Performance Evaluation and Optimization of Metacomputing Applications

Daniel Becker[1,2], Wolfgang Frings[1] and Felix Wolf[1,2]
{d.becker, w.frings, f.wolf} @fz-juelich.de

[1] Forschungszentrum Jülich, Jülich Supercomputing Centre (JSC), 52425 Jülich, Germany

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

Abstract

The combination of independent and potentially heterogeneous parallel machines creates a powerful metacomputer. Such a metacomputer can be used to run a single parallel application if a single machine does not provide enough CPUs. However, achieving satisfactory application performance on such a metacomputer is difficult since instances of grid-related as well as non grid-related performance properties may introduce various wait states during communication and synchronization. In our earlier work, we have introduced an extension to the SCALASCA tool set for recording event traces of metacomputing applications and searching them automatically for patterns of inefficient behavior related to wide-area communication. Here, we show how this extension in combination with statistical analyses and time-line visualization provided by VAMPIR can be applied to evaluate and optimize the performance of a multi-physics production code running on a heterogeneous and geographically dispersed metacomputer.

Keywords: Performance tools, grid computing, metacomputing, event tracing.

1 Introduction

The solution of critical numerical problems may require more processing power and memory capacity than is available on a single parallel machine. Often, coupling multiple independent parallel machines (i.e., metahosts) to form a more powerful metacomputer is the only method to increase the available resources for a single application.

However, although applications can benefit from the increased parallelism offered by a metacomputer, achieving satisfactory application performance is difficult. Algorithm design has to adapt to hierarchies of latencies and bandwidths in addition to the heterogeneous hardware architectures found in such environments. Hence, performance optimization is a crucial but non-trivial task that needs adequate tool support. A frequent problem that needs special attention are wait states that occur when the speed at which the computation progresses varies between metahosts or when message transfers are delayed by high network latency.

In our earlier work [1], we have shown that automatic pattern search in event traces is a suitable method to identify wait states that appear as a result of using a metacomputer consisting of multiple geographically dispersed metahosts. There, we have extended the trace-analysis tool SCALASCA [10] so that it can be used in metacomputing environments. Challenges addressed by our extension include performing the pattern analysis in the absence of a shared file system between metahosts, the synchronization of time stamps in hierarchical networks, and the definition of grid-specific patterns that target communication and synchronization across metahost boundaries.

In this paper, we demonstrate that not only performance evaluation but also performance optimization of applications running on a heterogeneous and geographically dispersed metacomputer are feasible. Using the grid-enabled tracing and analysis capabilities of the SCALASCA tool set, we determine relevant performance properties and demonstrate how this information can be used to significantly improve the performance of MetaTrace [5], a grid-enabled multi-physics application that simulates the transport of pollutants in groundwater.

Starting point of our study are event traces generated using the enhanced SCALASCA measurement infrastructure. First, we evaluate how the bandwidth and latency requirements of our application are met by the wide-area connection in our grid testbed using the statistical trace-analysis capabilities of VAMPIR [8]. Second, we show how the localization, classification, and quantification of wait states performed by the SCALASCA trace analyzer assists us in eliminating a major fraction of waiting times, leading to

a significant improvement of the overall performance. Finally, by running the application on a homogeneous cluster and comparing the results with those obtained on the metacomputer, we verify that some of the performance problems we have identified are indeed the consequence of using a metacomputer.

The outline of this article is as follows: We start in Section 2 with a short description of VIOLA, the metacomputer testbed we used for our experiments, and the application MetaTrace. In Section 3, we describe the methods and tools used during the optimization process. Then, in Section 4, we summarize our network analysis followed by an outline of the incremental optimization process. Finally in Section 5, we conclude our paper.

2 The VIOLA metacomputer

VIOLA [4] is a project funded by the German Ministry for Education and Research, which provides a testbed for advanced optical network technology. A major focus is the enhancement and test of advanced grid applications.

2.1 Network topology and hardware architecture

The network behind the VIOLA grid consists of a 10 Gbps backbone network with connections to workstations and compute clusters located at various sites in Germany including Sankt Augustin, Jülich, Bonn, Nürnberg, and Erlangen. The nodes of the connected compute clusters are linked to the backbone with 1 Gbps adapters. The high bandwidth of the backbone can only be used if the data transmission between the clusters is done in parallel.

