John von Neumann Institute for Computing

Distribution of Periscope Analysis Agents on ALTIX 4700

Michael Gerndt, Sebastian Strohhäcker

published in

Parallel Computing: Architectures, Algorithms and Applications ,
C. Bischof, M. Bücker, P. Gibbon, G.R. Joubert, T. Lippert, B. Mohr,
F. Peters (Eds.),
John von Neumann Institute for Computing, Jülich,
NIC Series, Vol. **38**, ISBN 978-3-9810843-4-4, pp. 113-120, 2007.
Reprinted in: Advances in Parallel Computing, Volume **15**,
ISSN 0927-5452, ISBN 978-1-58603-796-3 (IOS Press), 2008.

http://www.fz-juelich.de/nic-series/volume38

Distribution of Periscope Analysis Agents on ALTIX 4700

Michael Gerndt, Sebastian Strohhäcker

Technische Universität München, Fakultät für Informatik I10
Boltzmannstr.3, 85748 Garching, Germany
E-mail: gerndt@in.tum.de

Periscope is a distributed automatic online performance analysis system for large scale parallel systems. It consists of a set of analysis agents distributed on the parallel machine. This article presents the approach taken on the ALTIX 4700 supercomputer at LRZ to distribute the analysis agents and the application processes on to the set of processors assigned to a parallel job by the batch scheduling system. An optimized mapping reducing the distance between the analysis agents and the application processes is computed based on the topology information of the processors. This mapping is then implemented via the dplace command on the Altix.

1 Introduction

Performance analysis tools help users in writing efficient codes for current high performance machines. Since the architectures of today's supercomputers with thousands of processors expose multiple hierarchical levels to the programmer, program optimization cannot be performed without experimentation.

Performance analysis tools can provide the user with measurements of the program's performance and thus can help him in finding the right transformations for performance improvement. Since measuring performance data and storing those data for further analysis in most tools is not a very scalable approach, most tools are limited to experiments on a small number of processors.

Periscope[1–3] is the first distributed online performance analysis tool. It consists of a set of autonomous agents that search for performance properties. Each agent is responsible for analyzing a subset of the application's processes and threads. The agents request measurements of the monitoring system, retrieve the data, and use the data to identify performance properties. This approach eliminates the need to transport huge amounts of performance data through the parallel machine's network and to store those data in files for further analysis.

The focus of this paper is on the distribution of application processes and analysis agents in Periscope. Large scale experiments with Periscope are executed in form of batch jobs where in addition to the processors for the application additional processors are allocated for the analysis agents. The number of additional processors is currently decided by the programmer. The analysis agents provide feedback to the user, whether they were overloaded with the number of processes. In such a situation, the programmer might decide to use more processors for the analysis in a next experiment.

During startup of the experiment, Periscope determines the mapping of application processes and analysis agents to the processors. It is the goal, to place analysis agents near to the controlled processes to reduce the communication overhead. This paper describes the concepts and the implementation used on the ALTIX 4700 supercomputer at LRZ for placement.

The next section gives a short overview of related work. Section 3 presents Periscope's architecture. Section 4 introduces our target system. The mapping features of the batch system on the ALTIX are introduced in Section 5. Section 6 presents the details on the distribution of the application processes and the analysis agents. Section 7 presents results of our simulated annealing-based optimization approach.

2 Related Work

Several projects in the performance tools community are concerned with the automation of the performance analysis process. Paradyn's[4] Performance Consultant automatically searches for performance bottlenecks in a running application by using a dynamic instrumentation approach. Based on hypotheses about potential performance problems, measurement probes are inserted into the running program. Recently MRNet[5] has been developed for the efficient collection of distributed performance data. However, the search process for performance data is still centralized. To remedy the bottleneck this centralized approach presents for large-scale machines, work towards a Distributed Performance Consultant (DPC)[6] was recently conducted.

The Expert[7] tool developed at Forschungszentrum Jülich performs an automated postmortem search for patterns of inefficient program execution in event traces. Potential problems with this approach are large data sets and long analysis times for long-running applications that hinder the application of this approach on larger parallel machines. More recently, Scalasca[8] was developed as a parallelized version of Expert.

Aksum[9], developed at the University of Vienna, is based on a source code instrumentation to capture profile-based performance data which is stored in a relational database. The data is then analyzed by a tool implemented in Java that performs an automatic search for performance problems based on JavaPSL, a Java version of ASL.

