Scalable Processing and Communication Performance in a Multi-Media Related Context

John Markus Bjørndalen[1], Otto J. Anshus[1]
Tore Larsen[1], Lars Ailo Bongo[1], Brian Vinter[2]

[1)] Department of Computer Science
University of Tromsø

[2)] Department of Mathematics and Computer Science
University of Southern Denmark

Abstract

The PATHS system for configuring an application on one or multiple clusters in a GRID is described and then used on three applications to demonstrate scalability with regards to processing and communication.

The PATHS system use a "wrapper" to provide a level of indirection to the actual run-time location of data. A wrapper specify where data is located, how to get there, and which protocols to use. Wrappers are also used to add or modify methods accessing data. Wrappers are specified dynamically. A "path" is comprised of one or more wrappers. Sections of a path can be shared among two or more paths.

The PATHS system is used to configure a global sum, a wind tunnel, and a video distribution application with the purpose of scaling processing performance when the number of processors increase, and scaling data distribution performance when the number of clients increase.

The performance measurements show that the PATHS system can be used to both scale the processing and communication performance.

1 Introduction

A key challenge when running distributed high performance applications is to establish mappings of processes to hosts that achieve high performance or ef cient execution. Attacking this challenge requires balancing the con icting goals of distributing threads for improved load balancing, while reducing communication and synchronization overheads. High performance and good scalability with respect to processing and communication typically requires manual ne-tuning of the mapping in order to balance the factors signi cantly in uencing the performance.

Mappings may be speci ed by directives in the application source-code, or determined by communication libraries, middleware or the operating system. For the purpose of this paper, and due to space restrictions, we will focus on static mappings given to the application by the programmer.

In [2] three categories of useful tools were found when tuning the performance of NOW-Sort, a parallel disk-to-disk sorting algorithm on a cluster system: tools that help set expectations and con gure the application to different hardware parameters, visualization tools that animate performance counters, and search tools that track down performance anomalies. We are developing similar tools.

This paper describes some components on our work on a middleware extension inspired by method combination[10] and remote procedure calls[3] which allows the communication topology to be directed by specifying meta-code and meta-data, without introducing any modi cations to the application code. The extended middleware also allows computations to be speci ed and executed along the access paths to data. For now, we assume that the application under study runs alone on all hosts in the system; i.e. there are no other applications competing for host resources. The goal of the mapping is to achieve high performance for a particular application having exclusive access to all resources.

We provide tools for specifying and experimenting with load-time mappings, and demonstrate how one through a few experiments may identify exible location policies achieving high performance and good scalability.

Section 2 describes how wrappers are used and combined to specify and identify access paths. Section 3 present the applications we have used to experiment with, section

Proceedings of the 28 th Euromicro Conference (EUROMICRO'02)
1089-6503/02 $17.00 © 2002 IEEE

4 describes our experiments using three different clusters located at the University of Tromsø, Norway and the University of Southern Denmark, Odense. Section 5 presents related work, and section 6 presents our conclusions.

2 PATHS: Con gurable Orchestration and Mapping

Our research platform is PastSet[1][15], a structured distributed shared memory system in the tradition of Linda[7]. A PastSet *element* is a sequence of tuples that are of the same or equivalent type. Tuples can be read from and written to the element using the *move* and *observe* operations. Each element is globally accessible by specifying the element's name.

PATHS[4], is an extension of PastSet that allows for mapping of processes to hosts at load time, selection of physical communication paths to each element, and distribution of computations along the path. Figure 1 shows an example where a path is created between a thread and an element. A thread only references the toplevel stage in the path, and invokes operations through that stage.

Paths can be joined (forming a tree structure) to amortize communication overhead. As an example, gure 2 shows partial sums being computed in each node before being forwarded to the server containing the element.

A thread may use multiple paths to the same element, each of which can be speci ed and built dynamically. Each stage in the path is implemented using a *Wrapper* of a given type.

Since only leaf wrappers are referenced in the application source code, applications can be mapped onto arbitrary cluster con gurations without changing source code or recompiling. Fitting and optimizing applications to any particular con guration is instead done by changing path-specifying meta-data and code.

The wrappers are partly inspired by the PastSet X-functions[16], which are operation modi ers speci ed by the programmer and associated with speci c elements to modify the semantics of the elements operations.

