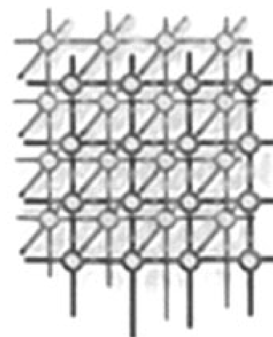
CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2010; **22**:2212–2229
Published online 20 April 2010 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/cpe.1585



Performance measurement and analysis tools for extremely scalable systems

B. Mohr[1,*,†], B. J. N. Wylie[1] and F. Wolf[1,2,3]

[1]*Jülich Supercomputing Centre, Institute for Advanced Simulation, Forschungszentrum Jülich, 52425 Jülich, Germany*
[2]*German Research School for Simulation Sciences, Aachen, Germany*
[3]*Computer Science Department, RWTH Aachen University, Germany*

SUMMARY

High-performance computing systems continue to employ more and more processor cores. Current typical high-end machines in industry, university, and government research laboratory computing centers feature thousands of computing cores. While these machines promise ever more compute power and memory capacity to tackle today's complex simulation problems, they force application developers to greatly enhance the scalability of their codes to be able to exploit it. To better support them in their porting and tuning process, many parallel-tools research groups have already started to work on scaling their methods, techniques, and tools to extreme processor counts. In this paper, we survey existing profiling and tracing tools, report on our experience in using them in extreme scaling environments, review working and promising new methods and techniques, and discuss strategies for solving open issues and problems.
Copyright © 2010 John Wiley & Sons, Ltd.

Received 22 December 2009; Revised 25 January 2010; Accepted 15 February 2010

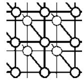
KEY WORDS: performance analysis; parallel programming; scalability

INTRODUCTION AND MOTIVATION

The number of processor cores available in high-performance computing systems is steadily increasing, and a major factor is the current trend to use multi-core and many-core processor chip architectures as compute nodes in large configurations. Since June 2009, the *Jugene* IBM Blue Gene/P in Jülich Supercomputing Centre comprising 72 racks each with 1024 quad-core PowerPC processors—294 912 (288 k) cores in total—has been efficiently running a diverse workload of

*Correspondence to: B. Mohr, Jülich Supercomputing Centre, Institute for Advanced Simulation, Forschungszentrum Jülich, 52425 Jülich, Germany.
†E-mail: b.mohr@fz-juelich.de

Copyright © 2010 John Wiley & Sons, Ltd.



highly parallel applications [1]. While uniquely providing more than 256 k cores, an additional 4 systems from Cray and IBM offer more than 128 k cores, and the average number of cores per system in the November 2009 Top500 list is over 9300 [2].

Ever-increasing compute power and memory capacity available to tackle today's complex simulation problems forces application developers to greatly enhance the scalability of their codes to be able to exploit it. This often requires new algorithms, methods or parallelization schemes as many well-known and accepted techniques stop working at such large scales. It starts with simple things such as opening a file per process to save checkpoint information, or collecting simulation results of the whole program via a gather operation on a single process, or previously unimportant order $O(n^2)$-type operations that now quickly dominate the execution. Unfortunately, many of these performance problems only show up when executing with very high numbers of processes and often cannot be easily diagnosed or predicted from measurements at lower scales. Detecting and diagnosing these performance and scalability bottlenecks requires sophisticated performance instrumentation, measurement, and analysis tools. Simple tools typically scale very well but the information they provide proves to be less and less useful at higher scales. Clearly, understanding performance and correctness problems of applications requires running, analyzing, and gaining insight into these issues at the largest scale.

Consequently, a strategy for software development tools for extreme-scale systems must address a number of dimensions. First, the strategy must include elements that directly address extremely large task and thread counts. Such a strategy is likely to use mechanisms that reduce the number of tasks or threads that must be monitored. Second, less clear but equally daunting, is the fact that several current systems are composed of heterogeneous computing devices and more are being planned. Performance and correctness tools for these systems are currently very immature. Third, the strategy requires a scalable and modular infrastructure that allows rapid creation of new tools that respond to the unique needs that may arise as extreme-scale systems evolve. Furthermore, a successful tools strategy must enable productive use of systems that are by definition unique. Thus, it must provide the full range of traditional software development tools including debuggers and other code correctness tools such as memory analyzers, performance analysis tools as well as build environments for complex codes that rely on a diverse and rapidly changing set of support libraries.

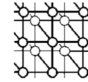
Many parallel-tools research groups have been working for several years on scaling their methods, techniques, and tools to extreme processor counts. In 2007 we investigated performance measurement and analysis tools for large-scale systems, and evaluated those installed on the predecessor capability computing system at JSC—the *Jubl* IBM Blue Gene/L with 16 384 cores. In this paper, we survey the profiling and tracing tools, and report on their use in extreme scaling environments, including our own experience with the JSC Blue Gene/L and Blue Gene/P systems. We also review established and emerging methods and techniques, and discuss strategies for solving open issues and problems.

SCALABLE PROFILING APPROACHES

Profiling tools aggregate metrics collected during the execution of a program. By summarizing the events influencing the performance of a program instead of recording every event instance separately (as done when tracing), the amount of data collected remains constant—independent

Copyright © 2010 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe



of the runtime of the observed program. Therefore, profiling tools can also be used for very long running programs working on realistic input data sets and models. The amount of data can also be controlled by selecting for what kind of program objects data is collected, e.g. for which functions, function call-sites, basic blocks, or statements. Similarly, for parallel programs, data can be collected for each thread or process or can be aggregated over the nodes or even the whole machine. So in the context of extremely scalable machines, profiling has the advantage that by choosing the right aggregation levels, profiling can be made to work for very large programs on many processors. However, it is clear that the more measurements are aggregated, the higher the risk that important details of performance problems to be identified are obscured.

In the following, we give an overview of profiling tools and frameworks which were known to work on at least 8000 processors and describe the tradeoff decisions governing their design.

MPI-only profiling libraries

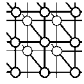
MPI is the predominant parallel programming model used in scientific computing which is demonstrated to work on the largest computer systems available. With PMPI, there exists a standardized, and therefore portable, MPI monitoring interface. The simplest performance tool is a collection of wrapper routines that collect profile data for each MPI call. At the end of the program execution, e.g. during `MPI_Finalize`, the collected data is potentially aggregated across all processes using MPI communication and then written to disk. Data on user defined routines is ignored in this case, which has the big advantage that for the measurement the user application only has to be re-linked against the PMPI wrapper library.