Hercule[10] is a prototype automatic performance diagnosis system that implements the model-based performance diagnosis approach. It operates as an expert system within the performance measurement and analysis toolkit TAU. Hercule performs an offline search based on model-specific measurements and performance knowledge.

Periscope goes beyond those tools by performing an automatic online search in a distributed fashion via a hierarchy of analysis agents.

3 Architecture

Periscope consists of a frontend and a hierarchy of communication and analysis agents – Fig. 1. Each of the analysis agents, i.e., the nodes of the agent hierarchy, searches autonomously for inefficiencies in a subset of the application processes.

The application processes are linked with a monitoring system that provides the Monitoring Request Interface (MRI). The agents attach to the monitor via sockets. The MRI allows the agent to configure the measurements; to start, halt, and resume the execution; and to retrieve the performance data. The monitor currently only supports summary information, trace data will be available soon.

The application and the agent network are started through the frontend process. It analyzes the set of processors available, determines the mapping of application and analysis

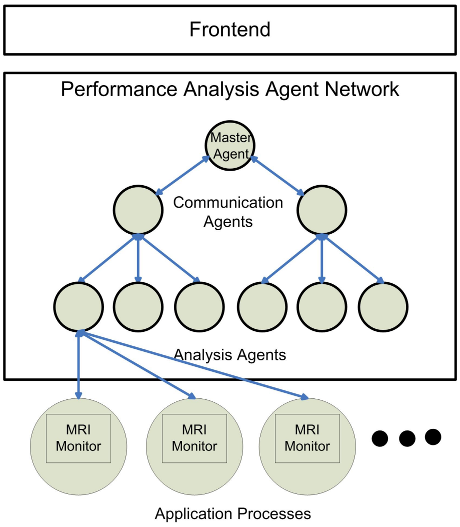


Figure 1. Periscope consists of a frontend and a hierarchy of communication and analysis agents. The analysis agents configure the MRI-based monitors of the application processes and retrieve performance data.

agent processes, and then starts the application. The next step is the startup of the hierarchy of communication agents and of the analysis agents. After startup, a command is propagated down to the analysis agents to start the search. At the end of the local search, the detected performance properties are reported back via the agent hierarchy to the frontend. The communication agents combine similar properties found in their child agents and forward only the combined properties.

To eliminate the need to transfer performance data through the whole machine, the analysis agents are placed near to their application processes. The next sections provide details on how the mapping is computed and implemented on the ALTIX 4700 supercomputer at LRZ, our main target system.

4 ALTIX 4700 Architecture

The ALTIX 4700 at the Leibniz Rechenzentrum (LRZ) in Garching consists of 19 NUMA-link4 interconnected shared memory partitions, each with 256 Intel Itanium 2 Montecito dual core processors and 2TB of memory. Within one partition the connection between processors is laid out as fat-tree.

Peak performance of the system with 9728 cores is 62,3 Teraflop/s. A large range of compilers, tools, libraries and other components is available for development and maintenance supporting OpenMP, various MPI flavours and other parallel programming paradigms.

5 PBS Jobs

The largest part of the ALTIX 4700 is reserved for batch processing. Batch jobs are annotated shell scripts where PBS-directives can be used to specify the required resources. This includes the number of processors, the amount of memory, and a runtime limit. The scripting part sets up the environment and starts the application.

MPI jobs can be run on processors from multiple partitions, while pure OpenMP jobs have to run within a single partitions and thus are limited to 512 threads. Hybrid application have the restriction that all threads of a single MPI process have to run in the same partition.

A job request is sent to the batch queue server using *qsub*. If enough resources are available they are assigned to the job and the script is executed. The environment contains information about the processors that are attributed to the current job in the form of cpusets. For each partition with processors selected for the current job, PBS specifies a cpuset which provides data about the physical CPU number of the allocated processors. Together with topological data of the ALTIX 4700 distance information can be generated.

MPI or hybrid applications are started from within a job script using *mpiexec*. It automatically starts application processes on all allocated processors (potentially spanning several nodes). In addition PBS allows starting remote application via *pbsdsh* which is used to place agents on their destined node.