These components form a very heterogeneous metacomputer layout with a hierarchy of different network latencies and varying characteristics of the compute clusters, which differ with respect to their operating systems (different versions of Linux) and compilers. It can be expected that the high latency of inter-machine communication as well as the heterogeneous hardware may adversely affect application performance.

2.2 Middleware

Running a parallel application on such a metacomputer needs middleware components for application startup and a wide-area communication library for the transfer of data between application processes residing on geographically dispersed metahosts. The middleware interacts with local resource managers to co-schedule jobs on different clusters. The communication library should support transparent high-bandwidth and low-latency message transfers between all nodes of the attached clusters.

The co-scheduling of jobs on different clusters in the VIOLA grid is managed by the grid middleware UNICORE [7] which has been enhanced by adding a meta-scheduler for the simultaneous allocation of compute and network resources. Bierbaum et al. [2] describe this UNICORE-based infrastructure supporting the co-allocation of metacomputing resources in more detail, with special emphasis on the intricate task of coordinating network allocation with application startup. This infrastructure provides seamless access to distributed grid resources through a graphical user interface, which is depicted in Figure 1.

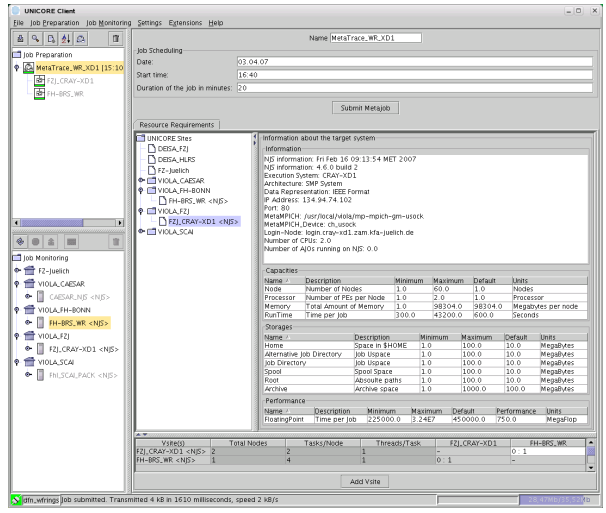


Figure 1. Meta-scheduler and the UNICORE graphical user interface.

Moreover, VIOLA uses MetaMPICH [3], the MPICH-based MPI-implementation developed at RWTH Aachen University, to establish direct connections to the external network from each node. MetaMPICH supports these direct connections through a multi-device architecture that allows external communication within the VIOLA-testbed with the maximum bandwidth of 1 Gbps per node across the wide-area network without the involvement of dedicated router processes.

2.3 Applications

Applications on the VIOLA grid cover various research disciplines including environmental research, the design of complex technological systems like biosensors and crystal growth for microchip wafer production, and structural mechanics in engineering.

MetaTrace, one of the applications running on the VIOLA-testbed, simulates the transport of pollutants in groundwater. MetaTrace is a combination of two parallel simulation submodels, Trace and Partrace. Whereas

Trace simulates water flow in porous media, Partrace computes the transport of solutes in this water flow. Trace applies a three-dimensional domain decomposition (in our case $192 \times 32 \times 32$ m^3) with nearest-neighbor communication, whereas Partrace tracks individual particles. For simulating pollutant transport in non-steady flows, the simultaneous execution of both submodels is crucial. MetaTrace couples the two submodels through a parallel connection between the two submodels. This connection is mainly used in one direction for the transfer of the distributed three-dimensional velocity field from Trace to Partrace whenever Trace completes a simulation step. The unidirectional communication scheme makes MetaTrace suitable to run efficiently on a computational grid. Running each submodel on a single metahost allows the internal communication to benefit from the low-latency network whereas only synchronization as well as data exchange between the two submodels have to use the high-latency network. The unidirectional and low-frequency communication between the two submodels is done synchronously over the VIOLA backbone network through the node-local network adapters. After receiving the data, Partrace replicates the received velocity field on each node by synchronously distributing it across all Partrace nodes using a systolic loop.

3 Performance measurement and analysis

In this section, we illustrate our performance measurement and analysis method used to optimize the application. We focus on the SCALASCA tool set and its recent extension that can be used to analyze metacomputing applications. In addition, we briefly describe the VAMPIR graphical trace browser.

