SMG: SHARED MEMORY FOR GRIDS

John P. Ryan, Brian A. Coghlan
Computer Architecture Group
Dept. of Computer Science
Trinity College Dublin
Dublin 2, Ireland
email: {john.p.ryan,coghlan}@cs.tcd.ie

ABSTRACT
This paper discusses some of the salient issues involved in implementing the illusion of a shared-memory programming model across a group of distributed memory processors from a cluster through to an entire Grid. This illusion can be provided by a distributed shared memory (DSM) runtime system.
Mechanisms that have the potential to increase the performance by minimizing high-latency intra site messages & data transfers are highlighted. Relaxed consistency models are investigated, as well as the use of a grid information system to ascertain topology information. The latter allows for hierarchy-aware management of shared data and synchronization variables. The process of incremental hybridization is also explored, where more efficient message-passing mechanisms can incrementally replace DSM actions when circumstances dictate that performance improvements can be obtained.
In this paper we describe the overall design/architecture of a prototype system, SMG, which integrates DSM and message passing paradigms and may be the target of an OpenMP compiler. Our initial findings based on some trivial examples indicate some of the potential benefits that can be obtained for grid Applications.

KEY WORDS
DSM, MPI, OpenMP, Grid, Incremental Hybridization

1 Introduction

The message passing programming paradigm provides simple mechanisms to transfer data structures between distributed memory machines to enable the construction of high performance and highly scalable parallel applications. However, there is considerable burden placed on the programmer whereby send/receive message pairs must be explicitly declared and used, and this is often the source of errors. Implementations of message passing paradigms exist for grids [1].
Shared memory is a simpler paradigm for constructing parallel applications, as it offers uniform access methods to memory for all user threads of execution. Therefore it offers an easier way to construct applications when compared to a corresponding message passing implementation. The disadvantage is limited scalability. But nonetheless, vast quantities of parallel codes have been written in this manner. OpenMP, promoted by multiple vendors in the high performance computing sector, has emerged as a standard for developing these shared memory applications [2]. Through the use of compiler directives, serial code can be easily parallelized by explicitly identifying the areas of code that can be executed concurrently. This parallelization of an existing serial application can be done in an incremental fashion. This has been an important feature in promoting the adoption of this standard among parallel application developers.
These are the two predominant models for parallel computing and there are many implementations of both for different architectures and platforms. Shared memory programming has the easier programming semantics, while message passing has more scalability and efficiency (communication is explicitly defined and overheads such as control messages can be reduced dramatically or even eliminated). Previous work [3] examined approaches to combine the message passing and shared-memory paradigms in order to leverage the benefits of both approaches, especially when the applications are executed in an environment such as a cluster of SMPs, but not on grids.
Distributed Shared Memory (DSM) implementations aim to provide an abstraction of shared memory to parallel applications executing on ensembles of physically distributed machines. The application developer therefore obtains the benefits of developing in a style similar to shared memory while harnessing the price/performance benefits associated with distributed memory platforms. Throughout the 1990's research into Software-only Distributed Shared Memory (S-DSM) became popular, resulting in numerous implementations, e.g. Munin [4], Midway [5], Treadmarks [6], and Brazos [7]. However it has met with little success due to poor scalability and the lack of a common Application Programming Interface (API) [8].
As Grids are composed of geographically distributed memory machines, traditional shared memory may not execute across multiple grid sites. DSM offers a potential solution, but its use in a Grid environment would be no small accomplishment as numerous barriers exist to an efficient implementation that could run in such a heterogeneous environment. However, if the paradigm could be made available,