Exact processor binding is achieved through the *dplace* utility. It can be used in combination with *pbsdsh* as well as with *mpirun*, which replaces the automatic placement via *mpiexec* in the batch script. *dplace* allows to map processes to processors of a single partition. If MPI or hybrid jobs span multiple partitions, the multi-program form of *mpirun* has to be used to start the MPI processes on each partition separately.

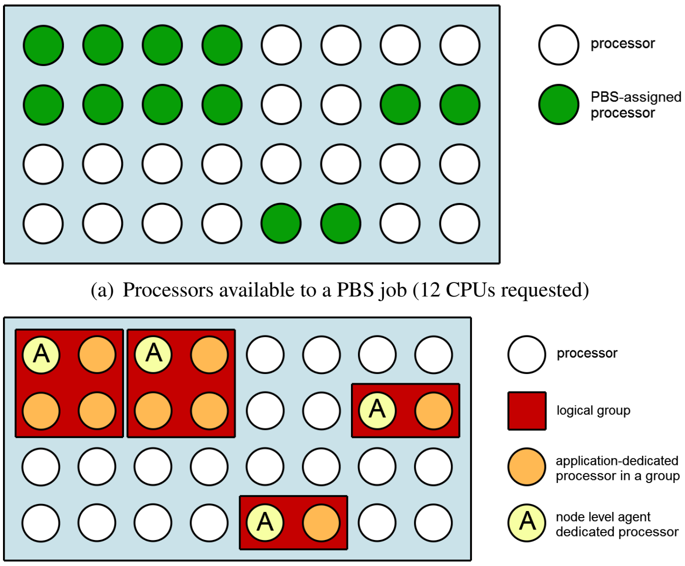
Hybrid programs have special properties both in their startup parameters as well as in the order processes and threads are created. When a process is started, it instantiates a number of helper threads before the actual threads are created. This information becomes relevant when *dplace* is used to force exact placement of both MPI processes and OpenMP threads as it relies on the order of process/thread creation.

6 Application and Agent Distribution

Performance measurement with Periscope is initiated by starting the frontend component. It requires parameters like the target application and the number of MPI processes and OpenMP threads (where their total number has to be smaller than the amount of requested processors so the agents can be started on unoccupied CPUs). Also some performance search configuration values can be overridden by parameters.

First the frontend evaluates data about the environment gathered from the PBS system, cpusets, and machine-dependent resources. It computes the mapping of application processes and analysis agents to the processors allocated to the job and starts the instrumented target application. The initialization code of the monitoring library inserted by source-level instrumentation halts the application processes so that the analysis agents can connect to and control the application processes.

Then a master agent is started with information about the target application components, i.e., processes and threads. It is the top level agent in the hierarchy, a recursive partitioning algorithm starts further communication agents until a single communication


processor
PBS-assigned processor
processor
logical group
application-dedicated processor in a group
A
node level agent dedicated processor

(a) Processors available to a PBS job (12 CPUs requested)

(b) Assignment to analysis agents and program components

Figure 2. The process of aggregating logical clusters. Available processors as depicted in (a) are grouped together depending on their distance values (not shown here). This information is used for the actual program component and analysis agent placement which can be seen in (b).

agent resides on each partition. Then, instead of deepening the hierarchy of communication agents, a number of analysis agents are started and connected to disjoint subsets of the application processes. The analysis agents notify their successful startup by sending a message to the parent in the agent hierarchy. When all children of a communication agent are started this information is handed upwards until the frontend is aware of all agents being ready. Then a start message is sent to all analysis agents to begin with their performance property search.

The agent hierarchy is in the shape of a tree consisting of communication agents in its upper part and analysis agents which are connected to the target application components at the lowest level. Depending on the architecture of the machine the branching factor of the hierarchy is chosen, for the ALTIX 4700 rather few communication agents are required in the upper layers due to the low number of partitions. The amount of analysis agents is determined by the programmer by choosing more processors for the job than application processes or threads are started.

In order to keep the impact of the performance monitoring low, the analysis agents are to be placed close to the application processes or threads, they are responsible for. Thus the available processors are grouped into evenly-sized clusters where one processor slot is reserved for the analysis agent – see Fig. 2. This first step is guided by a specification of the maximum cluster size. The real cluster size is based on the number of CPUs in the partition, the number of desired clusters, and possibly by requirements from using a hybrid programming model.