Often, parallel applications which are free of computational errors need to be optimized. This requires the information which component of the program is responsible for what kind of inefficient behavior. Performance analysis is the process of identifying those parts, exploring the reasons for their unsatisfactory performance, and quantifying their overall influence. To do this, performance data are mapped onto program entities. A developer can now investigate application's runtime behavior using software tools. Thus, the developer is enabled to understand the performance behavior of his application. The process of gathering performance data is called performance measurement and forms the basis for subsequent analysis.

Event tracing is a technique for post-mortem performance analysis of parallel applications. Time-stamped events, such as entering a function or sending a message, are recorded at runtime and analyzed afterwards with the help of software tools. The information recorded for an event includes at least a time stamp, the location (e.g., the process or node) where the event happened and the event type. Depending on the type, additional information may be supplied, such as the function identifier for function call events. Message event records typically contain details about the message they refer to (e.g., the source or destination location and message tag).

Graphical trace browsers, such as VAMPIR, allow the fine-grained, manual investigation of parallel performance behavior using a zoomable time-line display and provide statistical summaries of communication behavior. However, in view of the large amounts of data generated on contemporary parallel machines, the depth and coverage of the visual analysis offered by a browser is limited as soon as it targets more complex patterns not included in the statistics generated by such tools.

By contrast, the trace analyzer of the SCALASCA tool set [6] automatically searches event traces for patterns of inefficient behavior, classifies detected instances by category, and quantifies the associated performance penalty. To do this efficiently at larger scales and also to circumvent the obstacles arising from the absence of a shared file system in grid environments, the traces are analyzed in parallel by replaying the original communication using the same hardware configuration and the same number of CPUs as have been used to execute the target application itself.

For our experiments presented in Section 4, we used the SCALASCA tool set which has been extended to support the automatic performance analysis of metacomputing applications. Goal of these extensions was (i) to enable automatic trace analysis on a metacomputer and (ii) to help identify metacomputing-specific performance problems in applications. On a technical level, capabilities have been added to identify the metahost a process is running on, to synchronize time stamps across a hierarchical network with different latencies, and to analyze traces in the absence of a shared file system. In addition, special metacomputing patterns have been added to the existing pattern base. The interested reader can find a more detailed description in [1].

4 Performance evaluation and optimization

In this section, we present experimental results that show the feasibility of evaluating relevant performance metrics and of optimizing the performance of a real-world production code in metacomputing environments.

4.1 Experiment description

To demonstrate that performance measurement in combination with performance analysis can be used to identify inefficient performance behavior, we analyzed the aforementioned multi-physics application MetaTrace. For our experiments we used the VIOLA sites at FH Bonn-Rhein-Sieg Sankt Augustin (FH-BRS) and at Forschungszentrum

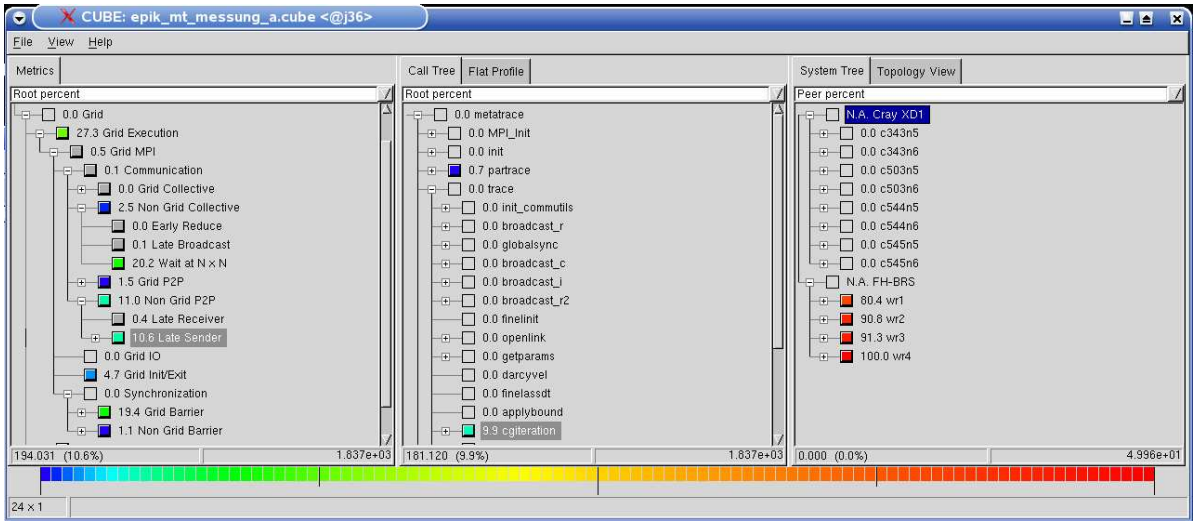


Figure 2. Analysis results of metacomputer experiment: Late Sender problem inside Trace function cgiteration() at FH-BRS.