then according to [9], grid programming would be reduced to optimizing the assignment and use of threads and the communication system.
Our aim has been to explore a composition of the environments, with an OpenMP-compatible DSM system layered on top of a version of the now ubiquitous Message Passing Interface (MPI) that can execute in a Grid environment. This choice reflects a desire for a tight integration with the message passing library and the extensive optimization of MPI communications by manufacturers. Such a system would need to be multi-threaded to allow the overlap of computation and communication to mask the high-latency. It has also been shown that the integration of an information and monitoring system can yield substantial gains in the implementation of MPI collective routines [10]. We believe that following a similar approach and integrating an information system into the DSM will result in performance improvements by allowing optimizations to, for example, per-site caching & write collection of shared data, and enable communication-efficient synchronization routines such as barriers. Hence, a further aim has been to use the information system to create topology/hierarchy awareness and runtime support of user applications, and to insulate the developer from this, except where they wish topology information to be exposed to the user application code to allow for optimizations at the application level.
A primary concern has been to minimize the use of the high-latency communication channels between participating sites where possible, and to favour a lower message count with higher message payload where not, and also to not require hardware support for remote memory access.

2 Shared Memory for Grids

The premise of grid computing is that distributed sites make available resources for use by remote users. A grid job may be run on one or more sites and each site may consist of a heterogeneous group of machines. In order to reduce the potential number of variations necessary, the DSM implementation should only use standard libraries such as the POSIX thread, MPI, and standard C libraries. No special compiler or kernel modifications should be required.
The system must present the programmer with an easy-to-use and intuitive API in order that the associated burden in the construction of an application is minimal. This requires that the semantics must be as close to that of shared memory programming as possible, and that it should borrow from the successes and learn from the mistakes of previous DSM implementations. All function calls therefore belong to one of the three following groups.

- Initialization/finalization
- Memory allocation
- Synchronization

For a DSM to gain acceptance, compliance with open standards is necessary, so it is also preferable that the DSM form the basis for a target of a parallelizing compiler, such as OpenMP. There are other projects have adopted this approach [11]. By examining some OpenMP directives one can identify some of the design requirements of the DSM. The following is an OpenMP code snippet that implements the multiplication of two matrices.

```
 /*** Begin parallel section ***/
#pragma omp for
for (i = 0; i < ROWSA; i++){
  for(j = 0; j < COLSB; j++){
    c[i][j] = 0;
    for (k = 0; k < COLSA; k++)
    {
      c[i][j] = c[i][j] +
                a[i][k] * b[k][j];
    }
  }
} /*** End parallel section ***/
```

Barriers are used implicitly in OpenMP, where any thread will wait at the end of the structured code block until all threads have arrived, except, for example, where a *nowait* clause has been declared. In order for concurrency to be allowed inside the parallel section the shared memory regions must be writable by multiple writers.
Mutual Exclusion is required in OpenMP. The main mutual exclusion directives are as follows.

```
/*** Only the master thread (rank 0)
     will execute the code  ***/
#pragma omp master
{...}

/*** Only one thread will execute the
     code ***/
#pragma omp single
{...}

/*** All threads will execute the
     code, but only one at a time ***/
#pragma omp critical
{...}
```

These imply that a distributed mutual exclusion device (or lock) is required to implement the latter of these directives. The first two are functionally equivalent and can be implemented using simple if-then-else structure.

2.1 Memory Consistency

The primary goal of minimizing communication between nodes is only achievable if a relaxed consistency model is employed, where data and synchronization operations are

clearly distinguished and data is only made consistent at synchronization points. The most common are the Lazy-Release (LRC), Scope (SC), and Entry Consistency (EC) models. The choice of which to use involves a trade-off between the complexity of the programming semantics and the volume of overall control and consistency messages generated by the DSM.

- **Release consistency** requires that the shared memory areas are only consistent after a synchronization operation occurs [12]
- **Lazy-Release consistency** is an extension of release consistency where modifications are not propagated to a process until an acquire operation has been performed by that process, and was first implemented in the Treadmarks DSM [6].
- **Entry consistency** is similar to (lazy)release consistency but more relaxed; shared data is explicitly associated with synchronization primitives and is made consistent when such an operation is performed. EC was first implemented in the Midway DSM [5].
- **Scope consistency** implements a scenario that lies between LRC and EC, whereby data is not explicitly associated with synchronization variables, but the bindings implied by the programmer are dynamically detected [13].

