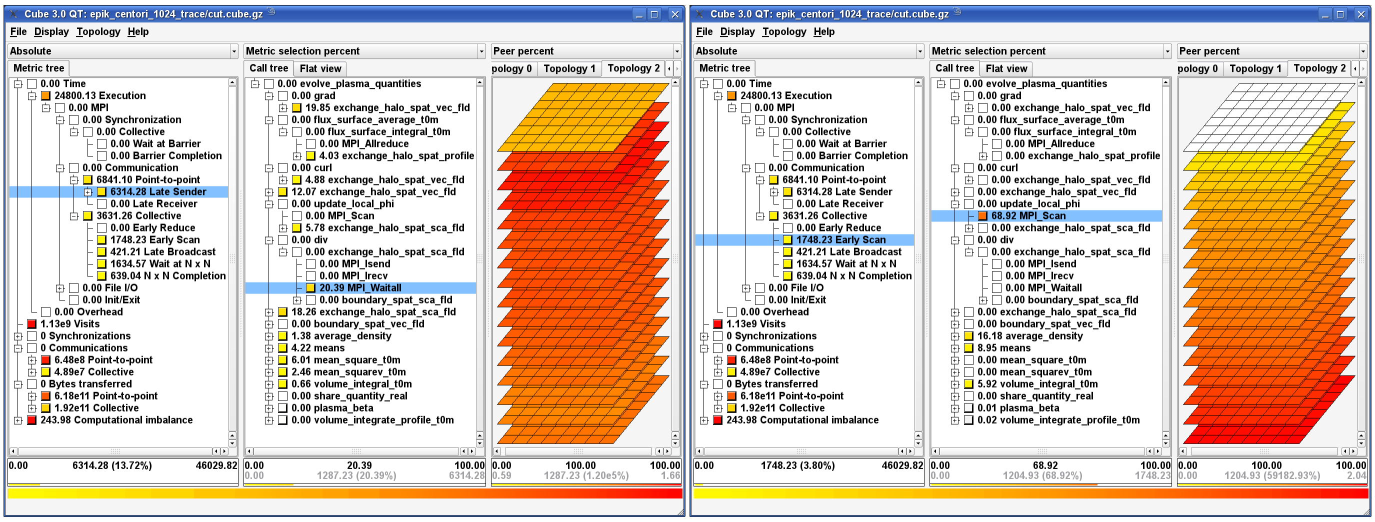
Although the IBM XL compilers and MPI library on BG/P are quite different from the PGI compilers and CNL used on the Cray XT4, the procedure for building an instrumented executable and running it under the control of the Scalasca measurement and analysis nexus are identical. Figure 7 compares trace experiments from both systems with 1,024 MPI processes, showing that roughly two-thirds of the time is spent in local computation in each, although the Cray XT execution is three times faster. Naturally, 2.3 GHz quad-core Opteron processors out-perform the quad-core PowerPC processors clocked at 850 MHz on computation, however, the BG/P torus/tree interconnect generally shows advantages in communication. This version of PEPC uses `MPI_Barrier` extensively, resulting in roughly one-third of time in "Wait at Barrier" for both. Most of the collective communication time is also time waiting for last ranks to enter `MPI_Allgather` and `MPI_Alltoall`, however, the imbalance for `MPI_Allgather` is twice as severe on BG/P, however, almost 50 times less for `MPI_Alltoall`. "N x N Completion" times for collective communication are also notably longer on Cray XT.

While careful examination of the call-trees for both executions will identify that the IBM and PGI compilers performed inlining and Fortran module name decoration differently, the generic representation and implementation of Scalasca analysis reports facilitate performance comparisons between architectures, compilers, optimization levels and algorithms.

4.5 CENTORI

The UKAEA CENTORI Fortran/MPI diffusive plasma simulation code [9] for plasma transport and turbulence studies of fusion reactors was analysed on the *HECToR* Cray XT4. After automatic instrumentation, a series of experiments were collected running a short simulation with 1,024 MPI processes. From an initial summary measurement, routines that do only purely local computation were identified, and these were then specified in a filter file to be excluded from subsequent measurements. Using the filter not only reduced measurement overhead, but also trace buffer requirements shrank from 1.4 GB to 41 MB per process. A trace measurement of CENTORI running for 55 seconds additionally required approximately 12 seconds to write the 40 GB trace, followed by 35 seconds to analyse it (using the same processors) and produce the trace analysis report.