Jülich (FZJ) to execute MetaTrace. That is, the metacomputer used for our measurements includes two metahosts, one at each site:

- A PC Linux cluster with 6 4-way AMD Opteron SMP nodes at 2 GHz with a usock over Myrinet interconnect located at FH-BRS.
- A Cray XD1 Linux cluster with 60 2-way AMD Opteron SMP nodes at 2.2 GHz with a usock over RapidArray interconnect located at FZJ.

In our first experiment, Partrace ran at FZJ, while Trace was executed at FH-BRS. To enable a comparison between a grid environment and a homogeneous cluster we performed a second experiment on an IBM AIX POWER 4+ cluster at Forschungszentrum Jülich. In both cases we used 24 processes in total.The detailed configurations of these experiments are listed in Table 1.

4.2 Experimental results

To generate the trace data needed to investigate the performance behavior, the instrumented program was executed on the VIOLA grid. MetaTrace was instrumented by manually inserting directives which were automatically translated into appropriate SCALASCA measurement API calls by a preprocessor. During the program run, the trace files were generated in the EPILOG format. The trace data were analyzed by SCALASCA's parallel analyzer to generate a profile of high-level performance properties. From the analysis results we derived our decisions which optimization we should apply to the application. For fine-grained visual trace analysis, the EPILOG event trace was converted to the OTF format.

Table 1. Detailed configurations of the two-metahost and one-metahost experiments.

	Experiment 1	**Experiment 2**
Partrace	FZJ: 8 nodes 1 processes/node	IBM AIX POWER 4+: 1 node 8 processes/node
Trace	FH-BRS: 4 nodes 4 processes/node	IBM AIX POWER 4+: 1 node 16 processes/node

4.2.1 Network characteristics of the VIOLA-testbed

For our initial performance measurement we used MetaTrace in the configuration described in Section 2. After applying an OTF converter to our EPILOG traces, we were able to determine several performance metrics of the VIOLA-testbed using VAMPIR's statistical summary functionality.

Partrace and Trace simulate the spread of groundwater pollution collaboratively, and thus, the two submodels exchange simulation data at synchronization points across the external network. That is, the total amount of data sent across the wide area network represents the use of VIOLA's infrastructure. Table 2 shows the total amount of data transferred across the internal and external network within the