FPMPI-2 is a portable, open-source, very light-weight, and scalable MPI profiling library from Argonne National Laboratory [3]. For each MPI function, it provides the average and the maximum of the sum of metrics over all processes in a single textual output file. The provided metrics are the number of calls and the total execution time of each MPI function. FPMPI-2 has two special features that set it apart from other MPI profiling libraries. First, for communication functions it also records not only the amount of data transferred but also the distribution of the message sizes (by using an adaptive 32 bins histogram). Second, it optionally tries to determine the actual synchronization time within blocking MPI calls by replacing the actual MPI implementation with a logically equivalent implementation using busy-wait on the user level, allowing the blocking time to be estimated. The largest experiment known to us so far is a 16 384-process run. Figure 1 shows an extract of the output report of a 8192-process run on the *Jubl* Blue Gene/L system analyzing the ASC SMG2000 benchmark [4].

mpiP is also a portable, scalable MPI profiling library originating from Lawrence Livermore National Laboratory but meanwhile maintained as an open-source project on sourceforge.net [5]. It profiles a larger set of MPI routines than FPMPI-2, and also provides the number of calls, total execution time, bytes sent for each MPI function, and optionally for each MPI call-site. Captured call-paths are determined by a user-specified traceback level. Also, it provides MPI file I/O statistics where applicable. The collected data is not aggregated across the processes but is collected in a scalable manner into one single output file that contains the complete data for all processes. Figure 2 shows a small portion of the measurement output of an ASC SMG2000 benchmark run on 8192 processes on the *Jubl* Blue Gene/L. The complete file consists of 295 075 lines (about 22MByte). A separate mpiPView GUI is provided to display and facilitate navigation in mpiP profile data.

Copyright © 2010 John Wiley & Sons, Ltd.

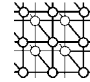
DOI: 10.1002/cpe




```
MPI Routine Statistics (FPMPI2 Version 2.1e)
Processes:       8192
Execute time:    52.77
Timing Stats: [seconds] [min/max]           [min rank/max rank]
  wall-clock: 52.77 sec 52.770000 / 52.770000      0 / 0
        user: 52.79 sec 52.794567 / 54.434833    672 / 0
         sys: 0 sec      0.000000 / 0.000000       0 / 0

                  Average of sums over all processes
Routine           Calls        Time Msg Length    %Time by message length
                                                0.........1........1........
                                                          K        M
MPI_Allgather  :       1    0.000242          4 0*0000000000000000000000000
MPI_Allgatherv :       1     0.00239         28 0000*00000000000000000000000
MPI_Allreduce  :      12    0.000252         96 00*0000000000000000000000000
MPI_Reduce     :       2       0.105          8 0*0000000000000000000000000
MPI_Isend      :  233568        1.84   2.45e+08 01.....1112111...00000000000
MPI_Irecv      :  233568       0.313   2.45e+08 02...111112......00000000000
MPI_Waitall    :   89684        23.7
MPI_Barrier    :       1    0.000252

Details for each MPI routine
                                                % by message length
                            (max over           0.........1........1........
                             processes [rank])            K        M
MPI_Isend:
  Calls       :      233568        436014 [3600] 02...111122......00000000000
  Time        :        1.84          2.65 [3600] 01.....1112111...00000000000
  Data Sent   :    2.45e+08    411628760 [3600]
  By bin      : 1-4               [13153,116118]  [     0.0295,      0.234]
              : 5-8               [2590,28914]    [    0.00689,     0.0664]
...
              : 131073-262144     [8,20]          [     0.0162,     0.0357]
  Partners    :          245 max 599(at 2312) min 47(at 1023)
MPI_Waitall:
  Calls       :       89684
  Time        :        23.7
  SyncTime    :        6.07

// Similar details for other MPI routines
```

Figure 1. FPMPI-2 measurement output of an ASC SMG2000 benchmark run with 8k processes on a Blue Gene/L system. The times given are the times to perform the operation. Average times, minimum, and maximum refer to all processes. Amount of data is computed in bytes. For synchronizing collective operations, the average, min, and max time spent synchronizing is shown next. Calls by message size show the fraction of calls that sent messages of a particular size. Each bin is represented by a single digit, representing the tens of percent of messages within this bin. A '0' represents precisely zero, a '.' (period) represents non-zero but less than 10%, and '*' represents 100%. Messages by message size show similar information, but for the total message size (with similar intermediate lines elided).

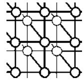
Copyright © 2010 John Wiley & Sons, Ltd.



```
@ mpiP
@ Version: 3.1.1
// 10 lines of mpiP and experiment configuration options
// 8192 lines of task assignment to BlueGene topology information

@--- MPI Time (seconds) ---------------------------------------------
Task    AppTime    MPITime     MPI%
   0       37.7       25.2    66.89
 ...
8191       37.6         26    69.21
   *   3.09e+05   2.04e+05    65.88

@--- Callsites: 26 --------------------------------------------------
 ID Lev File/Address      Line Parent_Funct             MPI_Call
  1   0 coarsen.c          542 hypre_StructCoarsen      Waitall
// 25 similiar lines

@--- Aggregate Time (top twenty, descending, milliseconds) ----------
Call                 Site       Time    App%    MPI%     COV
Waitall                21   1.03e+08   33.27   50.49    0.11
Waitall                 1   2.88e+07    9.34   14.17    0.26
// 18 similiar lines

@--- Aggregate Sent Message Size (top twenty, descending, bytes) ----
Call                 Site      Count      Total       Avrg  Sent%
Isend                  11  845594460   7.71e+11        912  59.92
Allreduce              10      49152   3.93e+05          8   0.00
// 6 similiar lines

@--- Callsite Time statistics (all, milliseconds): 212992 -----------
Name            Site Rank     Count     Max      Mean      Min    App%    MPI%
Waitall           21    0    111096     275       0.1 0.000707   29.61   44.27
   ...
Waitall           21 8191     65799     882      0.24 0.000707   41.98   60.66
Waitall           21    * 577806664     882     0.178 0.000703   33.27   50.49
// 213,042 similiar lines

@--- Callsite Message Sent statistics (all, sent bytes) -------------
Name            Site Rank     Count       Max      Mean    Min         Sum
Isend             11    0     72917 2.621e+05     851.1      8   6.206e+07
   ...
Isend             11 8191     46651 2.621e+05      1029      8   4.801e+07
Isend             11    * 845594460 2.621e+05     911.5      8   7.708e+11
// 65,550 similiar lines
```

Figure 2. mpiP measurement output of an ASC SMG2000 benchmark run with 8 k processes on a Blue Gene/L system, showing the basic structure of the reported data. A '*' represents all 8192 ranks and '. . .' indicates similar lines elided for the ranks from 1 to 8190. The call-site time and message statistics have a line for every MPI function call site and rank (only one of which is shown here).

Copyright © 2010 John Wiley & Sons, Ltd.



The largest reported experiment we are aware of is a 65 536 processor run on the LLNL Blue Gene/L analyzing the Qbox molecular dynamics code [6].

Finally, the MPI profiling library of the IBM High-Performance Computing Toolkit (**HPCT**) [7] available from the IBM Advanced Computing Technology Center for IBM customers uses yet another approach for minimizing the amount of data given to the user for extreme scaling experiments. The usual data is collected for each MPI call (and again optionally for each MPI call site via a user-specified traceback level) and process, but only the processes with the MPI rank zero and with the ranks representing the minimum, maximum, and average total MPI execution time are written to file. Figure 3 shows an extract of such a report produced by an SMG2000 measurement of 8192 processes on the *Jubl* Blue Gene/L.