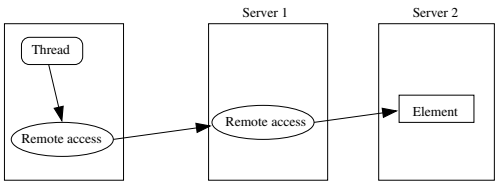


Figure 1. A path between a thread and an element

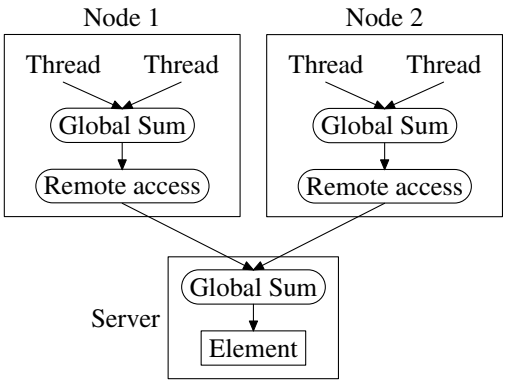


Figure 2. Four threads on two hosts accessing a shared global sum element on a separate server. Each host computes a partial sum that is forwarded to the global-sum wrapper on the server.

2.1 Building and specifying paths

After deciding on a mapping of processes onto hosts, setting up access from one thread to a PastSet element involves the following two stages:

Specify the path. This involves examining information such as process mappings, cluster topologies, and location of target elements.

Build the path from the description. This involves creating and binding wrappers with parameters speci ed in the path description.

Each wrapper in the path description is parameterized. Some parameters are common among all wrappers (such as whether the wrapper need to use thread synchronization mechanisms), whereas others are provided only for speci c wrapper types (i.e. what protocol to use, server address, and service requirements for remote access wrappers).

An example path description used by one of the nodes in Figure 2 is included in Figure 3.

Each thread creates (or is given) its path description, before calling `build_path` which returns reference to the top-level wrapper in the path. `Build_path` checks path descriptions, and merges paths when it is feasible to share portions of the paths.

Pro ling is provided via trace wrappers that log the start and completion times of operations that are invoked through each wrapper. The trace wrappers may also be used to store information about operation parameters, and tuples provided and returned in the operation. Any number of trace wrappers may be inserted anywhere in the path trees.

```
path = make_path(stage("reduce-sum", num_threads=2),
                 stage("remote", proto=TCP, host="p0"),
                 stage("reduce-sum", num_threads=2),
                 stage("core", name="PI-SUM1"))
elm = build_path(path)
```

Figure 3. Example path specifictation and building in Python

Traces stored with the corresponding path speci cations are later read by specialized tools to examine the performance aspects of an application.

3 Applications

The applicability of PATHS is tested using three parallel applications, Global sum, Wind tunnel, and Video.

Global sum computes a global sum that is aggregated from parallel threads providing partial sums. The algorithm works essentially as the MPI *Allreduce* function. Global Sum is used in experimenting with the effects of thread mappings, communication parameters, and hierarchal reduction of the latencies of global reductions. In Section 4.2 we report our ndings from experimenting with a hierarchal global reduction to reduce latencies. Earlier experiments using Global sum can be found in [4].

The **Wind tunnel** application is a Lattice Gas Automaton doing particle simulation. It uses eight matrices in which particles are shifted around to simulate the ow of air. The parallel version splits each matrix into slices, which are then assigned to threads. When running, each thread exchanges the border entries of its slices with threads computing on neighboring slices. We demonstrate how we can map the application to different cluster con gurations and change the mapping to improve the scaling and performance of the system.

The **Video** application demonstrates parallel distribution of one video feed. A feeder application captures images from a frame grabber, converts each image into a jpeg, and stores each jpeg as a tuple in a PastSet element. Each video client uses the *last-observe* wrapper to retrieve the latest available new jpeg. If no jpeg exists that is newer than the ones already observed, last-observe blocks until one arrives. A client program decompresses the images and displays them in a TKinter window.

Organizing the PastSet servers and paths hierarchically, we demonstrate that no jpeg image needs to be transmitted more than once down any wire, that clients which cannot consume images at full speed only retrieves the latest available image (and skip older images), and that faster clients pre-fetch images for the slower clients; essentially turning the last-observe wrapper into a cache.

4 Experiments

Experiments were done using Global sum and Wind tunnel, to study the ability of PATHS to improve processing performance by re-con guring applications at load time. Additional experiments were done using the Video application to study the ability of PATHS to provide scalable distribution of one video-feed.

Wrappers were used to extract performance data which were visualized using our prototype performance data tools[5].