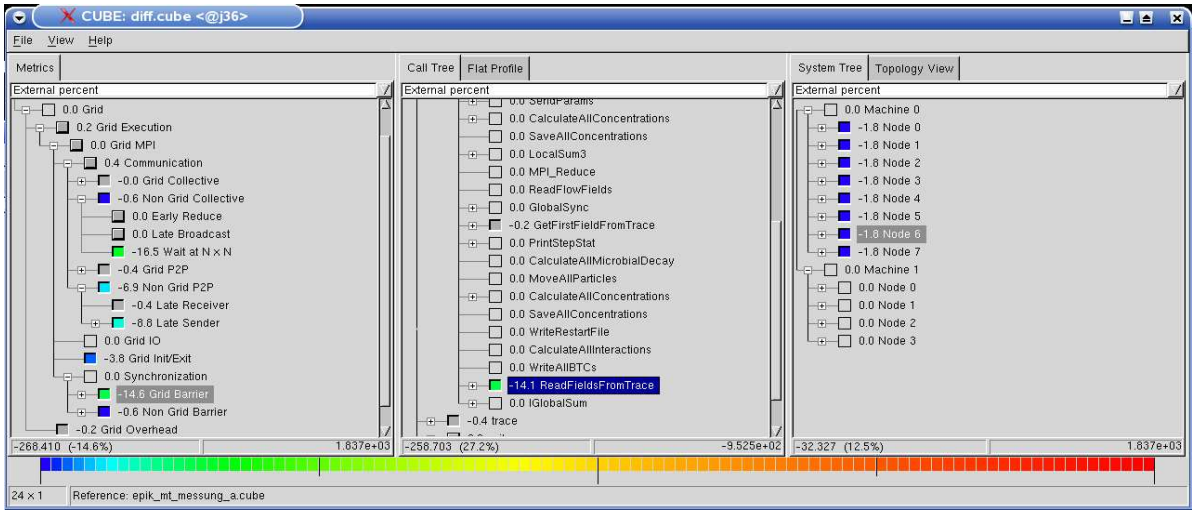


Figure 3. Analysis results of metacomputer experiment: Difference experiment obtained by subtracting the original version from the optimized version.

VIOLA-testbed. As can be seen in our experiment, Trace at FH-BRS sent in total 547.8 MByte of data across the external network to Partrace at FZJ. Thus, each Partrace process received in average 68.5 MByte of data from Trace across the external network. It should be mentioned that Partrace sent only minor control and status information back to Trace.

Table 2. Total amount of data transferred across the internal and external network in the VIOLA-testbed in MByte.

	FZJ	FH-BRS
FZJ	4320.0	0.0
FH-BRS	**547.8**	1120.0

To clarify whether the data transfer used the full bandwidth offered by VIOLA's infrastructure, we determined the maximum data rate of the internal and external communication as well. Our measurements summarized in Table 3 show a maximum data transfer rate of 47.3 MByte/s between two corresponding processes at FH-BRS and FZJ. Each node at FH-BRS used a network link with the maximum bandwidth of 1 Gbps. Since we assigned 16 processes to Trace and 8 processes to Partrace, only two Trace processes on the same 4-way node could communicate in parallel with two corresponding Partrace processes during the data exchange. Given that these two Trace processes shared a single network link, each of the two could use half of the bandwidth (62.5 MByte/s per process) offered by VIOLA's network links, and thus, our measurements show that MetaTrace almost fully utilized the VIOLA network bandwidth.

Table 3. Maximum P2P communication rate of the internal and external communication in the VIOLA-testbed in MByte/s.

	FZJ	FH-BRS
FZJ	208.6	0.4
FH-BRS	47.3	511.7

In addition, Table 4 illustrates the minimum duration of the internal and external communication in the VIOLA-testbed. In our configuration, the external message transfer duration exceeded the internal message transfer duration by almost two orders of magnitude. During the communication between Trace and Partrace, the minimum message transfer duration was 862.0 μs. Given that the sites at FZJ and FH-BRS lie 100 km apart, the minimum message transfer time of roughly 333.0 μs can be calculated based on the speed of light. Hence, it can be concluded that the VIOLA network indeed offered a low-latency wide area network link between the sites used for our experiments.

Our measurements show that MetaTrace took advantage of the state-of-the-art network capabilites offered by the VIOLA grid. Solving larger input problems might necessitate further improvements of the underlying network technology.

Table 4. Minimum duration of the internal and external communication in the VIOLA-testbed.

	FZJ	FH-BRS
FZJ	27.3 μs	879.0 μs
FH-BRS	862.0 μs	30.3 μs

4.2.2 Incremental performance optimization

To optimize the performance of MetaTrace, we used SCALASCA to identify undesirable wait states hoping that they can be easily removed. The optimization was carried out in two cycles each consisting of a trace analysis using SCALASCA and a subsequent source-code modification.

The analysis of the unoptimized version showed an overall execution time of 1837.40 seconds aggregated across all processes, whereby a major fraction (72.1 %) was spent in MPI function calls. This MPI fraction is composed of the time used for actual communication (15.4 %) and the time spent waiting (56.7 %) for a communication partner. Obviously, the waiting time clearly dominated the overall communication behavior making it the most promising target for our optimization efforts. Often, reasons for such wait states can be found in the scheduling of communication operations or in the distribution of work among the processes involved.

Figure 2 shows a screen shot of SCALASCA's trace analysis results. Apparently, the application suffered from grid-specific *Wait at Barrier* situations (i.e., global) and non grid-specific *Late Sender* and *Wait at N×N* situations (i.e., local), when communicating or synchronizing. As the display indicates, the global *Wait at Barrier* problem consumed 18.7 % of the overall execution time. In addition, the local *Late Sender* problem consumed 10.6 % of the overall execution time. Finally, the local *Wait at N×N* problem caused 20.2 % of the overall execution time. For a description of these patterns, the reader may refer to [1].