TAU profiling

The **TAU** framework [8,9] from the University of Oregon is a very versatile and portable tool set for the performance analysis of parallel multi-threaded applications. Its components support the measurement of profiling and tracing data based on the same integrated set of program instrumentation facilities. Collected data includes a wide variety of metrics including time, hardware counter values, and memory usage. The large set of instrumentation facilities provide source and object code as well as static and dynamic instrumentation for many languages (among them C, C++, Fortran, Java, Python), threading models (e.g. pthreads, Java, Win32, OpenMP) and message passing (MPI, SHMEM). For profiling, it supports flat, call-path, and incremental profiles.

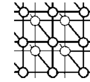
Profiles are collected and stored on a per-thread basis. After the measurement, the profile data is usually loaded in a profile experiment database called PerfDMF [10]. The use of a database not only allows performing a wide range of analyses of the collected profile data but also comparisons between experiments. Aggregation happens on-the-fly while processing the analysis queries. The database and special 3D triangle mesh, scatter, and bar plots ensure the scalability of the performance analysis. Because of the widespread use of TAU at open and restricted computer sites, it is not clear how large the largest successful profiling experiment with TAU was, but it was certainly 64 000 processors or more. Figure 4 shows a 3D bar plot of a measurement of the Miranda code executed with 16 384 processes on a Blue Gene/L system [11].

Scalasca profiling

The **Scalasca** toolset [12,13] from Jülich Supercomputing Centre is a performance-analysis tool that has been specifically designed for use on large-scale systems including Blue Gene and Cray XT. Scalasca integrates both profiling and tracing in a stepwise performance diagnosis process, adopting a strategy of successively refined measurement configurations. The current version of Scalasca can be applied to MPI and OpenMP programs written in C/C++ and Fortran.

Compared to its predecessor **KOJAK** [14,15], Scalasca includes a call-path profiling component and uses a parallel and therefore more scalable approach to analyzing event traces. The current version takes full call-path profiles of MPI and user functions, as well as OpenMP parallel regions, and incremental profiling is supported in a prototype version. The metrics provided include wall-clock time, message counts, bytes communicated, and optionally hardware counters. On most systems, including Blue Gene, the instrumentation of user functions is performed completely

Copyright © 2010 John Wiley & Sons, Ltd.



```
elapsed time from clock-cycles using freq = 700.0 MHz
-----------------------------------------------------------------
MPI Routine                  #calls     avg. bytes      time(sec)
-----------------------------------------------------------------
MPI_Comm_size                 31696            0.0          0.009
MPI_Comm_rank                 36391            0.0          0.006
MPI_Isend                    125475          585.9          0.789
MPI_Irecv                    125242          681.9          0.136
MPI_Waitall                  121060            0.0         33.116
MPI_Barrier                       1            0.0          0.000
MPI_Allgather                     1            4.0          0.000
MPI_Allgatherv                    1           28.0          0.002
MPI_Allreduce                    12            8.0          2.591
-----------------------------------------------------------------
total communication time = 36.650 seconds.
total elapsed time       = 49.904 seconds.

-----------------------------------------------------------------
Message size distributions:

MPI_Isend                 #calls    avg. bytes      time(sec)
                           26279           4.0          0.054
                            7582           8.0          0.018
...
                               3      131072.0          0.006
                               5      262144.0          0.004

// Similar details for other MPI routines
-----------------------------------------------------------------

Communication summary for all tasks:

  minimum communication time = 27.538 sec for task 3600
  median  communication time = 32.563 sec for task 4027
  maximum communication time = 37.557 sec for task 63

MPI tasks sorted by communication time:
taskid  xcoord  ycoord  zcoord  procid    total_comm(sec)
  3600      16       0      14       0         27.538
...
    63      31       1       0       0         37.557
```

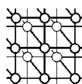
Figure 3. HPCT mpitrace measurement output of an ASC SMG2000 benchmark run with 8 k processes on a Blue Gene/L system, showing the basic structure of the reported data. '. . .' indicates similar lines elided for intermediate message sizes and tasks. Only the report for rank zero includes the communication summary at the bottom, whereas the reports for the ranks with minimum/median/maximum communication times only include the MPI profile output at the top.

automatically by the compiler; on other systems a mix of manual and automatic instrumentation mechanisms is offered.

To avoid inefficient use of the file system, the Scalasca profiler emits only a single report file at the end of the run, which is collated in parallel using MPI collective operations. For the same reason, the

Copyright © 2010 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe



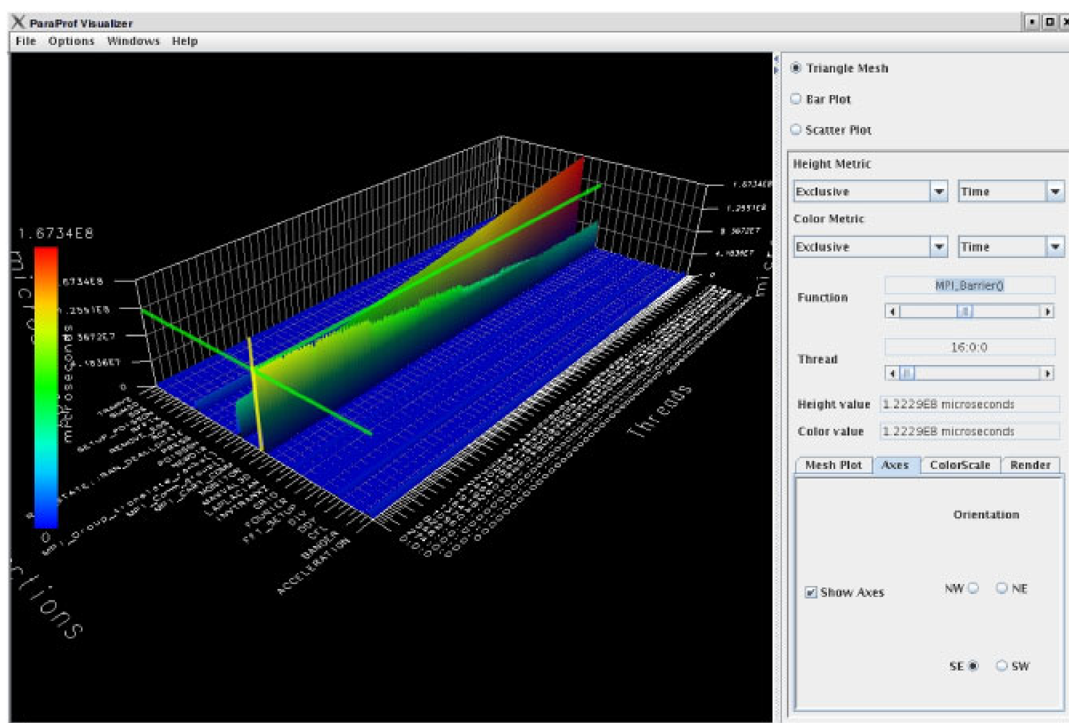


Figure 4. TAU 3D bar plot of a measurement of the Miranda code executed with 16 k processes on a Blue Gene/L system. A metric (here *Exclusive Time*) is plotted for all threads (axis on the right) and all monitored functions (axis on the left). The color coding could be used to represent another metric; in the picture above it is also showing *Exclusive Time*. A 3D cursor can be used to inquire values of conspicuous metrics.