Figure 8 shows views with the Scalasca analysis report ex-



```
0.00 Time
  24800.13 Execution
    0.00 MPI
      0.00 Synchronization
        0.00 Collective
          0.00 Wait at Barrier
          0.00 Barrier Completion
      0.00 Communication
        6841.10 Point-to-point
          6314.28 Late Sender
          0.00 Late Receiver
        3631.26 Collective
          0.00 Early Reduce
          1748.23 Early Scan
          421.21 Late Broadcast
          1634.57 Wait at N x N
          639.04 N x N Completion
      0.00 File I/O
      0.00 Init/Exit
    0.00 Overhead
1.13e9 Visits
0 Synchronizations
0 Communications
  6.48e8 Point-to-point
  4.89e7 Collective
0 Bytes transferred
  6.18e11 Point-to-point
  1.92e11 Collective
243.98 Computational imbalance

0.00        6314.28 (13.72%)        46029.82
```

```
0.00 evolve_plasma_quantities
  0.00 grad
    19.85 exchange_halo_spat_vec_fld
  0.00 flux_surface_average_t0m
    0.00 flux_surface_integral_t0m
      0.00 MPI_Allreduce
      4.03 exchange_halo_spat_profile
  0.00 curl
    4.88 exchange_halo_spat_vec_fld
  12.07 exchange_halo_spat_vec_fld
  0.00 update_local_phi
    0.00 MPI_Scan
    5.78 exchange_halo_spat_sca_fld
  0.00 div
    0.00 exchange_halo_spat_sca_fld
      0.00 MPI_Isend
      0.00 MPI_Irecv
      20.39 MPI_Waitall
      0.00 boundary_spat_sca_fld
  18.26 exchange_halo_spat_sca_fld
  0.00 boundary_spat_vec_fld
  1.38 average_density
  4.22 means
  6.01 mean_square_t0m
  2.46 mean_squarev_t0m
  0.66 volume_integral_t0m
  0.00 share_quantity_real
  0.00 plasma_beta
  0.00 volume_integrate_profile_t0m

0.00              20.39              100.00
0.00      1287.23 (20.39%)          6314.28
```

```
0.00              100.00             100.00
0.59      1287.23 (1.20e5%)            1.66
```

```
0.00 Time
  24800.13 Execution
    0.00 MPI
      0.00 Synchronization
        0.00 Collective
          0.00 Wait at Barrier
          0.00 Barrier Completion
      0.00 Communication
        6841.10 Point-to-point
          6314.28 Late Sender
          0.00 Late Receiver
        3631.26 Collective
          0.00 Early Reduce
          1748.23 Early Scan
          421.21 Late Broadcast
          1634.57 Wait at N x N
          639.04 N x N Completion
      0.00 File I/O
      0.00 Init/Exit
    0.00 Overhead
1.13e9 Visits
0 Synchronizations
0 Communications
  6.48e8 Point-to-point
  4.89e7 Collective
0 Bytes transferred
  6.18e11 Point-to-point
  1.92e11 Collective
243.98 Computational imbalance

0.00        1748.23 (3.80%)        46029.82
```

```
0.00 evolve_plasma_quantities
  0.00 grad
    0.00 exchange_halo_spat_vec_fld
  0.00 flux_surface_average_t0m
    0.00 flux_surface_integral_t0m
      0.00 MPI_Allreduce
      0.00 exchange_halo_spat_profile
  0.00 curl
    0.00 exchange_halo_spat_vec_fld
  0.00 exchange_halo_spat_vec_fld
  0.00 update_local_phi
    68.92 MPI_Scan
    0.00 exchange_halo_spat_sca_fld
  0.00 div
    0.00 exchange_halo_spat_sca_fld
      0.00 MPI_Isend
      0.00 MPI_Irecv
      0.00 MPI_Waitall
      0.00 boundary_spat_sca_fld
  0.00 exchange_halo_spat_sca_fld
  0.00 boundary_spat_vec_fld
  16.18 average_density
  8.95 means
  0.00 mean_square_t0m
  0.00 mean_squarev_t0m
  5.92 volume_integral_t0m
  0.00 share_quantity_real
  0.01 plasma_beta
  0.02 volume_integrate_profile_t0m

0.00              68.92              100.00
0.00      1204.93 (68.92%)          1748.23
```

```
0.00              100.00                100.00
0.00      1204.93 (59182.93%)             2.04
```