Trace and Partrace synchronize at a global barrier before Trace unidirectionally sends the velocity field to Partrace for further processing. However, because Trace and Partrace are essentially two different programs, each submodel invokes this barrier from a different function. As a result both functions are diagnosed with the global *Wait at Barrier*, although both occurrences are closely connected. Most of the waiting time was attributed to the Partrace function `ReadFieldsFromTrace()`, which had to wait until all processes in Trace had reached the corresponding barrier call in function `printtolink()`. That is, we detected an imbalance between Trace and Partrace, since Partrace went ahead of Trace. Moreover, Trace suffered from local *Late Sender* and *Wait at N×N* situations, which together represent most of the waiting time in internal communication.

The Trace-local *Late Sender* was concentrated in `cgiteration()`. All Trace processes performed calculations inside `cgiteration()` and subsequently distributed their local results to their nearest neighbors. Afterwards, a dot product was calculated using `MPI_Allreduce()`. Although the domain decomposition assigned equally-sized subdomains to every process, border processes were quicker because they had fewer neighbors to exchange border cells with. Given that these processes had fewer communication partners, they not only waited during the data exchange phase for their peers in the center but they could also leave the data exchange phase earlier. That is, this imbalance introduced two performance problems. The first problem occurred while all Trace processes were synchronizing in pairs to exchange their local results, causing a local *Late Sender* situation. The second problem occurred when Trace subsequently calculated the dot product, causing a local *Wait at N×N* situation.

The goal of our first optimization was to make Trace faster. More precisely, we assumed that reducing the Trace-local *Late Sender* problem inside `cgiteration()`, would allow Trace to reach the synchronization point with Partrace earlier, which would also decrease the barrier waiting time between the two submodels. We therefore replaced the synchronous communication operations in `cgiteration()` with their asynchronous counterparts, allowing more variability for the nearest-neighbor data exchange. Now, processes inside Trace would be able to process received results earlier. In addition, the Trace-local *Wait at N×N* situation would also be reduced, since processes in the center of the domain could leave the data exchange phase earlier as well.

After our first optimization cycle, we measured an overall execution time of 877.90 seconds, corresponding to a reduction by more than a factor of two. Now, only a fraction of 42.0 % of the overall execution time was spent in MPI function calls. In Figure 3, a screen shot of a difference experiment [9] obtained by subtracting the original version from the optimized one is depicted. Performance gains are represented by sunken reliefs (negative numbers), performance losses by raised reliefs (positive numbers). The numbers show the difference in execution time in percent relative to the unoptimized version. One can easily recognize that the global *Wait at Barrier* as well as the Trace-local *Late Sender* and *Wait at N×N* were significantly reduced. For instance, the figure shows that the global waiting time at the barrier inside `ReadFieldsFromTrace` was reduced by roughly 14.1% of the total execution time.

Moreover, in the optimized version the global *Wait at Barrier* problem consumed 8.6 % (18.7 % before) of the overall execution time and the Trace-internal *Late Sender* problem consumed 3.7 % (10.6 % before) of the overall ex-

ecution time. Finally, the Trace-local *Wait at N×N* problem caused 7.8 % (20.2 % before) of the overall execution time. By means of asynchronous communication, we were able to significantly reduce the *Late Sender* situation inside Trace since Trace did not wait at synchronization points inside `cgiteration()` during the internal data exchange. In addition, Trace now needed less time for a single iteration and so Trace reached the synchronization point with Partrace earlier, which reduced the global *Wait at Barrier* problem. Finally, the waiting time at the Trace-local *Wait at N×N* situation was notably decreased as well, which was caused by the elimination of synchronization points during the preceeding data exchange phase.