Generation of the clusters works per partition with processors assigned to the job. To achieve tight grouping, the processor distance information of a partition is extracted from the topology file. Based on nearest processor assignment the clusters are created, after that several optimizing techniques can be deployed to minimize the distance from an analysis agent CPU to the other CPU in the cluster (Section 7).

Based on the computed mapping information the target application is started using *dplace* to enforce exact placement of all components. After the application is fully started up, the same clustering data are used for the analysis agent placing as well: when creating the agent hierarchy, placement strings and identifiers of application processes are passed along to be used at the lowest level to start the analysis agent on its dedicated processor and to connect its to the application processes.

7 Agent Placement Optimization

In order to keep the communication overhead low, two different methods of optimization are performed. First, the logical group tightness is improved by rearranging the mapping of application processes/threads and the analysis agent to processors within a cluster (Section 6). Second, the possibly available information about the message traffic between agents is used to calculate a good distribution of analysis agents over the logical groups.

The task of optimizing the initial clusters is modeled as the minimization of a sum of distances. This sum consists of all distance values from an application CPU to its respective analysis agent CPU. First, the processors in a cluster are treated without distinguishing between their purpose. For each cluster the optimal CPU number for the node-level agent is determined by calculating the sum of distances from one CPU to the others. The configuration that minimizes this sum is used to define the new processor number that will host the node-level agent.

During the second step the node-level agent position is fixed (as determined previously). The algorithm iterates over all pairs of clusters and over all pairs of CPUs selecting one processor number from each cluster. Then the theoretical improvement by swapping the CPU pair is calculated as the sum of the distances from the processors to the node-level agent dedicated processor. If the sum of this measure for both clusters is greater than zero the swap is performed. By this the total measure never increases.

To avoid local minima an exchange can also be performed if the total sum is raised, but only with a certain probability which is high at the beginning of the algorithm and decreases over time. This is similar to the optimization of a function using simulated annealing.

Optimizing the distribution of the analysis agents requires knowledge about the communication traffic. This might be information gathered from previous test runs, or a reasonable estimate. The underlying algorithm works similar to the simulated annealing based approach above. Instead of calculating the total communication costs for all permutations of the analysis agent to processor mapping, starting at an arbitrary configuration, analysis agent CPUs are exchanged (which implies a change of the mapping) if this results in lower total costs. Swap operations that raise the costs are only accepted with a decreasing probability.

In Fig. 3 the effect of the optimization based on simulated annealing is depicted. A configuration of 108 analysis agents is used, but the algorithms have been verified to work correctly and efficiently for much larger amounts of agents. The distance information of the Altix 4700 in form of the topology data is used, whereas random data for the communi-

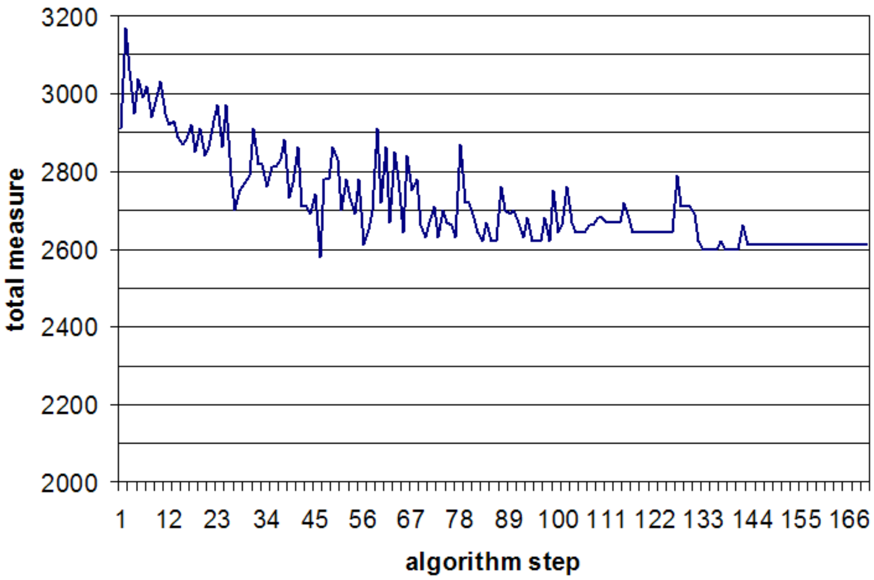


Figure 3. Result of optimization.

