Performance Analysis of a Parallel Application in the GRID

Holger Brunst[2], Edgar Gabriel[1,3], Marc Lange[1], Matthias S. Müller[1], Wolfgang E. Nagel[2], and Michael M. Resch[1]

[1] High Performance Computing Center Stuttgart
Allmandring 30, D-70550 Stuttgart, Germany
{gabriel,lange,mueller,resch}@hlrs.de

[2] ZHR, Dresden University of Technology
D-01062 Dresden, Germany
{brunst,nagel}@zhr.tu-dresden.de

[3] Innovative Computing Laboratories,
Computer Science Department,
University of Tennessee, Knoxville, TN, USA

Abstract. Performance analysis of real applications in clusters and GRID like environments is essential to fully exploit the performance of new architectures. The key problem is the deepening hierarchy of latencies and bandwidths and the growing heterogeneity of systems. This paper discusses the basic problems of performance analysis in such clustered and heterogeneous environments. It further presents a software environment that supports the user in running codes and getting more insight into the performance of the application. In order to give a proof of the concept a code for direct numerical simulation of reactive flows is run in a GRID like hardware environment, and the performance analysis is presented.

1 Introduction

For some time, parallel programming has been mainly a problem of splitting the work of a code into a number of equally sized pieces and distributing them correctly to all available processors. In addition, communication issues had to be taken care of. This required a substantial amount of work but at least the programmer could safely assume that the code would run on a homogeneous architecture. The speed of processors would be the same as well as the speed of communication between individual processors. Moving from single processor systems to parallel computers was thus a cumbersome task but added just one additional level of complexity.

With the advent of clusters [1] and the concept of the GRID [2] the situation has changed. The new hardware concept is to put together basic building blocks to form a new system. Such basic building blocks can be anything from a single processor PC or a multi processor SMP to a full cluster. In that sense what we

P.M.A. Sloot et al. (Eds.): ICCS 2003, LNCS 2658, pp. 285–294, 2003.

get are systems that are more and more dynamic in composition. In addition, the programmer is faced with a hierarchy of bandwidths, latencies and protocols that is getting deeper and more heterogeneous [3]. From the point of view of performance analysis for real applications this architectural trend represents a challenge both for the programmer and the tool provider.

This paper deals with the performance analysis of a parallel application from the area of combustion simulation, which was ported to distributed GRID environments due to its tremendous demand on resources. During this procedure two problems had to be solved: first, we had to port the application to a GRID environment. Second, we wanted to evaluate the potential of this application in a heterogeneous GRID environment. Both steps required detailed analysis of the communication patterns of the application, using a performance analysis tool.

The structure of the paper is like follows: the problem of performance analysis in GRID environments and the performance analysis tool Vampir is presented in Sect. 2. Section 3 presents the application used throughout the rest of the paper. Section 4 presents then the performance analysis of this application and how we solved the identified problems. Finally, in Sect. 5 we summarize the paper and present the ongoing work in this area.

2 How to Adapt Existing Analysis Technology

Today, clustered parallel computers are available to a large user comunity that deals with numerical simulation. These systems mostly manufactured from standard components off the shelf (COTS) often provide a very good price/performance ratio. Therefore, parallel computing more and more also becomes attractive for commercial applications and thus widens the community of parallel application developers who need tool support.

For increased requirements of resources, clusters can be interconnected via a GRID middleware. Unfortunately, the different communication layers in such a hybrid system make it highly complicated to develop parallel applications performing well. Therefore, it is crucial to gain insight into the application's internal behavior when executed; as this is the only possibility to locate performance bottlenecks in many cases.