Figure 8: Scalasca analysis report explorer presentations of CENTORI trace experiments with 1,024 processes on the *HECToR* Cray XT4 system, showing the most severe performance inefficiencies of the plasma evolution simulation phase. On the left is the "Late Sender" time in `MPI_Waitall` of halo exchange, and on the right is the "Early Scan" time of `MPI_Scan` operations in `update_local_phi`.

plorer GUI, where the plasma evolution phase has been extracted to exclude the less interesting initialization and finalization phases. 54% of the total execution time is in purely local computation, with the remainder MPI communication time. Point-to-point communication constitutes the majority of this, and almost half is identified as being due to "Late Sender" situations where receivers were blocked waiting for sends to be initiated. The left view shows one-fifth of the "Late Sender" time in `exchange_halo_spat_sca_fld` called from the `div` routine. Using the application's $16 \times 8 \times 8$ logical topology reveals the distribution of metric values by process, where x-planes have similar values but there are significant differences for each x-plane (here stacked vertically). These are explained by the absence of wraparound boundary conditions in the x-dimension and correspondingly fewer communication operations for those faces. This planar performance variation is also evident in collective operations, such as the `MPI_Scan` in the `update_local_phi` routine of the right view, where the "Early Scan" time metric shows that processes with higher ranks wait longer for values from lower ranks. From these and other insights provided by the Scalasca analyses, code performance could be improved as part of the performance engineering collaboration between EPCC and the application developers.

4.6 GemsFDTD

The GemsFDTD computational electromagnetics code from KTH-PSCI [8] solves the Maxwell equations using the finite-difference time-domain method. A subset of this code computing the radar cross-section of a perfectly conducting object that was included in the SPEC MPI2007 benchmark suite (v1.1) was found to perform particularly poorly at larger scales on distributed-memory computer systems, and investigation with the Scalasca toolset pinpointed aspects in the application's initialization phase that limited scalability to only 128 processes and made execution prohibitive with larger numbers. This insight led the application developers to rework the initialization and further optimize the time-stepping loop, to realize substantial execution performance and overall scalability improvements, ultimately resulting in an entirely updated version of the benchmark (v2.0) which could be run with 2,048 processes and scaled well to 512 processes.

Scalasca summary experiments with varying numbers of MPI processes on the *HECToR* Cray XT4 with both versions of GemsFDTD and the 'ltrain' dataset were collected, with the analysis reports providing a breakdown of the total run time into

