Scalable Performance Analysis Methods for the Next Generation of Supercomputers

Felix Wolf[1,2], **Daniel Becker**[1,2], **Markus Geimer**[1], **and Brian J. N. Wylie**[1]

[1] Forschungszentrum Jülich,
Jülich Supercomputing Centre, 52425 Jülich, Germany
E-mail: {f.wolf, d.becker, m.geimer, b.wylie}@fz-juelich.de

[2] RWTH Aachen University,
Department of Computer Science, 52056 Aachen, Germany

Facing increasing power dissipation and little instruction-level parallelism left to exploit, computer architects are realizing further performance gains by using larger numbers of moderately fast processor cores rather than by further increasing the speed of uni-processors. As a consequence, supercomputing applications are required to harness much higher degrees of parallelism in order to satisfy their growing demand for computing power. However, writing code that runs efficiently on large numbers of processors remains a significant challenge.

To address this challenge, the Helmholtz-University Young Investigators Group *Performance Analysis of Parallel Programs* at the Jülich Supercomputing Centre develops performance-analysis tools to diagnose inefficiencies in supercomputer applications and works with application developers to analyze and improve the performance of their codes. In this contribution, we highlight the research activities of our group during the past two years and give an outlook on future work. At the center of our report lies the development of SCALASCA, a performance-analysis tool that has been specifically designed for large-scale systems and that allows the automatic identification of harmful wait states in applications running on thousands of processors.

1 Introduction

Supercomputing is a key technology pillar of modern science and engineering, indispensable to solve critical problems of high complexity. The extension of the ESFRI road map to include a European supercomputer infrastructure in combination with the creation of the PRACE consortium acknowledges that the requirements of many critical applications can only be met by the most advanced custom-built large-scale computer systems. However, as a prerequisite for their productive use, the HPC community needs powerful and robust development tools. These would not only help improve the scalability characteristics of scientific codes and thus expand their potential, but also allow domain scientists to concentrate on the underlying models rather than to spend a major fraction of their time tuning their application on a particular machine.

As the current trend in microprocessor development continues, this need will become even stronger in the future. Facing increasing power dissipation and little instruction-level parallelism left to exploit, computer architects are realizing further performance gains by using larger numbers of moderately fast processor cores rather than by further increasing the speed of uni-processors. As a consequence, supercomputer applications are being required to harness much higher degrees of parallelism in order to satisfy their growing demand for computing power. With an exponentially rising number of cores, the often substantial gap between peak performance and the performance level actually sustained

by production codes is expected to widen even further. Finally, increased concurrency levels place higher scalability demands not only on applications but also on parallel programming tools. When applied to larger numbers of cores, familiar tools often cease to work in a satisfactory manner (e.g., due to escalating memory requirements or limited I/O bandwidth).

To overcome this challenge, the Helmholtz-University Young Investigators Group *Performance Analysis of Parallel Programs* at the Jülich Supercomputing Centre develops performance-analysis tools to diagnose inefficiencies in supercomputer applications running on thousands of processors and works directly with application developers to improve the performance of their codes. The main product of the group is the software package SCALASCA[1], a comprehensive performance-analysis tool that has been specifically designed for use on large-scale systems, such as the Blue Gene/L JUBL in Jülich and its successor Blue Gene/P JUGENE. When scaling message-passing applications to thousands of processors, their performance is often affected by wait states that occur when processes fail to cooperate efficiently. Building upon earlier experience obtained during the KOJAK project[2], SCALASCA is able to efficiently identify such wait states at the previously inaccessible scale of 16,384 processes and shows potential even for larger configurations.

In this article, we highlight the research accomplishments of our group during the past two years. We explain SCALASCA in detail and demonstrate how it can be used to diagnose performance problems in large-scale parallel applications. We also make a side trip to computational grids and demonstrate that SCALASCA's novel architecture proves beneficial there as well. At the end, we outline our research goals for the coming years.

2 SCALASCA

In message-passing (i.e., MPI) applications, which still constitute the major portion of large-scale applications running on systems such as IBM Blue Gene or Cray XT, processes often require access to data provided by remote processes, making the progress of a receiving process dependent upon the progress of a sending process. As a consequence, 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. Especially when trying to scale communication-intensive applications to large processor counts, such wait states can present severe challenges to achieving good performance. As a first step in reducing the impact of wait states, application developers need a diagnostic method that allows their localization, classification, and quantification especially at larger scales. Because wait states cause temporal displacements between program events occurring on different processes, their identification can be accomplished by logging those events with timestamps into so-called *event traces* and then searching these traces for patterns indicating situations where a process waits for input from one or more other processes.

2.1 Scalability

Compared to SCALASCA's predecessor KOJAK, the novel approach taken in SCALASCA is that the event traces are searched in a much more scalable way by exploiting 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. Since trace processing capabilities (i.e., processors and memory) grow proportionally with the number of application processes, we can achieve good scalability at previously intractable scales. In brief, to meet the scalability requirements of next-generation supercomputers, SCALASCA is a parallel program in its own right.

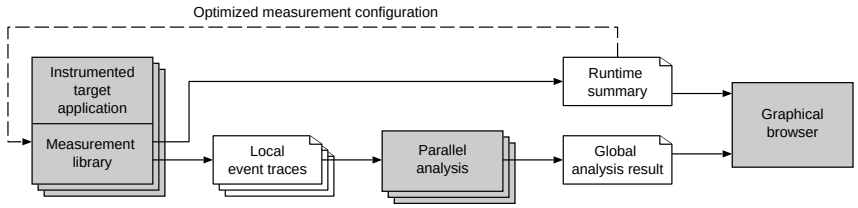

Figure 1. Parallel trace analysis in SCALASCA.

Figure 1 shows the basic analysis workflow carried out by SCALASCA. Before any performance data can be collected, the target application must be *instrumented*, that is, extra code must be inserted to record the events whenever they occur. On some systems including Blue Gene, this can be done completely automatically using compiler support; on other systems a mix of manual and automatic instrumentation mechanisms is offered. When running the instrumented code on the parallel machine, the user can choose between two options: (i) generating a runtime summary with aggregate performance metrics for individual function-call paths or (ii) generating event traces to record individual runtime events. The first option is useful to get an overview of the performance behavior and also to optimize the instrumentation for later trace generation. Since traces tend to become very large, this step is usually recommended before choosing the second option. When tracing is enabled, each process generates a trace file containing records for all its process-local events. After program termination, SCALASCA loads the trace files into main memory and analyzes them in parallel using as many CPUs as have been used for the target application itself. During the analysis, SCALASCA classifies detected pattern instances by category and quantifies their significance for every function-call path and system resource involved. Both results of the trace analysis as well as runtime summaries can be interactively explored in a graphical browser (Figure 5).

To demonstrate the scalability of the parallel analysis approach taken in SCALASCA, we compared it to the sequential approach taken in KOJAK. The ASCI SMG2000 benchmark, a parallel semi-coarsening multigrid solver that uses a complex communication pattern, served as a test case. SMG2000 performs a large number of non-nearest-neighbor point-to-point communication operations and can be considered to be a stress-test for the network subsystems of a machine. Applying a weak scaling strategy, a fixed $64 \times 64 \times 32$ problem size per process with five solver iterations was configured, resulting in a nearly constant application runtime as further CPUs were added.

Figure 2 charts wall-clock execution times for the uninstrumented benchmark and the analyses of trace files generated by an instrumented version with a range of process num-

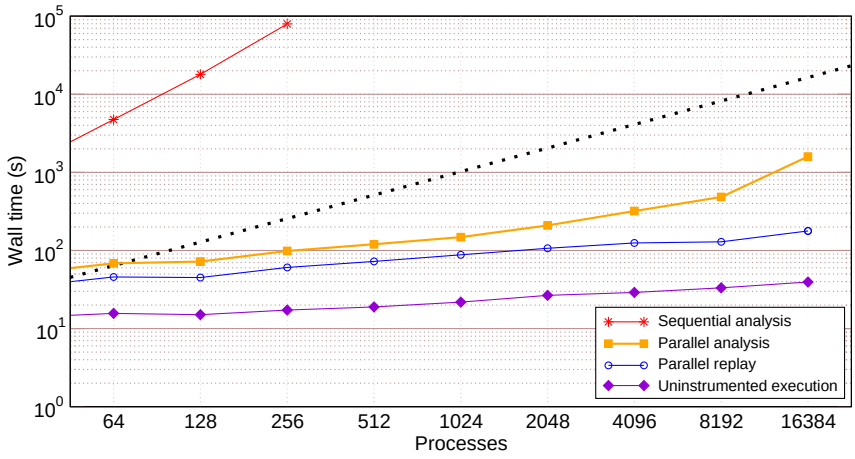

Figure 2. Comparison of application execution, sequential trace analysis (KOJAK) and parallel trace analysis (SCALASCA) times for the ASCI SMG2000 benchmark on up to 16,384 processors of Blue Gene/L. Linear scaling is represented by the bold dotted line.

bers on JUBL. The 8-fold doubling of process numbers necessitates a log–log scale to show the corresponding range of times, particularly for the old sequential analysis (which furthermore becomes impractical for the largest traces). The figure shows the total time needed for the parallel analysis and the time taken by the parallel replay itself without file I/O. It can be seen that the total analysis time including loading the traces into the analyzer never exceeded 30 minutes.

The actual procedure of replaying and analyzing the event traces without file I/O exhibits smooth scaling behavior up to very large configurations. Because of its replay-based nature, the time needed for this part of the analysis depends on the communication behavior of the target application. Since communication is a key factor in the scaling behavior of the target application as well, similarities can be seen in the way both curves evolve as the number of processes increases. Notably, the total time for the new analysis approach is orders of magnitude faster than the sequential analysis based on KOJAK even at modest process counts, making it possible to examine traces at previously intractable scales in a reasonable time.

2.2 Clock Synchronization

Identifying wait states in event traces of message-passing applications requires measuring temporal displacements between concurrent events, although many parallel systems, such as PC clusters, do not provide synchronized hardware clocks. In these cases, linear interpolation techniques can already account for differences in offset and drift, assuming that the drift of an individual processor is not time dependent. However, inaccuracies and drifts varying in time can still cause violations of the logical event ordering that are harmful to the accuracy of our analysis. The *controlled logical clock* algorithm by Rabenseifner[3] compensates for such violations in point-to-point communication by shifting message events in time as much as needed while trying to preserve the length of intervals between lo-

cal events. Our group extended this method to collective communication to enable a more complete correction of realistic message-passing traces. In addition, we designed a parallel version of the algorithm that scales to thousands of application processes.[4]

2.3 Computational Grids

If a single machine does not provide enough CPUs to solve a given problem, multiple independent parallel machines can be combined into a so-called *metacomputer* that appears to the application as a single coherent system. However, achieving satisfactory application performance on such a metacomputer is hard because high latency of inter-machine communication as well as differences in hardware of constituent machines may introduce various types of wait states. Since the analyses offered by SCALASCA could prove especially beneficial in such a grid-like environment, we extended[5] our tool in such a way that it can cope with typical metacomputer limitations, such as a missing global file system and varying network latencies. In addition, we added metacomputing-specific patterns to SCALASCA's pattern base. Using this grid-enabled version of SCALASCA, we were able to remove harmful wait states from an environmental-science application running on the VIOLA[6] grid testbed.[7]

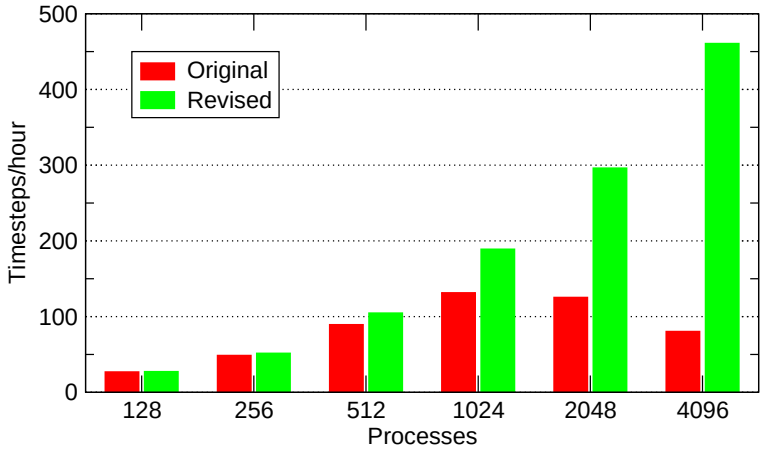

Figure 3. Performance improvement of XNS after removing redundant message traffic.

3 Application Engagement

In addition to engineering performance tools, our group also actively works with application developers to analyze and improve the performance of their codes. A recent example is the XNS fluid dynamics application being developed at the Chair for Computational Analysis of Technical Systems at RWTH Aachen University. XNS can be used for effective simulations of unsteady fluid flows, including microstructured liquids, in situations involving significant deformations of the computational domain, as they occur in blood pumps

deployed to support human heart function. The algorithm is based on finite-element techniques on irregular three-dimensional meshes[8].

Initially, XNS was successfully running on JUBL, however, its scalability was unsatisfactory and resisted scaling beyond approximately 900 processes until the first Jülich Blue Gene Scaling Workshop in December 2006 provided an opportunity for the application developers and our group to start working together. After a first investigation of the solver using basic profiling tools already hinted toward redundant message traffic (i.e., zero-sized messages), a statistical trace analysis using the SCALASCA infrastructure showed that the number of zero-sized messages were rapidly growing with the number of processes employed. Subsequent remediation[9] allowed the application to continue scaling with a fourfold simulation performance improvement at 4,096 processes (Figure 3), demonstrating the benefits of interdisciplinary collaboration between domain scientists and performance analysts.

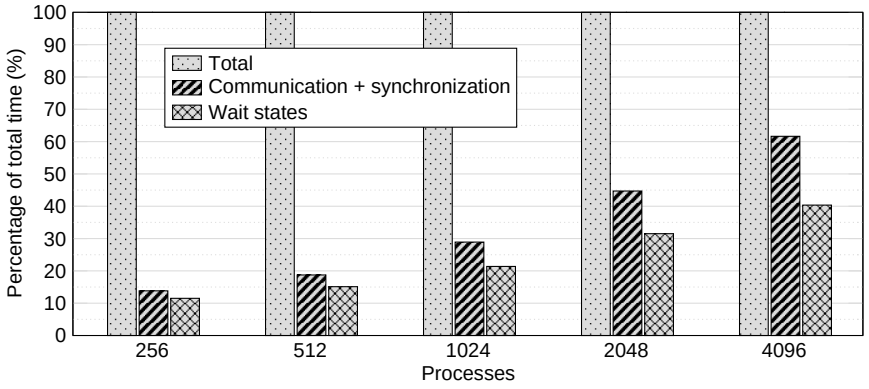

Figure 4. Performance behavior of XNS (tuned version) at a range of scales from 256 to 4096 processes. The bars show the percentage of time the application spent in communication & synchronization operations including associated wait states.

Continuing our investigation, we applied our trace analysis to the tuned version, which exhibits almost perfect scaling behavior on up to 512 processors. However, as the number of processors is raised further, the parallel efficiency continuously degrades, although even at the largest configuration of 4096 processes a clear speedup can still be observed (Figure 3). Beyond 512 processes, as we can see in Figure 4, the communication and synchronization overhead grows steeply. Yet the primary result of our analysis is that the biggest fraction of this overhead is actually waiting time, in the 4096 CPU case amounting to roughly 40 % of the total time (at least 25 % when accounting for intrusion overhead), illustrating that the wait states we are targeting can constitute principal performance problems at larger scales. The SCALASCA display in Figure 5 shows a function-call path that appears to be a major source of the wait states diagnosed during our analysis. Of course, with this finding we hope that the performance of XNS can be further improved.

4 Outlook

Potential causal connections between different wait state instances or related phenomena, such as load imbalance, are currently not covered in a systematic way. Understanding such connections, however, could prove essential for more effective scaling strategies. One approach of establishing links between different wait-state instances would be to define hypotheses and subsequently verify them using a trace-based simulator. Similar in spirit to approaches such as Dimemas[10], the simulator could leverage SCALASCA's parallel archi-

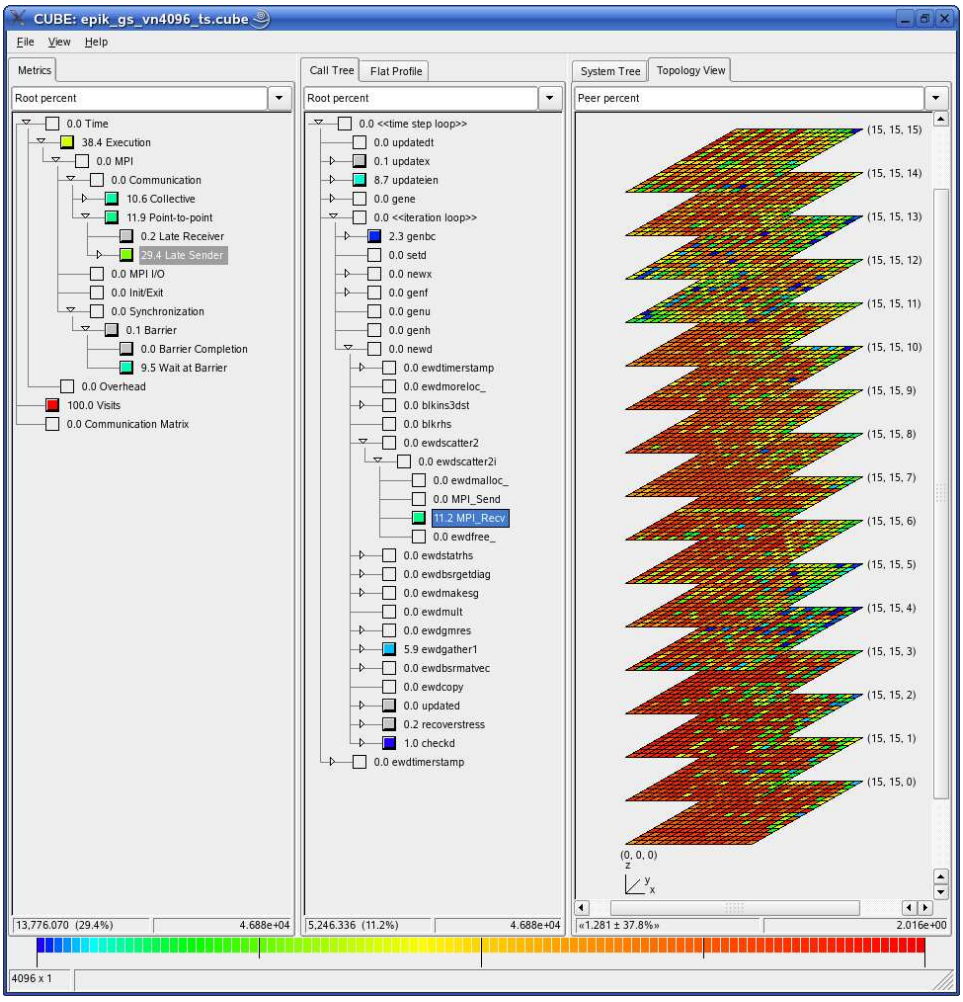

Figure 5. XNS (tuned version) wait states with 4096 processes in `ewdscatter2()`. The middle pane shows the distribution of waiting times across the call tree as the percentage of the time spent in the time step loop. The right pane visualizes how the time incurred by the selected call path is spread across the physical Blue Gene/L torus topology.

tecture to achieve the required scalability.

To institutionalize our major national and international collaborations on HPC programming tools, we recently founded the Helmholtz Virtual Institute - High Productivity Supercomputing together with the Technical University of Dresden, RWTH Aachen University, and the University of Tennessee. The mission of VI-HPS is devoted to the development and deployment of advanced and integrated performance-analysis and error-detection tools for high-performance computing applications. A significant portion of our resources is also assigned to support and training activities. We expect that this initiative will further promote the quality and accelerate the development process of complex simulation codes in science and engineering.

References

1. Markus Geimer, Felix Wolf, Brian J. N. Wylie, and Bernd Mohr, *Scalable Parallel Trace-Based Performance Analysis*, in: Proc. 13th European PVM/MPI Users' Group Meeting, vol. 4192 of *LNCS*, pp. 303–312, Springer, Bonn, Germany, September 2006.
2. F. Wolf and B. Mohr, *Automatic performance analysis of hybrid MPI/OpenMP applications*, Journal of Systems Architecture, **49**, no. 10-11, 421–439, 2003.
3. R. Rabenseifner, *Die geregelte logische Uhr, eine globale Uhr für die tracebasierte Überwachung paralleler Anwendungen*, PhD thesis, Universität Stuttgart, March 2000.
4. D. Becker, R. Rabenseifner, and F. Wolf, *Timestamp Synchronization for Event Traces of Large-Scale Message-Passing Applications*, in: Proc. of the 14th European Parallel Virtual Machine and Message Passing Interface Conference (EuroPVM/MPI), vol. 4757 of *LNCS*, pp. 315–325, Springer, Paris, France, September 2007.
5. D. Becker, F. Wolf, W. Frings, M. Geimer, B. Wylie, and B. Mohr, *Automatic Trace-Based Performance Analysis of Metacomputing Applications*, in: Proc. of the International Parallel & Distributed Processing Symposium (IPDPS), IEEE Computer Society, Long Beach, CA, March 2007.
6. BMBF (Ministry for Education and Research), *Vertically Integrated Optical Testbed for Large Applications in DFN (VIOLA)*, `http://www.viola-testbed.de/`.
7. D. Becker, W. Frings, and F. Wolf, *Performance Evaluation and Optimization of Parallel GRID Computing Applications*, in: Proc. of the 16th Euromicro Workshop on Parallel and Distributed Processing (PDP), Toulouse, France, February 2008, (to appear).
8. M. Behr, D. Arora, O. Coronado, and M. Pasquali, *Models and Finite Element Techniques for Blood Flow Simulation*, International Journal of Computational Fluid Dynamics, **20**, 175–181, 2006.
9. B. J. N. Wylie, M. Geimer, M. Nicolai, and M. Probst, *Performance analysis and tuning of the XNS CFD solver on BlueGene/L*, in: Proc. of the 14th European Parallel Virtual Machine and Message Passing Interface Conference (EuroPVM/MPI), vol. 4757 of *LNCS*, pp. 107–116, Springer, Paris, France, September 2007.
10. J. Labarta, S. Girona, V. Pillet, T. Cortes, and L. Gregoris, *DiP : A Parallel Program Development Environment*, in: Proc. of the 2nd International Euro-Par Conference, vol. 1124 of *LNCS*, pp. 665–674, Springer, Lyon, France, August 1996.