The performance evaluation and the subsequent optimization of a parallel program executed on a GRID imposes special requirements on the generation and visualization of performance data [4]. In order to allow for a largely transparent view of the application's behavior, there is a need for some background mechanism that collects and merges the performance data generated on the distributed nodes of the GRID. This task is not a simple one as one has to cope with the synchronization problem of multiple non-synchronous system clocks. Current performance tools are familiar with the analysis and visualization of performance data generated on a homogeneous parallel machine. So far, there is no graphical support for the specifics introduced by GRID computing. In our work we identified the following key issues towards the performance analysis of parallel GRID applications:

- Appropriate data base formats for parallel distributed trace data
- Unified and transparent access to trace data
- Post synchronization of asynchronous trace data
- Visualization of trace data with focus on additional communication layer

Conceptionally, a great deal of these issues has been veryfied with the following software framework that was the basis for this paper. For the communication part we used a GRID-aware implementation of the MPI standard called PACX-MPI. The performance optimization was carried out with an adapted version of the performance analysis tool Vampir [5–7] which was one of the results of the german UNICORE [8] project. The porting of the associated tracing library Vampirtrace onto PACX-MPI has been done in the scope of the European project METODIS [9] and is currently further enhanced in the DAMIEN [10] project. The following properties can be analyzed with this framework:

- Program state changes
- Collective operations
- Peer to peer communication
- Hardware performance monitors
- OpenMP parallel regions
- File I/O
- Environmental information
- Source code locators

Based on event traces, the framework provides timeline representations and accumulated performance measures for selected time intervals of a program run. The timelines 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 weak usage of the floating point units, the timelines are less useful. Hardware counters can be found in almost every processor now. Support of these counters in the form of hardware performance monitors is available in combination with process timeline diagrams.

The different performance categories named above need to be related to each other in order to create a complete view of an application's runtime behavior. Flexible grouping allows performance analysis from different perspectives. When it comes to the visualization task of a multi level GRID application, the grouping capability can be used to navigate through the trace data on different levels of abstraction. This approach works fine for both event trace oriented displays (timelines) and statistic displays for accumulated data.

3 A Real World Application Test Case

This section provides some background on the application from the field of reactive flow simulation which has been analysed in an GRID-like environment. In Sect. 3.1, we briefly discuss why one is interested in the use of clustered systems for this type of application. The basic properties of the code and the physical problem to which it is applied are described in Sect. 3.2.

The performance problems we were faced executing the first runs and the following performance analysis are described then in the subsequent sections.

3.1 Direct Numerical Simulation of Reactive Flows

Chemical reactions occuring in turbulent flows play an essential role in a variety of technical applications like e.g. combustion in automotive engines and gas turbines. There is obviously a large interest in an improved energy efficiency and reduced emissions of these processes. Better and more accurate models for turbulent combustion are needed to improve the quality of numerical simulations of such technical combustion processes. The information about the basic interaction mechanisms of turbulent transport and chemical reactions which is needed for the development of improved turbulent combustion models can be obtained by direct numerical simulations (DNS), in which turbulent reactive flows are computed with a great level of detail.

The set of coupled partial differential equations to be solved in DNS are the Navier-Stokes equations for reactive flows as given e.g. in [11]. These equations express the conservation of mass, momentum, and energy, which are also needed for the DNS of non-reacting flows, as well as the conservation of the mass fractions of all the chemical species. The number of species to be considered varies between nine for the hydrogen-air system and several hundreds in the case of higher hydrocarbons. The computation of the chemical source-terms and of the multicomponent diffusion velocities are the most time consuming parts of such DNS.

Due to the nature of turbulence and the chemical kinetics, a very high resolution is needed in space and time, which leads to a large computational effort. E.g., the DNS of the temporal evolution of a premixed methane flame over a physical time of a few milliseconds in a twodimensional domain with an area of about 3 cm^2 took about 60 hours using 256 PEs of a Cray T3E-900 [12]. As turbulence is an inherently three-dimensional phenomenon, some of it's aspects could only be reproduced precisely by performing similar DNS in 3D requiring at least a hundred times as many grid points. Making efficient usage of large distributed computing environments would be an essential contribution to reach this goal.

3.2 The Application Program PCM

PCM is a code developed for the direct numerical simulation (DNS) of reactive flows on parallel computers with distributed memory using message-passing communication [12, 13]. In favor of being able to utilize detailed models for the computation of chemical reaction kinetics and the molecular transport, it is currently restricted to the simulation of two-dimensional flow fields. While some features of turbulent flows are thus not present in the simulations, it has been shown that many other aspects of turbulent combustion processes are missing if oversimplified models for chemistry or molecular diffusion are used [12].

