

# Survey of optimizing techniques for parallel programs running on computer clusters

Espen S. Johnsen  
Otto J. Anshus  
John Markus Bjørndalen  
Lars Ailo Bongo

*Department of Computer Science, University of Tromsø*

September 29, 2003

## 1 Introduction

In the current field of high performance computing, clusters technologies plays an ever increasing role. This paper tries to summarize state-of-the techniques for optimization of parallel programs designed to run on computer clusters.

Optimizing parallel programs is a much harder task than optimizing sequential programs due to the increased complexity caused by communication and synchronization between threads. As a consequence of this, tools and techniques traditionally used for sequential application are not always very helpful in detecting and solving performance bottlenecks in parallel application. This survey present an overview of some of the available techniques and tools for optimization of parallel programs.

## 2 Optimizing techniques for parallel programs

Quite a few techniques exists that may be used in optimization of parallel programs. In this paper, these techniques are roughly classified into the following categories:

- program transformations
- optimizing communication links
- optimizing message passing
- optimizing cluster configuration
- self adapting software
- load balancing

### 2.1 Program transformation

Program transformation is the act of changing a program into another without altering the original semantics. In this context, it often means parallelizing of sequential applications, either manually or automatically.

#### 2.1.1 Application level optimization

One of the most obvious techniques for optimizing of both parallel and sequential programs is tuning and transforming of the algorithms for the underlying computation – in other words, choosing better algorithms. As this technique changes the way the application problem is solved, it is called application level optimization[6]

When developing new applications, using this technique does normally not lead to any problems. But it may not be feasible when dealing with legacy application or parallelizing existing sequential code. One reason for this is that in applications which are often very complex structures, the computations may not be well understood by the maintainer (who may be a computer scientist and not expert in the field of the application). Modifying the application in such a way may therefor lead to incorrect results or just doubt about the correctness of the computations. Application level optimization is also a very time consuming task as it can only be performed manually.

#### 2.1.2 Program level optimization

Program level optimization[6] is a technique in which the application is analyzed at program level, in order to improve performance. Understanding of the underlying application problem is not necessary. Instead transformation is based on the source code, very similar to how a parallelizing compiler does optimization. This

kind of technique is especially suited when porting sequential code to parallel form.

## 2.2 Optimizing communication links

Effective communication between the nodes in a cluster, is essential for best utilization of the available resources, eg. not leaving nodes idle while waiting for data to process. A lot of effort is currently being put into research around these matters. This section summarizes some this work.

### 2.2.1 Network topology

An important aspect of clusters is the physical topology of the network connecting nodes. By changing the interconnecting links, significant improvements in performance may be achieved. *Minimal Distance Mesh with Wrap-around*[13] (Midimew) is an optimal network topology where every node is connected to four other nodes in a such way that a circular graph is created. Compared to other topologies, Midimew has shorter average distance when the number of nodes is large and should be a very good network topology when building large clusters and massive parallel computers.

### 2.2.2 Compiled communication

Compiled communication is a technique which seeks to optimize network efficiency by eliminating as much as possible of runtime configuration (ie. routing decision) by managing network resources statically at compile time. The technique depends on communication patterns in applications being mostly static. Scientific applications is generally regarded as having mostly static communication patterns[17].

CC-MPI[10] is an implementation of a compiled communication capable MPI prototype for switched ethernet clusters. CC-MPI optimizes the communication routines for many of the collective operators. To achieve this CC-MPI, extends the MPI standard by separating network control from data transmission and introducing routines to the application programmer for controlling and optimizing communication. One-to-many and many-to-many communication is implemented using reliable multi-cast based on compiler-assisted group management.

The conclusion drawn is that when using compiled communication the performance of switched ethernet clusters, can be significantly improved compared to traditional MPI implementations. One drawback with the technique is that it requires detailed knowledge of the underlying network, which the typical application programmer may not have.

### 2.2.3 Reliable IP multicast

As shown in [10] multi-cast can greatly improve the communication performance of MPI collective operations on LANs. A comprehensive study of different reliable multicast protocols and how well they perform in LAN clusters is given in [11]. The protocols studied are ACK-based, NAK-based with polling, ring based and tree based.

## 2.3 Optimizing message passing

Other methods for reducing network overhead also exists. Message vectorization[7] is an optimization technique for distributed memory access through message passing, in where *send* and *receive* operations is moved out of loops and concatenated to form larger messages. Since the overhead in initiating a communication operation is quite large compared to the actual data transfer, this should reduce the total time spend in communication considerably. Message coalescing and message aggregation is similar techniques that combines references to the same data from different places and references to different parts of the data into single messages.

These techniques does only give local optimization (ie. optimization of a single loop nest) and doesn't consider references across different nests. To solve this problem, different methods based on data-flow analysis have been proposed. On particular method for data-flow analysis which uses linear algebra[9], allows both traditional loop based optimization and global optimization techniques to be applied.