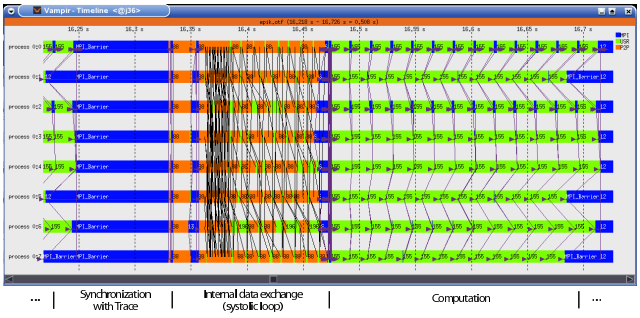


Figure 4. Vampir display: Event traces of all Partrace processes during one simulation cycle.

However, the application still suffered from a global *Wait at Barrier* situation apparent in the two functions mentioned earlier. We decided to perform a second optimization cycle. Trace has variable simulation time steps which depend on the accuracy of the respective calculation whereas Partrace uses constant time steps independent of the accuracy. Since the communication between the two submodels is essentially unidirectional and asynchronous by nature, we replaced the synchronous communication operations between Trace and Partrace including the barrier call with their asynchronous equivalents to eliminate the global *Wait at Barrier* problem. It is worth noting, that without the asynchronous communication scheme, removing the barrier call would cause waiting times during the data exchange. Also, although in our case a decreased runtime of Partrace increases the waiting time during the data transfer between Trace and Partrace, we applied an optimization to Partrace as well. Figure 4 visualizes the event traces of all Partrace processes during one simulation cycle by showing a time line for each process indicating its current execution state by color. Using VAMPIR's zooming capability we examined the runtime behavior further. Partrace used a systolic loop to distribute its simulation data internally. We decided to replace the original communication scheme with a collective communication since the collective operation `MPI_Allgather()` needs substantially less effort.

Table 5. Summary of performance measurements of the unoptimized version and after each optimization cycle.

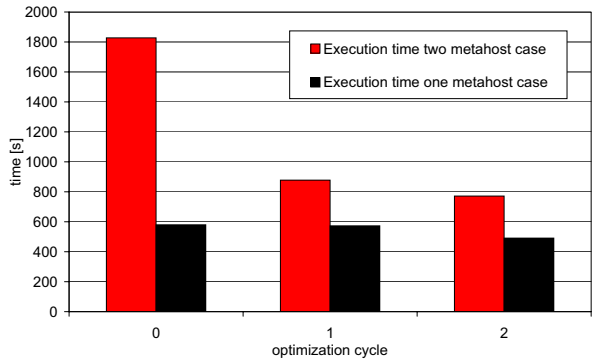
		Optimization	
	unoptimized	1	2
MPI fraction	72.1 %	42.0 %	34.4 %
Wait at Barrier	18.7 %	8.6 %	0.9 %
Late Sender	10.6 %	3.7 %	1.7 %
Wait at N×N	20.2 %	7.8 %	6.0 %

The results of our final performance measurement including the aforementioned optimizations showed only a fraction of 34.4 % of the overall execution time (771.50 seconds) spent in MPI function calls. Further, our analysis results showed that the global *Wait at Barrier* problem could be completely eliminated. Additionally, the Trace-local *Late Sender* version only consumed 3.9 % of the overall execution time and the Trace-local *Wait at N×N* problem caused 6.0 % of the overall execution time. Hence, the major performance problems were significantly reduced and, thus, the performance behavior was significantly improved. Table 5 summarizes the values of the respective performance problem after each optimization cycle according to the functions mentioned above.

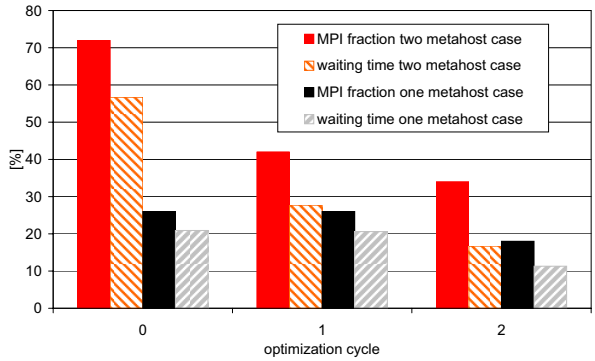
Finally, we compared the application performance on the VIOLA metacomputer achieved before and after our optimizations with the performance when running on the homogeneous IBM AIX POWER 4+ cluster. While Figure 5 (a) shows the the total execution time before and after one and two optimization cycles, Figure 5 (b) shows the corresponding percentage of the execution time spent in MPI function calls. In addition, the respective MPI waiting time is depicted. As can be seen, the overall execution time as well as its MPI fraction is smaller in each experiment performed on the homogenous cluster than on the metacomputer. The optimizations showed only minor influence on the application performance in the homogeneous case. We were able to significantly reduce the total execution time from 1837.40 seconds to 771.50 seconds on the metacomputer. Hence, we were able to significantly reduce grid-specific performance problems of a parallel computational grid application by eliminating the major fraction of waiting times in several optimization cycles.