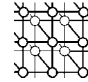
necessary unification of local identifiers used to denote program objects such as regions or call-paths is also done with MPI. The profiles are stored as a three-dimensional array with the dimensions metric, call-path, and process. Motivated by the need to analyze the performance behavior on different levels of granularity, each dimension is organized in a hierarchy. The profiles can be viewed in a flexible browser that uses coupled tree widgets to navigate through the hierarchical performance space. To facilitate the analysis of runs on thousands of processors, the browser provides a scalable two- or three-dimensional Cartesian grid display to visualize physical or virtual process topologies.

Cray tools profiling

The Cray performance analysis tools [16] provide an integrated infrastructure for measurement and analysis of computation, communication, I/O, and memory utilization. CrayPat Performance Collector is the data capture tool and Cray Apprentice2 Performance Analyzer is a post-processing data visualization tool that is used to explore and study the captured data. Data collection is supported

Copyright © 2010 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe



via external sampling and internal code instrumentation mechanisms. The toolset allows data to be recorded either as a summation of events over time (profile), or as a sequence of events over time (trace files). By using a drill-down approach, the combination of these mechanisms allows the Cray performance analysis tools to work at large scale. *Automatic Profiling Analysis* collects performance data from a running application using sampling, analyzes the data, and identifies the important areas for automatic code instrumentation with the pat_build utility. The Cray performance analysis tools have been used regularly at large scales on high-end Cray XT systems with more than 30 000 processors. One of the main focuses of the analysis engine is detecting possible scalability problems via load balance analysis [17], MPI synchronization time analysis, combined with profile-guided MPI rank placement suggestions.

SCALABLE TRACING APPROACHES

Whereas the strength of profiling lies in the compactness of the resulting data sets and the robustness of related tools, event tracing allows the in-depth study of parallel program execution behavior. Tracing is especially effective for observing the interactions between different processes or threads that occur during communication or synchronization operations and to analyze the way concurrent activities influence each other's performance. When an application execution is traced, performance-relevant events, such as entering functions or sending messages, are recorded at runtime and analyzed postmortem to identify potential performance problems.

Traditionally, developers of parallel programs use event traces to visualize the program behavior along the time axis in the style of a Gantt chart, where local activities are represented as boxes with a distinct color (Figures 5 and 6). Interactions between processes can be highlighted by drawing arrows or polygons to illustrate the exchange of messages or the involvement in a collective operation, respectively (not shown). Alternatively, event traces can be analyzed automatically by scanning them for characteristic patterns and extracting features of interest.

While event tracing enables the investigation of performance problems at a high level of detail, growing trace-file size often constrains its scalability and complicates management, analysis, and visualization of trace data. Users of tracing tools typically confront problems, such as massive storage requirements during trace generation and in-memory analysis, program perturbation when flushing event buffers to disk at runtime, limited I/O bandwidth when accessing trace files on disk, and failure, extended response times, and insufficient resolution of graphical displays.

The reasons for large traces can be roughly divided into three categories: (i) large numbers of processes or threads (trace width), (ii) high event frequencies in combination with long execution intervals to cover (trace length), and (iii) number of metrics recorded per event (trace depth). Below, we survey current approaches to handle large traces and classify them according to the primary issues they address and the primary benefits they offer.

Methods and approaches for scalable tracing

To allow for efficient zooming and scrolling of timeline visualizations, methods are needed to efficiently access trace data from files. The **Jumpshot** [18] trace browser from Argonne National Laboratory facilitates the scalable visualization of long traces by dividing the trace file into frames

Copyright © 2010 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

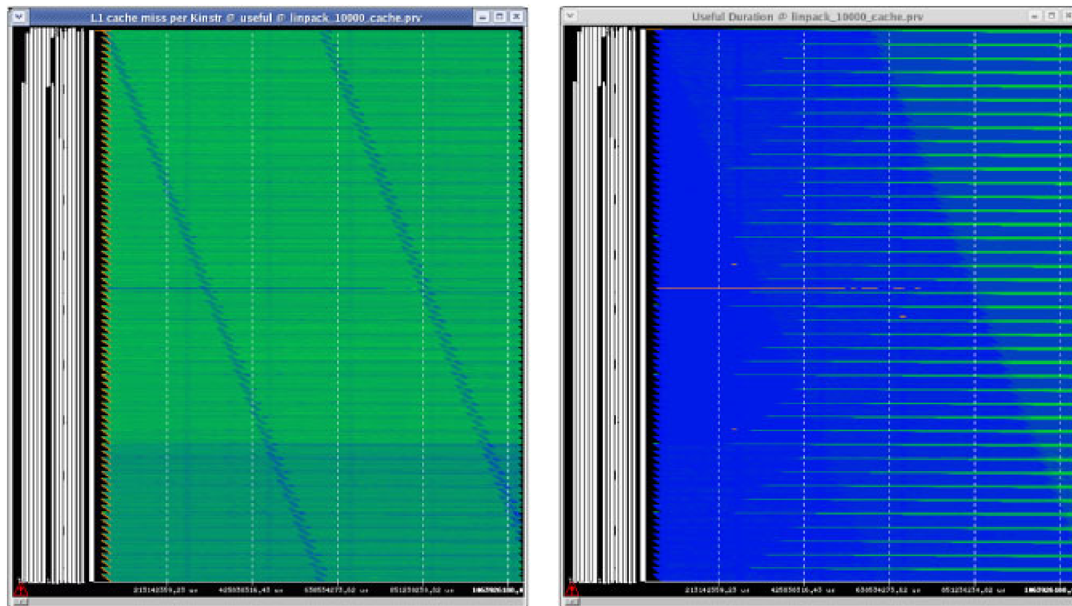
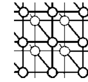


Figure 5. Paraver timeline plots of a measurement of the High-Performance Linpack run executed on Barcelona Supercomputing Center's MareNostrum machine. The x-axis covers all 1700 s of the execution while the y-axis shows all of the 10 000 processors used. Zooming in on any rectangular subregion of the plot allows analyzing the measured data in a flexible yet scalable way. The color encodes the value of a selected metric (left: *L1 Cache Misses*, right: *Useful Duration* of DGEMM) over time.

representing intervals that can be separately loaded and analyzed [19]. In this way, the time needed and the amount of memory required to load and display a given interval from a trace file in Jumpshot's native SLOG format depend only on the number of graphical objects to be displayed. The visual performance can be further improved by arranging drawable objects into a binary tree of bounding boxes, which provides better support for drawing coarse-grained previews [20]. After the desired interval has been loaded, the user can unclutter the timeline visualization by letting Jumpshot replace larger sets of related message arrows with summary arrows.

The **Paraver** [21] trace-browser framework from Barcelona Supercomputing Center employs a variety of scalability-enhancing techniques [22], covering all steps of the trace analysis from instrumentation through post-processing to visualization. To reduce trace-file size, Paraver uses a system of filters that stepwise transform larger traces into smaller ones both by eliminating dispensable features as well as by summarizing unnecessary details using a mechanism they call soft counters. A key method in this context is the automatic extraction of structural properties using signal processing techniques [23]. To avoid long traces in the first place, Paraver can skip the recording of repetitive behavior based on a dynamic periodicity detection algorithm [24]. This is applied locally and therefore assumes structural similarities between concurrent control flows, which is why this technique has so far been applied only to OpenMP applications. Finally, in contrast to other trace browsers, Paraver uses the same compressing, zooming, and panning facilities in the

Copyright © 2010 John Wiley & Sons, Ltd.