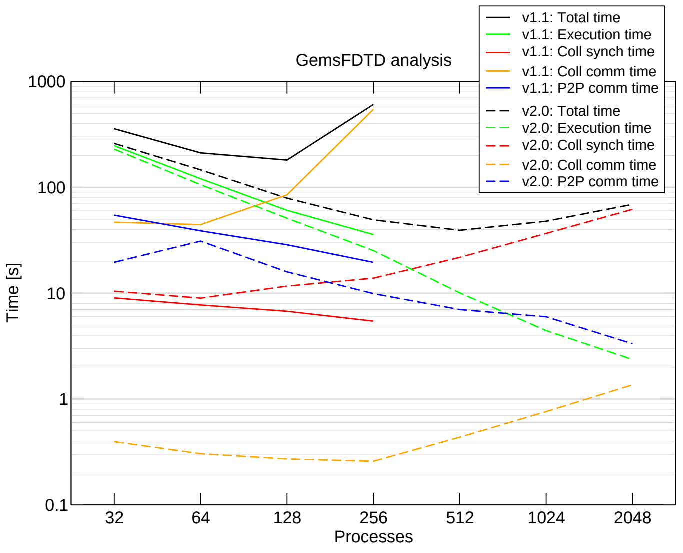
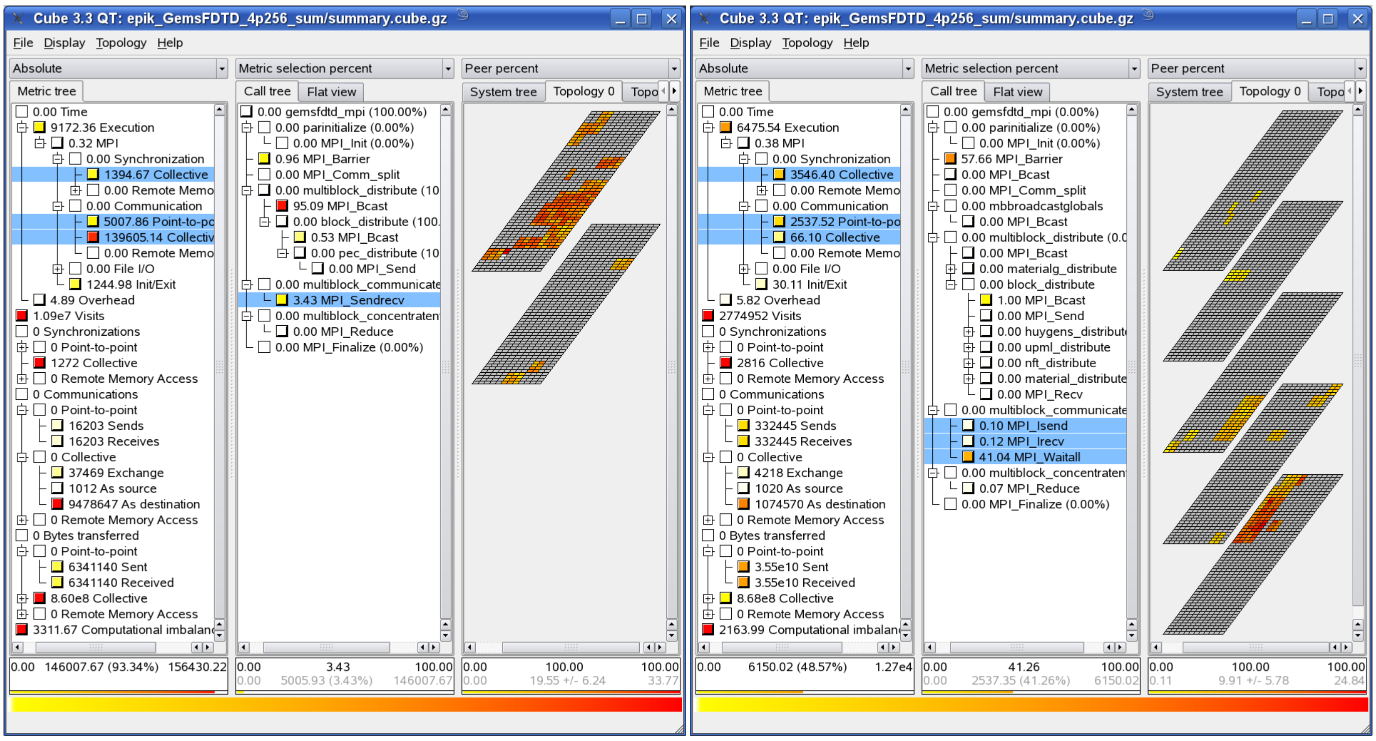