We are now experimenting with the Video Distribution system using the Wind Tunnel application output with the purpose of prototyping a multimedia application scaling well both with regards to computation and data distribution.

4.1 Hardware Platform

The hardware platform consists of three geographically dispersed clusters, each with 32 processors:

- 2W: 16*2-Way Pentium III 450 MHz, 256MB RAM (Odense, Denmark).
- 4W: 8*4-Way Pentium Pro 166 MHz, 128MB RAM, (Tromsø, Norway).
- 8W: 4*8-Way Pentium Pro 200 MHz, 2GB RAM, (Tromsø, Norway).

All clusters run intra-cluster communication using TCP/IP over 100 Mbps Ethernet. 4W has an additional internal 100 Mbps 100 VG-AnyLAN connection that is used for some experiments. Communication between 4W and 8W used a 100 Mbps 100 VG AnyLan connection. All communication between the cluster in Odense (2W) and the two clusters in Tromsø (4W and 8W), uses institutional resources at the respective sites, each country's national research and educational backbone, and the Nordic interconnection of national research networks (NORDUnet). Using the PastSet system, the latency between two threads on different nodes using the 100 Mbps networks is typically around 100-250 microseconds depending on the systems and protocols used. Using the network between Tromsø and Odense, the latency is typically around 40 milliseconds.

4.2 Global Sum Experiments

In the Global sum experiment we measured the average execution time of a global sum computation. The number of threads applied is equal to the number of values to be added. The Global sum experiment is run on 2W only.

```
barrier_sync();
gettimeofday();
TS(0);
for (i = 1; i= < iters; i++)
{
     sum = gsum(i);
     TS(i);
}
gettimeofday();
```

Figure 4. Pseudocode for the Global Sum benchmark. Only one thread runs the timestamp code, all other threads run the same code with the timestamp code removed.

Figure 4 shows pseudocode for the Global sum benchmark. The *TS()* macro samples the Pentium Pro timestamp counter, and stores the timestamp in an array. *Gettimeofday()* samples the real-time clock on the host computer with microsecond resolution. The gsum benchmark was run with an iters of 1.000. For each test, the average execution time of ve runs is plotted on the graphs.

In one experiment we measured the performance using the *even distribution* algorithm. Using this algorithm, threads are distributed as evenly as possible among all hosts in the cluster; i.e. the number of threads on any two hosts differs by at most one. One single thread computes the aggregate of the partial sums computed by every other thread. More elaborate algorithms are investigated in[4].

In [4] we observed that increasing the number of threads per host beyond three or four would reduce the computational ef ciency of each host. This observation suggests that using four threads per sum wrapper and arranging the threads and wrappers hierarchically as shown in 5 gives an organization that maps easily onto the cluster, while limiting the number of threads per wrapper to four. Compared to a at organization, the hierarchical organization offers potential performance improvements due to a higher degree of parallelism.

Figure 6 shows a plot of the execution times using the hierarchical and even organizations as the number of threads increases from one up to the number of CPUs in the cluster. The number of values to be added is increased linearly for each thread added.

Using hierarchical organization, we measure double or

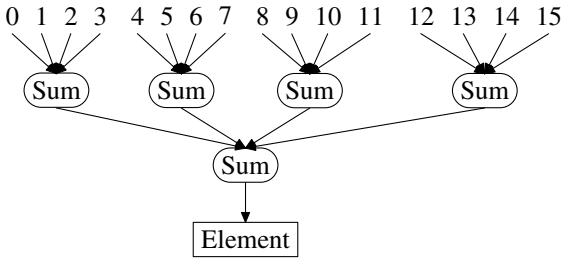
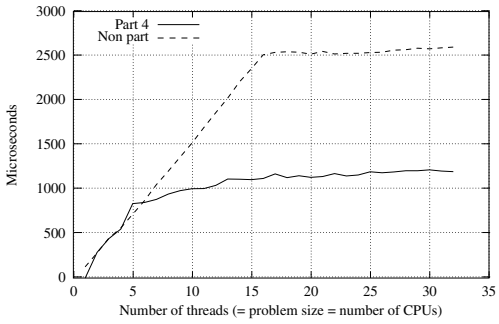


Figure 5. Hierarchial global reduction sum tree. The numbers represent threads. The upper layer of sum wrappers computes partial sums used in the lowermost sum wrapper.