The selection of the consistency model is a prime determinant in the overall performance of the DSM system. Studies show that entry consistency and lazy release consistency can on average produce the same performance on clusters of computers of a modest size [14], the application's access pattern to shared memory being a determining factor. Development involving entry consistency introduces additional programming complexity, but [14] demonstrates that EC may generate an order of magnitude less inter node messages for the same application. This is an important consideration if an application is executing across multiple sites with high-latency communication links.
Multiple-writer protocols attempt to increase concurrency by allowing multiple writes to locations residing on the same coherence unit. It has been demonstrated that significant performance gains can be achieved by employing this technique [15]. This addresses the classical DSM problem of false sharing. Multiple-writer entry consistency attempts to combine the benefits of both models while addressing associated problems that have been identified for LRC [16] and EC [15] protocols. In order to provide true concurrency then this protocol must be provided by the DSM.

2.2 Communication

Exploring the use of MPICH [17] for the communication between processes executing on distributed nodes allows for the exploitation of an optimized and stable message passing library, and leveraging of useful MPI resources such as profiling tools and debuggers. Its use also insulates the system from platform dependencies and will ease porting to other architectures and platforms in the future.
Unfortunately the current Grid enabled version of MPICH, MPICH-G2 [18], is based on the MPICH distribution (currently version 1.2.5.2), which has no support for multi-threaded applications. This makes hybridizing hard, as the DSM system thread requires the MPI communication channel, and so can only be used in MPI_THREAD_FUNNELLED mode. Implementations exist that provide a thread-safe MPI implementation, however they are not grid-enabled. Future MPI implementations can be expected to support multi-threading.

2.3 Information & Monitoring

There are many tools for the monitoring of message passing applications, such as MPICH's Jumpshot, but these lack support for execution across geographically dispersed sites. Alternatively, one can rely on a grid information system.
The MDS component of the Globus Toolkit is based on the hierarchical LDAP system. It has been shown that when MDS is integrated with a MPI implementation, dramatic improvements are obtainable in the performance of a number of MPI collective operations through hierarchy awareness [19, 10].

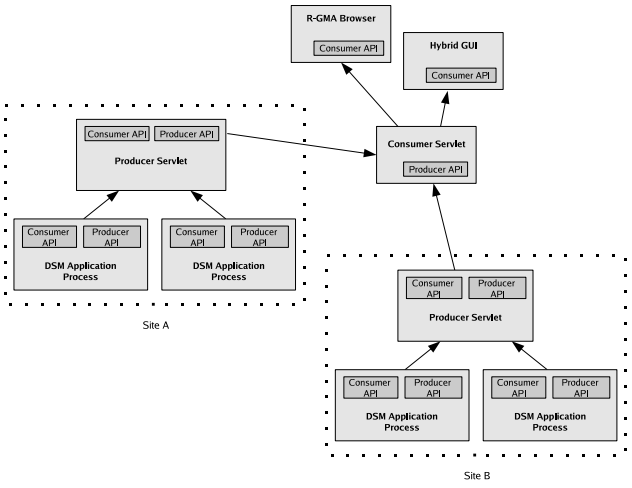

Figure 1. Hierarchy Information using R-GMA

The Relational Grid Monitoring Architecture (R-GMA) [20], is a relational implementation of the GGF GMA [21], where information is published via producers and accessed by consumers as shown in Figure 1. R-GMA has successfully been used to enable the monitoring of MPI applications in a grid environment [22].
Both MDS and R-GMA adhere to the GLUE schema [23]. Either (or both) are a good basis for hierarchy awareness.

3 Implementation

The prototype DSM system is called SMG (Shared Memory for Grids). As noted previously it utilises the message passing library for communication between processes. The SMG API implementation draws on previous DSM APIs and consists of initialization & finalization function (essentially wrappers around the complementary MPI calls), shared memory allocation functions, and synchronization operations (lock and barrier). No special compilers or kernel modifications are required.