Figure 9: Scalasca summary analysis breakdown of executions of GemsFDTD versions on the *HECToR* Cray XT4.



```
Cube 3.3 QT: epik_GemsFDTD_4p256_sum/summary.cube.gz
File  Display  Topology  Help
Absolute | Metric selection percent | Peer percent
Metric tree | Call tree  Flat view | System tree  Topology 0  Topo

0.00 Time                              0.00 gemsfdtd_mpi (100.00%)
 9172.36 Execution                      0.00 parinitialize (0.00%)
  0.32 MPI                                0.00 MPI_Init (0.00%)
   0.00 Synchronization                 0.96 MPI_Barrier
    1394.67 Collective                  0.00 MPI_Comm_split
    0.00 Remote Memo                    0.00 multiblock_distribute (10
   0.00 Communication                     95.09 MPI_Bcast
    5007.86 Point-to-pc                   0.00 block_distribute (100.
    139605.14 Collectiv                     0.53 MPI_Bcast
    0.00 Remote Memo                      0.00 pec_distribute (10
   0.00 File I/O                            0.00 MPI_Send
   1244.98 Init/Exit                    0.00 multiblock_communicate
 4.89 Overhead                            3.43 MPI_Sendrecv
1.09e7 Visits                           0.00 multiblock_concentraten
0 Synchronizations                        0.00 MPI_Reduce
 0 Point-to-point                       0.00 MPI_Finalize (0.00%)
 1272 Collective
 0 Remote Memory Access
0 Communications
 0 Point-to-point
  16203 Sends
  16203 Receives
 0 Collective
  37469 Exchange
  1012 As source
  9478647 As destination
 0 Remote Memory Access
0 Bytes transferred
 0 Point-to-point
  6341140 Sent
  6341140 Received
 8.60e8 Collective
 0 Remote Memory Access
3311.67 Computational imbalanc

0.00  146007.67 (93.34%)  156430.22 | 0.00  3.43  100.00 | 0.00  100.00  100.00
                                    | 0.00  5005.93 (3.43%)  146007.67 | 0.00  19.55 +/- 6.24  33.77
```

```
Cube 3.3 QT: epik_GemsFDTD_4p256_sum/summary.cube.gz
File  Display  Topology  Help
Absolute | Metric selection percent | Peer percent
Metric tree | Call tree  Flat view | System tree  Topology 0  Topo

0.00 Time                              0.00 gemsfdtd_mpi (0.00%)
 6475.54 Execution                      0.00 parinitialize (0.00%)
  0.38 MPI                                0.00 MPI_Init (0.00%)
   0.00 Synchronization                 57.66 MPI_Barrier
    3546.40 Collective                  0.00 MPI_Bcast
    0.00 Remote Memo                    0.00 MPI_Comm_split
   0.00 Communication                   0.00 mbbroadcastglobals
    2537.52 Point-to-pc                   0.00 MPI_Bcast
    66.10 Collective                    0.00 multiblock_distribute (0.0
    0.00 Remote Memo                      0.00 MPI_Bcast
   0.00 File I/O                          0.00 materialg_distribute
   30.11 Init/Exit                        0.00 block_distribute
 5.82 Overhead                              1.00 MPI_Bcast
2774952 Visits                              0.00 MPI_Send
0 Synchronizations                          0.00 huygens_distribute
 0 Point-to-point                           0.00 upml_distribute
 2816 Collective                            0.00 nft_distribute
 0 Remote Memory Access                     0.00 material_distribute
0 Communications                            0.00 MPI_Recv
 0 Point-to-point                       0.00 multiblock_communicate
  332445 Sends                            0.10 MPI_Isend
  332445 Receives                         0.12 MPI_Irecv
 0 Collective                             41.04 MPI_Waitall
  4218 Exchange                         0.00 multiblock_concentraten
  1020 As source                          0.07 MPI_Reduce
  1074570 As destination                0.00 MPI_Finalize (0.00%)
 0 Remote Memory Access
0 Bytes transferred
 0 Point-to-point
  3.55e10 Sent
  3.55e10 Received
 8.68e8 Collective
 0 Remote Memory Access
2163.99 Computational imbalanc

0.00  6150.02 (48.57%)  1.27e4 | 0.00  41.26  100.00 | 0.00  100.00  100.00
                               | 0.00  2537.35 (41.26%)  6150.02 | 0.11  9.91 +/- 5.78  24.84
```

Figure 10: Scalasca analysis report explorer presentations of GemsFDTD summary experiments for 256 processes on *HECToR* Cray XT4 (v1.1 on left and v2.0 on right). MPI communication and synchronization times are selected from the metric trees in the leftmost panes, and MPI communication operations inside the `multiblock_communicate` routine from the central call-tree panes, with the distribution of times per process shown in the right panes with the Cray XT machine topology (which was different for the two executions).

pure (local) computation time and MPI time, and the latter split into time for collective synchronization (i.e., barriers), collective communication and point-to-point communication, as detailed in Figure 9. Both code versions show good scaling of the computation time, with the new version demonstrating better absolute performance (at any particular scale) and better scalability overall.