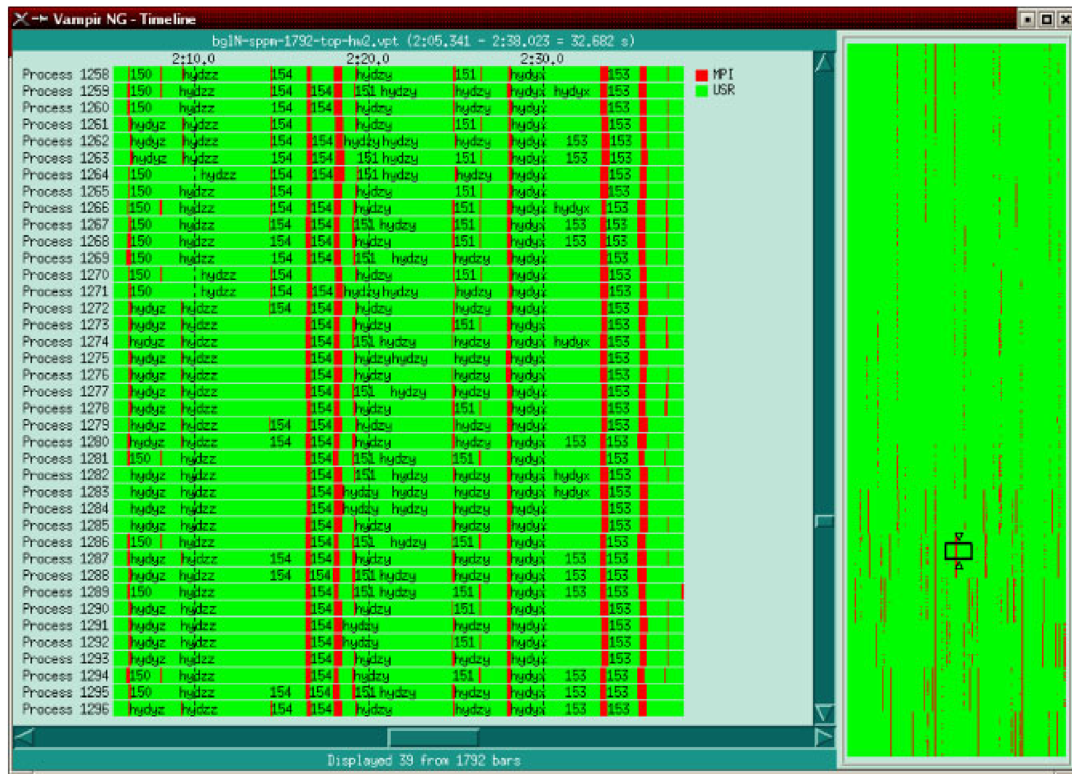


Figure 6. Vampir timeline display of a measurement of the ASC sPPM benchmark executed with 1792 processes on a Blue Gene/L. The thumbnail pane on the right of the window gives a rough approximation of the complete measurement. The main pane in the left shows the currently selected portion (which is indicated by a black rectangle in the overview thumbnail).

processor (vertical) axis as for the time (horizontal) axis. Other browsers use a simple scrollbar for this purpose, allowing only a subset of the processes to be shown and examined at a time and making it difficult to obtain an overview of the global state. This makes Paraver's timeline visualization much more scalable for very large number of processors. For example, Figure 5 shows the complete timeline plot of a High-Performance Linpack run on 10 000 processors of the MareNostrum machine [25], from which the broad patterns of execution are still distinguishable.

Likewise, the scalability of the **Vampir** [26] trace browser from TU Dresden has been enhanced in several ways. The current version, called VampirServer [27,28], is based on the principle of keeping performance data close to the location where they were created and of exploiting distributed memory for the analysis. It consists of a parallel analysis server and a potentially remote visualization client. The server is submitted postmortem as a separate parallel job usually on the machine where the trace data were created, while the client runs on the user workstation and interacts with the server via the network. A screenshot of the client display in Figure 6 shows a subsection of the timeline of a 1792 processor measurement of the ASC sPPM benchmark on a Blue Gene/L system. Moreover,

Copyright © 2010 John Wiley & Sons, Ltd.

unlike common linear storage schemes for event data, a tree-based main memory data structure called *compressed Complete Call Graph* (cCCG) [29] allows potentially lossy compression of long traces while observing specified deviation bounds.

Another approach for trace compression has been developed in the **ScalaTrace** project [30], which applies section descriptors to perform both intra-node and inter-node compression, thus addressing long as well as wide traces. For some applications, their original scheme, which does not support timestamped events, reduces traces to near constant size independent of the number of nodes. A more recent extension aims at retaining the relative distance of events at least to some degree by introducing delta-time histograms [31].

Scalasca tracing

In message-passing (i.e. MPI) applications, a significant fraction of the time spent in communication and synchronization routines can often be attributed to wait states that occur when processes fail to reach implicit or explicit synchronization points in a timely manner, for example, as a result of unevenly distributed workloads. Because wait states cause temporal displacements between program events occurring on different processes, they can be identified by automatically searching event traces for corresponding patterns [14,15]. In addition to being usually faster than a manual analysis using a trace browser, this approach is also guaranteed to cover the entire event trace and not to miss any instances.

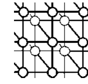
To accomplish the search is a scalable way, Scalasca exploits both distributed memory and parallel processing capabilities available on the target system. Instead of sequentially analyzing a single global trace file, as KOJAK does, Scalasca analyzes separate process-local trace files in parallel by *replaying* the original communication on as many CPUs as have been used to execute the target application itself. During the search process, Scalasca classifies detected pattern instances by category and quantifies their significance for every program phase and system resource involved. Since trace processing capabilities (i.e. processors and memory) grow proportionally with the number of application processes, the Scalasca analysis replay has achieved good scalability even at previously intractable scales. Additionally, to allow accurate trace analyses on systems without globally synchronized clocks, the trace analyzer provides the ability to synchronize inaccurate timestamps postmortem using the same scalable replay mechanism [32].

EXPERIMENTAL RESULTS

ASC benchmark SMG2000 [4] is a semi-coarsening multi-grid solver which is known to scale well on Blue Gene/L (in weak scaling mode where the problem size per process is constant) but pose considerable demands on MPI performance analysis tools due to huge amounts of non-local communication [33]. It therefore provides a challenge for comparing performance tools operating at large scales.

Figure 7 summarizes measurements with a selection of the performance analysis tools available in mid-2007 that we were able to apply to the ASC SMG2000 benchmark with 8192 processes on the *Jubl* Blue Gene/L. The first bar (marked ‘uninst’) is the execution time of the uninstrumented optimized application, which is used as a reference in determining instrumentation and measurement

Copyright © 2010 John Wiley & Sons, Ltd.