The spatial discretization in PCM is performed using a finite-difference scheme with sixth-order central-derivatives, avoiding numerical dissipation and leading to very high accuracy. The integration in time is carried out using a fourth-order fully explicit Runge-Kutta method with adaptive control of the time step. This

fully explicit formulation leads to a parallelization strategy, which is based on a regular two-dimensional domain decomposition of the physical space, projected onto a corresponding two-dimensional processor topology. For a given computational grid and number of processors it is tried to minimize the length of the subdomain boundaries and thus the amount of communication. However, the different communication speeds in a GRID-computing environment are not considered in the initial distribution up to now. After this initial decomposition, each processor node controls a rectangular subdomain of the global computational domain. In addition to the grid points belonging to a node's subdomain, a three points wide halo region is stored on each node. Using the values at the grid points of this halo region an integration step on the subdomain is carried out independently from the other nodes. After each integration step, the new values of the domain boundaries are exchanged between neighboring nodes using MPI.

The application case which has been used for the analysed runs is the DNS of a turbulent mixing-layer with a cold H_2/N_2 mixture on one side and heated air on the other. Periodic boundary conditions are used perpendicular to the mixing layer. This is a simple model configuration to study the influence of turbulence on autoignition processes, which is important in combustion systems like Diesel engines where fuel autoignites after being released into a turbulent oxidant of elevated temperature. More details on this configuration and some results regarding the physics can be found in [12, 13].

Recently, our main platform for production runs have been Cray T3E systems and large PC-clusters on which a high performance and a very good scaling behavior has been achieved [14]. While our first experiences with PCM in a metacomputing environment showed a great potential of such distributed platforms for this type of application, it became also obvious that a considerable optimization effort will be needed to fully exploit this potential [15].

4 Analysis of PCM with Vampir

For the tests presented in this paper, 24 processors on a Cray T3E at the High Performance Computing Center Stuttgart were used. For the distributed tests, we used two partitions of the same machine, creating a virtual metacomputer. The first 12 processes were running on the first partition (T3E-A), while the other 12 processes were running on the second partition (T3E-B). The internal latency between two processes on the same machine was 7 μs, the achievable bandwidth was around 300 MBytes/s, while the external latency between the two partitions was in the range of 4 ms, the achievable external bandwidth was approximately 10 MBytes/s. The latter values are typical in local area networks (LAN). Thus, the communication characteristics of this virtual metacomputer are similar to GRID environments with respect to the fact, that we have to deal with the same hierarchies in the quality of communication. Coupling two partitions of the same machine has additionally one major advantage for our analysis compared to using two different machines: we could exclude most effects due to network traffic of other users, which made our results more easily comparable to each other. The

communication library used to make PCM run across distributed environments is PACX-MPI [16], an implementation of the MPI standard optimized for GRID environments.

The following analysis is centered around two questions: During the initial porting of the application onto PACX-MPI, we were facing the problem, that the unexpected message queue in PACX-MPI had from time to time an unexpected overrun, which neither the developers of the library, nor the application developers could explain. Second, we wanted to evaluate the potential of this application for future use in distributed environments, since the computational demand of realistic simulations is tremendous.

4.1 Processing Part

The application can be split into three parts: preprocessing, processing, and application checkpointing. The preprocessing part is dominated by communication, since all file-operations are handled by the process with rank zero. The execution time for the preprocessing is increasing from approximately 3.5 seconds for the Cray-T3E run to about 6 seconds for the metacomputing run. This is due to the fact that the broadcast-operations require external communication. However, the preprocessing-part does not have any dramatic influence on the performance of a real production run.