(a) 2-Way cluster

Figure 6. Global sum, cluster partition, even distribution

better performance for twelve or more threads, compared to the even distribution algorithm. Observations from [4] were applied in [16] when comparing the performance characteristics of PastSet using the path framework with MPI[17] (LAM-MPI). Results demonstrated that PastSet was 1.83 times faster than LAM-MPI on global reductions. We are currently experimenting with a PATHS-inspired con guration mechanism for LAM-MPI, and preliminary results show that we are able to signi cantly improve some group operations.

4.3 Wind Tunnel Experiments

We used the Wind Tunnel benchmark to evaluate whether PATHS could be used effectively to experiment with different mappings and identify mappings that scaled well.

Using an even distribution of threads to CPUs, we experienced linear speedups for this application on every cluster by itself, and when combining both clusters in Tromsø (4W

and 8W). Combining the Odense cluster (2W) with any of the Tromsø clusters gave us less than linear speedups.

From an initial even mapping of one thread per 80th Mhz, we used PATHS to quickly set up several variant mappings that we experimented with. For these experiments, each thread carries the same effective workload regardless of mapping.

- Increasing the relative workload at 2W by evenly increasing the number of threads from 12 to 14, reduced the performance.
- Decreasing the relative workload at 2W by evenly decreasing the number of threads from 12 to 10, had no performance effect.
- Decreasing the relative workload at 2W further, by evenly decreasing the number of threads to eight reduced the performance.
- Reducing the load on the single node on 2W that handled all communication external to the cluster, slightly improved the performance.
- Reducing the number of sequential remote operations by moving data to 2W had no performance effect.
- Increasing the problem size to get a higher processing to communication ratio, resulted in improved performance. Effectively scaling our experiments along this approach was limited by the memory size (128 MB) of each host in 4W.

Using a prototype PATHS performance data visualization tool revealed that after some time, the progression of every thread in the clusters was effectively reigned in by the progression of the 2W inter cluster communication node. This phenomenon will be investigated further.

4.4 Video Distribution System

The Video Distribution experiment was designed to investigate if PastSet and PATHS could be applied in quickly developing a video broadcast system where each client that requests to receive the broadcast, receives its feed at a rate it can process, and without loading the server and network by delivering frames that would otherwise be skipped by the client.

The video broadcast system was con gured to have one dedicated root node on each cluster. (see Figure 8). Every participating client thread has a path that goes through a host-local PastSet server, on through the root node in the cluster, and nally to the broadcast server that holds the original video data.

The last-observe wrapper is used to cache frames pulled down from a server further up in the hiearchy. A new frame

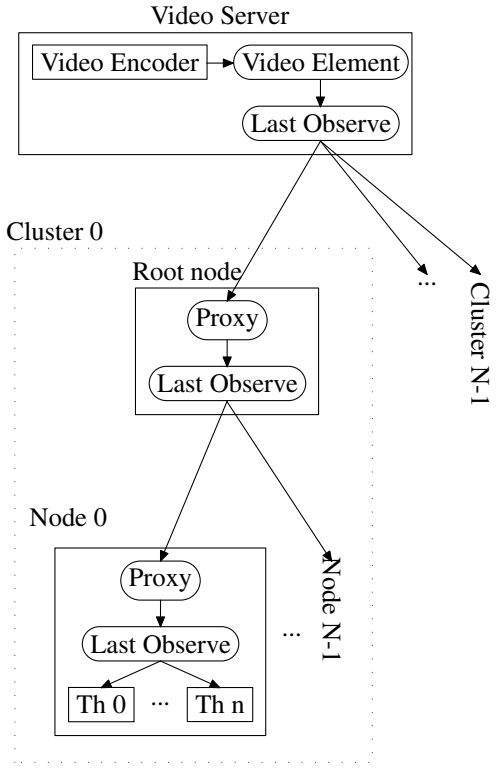


Figure 8. Hierarchial Video Application

is only requested from a server further up when the last-observe wrapper does not have a frame as new as, or newer than the one being requested by a client further down. Only the latest available frame will be returned by the wrapper. Older frames are dropped.

To test the scalability of this design, we started with one client per processor in the clusters and dynamically increased the number of clients while monitoring the frame rate at the server process and at every client process. The server process load was also monitored. The broadcast server was located in Tromsø, but external to 4W and 8W. The stream of jpegs consisted of 320x240 pixel images, delivered at 12.5 Hz. The client processes were evenly mapped across 2W, 4W, and 8W.

At 960 clients there was no noticeable degradation on the server or any client. Every client, including the 320 at 2W in Odense, ran at the full frame rate at 12.5 Hz.