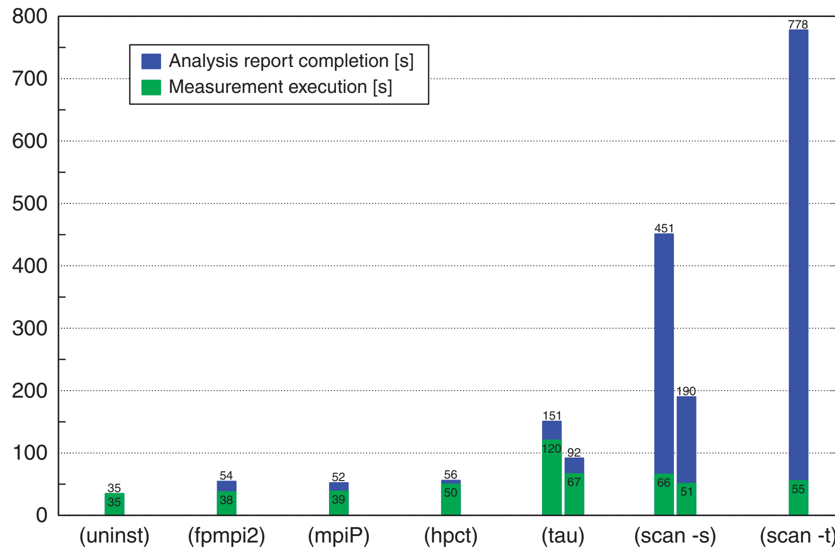


Figure 7. Measurement and analysis times for the ASC SMG2000 benchmark execution with 8192 processes on Blue Gene/L using a selection of tools. Measured executions take longer than the uninstrumented execution, due to the dilation effect of instrumentation and measurement during application execution combined with additional analysis and reporting time at completion.

dilation. The next three bars are for measurements taken with the three MPI profiling libraries—FPMPI-2, mpiP and HPCT— and show only small differences in measurement and reporting time, with dilation of the application execution (prior to finalization) starting at around 10%. Notably, mpiP could only be run without callstack traceback, due to excessive memory requirements during final report collation with more than 1024 processes, and FPMPI-2 had the same limit on providing point-to-point message distance information as part of its report.

The TAU profiler supports a large variety of measurements configurations, and along with the default cell-path profile measurement configuration a second one employing measurement throttling was also used. Throttling reduces measurement overhead by dynamically disabling measurements for frequently executed functions such as `MPI_Isend` and `MPI_Irecv`. Final reporting produces a separate profile file for each of the 8192 processes, and these must be unified and collated during analysis presentation. Unfortunately, our Blue Gene/L p720 frontend had insufficient memory to be able to load all of these files, however, subsets could be examined.

Scalasca measurement collection and analysis (marked ‘scan’) was also performed in several configurations, all using the same executable prepared with automatic function instrumentation along with an instrumented MPI library. The initial default measurement produced a complete runtime summarization profile with 3084 distinct call-paths, from which a filter was created listing some 32 frequently executed functions that were not found on MPI call-paths. Subsequent measurement times with this filter were thereby reduced both by fewer measurements being taken and only 580 call-paths remaining in the summary report, considerably reducing the time to unify the set of call-path definitions and collate the corresponding profile report. Measurement dilation was reduced

Copyright © 2010 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

below 50%, and although this is more than that of the pure MPI profiling libraries, it reflects the additional call-path information and timing of user functions.

Using this filter during trace collection ('scan -t') also reduced per-process trace sizes, allowing them to be stored entirely in memory and avoiding highly disruptive intermediate flushes to disk. Along with definition unification (98 s) and call-path profile collation (46 s), as for pure summarization, additional time is required for trace handling (453 s) and event replay (109 s) during parallel trace analysis to produce the final call-path profile augmented with inefficiency metrics.

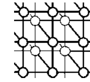
The different tools tested are seen to provide increasingly detailed analysis of the parallel application execution performance on a corresponding scale of cost. While the simplest MPI profiling tools report a large proportion of time in `MPI_Wait` operations, more comprehensive tools identify which of the various `MPI_Wait` call-sites or call-paths are responsible, and which of the processes are affected by the highly imbalanced execution behavior. Further analysis can distinguish potentially avoidable waiting time from the necessary communication and synchronization times to isolate the performance problems which are most amenable to remediation.

Experience with Scalasca

Scalasca trace collection and analysis scalability has been improved as inefficiencies were identified and addressed. Replay-based parallel trace analysis, able to exploit per-process event traces without merging or rewriting them, is one approach for scalable trace analysis. It has been complemented with a runtime summarization capability sharing the infrastructure for online unification of definition identifiers and collation of per-process analysis reports, that provides a more convenient starting point for general analysis from which trace analysis can be targeted effectively. Reducing the number of files written when tracing to a single file per process (plus two global files for a set of unique object definitions referenced by event records and for identifier mapping tables), which is written only once directly into a dedicated experiment archive, has also been necessary. Since filesystem performance is expected to continue to lag behind that of computer systems in general, even when parallel I/O is employed, elimination of unnecessary files provides benefits that grow with scale and are well-suited for the future.

As a specific example, SMG2000 using 65 536 processes on the *Jugene* Blue Gene/P was analyzed with the 1.0 (June 2008) release version of Scalasca. The uninstrumented version of the application ran in 17 min, including 12 min to launch this number of processes. From an initial runtime summary of the fully instrumented application, where execution time was dilated by 25% to 6 min, a file listing a number of purely computational functions to be filtered from the trace was determined. Collection and analysis of the resulting 3.33 TB trace took 100 min (two-thirds collection and one-third analysis), including 30 min for the two launches. Unifying identifier definitions and writing associated maps took 12 min, whereas writing the traces in parallel to GPFS achieved 6.2 GB/s and took 10 min, saturating the available transfer bandwidth. The major bottleneck, however, was the 25 min required to initially open and create one trace file per process. The solver section of the resulting 2.3 GB analysis report was subsequently extracted, and Figure 8 shows the distribution of the *Late Sender* metric for the call-path taking the longest time. With its 50% standard deviation there is clearly a significant imbalance that closely corresponds to the physical racks of the Blue Gene/P system. Furthermore, only 0.2% of the total number of late senders are on this call-path that requires one-third of the total *Late Sender* time, so there is potentially a

Copyright © 2010 John Wiley & Sons, Ltd.