In a real production run, the overall execution time is dominated by the processing part, since the application will execute several thousand iterations, instead of the 20 iterations shown in this analysis. Zooming into this section shows, that the ratio of communication to computation for this part of the application is a lot better. Using the summary chart, the exact times spent in communication and in computation can be determined. For the processing part, approximately 20 % of the execution time is just spent in communication. Taking into account, that we have to deal with a tightly coupled application using an environment with latencies and bandwidths typical for local-area networks, this result is quite good.

The message statistic display in Fig. 1 shows the communication pattern in the processing part. Two different patterns can be identified in the left part of the figure:

1. The penta-diagonal structure reflects the neighborhood communication in the application, which again is the result of the 2D domain decomposition.
2. Additionally, a much lower amount of communication is visible between the MPI-process with the rank zero and all other processes. A part of this additional communication is due to the load-balancing integrated in the application. Node zero collects the execution time of each process for a certain number of iterations. Depending on the fact, whether the load imbalance between processes is higher than a user defined limit, work will be redistributed between the nodes.

The key-issue for an application to achieve high performance in a clustered environment is high data locality. The assignment of sub-domains to processors

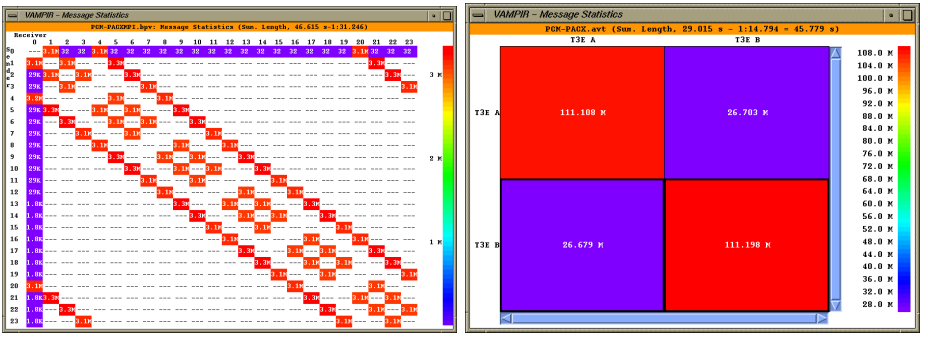

Fig. 1. Communication pattern in the processing part of PCM for a $\mathbf{4} \times 6$ process topology, process view (left) and cluster-view (right).

should reflect the hardware topology. One possible approach is to use an optimized MPI_Cart_create. However, in our case the application performs the mapping itself, because it also supports other message passing libraries. Nevertheless an optimal result may be achieved, because the assumption that processes with ranks that have small differences are close to each other is fulfilled by PACX-MPI. But in general the configuration has to be optimized concerning the communication pattern with respect to periodic boundary conditions, orientation of domain and others. Here, further analysis with Vampir is needed. The result of one configuration can be seen in in the right part of Fig. 1, where the total amount of data is shown, which has been sent between processes inside each machine, and the total amount of data transfered between the machines. The communication inside each cluster-node is dominating compared to the communication between both cluster-nodes. While totally 109 MBytes of data have been sent inside of each cluster-node, just around 26 MBytes of data have crossed the boundaries of the machines.

The good performance behavior of this part of the application is also visible in Table 1, where we compare the execution time for five iterations on the virtual metacomputer with the same number of iterations on the Cray T3E. While the execution time increases slightly, it is still in the same range as on the Cray T3E. This is mainly the result of the regular communication pattern shown in Fig. 1 (left) and of the high data locality shown in Fig. 1 (right).

4.2 Application Checkpointing

Long production runs have to save regularly the current status of the run in order to avoid the loss of results in case of a machine crash. In case several machines and networks are involved in such a simulation, like given in GRID-environments, the probability of loosing a subsystem even increases. For the application under examination here, a regular checkpointing is in addition required for tracking the temporal evolution of the solution.

Table 1. Comparison of the average execution time for 5 time steps on a single machine with vendor-MPI and on two partitions with PACX-MPI.