The extremely poor scalability of the original code version is due to the dramatically increasing time for collective communication, isolated to the numerous `MPI_Bcast` calls during initialization. Collective communication in the revised version is seen growing significantly at the largest scale, however, the primary scalability impediment with it is the increasing time for explicit barrier synchronization (which is not a factor in the original version). Point-to-point communication time is notably reduced in the revised version, but also scales less well than the local computation, which it exceeds for 1,024 or more processes.

A closer comparison of the analysis reports from GemsFDTD v1.1 and v2.0 experiments with 256 processes is possible by studying Figure 10 which is configured to hide non-MPI callpaths. Pure computation time (shown as “Execution”) is improved by more than 40%, however, there is an even more dramatic reduction in MPI time. While MPI collective synchronization time increased by a factor of 2.5, collective communication which dominates the original version is almost entirely eliminated. In particular, MPI point-to-point communication time is halved to an average of 9.91 seconds from 19.55 seconds by substituting non-blocking operations for the `MPI_Sendrecv` originally in the `multiblock_communicate` routine. Notably, improved performance was achieved by increasing the numbers of point-to-point communications and bytes transferred.

The comprehensive analyses provided by the Scalasca toolset were instrumental in directing the developer’s performance engineering of a much more efficient and highly scalable GemsFDTD code. Since these analyses show that there are still significant optimization opportunities at larger scales, further engineering improvements may be investigated in future.

5 Conclusions

A wide variety of applications have been analysed with the Scalasca toolset on a number of Cray XT and other HPC platforms, from which a selection have been reviewed in this paper. Performance analysts and application developers have thereby benefited from new insights into performance problems and have been able to exploit optimization opportunities to improve their codes.

While the Scalasca performance analysis toolset was designed for scalability from the outset, it has required regular redesign and re-engineering of components where scalability impediments were identified as system and application scales and complexities have grown. Cray XT systems have been no exception, but have been able to benefit from improvements originally targeted at other platforms. Specific incorporation of support for the rich set of Cray programming environments, including the OpenMP/MPI hybrid paradigm, ensures that Scalasca remains a versatile and portable complement to Cray's proprietary performance analysis tools.

Acknowledgements

This work would not have been possible without the assistance of numerous application developers, performance analysts and support staff, from which Luiz de Rose, Jason Beech-Brandt & Heidi Poxon (Cray Inc.), Ulf Andersson (KTH-PDC), Lukas Arnold & Paul Gibbon (JSC), Tom Edwards, Xu Guo & Joachim Hein (EPCC), Darren Kerbyson (LANL/PNNL), Mahin Mahmoodi (PSC), Jean-Guillaume Piccinali (CSCS), Sebastian van Alfthan (CSC) and Pat Worley (ORNL) warrant particular mention.

Part of this work has been performed under the HPC-Europa2 project (number 228398) with the support of the European Commission – Capacities Area – Research Infrastructures. EPCC staff provided generous hospitality and valuable assistance to use their Cray XT4 system (*HECToR*) in Edinburgh, Scotland. Access to the Cray XT systems at CSC in Espoo, Finland (*Louhi*) and CSCS in Manno, Switzerland (*Rosa*) was provided through the PRACE project of the European Union. This research was supported in part by the US National Science Foundation through TeraGrid resources (*Kraken*) provided by the National Institute for Computational Sciences under grant number TG-ASC080038N. This research also used resources (*Jaguar*) of the National Center for Computational Sciences at Oak Ridge National Laboratory, which is supported by the Office of Science of the US Department of Energy under contract DE-AC05-00OR22725.

References

[1] Accelerated Strategic Computing Initiative. The ASC SMG2000 benchmark code. `http://www.llnl.gov/asc/purple/benchmarks/limited/smg/`, 2001.