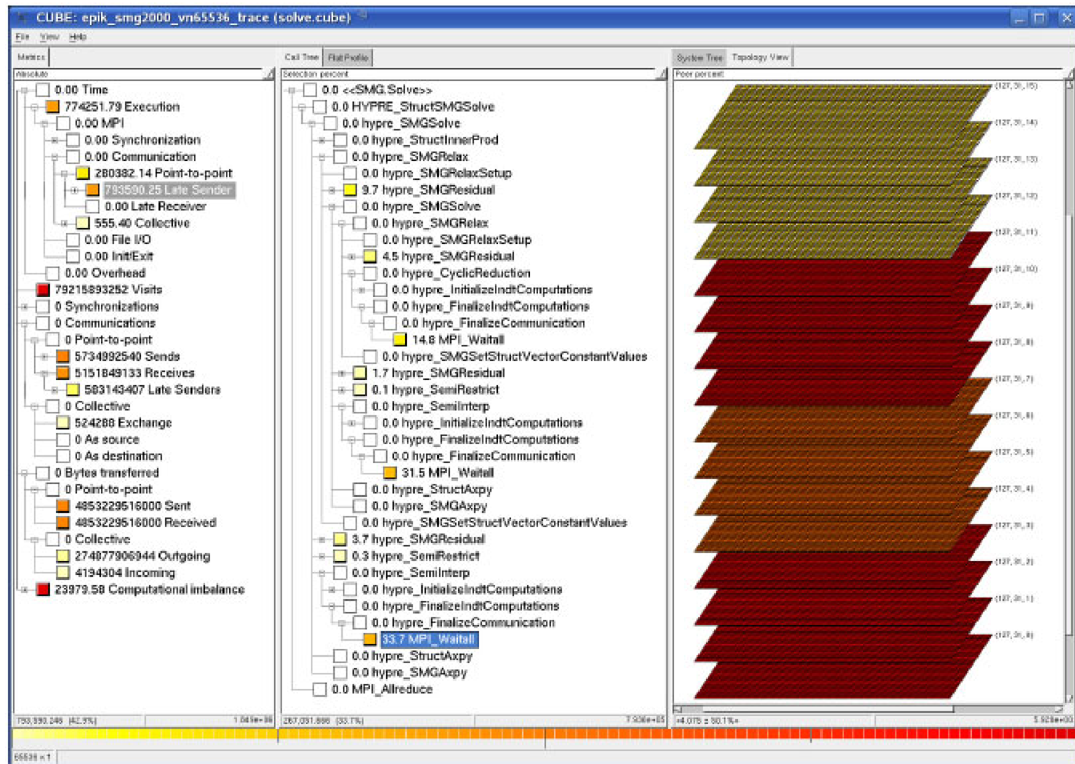
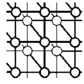


Figure 8. Scalasca analysis report explorer presentation of the solver extract of the trace analysis report for a 64k-process measurement of SMG2000 on a 16-rack Blue Gene/P, with *Late Sender* time selected from the metrics pane (left) amounting to 43% of total time (compared to 42% for *Execution* time excluding MPI). One third of *Late Sender* time is attributed to one `MPI_Waitall` from the call-tree pane (centre), and its distribution across the 65 536 processes shown on the Blue Gene physical topology (right). Metric values are colour-coded according to the scale at the bottom of the window to facilitate distinction of high (dark) and low (light) severities in both tree and topology displays.

great benefit from addressing this localized inefficiency, e.g. using a better mapping of processes to processors.

Beyond relatively simple benchmark kernels, Scalasca has also successfully been used to analyze and tune a number of locally important applications. The XNS simulation of flow inside a blood pump, based on finite-element mesh techniques, was analyzed using 4096 processes on Blue Gene/L, and after removing unnecessary synchronizations from critical scatter/gather and scan operations performance improved more than four-fold [34,35]. On the MareNostrum blade cluster, the WRF2 weather research and forecasting code was analyzed using 2048 processes, and occasional problems with seriously imbalanced exits from `MPI_Allreduce` calls that significantly degraded overall application performance were identified [36,37]. In both cases, the high-level call-path profile readily available from runtime summarization was key in identifying general performance

Copyright © 2010 John Wiley & Sons, Ltd.



issues that manifest at large scale, related to performance problems that could subsequently be isolated and understood with targeted trace collection and analysis.

CONCLUSION

In this paper we gave an overview of scalable profiling and tracing tools for today's extreme-scale computing systems. It shows that the main driving force in this area is an open community of tool developers at universities and government research laboratories. While IBM and Cray are still developing and providing tools for very large-scale supercomputers with funding from the U.S. HPCS program, it is not yet clear whether this is adequate to support the wider community of HPC users and systems.

Although it is difficult to directly compare the robustness and usability of research tools with commercial products, many of them reached a level where they can be effectively used to analyze 'real-world' applications on today's large machines. However, it is unclear whether this can be sustained in the future. It is hard to acquire the necessary access and amount of computing time necessary to test and optimize the tools on large numbers of processors. Complicating the situation is the fact that tool developers only get access to new machines at the same time as application developers, so the tools are not ported to or tuned for the new architecture when the application programmers need them most. Simpler tools are more readily ported to new systems than those requiring more demanding system functionality, e.g. Rice University HPCToolkit could not be included when we performed our evaluation in 2007 as the patched operating system kernels it requires for Blue Gene and Cray XT first became available in 2009 [38]. In comparison, first measurements in 2009 of hybrid OpenMP/MPI application executions on Cray XT5 systems and with 294 912 MPI processes on the JSC Blue Gene/P system *Jugene* have demonstrated both runtime summarization and automated trace analysis with the Scalasca toolset at an unprecedented scale [13,39].

However, providing more scalable tools is only half the solution. Equally important will be tools that can handle the heterogeneity of future systems exploiting accelerator hardware in various forms. Finally, performance measurement and analysis tools need to be complemented with equally scalable debuggers and other code correctness tools such as memory analyzers, to provide a comprehensive software development environment for today's and future complex application codes.

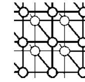
ACKNOWLEDGEMENTS

We wish to thank Forschungszentrum Jülich for making available the large number of compute hours to perform the measurements presented in this paper, and the JSC support staff for their professional help and patience in assisting our efforts.

REFERENCES

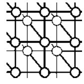
1. Mohr B, Frings W (eds.). Jülich Blue Gene/P Extreme Scaling Workshop 2009. *Technical Report FZJ-JSC-IB-2010-02*, Jülich Supercomputing Centre, 2010.
2. TOP500.org. TOP500 supercomputers. Available at: http://www.top500.org/ [16 November 2009].

Copyright © 2010 John Wiley & Sons, Ltd.