## 2.4 Optimizing cluster configuration

Cluster configuration in this context, does not mean hardware configuration, but how the software is mapped to the underlying cluster resources.

### 2.4.1 Threaded MPI

In a traditional implementation of MPI, individual instances of the application are separate processes, even when running on the same machine in a SMP-cluster. In such a setup, even communication between processes on a single machine needs to go through the operation system kernel. By running all MPI-nodes on a single machine, as threads within the same process, one may take advantage of faster context switching and using shared address space for communication and synchronization between threads. A an implementation of MPI utilizing this technique is TMPI[14]. Making such an implementation is not straight forward due to the fact that MPI is not thread safe. Access to some functions has to be serialized, and calls to low level parts of MPI can not be done. This is generally not

a problem since few applications make use of such low level functionality. Experiments shows that TMPI may outperforms MPICH substantially on SMP-cluster.

#### 2.4.2 Optimization of collective operations

Configuring hows nodes communicate with each other may significantly improve the efficiency of collective operations in a parallel MPI programs. In LAM-MPI broadcast and reduction operations are implemented using an algorithm where nodes are organized in a logarithmic spanning tree. This tree does not necessarily reflect the underlying topology of the cluster. Experiments[3] have shown that by adding configuration to LAM-MPI, efficiency of the allreduce operation could be improved by a factor as large as 1.98.

### 2.5 Self adapting software

Running the same application on different hardware platforms may require different settings of various performance parameters to achieve optimal performance. Manually tuning of large complex applications to new hardware architectures may be a very time consuming task. Especially in heterogeneous environments such as GRIDs where many different hardware architecture and configurations may be involved.

An alternative approach is to use self adaptive software[4], ie. the application decides at runtime which algorithm is best suited, and how to tune parameters to yield optimal performance. Information about the hardware it is running on, the current data set and empirical results of previous runs is used to make these decisions.

### 2.6 Load balancing

Load balancing is the task of distributing the work load evenly across the nodes in a cluster. Load-balancing may either be static or dynamic. Static load-balancing uses knowledge available before execution to partition tasks among the nodes, while dynamic load-balancing is an adaptive technique where tasks are scheduled based on knowledge gained during execution. The importance of load balancing for improving performance of parallel programs is thoroughly demonstrated in [12].

*Zoltan*[5] is a general purpose tool which aims provide an application independent way of doing dynamic load-balancing. The load-balancing algorithm is moved away from the application and into a library. The application programmer needs to provide callback functions for information gathering, to the library. He must also select a proper load-balancing method to used, but the actual load-balancing algorithms may be changed without any modification to applications.

Currently *Zoltan* only support geometric and graph-based methods, but other algorithms may be easily added. Migrating data to a new processor cannot be done by the library as it is application dependent, but *Zoltan* contains functionality to aid the application with this task.

DASH[1] is an other proposed general purpose load-balancing scheme using autonomous software agents. Each node runs an agent composed of a monitoring module, process execution module and process scheduler module. These modules is responsible exchanging information with nearby agents, running migrated task and scheduling or migrating task. The system is especially attractive to parallel application using the *Single Program Multiple Data Stream* (SPMD) model.

## 3 How to optimize parallel programs

This section gives some hints about how the optimization techniques described in this document may be applied. A more comprehensive list of tools for execution monitoring and performance analysis of parallel programs, is given in [8].

### 3.1 Compiler optimization

Many of these techniques may be applied automatically as compiler optimizations. EARTH-McCAT[18] is an optimizing/parallelizing C compiler focusing on reducing communication overhead. The compiler utilizes an analysis phase to find possible placement points for communication primitives. The following transformation phase would then select the best locations for these calls and perform appropriate code transformations.

### 3.2 The NOW-Sort experience

[2] present the experiences gained in developing and tuning the performance of NOW-Sort. Their methodology involved setting up optimistic performance expectations. By doing this they were able to put their optimizing efforts where it gave most effect, and also know when to stop optimizing. By explicit defining expectations, the tuning of the applications was said to be greatly simplified.

Performance tools were mainly used for three purposes: setting expectations, visualize measured performance and searching for anomalies. Performance counters (instrumentation) on all levels of the system were also found to be of great importance.

The development first focused on a single node. Only when the expectations for a single node implementation were reached, did the development focus change

to large scale parallelism. But only after the performance expectations were fulfilled on a small number of nodes, where the project scaled up to full cluster size.

### 3.3 Instrumentation

Code instrumentation is valuable technique as experienced in NOW-Sort, for collecting performance data that may be further analyzed for the purpose of locating bottlenecks in parallel (and sequential) programs.

*ULTRA*[15] is a tool using instrumentation techniques to collect traces from the execution of parallel MPI programs. This is done using the profiling wrapper mechanism in MPI, and does not require any modification to the program being instrumented. A log entry containing the number of instruction executed and other relevant information, is written for every communication operation. By using traces instead of time-based measurements, the performance of the application is measured independent of the actual cluster it is running on (ie. the performance of the cluster doesn't influence the result). These traces may therefor be used in simulations to predict how well the application will perform on a different cluster hardware.