cation intensity have been generated for the purpose of testing. Initially the probability of accepting exchanges that increase the total measure is set to 42% and decreased over time. The algorithm terminates if the probability drops below 0.6%, which results in a total runtime of under one second for up to 512 analysis agents (each shared memory partition of the Altix 4700 consists of 512 processor cores). The size of the decrements of the probability value as well as the final pre-defined probability can be used to adjust the runtime of the algorithm. Compared to testing all permutations of the analysis agent to processor mapping (which already becomes unfeasible for more than 120 agents) the exchange based optimization is fast and versatile.

Both optimization techniques are flexible to be used in various configuration scenarios. On large shared memory systems, like the Altix 4700, the logical grouping is the main target of optimization to reduce the access time to the shared data (ring buffers). The optimization of the communication traffic between the analysis agents is especially relevant on large distributed systems like IBM's Blue Gene.

8 Summary

Periscope is an automatic performance analysis tool for high-end systems. It applies a distributed online search for performance bottlenecks. The search is executed in an incremental fashion by either exploiting the repetitive behaviour of program phases or by restarting the application several times.

This article presented the startup of the parallel application and the agent hierarchy. We use a special algorithm to cluster application processes and analysis agents with respect to the machine's topology. The implementation of the startup on the ALTIX supercomputer at LRZ is based on the multi-program version of the *mpirun* command and the *dplace* utility for process to processor mapping.

In the current implementation, it is the responsibility of the programmer to choose the amount of resources for the analysis agent hierarchy. In future, we will work on an automatic selection based on historical information.

References

1. M. Gerndt, K. Fürlinger, and E. Kereku, *Advanced Techniques for Performance Analysis*, Parallel Computing: Current&Future Issues of High-End Computing (Proceedings of the International Conference ParCo 2005), Eds: G. R. Joubert, W. E. Nagel, F. J. Peters, O. Plata, P. Tirado, E. Zapata, NIC Series vol. **33**, ISBN 3-00-017352-8, pp. 15–26, (2006).
2. K. Fürlinger, *Scalable Automated Online Performance Analysis of Applications using Performance Properties*, PhD thesis, Technische Universität München, (2006).
3. E. Kereku, *Automatic Performance Analysis for Memory Hierarchies and Threaded Applications on SMP Systems*, PhD thesis, Technische Universität München, (2006).
4. 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 Tool*, IEEE Computer, **28**, 37–46, (1995).
5. Philip C. Roth, Dorian C. Arnold, and Barton P. Miller, *MRNet: A Software-Based Multicast/Reduction Network for Scalable Tools*, in: Proc. 2003 Conference on Supercomputing (SC 2003), Phoenix, Arizona, USA, (2003).
6. P. C. Roth and B. P. Miller, *The Distributed Performance Consultant and the Sub-Graph Folding Algorithm: On-line Automated Performance Diagnosis on Thousands of Processes*, in: Proc. ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'06), (2006).
7. F. Wolf and B. Mohr, *Automatic Performance Analysis of Hybrid MPI/OpenMP Applications*, in: 11th Euromicro Conference on Parallel, Distributed and Network-Based Processing, pp. 13–22, (2003).
8. M. Geimer, F. Wolf, B. J. N. Wylie and B. Mohr, *Scalable Parallel Trace-Based Performance Analysis*, in: Proc. 13th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface (EuroPVM/MPI 2006), pp. 303–312, Bonn, Germany, (2006).
9. T. Fahringer and C. Seragiotto, *Aksum: A Performance Analysis Tool for Parallel and Distributed Applications*, in: Performance Analysis and Grid Computing, Eds. V. Getov, M. Gerndt, A. Hoisie, A. Malony, B. Miller, Kluwer Academic Publisher, ISBN 1-4020-7693-2, pp. 189–210, (2003).
10. L. Li, A. D. Malony and K. Huck, *Model-Based Relative Performance Diagnosis of Wavefront Parallel Computations*, in: Proc. 2nd International Conference on High Performance Computing and Communications 2006 (HPCC 06), M. Gerndt and D. Kranzlmüller, (Eds.), vol. **4208** of *LNCS*, pp. 200–209, (Springer, 2006).