3.1 Coherency and Consistency

EC increases the programming burden when compared with other consistency models such as LRC. However, its semantics lends itself well to targeting by an OpenMP compiler and for this reason as well as its lower message volumes it is the consistency protocol of choice. Write-update coherency is naturally employed when an EC protocol is used. Multi-writer EC is supported where the shared data is associated with a barrier synchronization primitive while lock primitives support a single-writer.
The implementation of write-trapping and write collection follows a similar approach to that adopted in Treadmarks [6] with some small modifications. Write-trapping is achieved by setting the protection level of the shared object, where the minimum granularity is at the virtual page level up to the total size of the object if it occupies more than one page. Upon the first write to a variable in a shared region a twin is generated.
When the synchronization object that the shared memory object is bound to performs a release a diff is generated by comparing the twin and the current state. This is used to minimize the message traffic for coherence updates. If topology information is available then hierarchy awareness can be applied in barrier operations, i.e. as arrival notifications are received the diffs can be merged at intermediate nodes (as depicted in Figure 2), thus reducing the processing bottleneck at the root node level. Otherwise, a traditional tree-structured barrier is employed.
Multi-writer entry consistency can be easily supported for barrier primitives and this is equivalent to OpenMP parallel *for* constructs. Basic user-specified alteration to write trapping and write collection methods are allowed, and when more integration with the information system occurs, this user control will assist application-level optimizations where access patterns to shared data are irregular [24].

3.2 Environment Awareness

The information system is required if the DSM is to be hierarchy-aware, otherwise the topology assumes the flat model typical of an MPI application. The SMG DSM prototype uses R-GMA, principally for its support for relational queries and dynamic schemas. It uses the GLUE

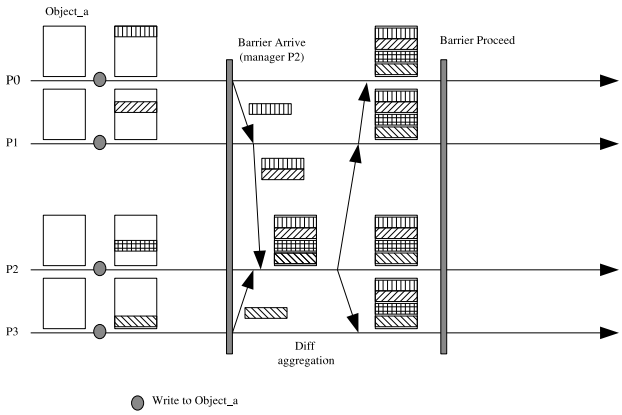

Figure 2. MW-EC Barrier write-collection

schema to obtain environmental information from GLUE compliant R-GMA producers. To enable monitoring of the user applications, SMG schemas are defined so that monitoring information can be published using the R-GMA producer API, and viewed using the standard R-GMA browser. Nonetheless the information system is hidden by wrappers, and other systems, such as MDS, can be used by implementing the required wrapper APIs.
Some of the benefits of environmental awareness include better global barrier implementation, load-balancing using per-site diff-merges, and the capability for runtime monitoring of applications in a similar manner to GRM [22].

4 Performance

The SMG DSM is currently being tested. As there is no grid-enabled MPI implementation that currently supports multi-threaded applications, system performance measurements are not representative, since the DSM system thread is unable to avail of blocking receive MPI calls.
Despite the lack of multi-threaded MPI, the usefulness of some the schemes we employ such as hierarchy-awareness, and alterable write trapping/collection techniques can be seen.
Coherence overheads are comparable with other DSM implementations but this will be improved when environmental awareness techniques are further improved.
There is significant memory overhead associated with any DSM, as not only are the storage requirements for twinning/diffing excessive, but so also are the storage requirements for the update catalogue for shared memory regions. This can be overcome at the expense of increased coherence message volumes.

4.1 Incremental Hybridization

Using the information system allows for the profiling of user applications, and potentially allows for tools for dead-

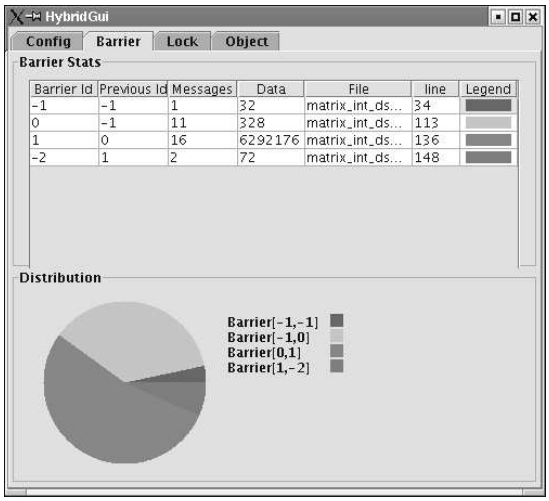

Figure 3. Monitoring Interface

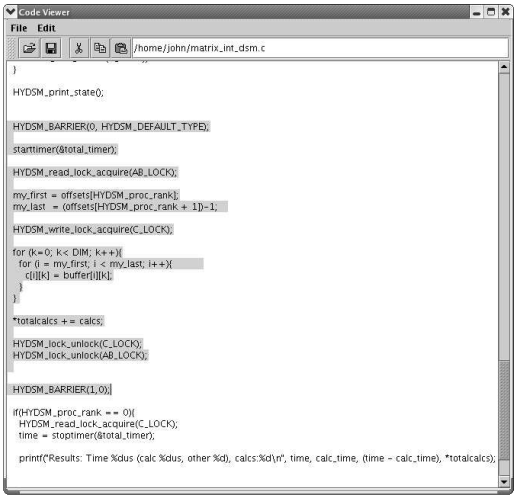

Figure 4. Code Identification

lock detection. We have constructed a user interface that can identify locations in an application where data access behaviour results in severe performance penalties. As we can monitor the variables and the code areas where these are used we can direct hybridization whereby the user replaces shared memory code with message passing, resulting in performance gains. This can be done in a localized fashion, possibly in conjunction with parallelization with OpenMP.

The hybridization GUI works by directing the user to the locations where the majority of communication occurs (so identifying the participating processes), and also to the shared variables that are responsible for the communication. The code browser illustrates the 'hot-spots' in the application and the the objects responsible. This allows for incremental and localized optimisations.

The developer can examine the interval between synchronization operations, such as between barriers, and view the amount of messages/data transfer generated between processes. The amount of communication between barriers in a matrix multiplication example is clearly illustrated in the monitoring interface in Figure 3, and the the actual application code that is responsible can be identified as in Figure 4. From other information gathered during the interval we can identify the memory objects causing the communication.

Another feature is the highlighting of fragmented shared memory use which may be the result of poor algorithm design/implementation. This can be identified when a shared memory region is repeatedly modified by multiple processes in a fragmented fashion. Such a scenario may occur in matrix multiplication when row and column order are mixed up by a developer who is unaware of the potential differences between C and Fortran languages.

If message passing and shared memory paradigms are used together for the same variables, then the shared regions must be made consistent after message passing is used.

5 Related work

DSM implementations such as Treadmarks, Munin, and Midway were written for the realm of compute clusters, not for Grid computing. Efforts at making hierarchy-aware DSM consistency protocols do exist [25], as well as efforts at implementing mixed mode applications composed of OpenMP and MPI codes [26]. The latter enables the exploitation of clusters of SMPs where shared memory is used for intra-node communication and message passing for inter-node communication.

Other research has attempted to allow OpenMP to run on distributed memory machines [3]. There have been other attempts at providing a shared memory model for wide area computing [27], and efforts are underway to implement the MPI 2 standard that includes specifications for remote memory access and one sided communications.

6 Conclusions and Future Work

Grid computing is starting to impact on mainstream parallel computing. If this trend is set to continue then more advanced tools and development environments must be implemented. We believe that there will be numerous approaches to constructing grid applications be it message passing, distributed shared memory and shared memory. Clearly none of these in isolation will provide the perfect fit, but rather an ensemble.

In the SMG system, we are attempting to demonstrate the potential advantages when message-passing and DSM programming paradigms are combined in the grid environment. The goal is to reduce the programming burden, and allow it to be followed by incremental optimisation. If this is achieved, it will promote the use of grids by allowing the exploitation of the vast collection of shared memory codes, and allow for easier parallelization/grid-enabling of serial

codes by using OpenMP directives.

Future support for heterogeneity is vital in order to further the cause. The problem is non-trivial, as shared data would need to be strongly typed, thus significantly increasing burden on the programmer. In addition, support for fault-tolerance is vital. Currently this is an active research field, FT-MPI [28] being one such project.

Future incremental hybridization would benefit from working in tandem with a source code control framework such as CVS, so that it could allow semi-automatic hybridization with backtracking as needed, perhaps guided by predicate logic.

It is clear that programming for the grid will be a challenge and that shared memory will ultimately assist. Here we have made a start.

References

[1] I. Foster and N. Karonis. A Grid-Enabled MPI: Message Passing in Heterogeneous Distributed Computing Systems. In *Proceedings of SC'98*. ACM Press, 1998.

[2] R. Chandra, L. Dagum, D. Kohr, D. Maydan, J. McDonald, and R. Menon. *Parallel Programming in OpenMP*. Morgan Kaufmann Publishers, 2001.

[3] E. Speight, H. Abdel-Shafi, and J. Bennett. An Integrated Shared-Memory/Message Passing API for Cluster-Based Multicomputing. In *Procs. of the Second International Conference on Parallel and Distributed Computing and Networks (PDCN)*, Brisbane, Australia, 1998.

[4] J. K. Bennett, J. B. Carter, and W. Zwaenepoel. Munin: Distributed Shared Memory Based on Type-Specific Memory Coherence. In *Proc. of the Second ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming (PPOPP'90)*, pages 168–177, 1990.

[5] B. Bershad, M. Zekauskas, and W. Sawdon. The midway distributed shared memory system. In *Procs. of COMPCON. The 38th Annual IEEE Computer Society International Computer Conference*, pages 528–537, Feb 1993.

[6] P. Keleher, S. Dwarkadas, A. L. Cox, and W. Zwaenepoel. TreadMarks: Distributed Shared Memory on Standard Workstations and Operating Systems. In *Proc. of the Winter 1994 USENIX Conference*, pages 115–131, 1994.

[7] W. E. Speight and J. K. Bennett. Brazos: A Third Generation DSM System. In *Proc. of the USENIX Windows NT Workshop*, 1997.

[8] J. B. Carter, D. Khandekar, and L. Kamb. Distributed Shared Memory: Where We Are and Where We Should Be Headed? In *Fifth Workshop on Hot Topics in Operating Systems (HotOS-V)*, pages 119–122, 1995.

[9] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan-Kaufmann, 1999.

[10] N. T. Karonis, B. R. de Supinski, I. Foster, W. Gropp, E. Lusk, and J. Bresnahan. Exploiting Hierarchy in Parallel Computer Networks to Optimize Collective Operation Performance. In *Proc. of the 14th International Conference on Parallel and Distributed Processing Symposium (IPDPS-00)*, pages 377–386, 2000.

[11] H. Lu, Y. C. Hu, and W. Zwaenepoel. OpenMP on network of workstations. In *Proc. of Supercomputing'98*, 1998.

[12] K. Gharachorloo, D. Lenoski, J. Laudon, P. Gibbons, A. Gupta, and J. Hennessy. Memory Consistency and Event Ordering in Scalable Shared-Memory Multiprocessors. In *25 Years ISCA: Retrospectives and Reprints*, pages 376–387, 1998.

[13] L. Iftode, J. P. Singh, and K. Li. Scope Consistency: A Bridge between Release Consistency and Entry Consistency. In *Proc. of the 8th ACM Annual Symp. on Parallel Algorithms and Architectures*, pages 277–287, 1996.

[14] S. Adve, A. Cox, S. Dwarkadas, R. Rajamony, and W. Zwaenepoel. A Comparison of Entry Consistency and Lazy Release Consistency Implementations. In *Proceedings of the Second High Performance Computer Architecture Conference*, pages 26–37, Feb 1996.

[15] H. S. Sandhu, T. Brecht, and D. Moscoso. Multiple Writers Entry Consistency. In *Proc. of the Int'l Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA'98)*, volume I, pages 355–362, 1998.

[16] C. Amza, A. Cox, S. Dwarkadas, and W. Zwaenepoel. Software DSM Protocols that Adapt between Single Writer and Multiple Writer. In *Proc. of the 3rd IEEE Symp. on High-Performance Computer Architecture (HPCA-3)*, pages 261–271, Feburary 1997.

[17] Homepage of MPICH MPI distribution. `http://www-unix.mcs.anl.gov/mpi/mpich`.

[18] N. Karonis, B. Toohen, and I. foster. MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface. *Journal of Parallel and Distributed Computing*, 63(5):551–563, May 2003.

[19] T. Kielmann, R. F. H. Hofman, H. E. Bal, A. Plaat, and R. A. F. Bhoedjang. MagPIe: MPI's collective communication operations for clustered wide area systems. *ACM SIGPLAN Notices*, 34(8):131–140, 1999.

[20] S. Fisher et al. R-GMA: A Relational Grid Information and Monitoring System. Technical Report WP3-2003-01-14, 2nd Cracow Grid Workshop, DATAGRID, Jan 2003.

[21] R. Aydt, D. Gunter, W. Smith, M. Swany, V. Taylor, B. Tierney, and R. Wolski. A Grid Monitoring Architecture. Technical Report gwd-perf-16-1, GGF, Jul 2001.

[22] N. Podhorszki and P. Kacsuk. Monitoring Message Passing Applications in the Grid with GRM and R-GMA. *Recent Advances in Parallel Virtual Machine and Message Passing Interface (PVM/MPI)*, pages 603–610, 2003.

[23] The GLUE Schema. `http://www.cnaf.infn.it/~sergio/datatag/glue/index.htm`.

[24] C. Amza, A. Cox, S. Dwarkadas, L.-J. Jin, K. Rajamani, and W. Zwaenepoel. Adaptive Protocols for Software Distributed Shared Memory. *Journal of the IEEE, Special Issue on Distributed Shared Memory*, pages 467–475, Mar 1999.

[25] G. Antoniu, L. Boug, and S. Lacour. Making a DSM Consistency Protocol Hierarchy-Aware: an Efficient Synchronization Scheme. In *Proc. Workshop on Distributed Shared Memory on Clusters (DSM'03)*, pages 516–523, May 2003.

[26] L. Smith and M. Bull. Development of Mixed Mode MPI / OpenMP Applications. *Scientific Programming*, 9:83–98, 2001. Presented at Workshop on OpenMP Applications and Tools (WOMPAT 2000), San Diego, Calif., July 6-7, 2000.

[27] D. Chen, C. Tang, X. Chen, S. Dwarkadas, and M. L. Scott. Multi-level Shared State for Distributed Systems. In *Procs. of 31st Int. Conference on Parallel Processing (ICPP'02)*, August 2002.

[28] G. Fagg and J. Dongarra. FT-MPI:Fault Tolerant MPI, Supporting Dynamic Applications in a Dynamic World. In *EuroPVM/MPI 2000*, pages 346–353. Springer-Verilag, 2000.