3. Argonne National Laboratory. The FPMPI-2 MPI profiling library. Available at: http://www-unix.mcs.anl.gov/fpmpi/ [3 September 2007].
4. Accelerated Strategic Computing Initiative. The ASC SMG2000 benchmark code. UCRL-MI-144211, 2001. Available at: https://asc.llnl.gov/computing_resources/purple/archive/benchmarks/smg/ [3 September 2007].
5. Vetter J, Chambreau C. The mpiP MPI profiling library. Available at: http://mpip.sourceforge.net/ [3 September 2007].
6. Knüpfer A, Kranzlmüller D, Schulz M, Klausecker C. Large-scale communication analysis. *Tutorial notes, SC'09*. ACM/IEEE Computer Society: Portland, OR, U.S.A., 2009.
7. IBM Advanced Computing Technology Center. High Performance Computing Toolkit. Available at: https://domino.research.ibm.com/comm/research_projects.nsf/pages/actc.index.html [3 September 2007].
8. University of Oregon. Tuning and Analysis Utilities. Available at: http://tau.uoregon.edu/ [3 September 2007].
9. Shende S, Malony A. The TAU parallel performance system. *International Journal of High Performance Computing Applications* 2006; **20**:287–331.
10. Huck K, Malony A, Bell R, Morris A. Design and implementation of a parallel performance data management framework. *Proceedings of the International Conference on Parallel Processing* (*ICPP*), Oslo, Norway. IEEE Computer Society: Silver Spring, MD, 2005; 473–482.
11. Malony A, Shende SS, Morris A. Performance technology for productive parallel computing. *Advanced Scientific Computing Research Computer Science FY2005 Accomplishments*, U.S. Department of Energy Office of Science, 2005.
12. Jülich Supercomputing Centre. The Scalasca toolset for scalable performance analysis of large-scale parallel applications. Available at: http://www.scalasca.org/ [16 November 2009].
13. Geimer M, Wolf F, Wylie BJN, Ábrahám E, Becker D, Mohr B. The Scalasca performance toolset architecture. *Concurrency and Computation*: *Practice and Experience* 2010; **22**:702–719. DOI: 10.1002/cpe.1556.
14. Wolf F, Mohr B. Automatic performance analysis of hybrid MPI/OpenMP applications. *Journal of Systems Architecture* 2003; **49**:421–439.
15. Wolf F, Mohr B, Dongarra J, Moore S. Automatic analysis of inefficiency patterns in parallel applications. *Concurrency and Computation*: *Practice and Experience* 2007; **19**:1481–1496.
16. DeRose L, Homer B, Johnson D, Kaufmann S, Poxon H. Cray performance analysis tools. *Tools for High Performance Computing*. Springer: Berlin, 2008; 191–199.
17. DeRose L, Homer B, Johnson D. Detecting application load imbalance on high end massively parallel systems. *Proceedings of Euro-Par 2007* (*Lecture Notes in Computer Science*, vol. 4641). Springer: Berlin, 2001; 151–159.
18. Zaki O, Lusk E, Gropp W, Swider D. Toward scalable performance visualization with Jumpshot. *High Performance Computing Applications* 1999; **13**:277–288.
19. Wu CE, Bolmarcich A, Snir M, Wootton D, Parpia F, Chan A, Lusk E, Gropp W. From trace generation to visualization: A performance framework for distributed parallel systems. *Proceedings of the Supercomputing Conference* (*SC*), Dallas, TX, U.S.A. ACM, IEEE Computer Society: New York, Silver Spring, MD, 2000.
20. Chan A, Gropp W, Lusk E. Scalable log files for parallel program trace data (DRAFT). *Technical Report*, Argonne National Laboratory, 2000.
21. Labarta J, Girona S, Pillet V, Cortes T, Gregoris L. DiP: A parallel program development environment. *Proceedings of the 2nd International Euro-Par Conference* (*Lecture Notes in Computer Science*, vol. 1124), Lyon, France. Springer: Berlin, 1996; 665–674.
22. Labarta J, Giménez J, Martinez E, González P, Servat H, Llort G, Aguilar X. Scalability of visualization and tracing tools. *Proceedings of Parallel Computing* (*ParCo*), Málaga, Spain, 2005; 869–876.
23. Casas M, Badia RM, Labarta J. Automatic phase detection of MPI applications. *Proceedings of Parallel Computing*, *ParCo* (*Advances in Parallel Computing*, vol. 15), Aachen/Jülich, Germany. IOS Press, 2007; 129–136.
24. Freitag F, Caubet J, Labarta J. On the scalability of tracing mechanisms. *Proceedings of the European Conference on Parallel Computing* (*Lecture Notes in Computer Science*, *Euro-Par*, vol. 2400), Paderborn, Germany. Springer: Berlin, 2002; 97–104.
25. Labarta J. Scalability of trace analysis tools. *Workshop on Tools for Petascale Computing*, Snowbird, Utah, U.S.A., 2007.
26. Nagel W, Weber M, Hoppe HC, Solchenbach K. VAMPIR: Visualization and analysis of MPI resources. *Supercomputer* 1996; **12**:69–80.
27. Brunst H, Nagel WE. Scalable performance analysis of parallel systems: Concepts and experiences. *Proceedings of the Parallel Computing Conference* (*ParCo*), Dresden, Germany, 2003; 737–744.
28. Brunst H. Integrative concepts for scalable distributed performance analysis and visualization of parallel programs. *Ph.D. Thesis*, TU Dresden, Shaker, Aachen, Germany, 2008. ISBN: 978-3-8322-6990-6.
29. Knüpfer A, Nagel WE. Construction and compression of complete call graphs for post-mortem program trace analysis. *Proceedings of the International Conference on Parallel Processing* (*ICPP*), Oslo, Norway. IEEE Computer Society: Silver Spring, 2005; 165–172.
30. Noeth M, Müller F, Schulz M, de Supinski BR. Scalable compression and replay of communication traces in massively parallel environments. *Proceedings of the IEEE International Parallel and Distributed Processing Symposium* (*IPDPS*), Long Beach, CA, U.S.A., 2007.

Copyright © 2010 John Wiley & Sons, Ltd.



31. Ratn P, Müller F, de Supinski BR, Schulz M. Preserving time in large-scale communication traces. *Proceedings of the 22nd International Conference on Supercomputing* (*ICS*), Kos, Greece. ACM: New York, 2008; 46–55.
32. Becker D, Rabenseifner R, Wolf F. Timestamp synchronization for event traces of large-scale message-passing applications. *Proceedings of the 14th European Parallel Virtual Machine and Message Passing Interface Conference* (*EuroPVM/MPI*) (*Lecture Notes in Computer Science*, vol. 4757), Paris, France. Springer: Berlin, 2006; 315–325.
33. Geimer M, Wolf F, Wylie BJN, Mohr B. A scalable tool architecture for diagnosing wait states in massively-parallel applications. *Parallel Computing* 2009; **35**:375–388.
34. Wylie BJN, Geimer M, Nicolai M, Probst M. Performance analysis and tuning of the XNS CFD solver on BlueGene/L. *Proceedings of the 14th European PVM and MPI Conference*, *EuroPVM/MPI* (*Lecture Notes in Computer Science* vol. 4757), Paris, France. Springer: Berlin, 2007; 107–116.
35. Wylie BJN, Geimer M, Wolf F. Performance measurement and analysis of large-scale parallel applications on leadership computing systems. *Journal of Scientific Programming* 2008; **16**:167–181.
36. Wylie BJN. Scalable performance analysis of large-scale parallel applications on MareNostrum. *Science and Supercomputing in Europe Report 2007*. HPC–EuropaTransnational Access, CINECA: Casalecchio di Reno (Bologna), Italy, 2008; 453–461.
37. Wylie BJN, Geimer M. Performance measurement & analysis of large-scale scientific applications on leadership computer systems with the Scalasca toolset. *Proceedings of the Seminar 07341 on Code Instrumentation and Modeling for Parallel Performance Analysis* (*CIMPPA*), Schloss Dagstuhl, Germany, 2007.
38. Tallent N, Mellor-Crummey JM, Adhianto L, Fagan MW, Krentel M. Diagnosing performance bottlenecks in emerging petascale applications. *Proceedings SC2009*, Portland, OR, U.S.A. ACM, IEEE Computer Society: New York, Silver Spring, MD, 2009.
39. Wylie BJN, Böhme D, Mohr B, Szebenyi Z, Wolf F. Performance analysis of Sweep3D on Blue Gene/P with the Scalasca toolset. *Proceedings of the 24th International Parallel and Distributed Processing Symposium*, *IPDPS*, Atlanta, GA, U.S.A. IEEE Computer Society: Silver Spring, MD, 2010.

Copyright © 2010 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe