# Performance Optimization for Large Scale Computing: The Scalable VAMPIR Approach

Holger Brunst[1], Manuela Winkler[1], Wolfgang E. Nagel[1], and Hans-Christian Hoppe[2]

[1] Center for High Performance Computing
Dresden University of Technology
D-01062 Dresden, Germany

[2] PALLAS GmbH
Hermühlheimer Str. 10
D-50321 Brühl, Germany

{brunst,nagel,winkler}@zhr.tu-dresden.de,
hch@pallas.com

**Abstract.** Performance optimization remains one of the key issues in parallel computing. Many parallel applications do not benefit from the continually increasing peak performance of todays massively parallel computers, mainly because they have not been designed to operate efficiently on the 1000s of processors of todays top of the range systems. Conventional performance analysis is typically restricted to accumulated data on such large systems, severely limiting its use when dealing with real-world performance bottlenecks. Event based performance analysis can give the detailed insight required, but has to deal with extreme amounts of data, severely limiting its scalability. In this paper, we present an approach for scalable event-driven performance analysis that combines proven tool technology with novel concepts for hierarchical data layout and visualization. This evolutionary approach is being validated by implementing extensions to the performance analysis tool Vampir.
**Keywords:** performance visualization, application tuning, massively parallel programming, scalability, message passing, multi-threading.

## 1 Introduction

Todays microprocessor technology provides powerful basic components normally targeted at the workstation market. Theoretically, this single processor technology enables an enormous peak performance when joined to a multiprocessor system consisting of 5000 or more processors. The multiplied peak performance gives the impression to most people that performance – and moreover performance optimization – is no longer an important issue. This assumption contradicts reality [12,20] when dealing with parallel applications. Only a few highly specialized and optimized scientific programs scale well on parallel machines that provide a couple of thousands processors. In order to widen the range of applications that benefit from such powerful computational resources, performance analysis and optimization is an essential. The situation is such that performance analysis and optimization for large scale computing itself poses crucial difficulties. In the following we will present an analysis/optimization approach that combines existing tool technology with new hierarchical concepts. Starting with a brief summary of evolving computer architectures and related work, we will present the concepts and a prototyped tool extension of our scalable performance analysis approach.

## 2 Evolving Architectures

In the near future, parallel computers with a shared memory interface and up to 32 or more CPU nodes (SMP Systems) are expected to become the standard system for scientific numerical simulation as well as for industrial environments. In contrast to the homogeneous MPP systems that were more or less exclusively designed and manufactured for the scientific community, SMP systems mostly offer an excellent price/performance ratio as they are typically manufactured from standard components off the shelf (COTS). The comparably low price also makes SMP systems affordable for commercial applications and thus widens the community of parallel application developers needing tool support.

In cases where outstanding computational performance is needed, SMP systems can be coupled to clusters interconnected by a dedicated high performance network. Projects like ASCI [1,3,4] funded by the Department of Energy or the LINUX based Beowulf [2,19] show that the coupling of relatively small SMP systems to a huge parallel computer consisting of approximately 10000 CPUs is feasible with current hardware technology. On the other hand the system and application software running on this type of machine still presents a lot of unanswered questions. It is a known fact that the different communication layers in such a hybrid system make it highly complicated to develop parallel applications performing well. Within this scope, *scalability* is one of the key issues.

Dealing with scalability problems of parallel applications in most cases requires performance analysis and optimization technology that is capable of giving detailed insight into an application's runtime behavior. Accumulative trace data analysis cannot fulfill this requirement as it typically does not explain the cause of performance bottlenecks in detail. We experienced that program event tracing is required for the location and solution of the majority of program scalability problems. Being aware of this implies that scalability also needs to be introduced into event based performance analysis and optimization tools as tracing the behavior of 1000-10000 processing entities generates an enormous amount of performance data. Today, the optimization tools are quite limited with respect to the number of processing entities and the amount of trace data that can be handled efficiently. The next section will give a brief summary on the current state of the art of performance tool development and its limits in order to provide a better understanding of our activities related to large scale performance analysis.

## 3 Current Analysis / Optimization Tools

Large machines with more than 1000 processing entities produce an enormous amount of trace data during a tracing session. Finding performance bottlenecks and their origin requires appropriate tools that can handle these GBytes of information efficiently. This implies data reduction, selection, archiving, and visualization on a large scale. Most of todays tools [6,22] cannot yet fulfill this task. We will now classify todays tools regarding their task, capabilities, and drawbacks.

### 3.1 Tools for Accumulated Data Analysis

This type of tool lists measures - typically in a long table - like summarized function dwell, communication rates, performance registers, etc. Representatives of this category are *prof, gprof, iostat, vmstat* known from the UNIX world. This type of data presentation is disadvantageous when dealing with large, complex, parallel applications. Nevertheless, it can help to identify a general performance problem. Once detected, more complex tools are needed to find out what's going on 'inside' the application.

Based on the same information acquisition methods as the text based tools, tools like *Xprofiler* [23] from IBM, *APPRENTICE* [8] from Cray Research or *Speedshop* [21] from SGI have standard spreadsheet constructs to graphically visualize accumulated information. Although this allows for a quicker overview of the total application's behavior, the identification of a problem's cause remains unresolved in most cases.

### 3.2 Tools for Event Trace Analysis

The tools that can directly handle program traces are typically capable of showing accumulated performance measures in all sorts of diagrams for arbitrary time intervals of a program run. This is already a major difference to the above mentioned tools which typically only show a fixed time interval which in most cases is the overall program duration. In addition to this, they offer a detailed insight into a program's runtime behavior by means of so-called timeline diagrams. These diagrams show the parallel program's states for an arbitrary period of time. Performance problems that are related to imbalanced code or bad synchronization can easily be detected with the latter as they cause irregular patterns. When it comes to performance bottlenecks caused by bad cache access or bad usage of the floating point units, the timelines are less useful. Similar performance monitors can be found in almost every CPU now, and would be very helpful when combined with these CPU timeline diagrams. Representatives of this tool category are *Jumpshot* [13], *Vampir* [17], and *Xpvm* [24].

## 4 New Scalability Issues

In the scope of ASCI, detailed program analysis over a long period of time (not just a few seconds as is possible today) has been identified as an important requirement. Independently of the method of data acquisition, this implies an enormous amount of data to be generated, archived, edited and analyzed. The data extent is expected to be more than one TByte for a moderate size (1000 CPUs, 1 hour runtime) program run, where 0.5 MByte/s trace data per CPU is a rule of thumb observed in our daily work if pretty much every function call is traced. Obviously, this amount of data is far too large to be handled by an analysis sub-system significantly smaller than the parallel master platform. Dynamic instrumentation as possible with DYNINST [7], which was developed in the scope of the Paradyn [16] project, connected to a visualization tool would allow to deselect uninteresting code parts during runtime and thereby reduce the trace data to a moderate size of 10-100 GByte/h.

The hierarchical design of large scale parallel systems will have an impact on the way this data is stored. Nobody would really expect a single trace file as the outcome of a one hour tracing session on 1000 CPUs. Distributed files each representing a single CPU or a cluster of CPUs are likely to be used as they allow for best distributed I/O performance. Efficient access to these files forms the basis of our work and will be discussed in the following.

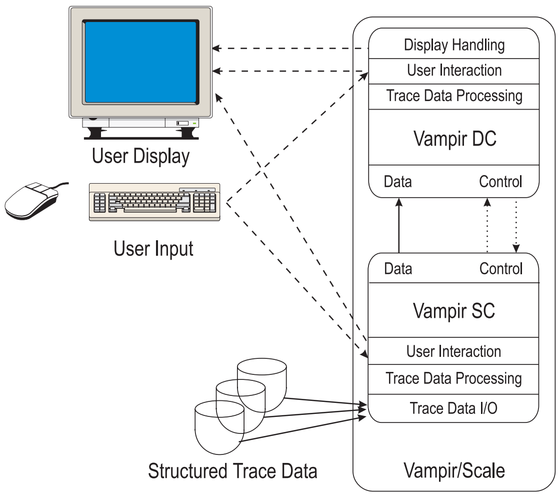


**Fig. 1.** A Scalable Optimization Tool Layout

## 4.1 Distributed Trace Data Preparation

The scalability requirements that arouse from the enormous amount of trace data to be archived, synchronized, edited, visualized and analyzed require considerable effort and time when incorporated into a fully new tool. As for smaller systems there are already a couple of good tools available on the market; we recommend a parallel data layer extension that acts in between the trace data base and the analysis tool. This data layer is responsible for all time consuming operations like for example:

- post-mortem synchronization of asynchronous trace data
- creation of statistics
- filtering of certain events, messages, CPUs, clusters
- summarizing the event history for long time periods

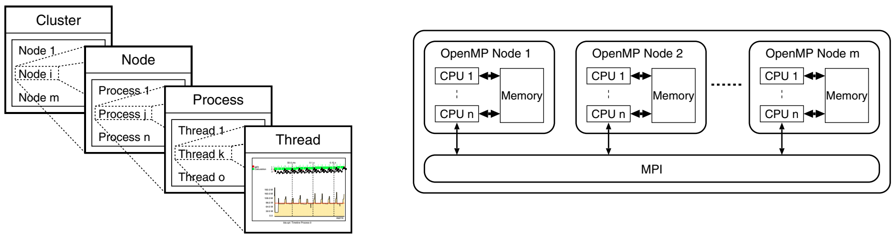
Outsourcing these activities from the visualization tool (which is typically a sequential program) allows for fast trace data access with limited changes to the user interface of the visualization tool. Our major goal was to keep the existing interface constant and to add scalability functionality in a natural way. Figure 1 illustrates the structure of this approach for the performance analysis tool Vampir.

The trace data can be distributed among several files, each one storing a ‘frame’ of the execution data as pioneered by jumpshot [13]. Frames can correspond to a single CPU, a cluster of CPUs, a part of the execution or a particular level of detail in the hierarchy (see 4.2). The frames belonging to a single execution are tied together by means of an index file. In addition to the frame references, the index file contains statistics for each frame and a ‘performance thumbnail’ which summarizes the performance data over time.

The analysis tool reads the index file first, and displays the statistics and the rough performance thumbnail for each frame. The user can then choose a frame to look at in more detail, and the frame file will be automatically located, loaded and displayed. Navigation across frame boundaries will be transparent.

### 4.2 Hierarchical Visualization

When it comes to visualization of trace data, dealing with 1000-10000 CPUs poses additional challenges. The limited space and resolution of a computer display allows the visualization of at most 200-300 processing entities at any one time. Acting beyond this limit seems to be useless especially when scrolling facilities are involved, as users become confused by too much data. Therefore, we propose a hierarchical approach where, based on the distributed system’s architecture, the user can navigate through the trace data on different levels of abstraction. This works for both event trace oriented displays (timelines) and statistic displays for accumulated data. The hierarchy in figure 2a allows the visualization of at least 10000 processing entities where a maximum of 200 independent objects need ever be drawn simultaneously on the same display. This prediction is based on a hypothetical cluster consisting of 64 SMP nodes with 200 processes each (and no threads). As our model provides 3 layers which can each hold at maximum 200 objects, almost any cluster configuration can be mapped in an appropriate way.



(a) Access Hierarchy (b) SMP Cluster with MPI/OpenMP

**Fig. 2.** Hierarchical View on Event Trace Data

## 4.3 Performance Monitors

Current processor architectures usually offer performance monitor functionality, mostly in the form of special registers that contain various performance metrics like the number of floating-point operations, cash misses etc. The use of these registers is limited because there is no relation to the program structure: an application programmer typically does not know which parts of the application actually cause bad cache behavior. To help here, the use of profiling techniques is necessary: cyclic sampling and association of the sampled values to code locations (à la prof) or combination of sampling with subroutine entry/exit events (à la gprof) will provide the insight into which parts of a program need further optimization.

For the optimization of large scale applications, performance monitors gain additional significance. As stated above, dealing with large amounts of performance data requires multiple abstraction layers. While the lower layers are expected to provide direct access to the event data by means of standard timeline and statistic views, the upper layers must provide aggregated event information. The aggregation needs to be done in a way that provides clues to performance bottlenecks caused in lower layers. Measures like cache performance, floating point performance, communication volume, etc. turned out to have good summarizing qualities with respect to activities on lower layers. We suggest introducing a chart display with $n$ graphs representing $n$ ($n < 64$) nodes as the entry point to a scalable performance optimization approach.

## 4.4 Hybrid Programming Paradigms: MPI + OpenMP

Parallel computers are typically programmed by using a message passing programming paradigm (MPI [5,15] PVM [10], etc.) or a shared memory programming paradigm (OpenMP[9,14,18] Multi Threading, etc.). Large machines are now often realized as clusters of SMPs. The hybrid nature of these systems is leading to applications using both paradigms at the same time to gain best performance. Figure 2b illustrates how MPI and OpenMP would cooperate on a SMP cluster consisting of $m$ nodes with $n$ CPUs each.

So far, most tools support either message passing or shared memory programming. For the hybrid programming paradigm, a tool supporting both models equally well is highly desirable. The combination of well established existing tools for both realms by means of a common interface can save development effort and spare the users the inconvenience of learning a completely new tool. In this spirit, Vampir and the GuideView tool by KAI [11] will be combined to support analysis of hybrid MPI/OpenMP applications. First results will be illustrated in the final paper.

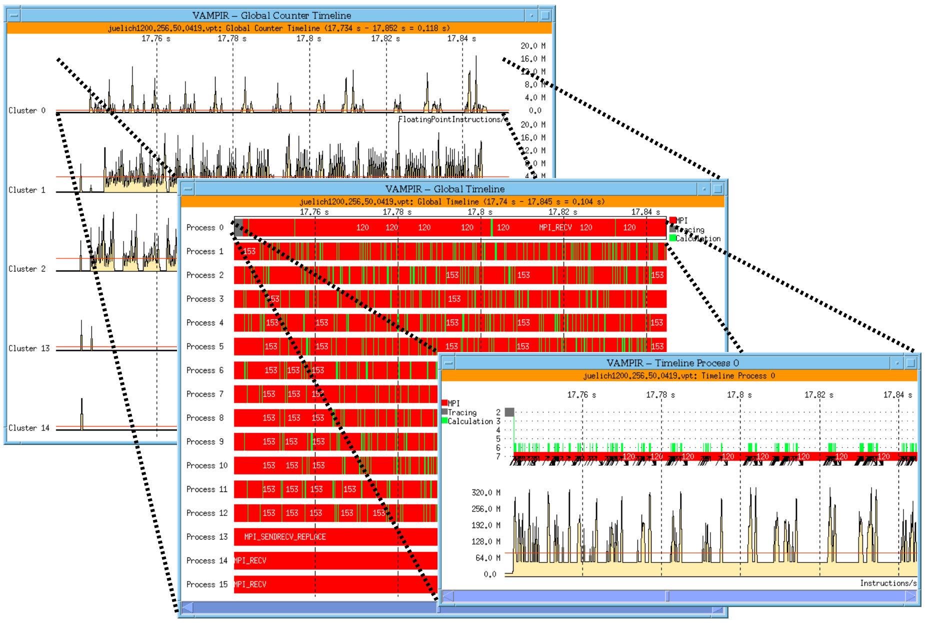

VAMPIR – Global Counter Timeline
juelich1200.256.50.0419.vpt: Global Counter Timeline (17.734 s - 17.852 s = 0.118 s)
Cluster 0
Cluster 1
Cluster 2
Cluster 13
Cluster 14
FloatingPointInstructions/s
VAMPIR – Global Timeline
juelich1200.256.50.0419.vpt: Global Timeline (17.74 s - 17.845 s = 0.104 s)
Process 0
Process 15
MPI_RECV
MPI_SENDRECV_REPLACE
MPI
Tracing
Calculation
VAMPIR – Timeline Process 0
juelich1200.256.50.0419.vpt: Timeline Process 0
Instructions/s

**Fig. 3.** Hierarchical View on 256 CPUs

## 5 Vampir - Scalability in Practice

The previous section introduced new ideas for performance analysis tools targeted towards the optimization of next generation applications. We already depend on such tools as part of our daily work, which is why we have put quite some effort into developing a prototype which implements many of the ideas mentioned above. The prototype is based on Vampir 2.5 [17], which is a commercial tool for performance analysis and visualization accepted in the field.

Based on real applications that were tuned at our center, we will now present one possible realization of a scalable performance analysis tool.

### 5.1 Navigation on GBytes of Trace Data

In section 4.2 we introduced the idea of a hierarchical view on event trace data. The reason for this was the large amount of event trace data generated by a large scale application running on hundreds of processing nodes over a longer period of time. Figure 3 illustrates the impact of this approach on trace data navigation for a test case generated on 256 (16 × 16) processors. The leftmost window depicts MFLOPS rates on the cluster level[1] for a time period of 0.1 s which was selected via a time based zoom mechanism. From this starting point we decided to get a closer view on cluster 1 and its processes. A simple mouse click opens up the window in the middle which depicts the state[2] changes for the processes 0 - 15. Further details to process 0 are available in the bottom right window of figure 3 which shows a detailed function call stack combined with a performance monitor showing the instructions per second rate of process 0. This type of hierarchical trace file navigation permits an intuitive access to trace files holding data for hundreds of clustered CPUs.

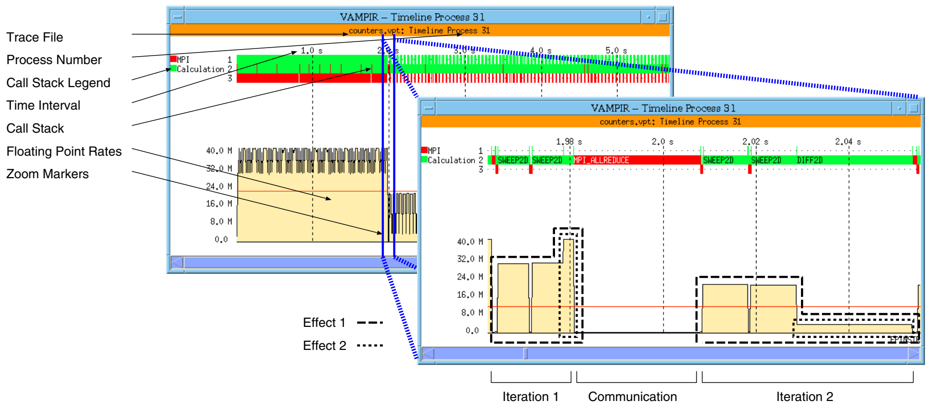


**Fig. 4.** Performance Monitor Combined with Call Stack

### 5.2 Finding Hot Spots by Means of Performance Monitors

Performance monitors can be of much use when dealing with performance bottlenecks of unknown origin. The following example was taken from a performance

[1] Cluster 3 - 12 were filtered to adapt to the small figure size in the paper

[2] Depending on the type of instrumentation, a state is a certain function or code block

optimization session, which we recently carried out on one of our customer's programs. For some reason, his parallel program started off performing as expected but suffered a serious performance decrease in its MFLOPS rate after two seconds.

Figure 4 shows the program's call stack combined with the MFLOPS rates for a representative CPU over the time period of 6 seconds with a close-up of the time interval the program behavior changes. We see two similar program iterations separated by a communication step. The first one is twice as fast as the second one. We can also see that the amount of work carried out in both iterations is identical, as their integral surfaces are the same (effect 1). A third aspect can be found in the finalizing part (function DIFF2D) of each iteration. Obviously the major problem resides here, as the second iteration is almost 10 times slower than the first one (effect 2). We eventually discovered that the whole problem was caused by a simple buffer size limit inside the program which lead to repeated date re-fetching.

## 6 Conclusion

This paper has presented concepts for scalable event-based performance analysis on SMP clusters containing 1000s of processing entities. The key issues are the distributed storage and handling of event traces, the hierarchical analysis and visualization of the trace data and the use of performance monitor data to guide detailed event based analysis. The existing implementation of parts of these within the Vampir framework has been discussed. Further extensions to Vampir that are currently being worked on will serve as a proof of concept, demonstrating the benefits of event based performance analysis to real-world users with large applications on the upcoming huge SMP systems.

## References

[1] Accelerated Strategic Computing Initiative (ASCI).
http://www.llnl.gov/asci.

[2] D. J. Becker, T. Sterling, D. Saverese, J. E. Dorband, U. A. Ranawak, and C. V. Packer. Beowulf: A Parallel Workstation for Scientific Computation. In *Proceedings, International Conference on Parallel Processing*, 1995.
http://www.beofulf.org.

[3] Blue Mountain ASCI Machine.
http://w10.lanl.gov/asci/bluemtn/bluemtn.html.

[4] Blue Pacific ASCI Machine.
http://www.llnl.gov/asci/platforms/bluepac.

[5] S. Bova, C. Breshears, H. Gabb, R. Eigenmann, G. Gaertner, B. Kuhn, B. Magro, and S. Salvini. Parallel programming with message passing and directives. *SIAM News*, 11 1999.

[6] S. Browne, J. Dongarra, and K. London. Review of performance analysis tools for mpi parallel programs.
http://www.cs.utk.edu/~browne/perftools-review.

[7] B. Buck and J. K. Hollingsworth. An API for Runtime Code Patching. Technical report, Computer Science Department, University of Maryland, College Park, MD 20742 USA, 1998.
http://www.cs.umd.edu/projects/dyninstAPI.

[8] Cray Research. *Introducing the MPP Apprentice Tool*, IN-2511 3.0 edition, 1997.

[9] D. Dent, G. Mozdzynski, D. Salmond, and B. Carruthers. Implementation and performance of OpenMP in ECWMF's IFS code. In *Proc. of the 5th SGI/CRAY MPP-Workshop*, Bologna, 1999.
http://www.cineca.it/mpp-workshop/abstract/bcarruthers.htm.

[10] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM: Parallel Virtual Machine*. The MIT Press, 1994.
http://www.epm.ornl.gov/pvm.

[11] The GuideView performance analysis tool.
http://www.kai.com.

[12] F. Hoßfeld and W. E. Nagel. Per aspera ad astra: On the way to parallel processing. In H.-W. Meuer, editor, *Anwendungen, Architekturen, Trends, FOKUS Praxis Informationen und Kommunikation*, volume 13, pages 246–259, Munich, 1995. K.G. Saur.

[13] The Jumpshot performance analysis tool.
http://www-unix.mcs.anl.gov/mpi/mpich.

[14] Lund Institute of Technology. *Proceedings of EWOMP'99, 1st European Workshop on OpenMP*, 1999.

[15] Message Passing Interface Forum. *MPI-2: Extensions to the Message-Passing Interface*, August 1997.
http://www.mpi-forum.org/index.html.

[16] B. P. Miller, M. D. Callaghan, J. M. Cargille, J. K. Hollingsworth, R. B. Irvin, K. L. Karavanic, K. Kunchithapadam, and T. Newhall. The Paradyn Parallel Performance Measurement Tools. *IEEE Computer*, 28(11):37–46, November 1995.
http://www.cs.wisc.edu/˜paradyn.

[17] W. E. Nagel, A. Arnold, M. Weber, H.-C. Hoppe, and K. Solchenbach. VAMPIR: Visualization and Analysis of MPI Resources. *Supercomputer 63*, XII(1):69–80, January 1996.
http://www.pallas.de/pages/vampir.htm.

[18] *Tutorial on OpenMP Parallel Programming*, 1998.
http://www.openmp.org.

[19] D. Ridge, D. Becker, P. Merkey, and T. Sterling. Beowulf: Harnessing the Power of Parallelism in a Pile-of-PCs. In *Proceedings, IEEE Aerospace*, 1997.
http://www.beofulf.org.

[20] L. Smarr. Special issue on computational infrastructure: Toward the 21st century. *Comm. ACM*, 40(11):28–94, 11 1997.

[21] The Speedshop performance analysis tool.
http://www.sgi.com.

[22] Pointers to tools, modules, APIs and documents related to parallel performance analysis.
http://www.fz-juelich.de/apart/wp3/modmain.html.

[23] The Xprofiler performance analysis tool.
http://www.ibm.com.

[24] The XPVM performance analysis tool.
http://www.netlib.org/utk/icl/xpvm/xpvm.html.