5 Conclusion

In this paper, we have shown that our extension to the SCALASCA tool set in combination with statistical analyses



(a) The total execution time before the optimization and after one and two optimization cycles.



(b) The percentage of the waiting time and execution time spent in MPI calls before the optimization and after one and two optimization cycles.

Figure 5. Optimization results on a homogeneous cluster and a metacomputer.

and time-line visualization provided by VAMPIR can be used to evaluate and optimize the performance of a multi-physics production code running on a heterogeneous and geographically dispersed metacomputer. Using the grid-enabled tracing and analysis capabilities of the SCALASCA tool set, we have determined relevant performance properties and have experimentally demonstrated that this information can be used to significantly improve performance.

First, we were able to verify that the bandwidth and latency requirements of our application are met by the wide-are connection in the VIOLA grid. Second, we presented a detailed description of the performance optimizations applied to MetaTrace. While MetaTrace fully utilized the entire network resources provided by the VIOLA grid, we have shown in several optimization cycles that our modifications eliminated the major fraction of waiting times. In addition, we compared results from a homogeneous cluster with those obtained on the metacomputer, confirming that some of the performance problems we identified are indeed the consequence of using a metacomputer.

Given the fact that performance optimization for just a single machine is already a non-trivial task that requires substantial tool support, we argue that this is even more important for grid environments. With grid-enabled tools developers are able to optimize their applications to achieve an appropriate performance level. Using MetaTrace as an example, we have shown that grid-enabled performance tools allow efficient execution of parallel applications in grid environments.

References

[1] D. Becker, F. Wolf, W. Frings, M. Geimer, B. Wylie, and B. Mohr. Automatic trace-based performance analysis of metacomputing applications. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, Long Beach, California, March 2007.

[2] B. Bierbaum, C. Clauss, T. Eickermann, L. Kirtchakova, A. Krechel, S. Springstubbe, O. Wäldrich, and W. Ziegler. Orchestration of distributed MPI-applications in a UNICORE-based grid with metampich and metascheduling. In *Proc. 13th European PVM/MPI Conference*, Bonn, Germany, September 2006. Springer.

[3] B. Bierbaum, C. Clauss, M. Pöppe, S. Lankes, and T. Bemmerl. The new multidevice architecture of MetaMPICH in the context of other approaches to grid-enabled MPI. In *Proc. 13th European PVM/MPI Conference*, Bonn, Germany, September 2006. Springer.

[4] BMBF (Ministry for Education and Research). *Vertically Integrated Optical Testbed for Large Applications in DFN (VIOLA)*. `http://www.viola-testbed.de/`.

[5] Forschungszentrum Jülich. *Solute Transport in Heterogeneous Soil-Aquifer Systems*. `http://www.fz-juelich.de/icg/icg-iv/modeling`.

[6] M. Geimer, F. Wolf, B. J. N. Wylie, and B. Mohr. Scalable parallel trace-based performance analysis. In *Proc. 13th European PVM/MPI Conference*, Bonn, Germany, September 2006. Springer.

[7] S. Haubold, H. Mix, W. E. Nagel, and M. Romberg. The UNICORE grid and its options for performance analysis. pages 275–288, 2004.

[8] W. Nagel, M. Weber, H.-C. Hoppe, and K. Solchenbach. VAMPIR: Visualization and analysis of MPI resources. *Supercomputer*, 12(1):69–80, 1996.

[9] F. Song, F. Wolf, N. Bhatia, J. Dongarra, and S. Moore. An algebra for cross-experiment performance analysis. In *Proc. of the International Conference on Parallel Processing (ICPP)*, Montreal, Canada, August 2004. IEEE Computer Society.

[10] F. Wolf and B. Mohr. Automatic performance analysis of hybrid MPI/OpenMP applications. *Journal of Systems Architecture*, 49(10-11):421–439, Nov. 2003.