Number of nodes	Execution time with Cray-MPI	Execution time with PACX-MPI
12	17.8 sec	19.1 sec
24	10.3 sec	12.4 sec

This part of the application shows a very different behavior on a single parallel machine (see left part of Fig. 2) compared to a cluster of T3E's (see right part of Fig. 2). The analysis with Vampir revealed the reason for this very different behavior and in addition explained the problem of gathering thousands of messages on the process with rank zero mentioned above.

On the Cray T3E, the execution of this part is dominated by the process with rank zero. As Fig. 2 shows all other processes start sending their part of the global data to the process zero approximately at the same time, and block then in a wait-operation. The reason for this blocking is, that for the used message length, Cray-MPI switches to the synchronous-send mode. Thus, the wait-operation can only return, when the according receive-operation has been started. This behavior causes all processes to be more or less synchronized with process zero.

With PACX-MPI, the processes sending their data to the process zero behave differently. They send all their messages in a row to the destination process without synchronizations between the individual sends. The reason for this is that PACX-MPI does not switch to a synchronous send-mode for external operations, unless explicitly required by the application by using *MPI_Ssend*. Additionally, the communication-daemons involved in the external operations can receive and transfer the messages faster than the process zero is able to handle the file-operations. Since the wait-operation to each of the initiated send-routines can

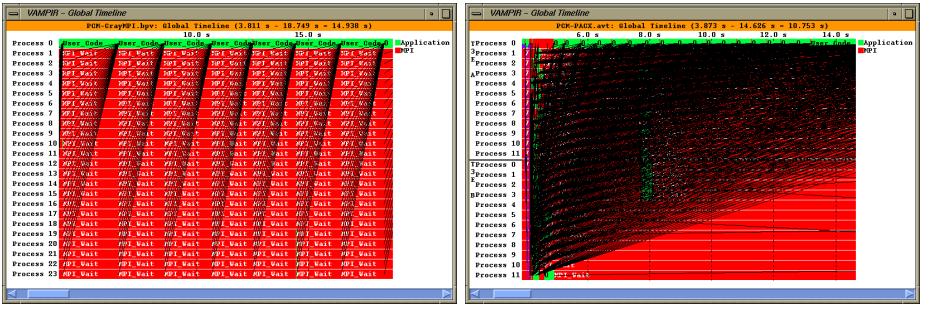

Fig. 2. Communication behavior of PCM during writing of results-files with Cray-MPI (left) and PACX-MPI (right)

return as soon as '*data is safely stored away*'[17], they are returning, as soon as the data is received by the communication-daemon handling all outgoing messages. Thus, the processes can proceed immediately with the next communication, without being synchronized with process zero. This behavior explains the gathering of much higher number of messages at the process doing the file I/O operations than in the single system case. To achieve a similar behavior with PACX-MPI like with Cray-MPI, the application will have to use the synchronous versions of the send-operations of MPI in this part of the application.

5 Conclusions

In this paper, the problems for performance analysis of parallel applications arising in clustered systems are briefly discussed. The components of a software environment have been presented which enables the user to run and analyse applications in GRID-like environments. The functionality of this software environment has been demonstrated for a real world application. In addition to an optimized, topology-aware application and an optimized, GRID-aware MPI library a thorough understanding of the MPI implementation was required to achieve good performance.

The visualization capabilities of Vampir enabled us to analyse the communication pattern of the application with respect to the different levels of communication, which helps to find the best application settings for a given GRID environment. The analysis with Vampir was not only useful for application tuning but also revealed the reason for the buffering of an unexpectedly high number of messages with PACX-MPI. The reason for this problem could not be found without the help of a runtime analysis tool. Once found, it was relatively easy to implement a solution for this problem.

Future extension of Vampir will include further features that are necessary to understand possible performance problems. Examples are a visualization of the protocol types which have been used for the individual messages, or the gathering of environment variables on the different platforms in the Grid that influence the behavior of the MPI library, like MPI_BUFFER_MAX.

Acknowledgments

The authors would like to thank their home institutions for providing their machines for this study. We would also like to thank Hans-Christian Hoppe from PALLAS GmbH for the support for VAMPIRtrace.