At 2016 client processes, there was still no noticeable degradation of the server or of the processes on 2W in Odense. At this load however, the clients running on the clusters in Tromsø (next room to the video server) did exhibit slight degradation, dropping the frame rate to about 12 Hz on 4W, and about 10-11 Hz on 8W.

We observe that of the two clusters in Tromsø, processes at 4W (166MHz four-way nodes) perform better than processes at 8W (200MHz eight-way nodes) when the number

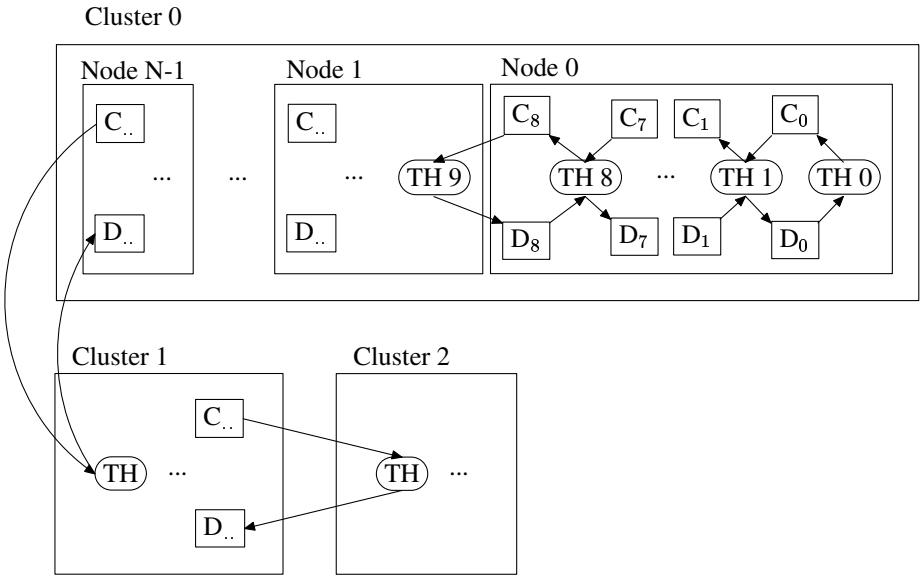


Figure 7. Multi-cluster Wind tunnel Experiment

of client processes is increased beyond a threshold level. We suspect this indicates processor-memory bus contention on the eight-way systems due to the high communication load of this benchmark. Processes at the faster 2W cluster perform best, and are not hindered by the slow Tromsø-Odense connection.

5 Related work

Accurate and ef cient performance prediction of existing distributed and parallel applications on target con gurations with potentially thousands of processors is hard. A number of simulators have been developed, including Parallel Proteus[11], LAPSE[8], SimOS[13], and Wisconsin Wind Tunnel[12]. The slowdowns range from 2 to 100. Few simulators simulate both computation and I/O operations. In contrast to simulators, our approach execute the actual application code several times, each time with a different mapping.

In [9] both processor and memory load balancing are used to support low contention and good scaling to hundreds of processors. Gang-scheduling is used to avoid wasting cycles spinning for a lock held by a descheduled process. In contrast, our system is much simpler and provides for much less or no automatic support at the present time.

In [14] it is shown that there is a communication and load balance trade-off when partitioning and scheduling sparse matrix factorization on distributed memory systems. Block based methods result in lower communication costs and worse load balancing, whereas a "round robin"-based scheme where all threads are distributed over the processors gives better load balance but higher communication costs.

In [18] an approach to load balancing for general-purpose simulations is reported in which little modi cation is needed to the user's code. Their approach uses run-time measurements and demonstrates better load-balancing than approaches without such measurements. This approach is similar to ours in that little modi cation of the user's code is needed. As they do, we also use different mappings and leave it to the application to control them. Our approach differ in that we can both try different mappings and add arbitrary code along the access path to data. Also, we differ in that we do a prerun of a few mappings, and then we choose a single one and we let the application use the selected mapping without incurring further overhead. Of course, we take all the overhead when choosing a mapping. For clusters where they can be dedicated to applications running often, this con guration hunting overhead will be amortizised over time.

In [6] it is shown that dramatic reductions in the bandwidth demand on the underlying server operating system can be gained via application-level data caching. This is in accordance with our work.