An other tool using instrumentation for performance measuring and analysis of parallel programs is *Paradyn*. [16]. The technique described, called dynamic instrumentation, inserts code to collect performance data at the appropriate points based on dynamic control. The user is assisted in deciding which performance bottlenecks to search for, or the system can automatically detect such bottlenecks.

## References

- [1] A. RAJAGOPALAN, S. H. An agent based dynamic load balancing system. In *Autonomous Decentralized Systems, 2000. Proceedings. 2000 International Workshop on , 21-23 Sept. 2000* (2000), IEEE, pp. 164–171.
- [2] ARPACI-DUSSEAU, A. C., ARPACI-DUSSEAU, R. H., CULLER, D. E., HELLERSTEIN, J. M., AND PATTERSON, D. A. Searching for the sorting record: experiences in tuning now-sort. In *Proceedings of the SIGMETRICS symposium on Parallel and distributed tools* (1998), ACM Press, pp. 124–133.
- [3] BJØRNDALLEN, J. M., ANSHUS, O., VINTER, B., AND LARSEN, T. The Performance of Configurable Collective Communication for LAM-MPI in Clusters and Multi-Clusters. *NIK 2002, Norsk Informatikk Konferanse, Kongsberg, Norway* (November 2002).
- [4] CHEN, Z., DONGARRA, J., LUSZCZEK, P., AND ROCHE, K. Self adapting software for numerical linear algebra and lapack for clusters, 2003.
- [5] DEVINE, K., HENDRICKSON, B., BOMAN, E., JOHN, M. S., AND VAUGHAN, C. Design of dynamic load-balancing tools for parallel applications. In *Proceedings of the 14th international conference on Supercomputing* (2000), ACM Press, pp. 110–118.
- [6] EIGENMANN, R. Toward a methodology of optimizing programs for high-performance computers. In *Proceedings of the 7th international conference on Supercomputing* (1993), ACM Press, pp. 27–36.
- [7] HALL, M. W., HIRANANDANI, S., KENNEDY, K., AND TSENG, C.-W. Interprocedural compilation of fortran d for mimd distributed-memory machines. In *Proceedings of the 1992 ACM/IEEE conference on Supercomputing* (1992), IEEE Computer Society Press, pp. 522–534.
- [8] JOHNSEN, ANSHUS, B., AND BONGO. Survey of execution monitoring tools for computer clusters.
- [9] KANDEMIR, M., BANERJEE, P., CHOUDHARY, A., RAMANUJAM, J., AND SHENOY, N. A global communication optimization technique based on data-flow analysis and linear algebra. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 21, 6 (1999), 1251–1297.
- [10] KARWANDE, A., YUAN, X., AND LOWENTHAL, D. K. Cc-mpi: a compiled communication capable mpi prototype for ethernet switched clusters. In *Proceedings of the ninth ACM SIGPLAN symposium on Principles and practice of parallel programming* (2003), ACM Press, pp. 95–106.
- [11] LANE, R. G. A comprehensive study of reliable multicast protocols over ethernet-connected networks.
- [12] MATEESCU, G. Parallel sorting on heterogeneous platforms. In *High Performance Computing Systems and Applications, 2002. Proceedings. 16th Annual International Symposium on , 16-19 June 2002* (2002), IEEE, pp. 116–117.
- [13] PUENTE, V., IZU, C., GREGORIO, J. A., BEVIDE, R., PRELLEZO, J. M., AND VALLEJO, F. Improving parallel system performance by changing the arrangement of the network links. In *Proceedings of the 14th international conference on Supercomputing* (2000), ACM Press, pp. 44–53.

- [14] TANG, H., AND YANG, T. Optimizing threaded mpi execution on smp clusters. In *Proceedings of the 15th international conference on Supercomputing* (2001), ACM Press, pp. 381–392.
- [15] W. E. COHEN, W. D. G., AND GAEDE, R. K. Parallel program traces for accurate prediction of proposed cluster performance.
- [16] WAHEED, A., ROVER, D. T., AND HOLLINGSWORTH, J. K. Modeling, evaluation, and testing of paradyn instrumentation system. In *Proceedings of the 1996 ACM/IEEE conference on Supercomputing (CDROM)* (1996), ACM Press, p. 18.
- [17] YUAN, X., MELHEM, R., AND GUPTA, R. Compiled communication for all-optical tdm networks. In *Proceedings of the 1996 ACM/IEEE conference on Supercomputing (CDROM)* (1996), ACM Press, p. 25.
- [18] ZHU, Y., AND HENDREN, L. J. Communication optimizations for parallel c programs. In *Proceedings of the ACM SIGPLAN 1998 conference on Programming language design and implementation* (1998), ACM Press, pp. 199–211.