References

1. R. Buyya. *High Performance Cluster Computing, Volume 1: Architectures and Systems, Volume 2: Programming and Applications* . Prentice Hall, 1999.
2. Ian Foster and Carl Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1999.

3. S. Pickles, F. Costen, J. Brooke, E. Gabriel, M. Müller, M. Resch and S. Ord. *Metacomputing Across Intercontinental Networks*. Future Generation Computer Systems, 7, 2001, 911–918.
4. H. Brunst, W. E. Nagel, and H.-C. Hoppe. Group-Based Performance Analysis for Multithreaded SMP Cluster Applications. In R. Sakellariou, J. Keane, J. Gurd, and L. Freeman, Eds., *Euro-Par 2001 Parallel Processing*, LNCS 2150, 148–153, Springer, 2001.
5. H. Brunst, M. Winkler, W. E. Nagel, and H.-C. Hoppe. Performance Optimization for Large Scale Computing: The Scalable VAMPIR Approach. In V. N. Alexandrov, J. J. Dongarra, B. A. Juliano, R. S. Renner, and C. K. Tan, Eds., *Computational Science – ICCS 2001, Part II*, LNCS 2074, 751–760, Springer, 2001.
6. S. Moore, D. Cronk, K. London, and J. Dongarra. Review of Performance Analysis Tools for MPI Parallel Programs. In Y. Cotronis and J. Dongarra, Eds., *Recent Advances in Parallel Virtual Machine and Message Passing Interface, 8th European PVM/MPI Users' Group Meeting, Proceedings*, LNCS 2131, 241–248, Springer, 2001.
7. W. E. Nagel, A. Arnold, M. Weber, H.-C. Hoppe, and K. Solchenbach. *VAMPIR: Visualization and Analysis of MPI Resources*. Supercomputer, 12(1), 1996, 69–80. http://www.pallas.de/pages/vampir.htm.
8. J. Almond, D. Snelling. *UNICORE: Secure and Uniform Access to Ditributed Resources via the World Wide Web*. 1998. http://www.fz-juelich.de/unicore/whitepaper.ps.
9. *METODIS – Metacomputing Tools for Distributed Systems*. http://www.hlrs.de/organization/pds/projects/metodis.
10. *DAMIEN – Distributed Applications and Middleware for Industrial Use of European Networks*. http://www.hlrs.de/organization/pds/projects/damien.
11. F. A. Williams. *Combustion Theory*. second edition, Benjamin/Cummings, 1985.
12. M. Lange and J. Warnatz. Investigation of Chemistry-Turbulence Interactions Using DNS on the Cray T3E. In E. Krause and W. Jäger, Eds., *High Performance Computing in Science and Engineering '99*, 333–343, Springer, 2000.
13. M. Lange. Parallel DNS of Autoignition Processes with Adaptive Computation of Chemical Source Terms. In C. B. Jenssen, T. Kvamsdal, H. I. Andersson, B. Pettersen, A. Ecer, J. Periaux, N. Satofuka, and P. Fox, Eds., *Parallel Computational Fluid Dynamics: Trends and Applications*, 551–558, Elsevier Science, 2001.
14. M. Lange. Massively Parallel DNS of Flame Kernel Evolution in Spark-Ignited Turbulent Mixtures. In E. Krause and W. Jäger, Eds., *High Performance Computing in Science and Engineering '02*, 425–438, Springer, 2002.
15. E. Gabriel, M. Lange and R. Rühle. Direct Numerical Simulation of Turbulent Reactive Flows in a Metacomputing Environment. In *Proceedings of the 2001 ICPP Workshops*, 237–244, IEEE Computer Society, 2001.
16. E. Gabriel, M. Resch and R. Rühle. Implementing MPI with Optimized Algorithms for Metacomputing. In A. Skjellum, P. V. Bangalore, Y. S. Dandass, Eds., *Proceedings of the Third MPI Developer's and User's Conference*, MPI Software Technology Press, 1999.
17. Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard (Version 1.1)*. 1995. http://www.mpi-forum.org.