[2] D. Becker, R. Rabenseifner, and F. Wolf. Implications of non-constant clock drifts for the timestamps of concurrent events. In *Proc. IEEE Cluster Conference (Cluster 2008, Tsukuba, Japan)*, pages 59–68. IEEE Computer Society, September 2008.

[3] W. Frings, F. Wolf, and V. Petkov. Scalable massively parallel I/O to task-local files. In *Proc. 21st ACM/IEEE SC Conference (SC09, Portland, OR, USA)*. ACM, November 2009.

[4] M. Geimer, S. Shende, A. Malony, and F. Wolf. A generic and configurable source-code instrumentation component. In *Proc. Int'l Conf. on Computational Science (ICCS, Baton Rouge, LA, USA)*, volume 5545 of *Lecture Notes in Computer Science*, pages 696–705. Springer, May 2009.

[5] M. Geimer, F. Wolf, B. J. N. Wylie, E. Ábrahám, D. Becker, and B. Mohr. The Scalasca performance toolset architecture. *Concurrency and Computation: Practice and Experience*, 22(6):702–719, Apr. 2010.

[6] Jülich Supercomputing Centre. PEPC: A multi-purpose parallel tree-code. `http://www.fz-juelich.de/jsc/pepc/`, 2005.

[7] Los Alamos National Laboratory. ASCI SWEEP3D v2.2b: 3-dimensional discrete ordinates neutron transport benchmark. `http://wwwc3.lanl.gov/pal/software/sweep3d/`, 1995.

[8] Parallel & Scientific Computing Institute (PSCI), Sweden. GEMS: General ElectroMagnetic Solvers project, 2005. `http://www.psci.kth.se/Programs/GEMS/`.

[9] A. Thyagaraja. Numerical simulations of tokamak plasma turbulence and internal transport barriers. *Plasma Physics and Controlled Fusion*, 42:B255–B269, 2000.

[10] R. F. Van der Wijngaart and H. Jin. NAS Parallel Benchmarks, Multi-Zone versions. Technical Report NAS-03-010, NASA Ames Research Center, Moffett Field, CA, USA, July 2003. `http://www.nas.nasa.gov/Software/NPB/`.

[11] F. Wolf, B. J. N. Wylie, E. Ábrahám, D. Becker, W. Frings, K. Fürlinger, M. Geimer, M.-A. Hermanns, B. Mohr, S. Moore, M. Pfeifer, and Z. Szebenyi. Usage of the Scalasca toolset for scalable performance analysis of large-scale parallel applications. In *Proc. 2nd HLRS Parallel Tools Workshop (Stuttgart, Germany)*, pages 157–167. Springer, July 2008.

[12] B. J. N. Wylie. Improved Scalasca toolset support for performance analysis of Cray XT systems. In *Science and Supercomputing in Europe Report 2009*. HPC-Europa2 Transnational Access, CINECA, Casalecchio di Reno (Bologna), Italy. To appear.

[13] B. J. N. Wylie, D. Böhme, B. Mohr, Z. Szebenyi, and F. Wolf. Performance analysis of Sweep3D on Blue Gene/P with the Scalasca toolset. In *Proc. 24th Int'l Parallel & Distributed Processing Symposium, Workshop on Large-Scale Parallel Processing (IPDPS–LSPP, Atlanta, GA, USA)*. IEEE Computer Society, April 2010.

[14] B. J. N. Wylie, M. Geimer, and F. Wolf. Performance measurement and analysis of large-scale parallel applications on leadership computing systems. *Scientific Programming*, 16(2–3):167–181, 2008.

About the authors

Brian J. N. Wylie is a research scientist in the team developing and supporting the Scalasca toolset at Jülich Supercomputing Centre of Forschungszentrum Jülich, Germany. He has experience developing sophisticated parallel performance measurement and analysis tools in both commercial and academic contexts, as well as exploiting them in numerous application performance and scalability engineering teams. E-mail: `b.wylie@fz-juelich.de`.

Scalasca is developed in a collaboration led by Felix Wolf between Forschungszentrum Jülich and the German Research School for Simulation Sciences. Software and documentation can be found at `www.scalasca.org` and the development team can be reached at `scalasca@fz-juelich.de`.