6 Conclusion

Fine-tuning the performance of high-performance distributed applications through analytical means or simulation is hard, requiring detailed insights into complicated factors including the tradeoffs and effects of caching, synchronization, locality, load balancing, communication demands, and how network protocols and synchronization mechanisms

have been implemented. Using the PATHS system different mappings of an applications communication and computations can quickly be tried out without changing the application code.

Through several experiments we discovered some of the factors contributing negatively to the performance of a set of applications, and then we remapped the applications to nd con gurations with better performance.

We believe that our system can improve performance by nding a con guration where the resource usage better avoids hot spots, bottlenecks, and expensive waiting times for processor, memory, cache, and I/O by trading between load sharing and communication. The exibility of using maps, paths and wrappers also make it possible to monitor the application and provide data for visualization of both behaviour and performance.

References

[1] ANSHUS, O. J., AND LARSEN, T. Macroscope: The abstractions of a distributed operating system. *Norsk Informatikk Konferanse* (October 1992).

[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. *Proceedings of the SIGMETRICS symposium on Parallel and distributed tools (SPDT 98), USA* (1998), pp. 124 133.

[3] BIRRELL, A. D., AND NELSON, B. J. Implementing remote procedure calls. In *Proceedings of the ninth ACM Symposium on Operating Systems Principles* (1983).

[4] BJØRNDALEN, J. M., ANSHUS, O., LARSEN, T., AND VINTER, B. Paths - integrating the principles of method-combination and remote procedure calls for run-time con guration and tuning of high-performance distributed application. *Norsk Informatikk Konferanse* (November 2001), 164 175.

[5] BONGO, L. A. Steps: A performance monitoring and visualization tool for multicluster parallel programs, June 2002. Large term project, Department of Computer Science, University of Tromsø.

[6] BRADSHAW, M. K., WANG, B., SEN, S., GAO, L., KUROSE, J., SHENOY, P., AND TOWSLEY, D. Periodic broadcast and patching services - implementation, measurement, and analysis in an internet streaming video testbed*. ACM MM'01, Ottawa, Canada.

[7] CARRIERO, N., AND GELERNTER, D. Linda in context. *Commun. ACM 32*, 4 (April 1989), 444 458.

[8] DICKENS, P., HEIDELBERGER, P., AND NICOL, D. Parallel direct execution simulation of message-passing parallel programs. *IEEE Transactions on Parallel and Distributed System* (1996).

[9] GOVIL, K., TEODOSIU, D., AND YONGQIANG HUANG AND, M. R. Cellular disco: resource management using virtual clusters on shared-memory multiprocessors. *ACM Symposium on Operating Systems Principles (SOSP'99), published in Operating Systems Review 34(5)* (December 1999), pp 154 169.

[10] KICZALES, G., DES RIVIERES, J., AND BOBROW, D. G. *The Art of the Metaobject Protocol*. MIT Press, 1991.

[11] LUO, Y. Mpi performance study on the sgi origin 2000. *Pacific Rim Conference on Communications, Computers and Signal Processing* (1997), pp 269 272.

[12] REINHARDT, S., HILL, M. D., LARUS, J., LEBECK, A., J.C, LEWIS, AND WOOD, D. The wisconsin wind tunnel: Virtual prototyping of parallel computers. *Proceedings of the 1993 ACM SIGMETRICS Conference* (May 1993).

[13] ROSENBLUM, M., BUGNION, E., DEVINE, S., AND HERROD, S. Using the simos machine simulator to study complex computer systems. *ACM Trans. On Modeling and Computer Simulation Vol. 7*, No. 1 (January 1997), pp. 78 103.

[14] VENUGOPAL, S., AND NAIK, V. K. Effects of partitioning and scheduling sparse matrix factorization on communication and load balance. *Proceedings of the 1991 conference on Supercomputing* (1991), pp. 866 875.

[15] VINTER, B. *PastSet a Structured Distributed Shared Memory System*. PhD thesis, Tromsø University, 1999.

[16] VINTER, B., ANSHUS, O. J., LARSEN, T., AND BJØRNDALEN, J. M. Extending the applicability of software dsm by adding user rede nable memory semantics. *Parallel Computing (ParCo) 2001, Naples, Italy* (September 2001).

[17] WALKER, D. W. The design of a standard message-passing interface for distributed memory concurrent computers. In *Parallel Computing, Vol. 20*. April 1994, pp. 657 673.

[18] WILSON, L. F., AND NICOL, D. M. Experiments in automated load balancing. *Proceedings of the 10th Workshop on Parallel and Distributed Simulation (PADS '96)* (1996).