Kerrighed: A SSI Cluster OS Running OpenMP

David Margery[1], Geoffroy Vallée[2], Renaud Lottiaux[1],
Christine Morin[1], Jean-Yves Berthou[2]
PARIS research group, IRISA, Campus de Beaulieu
35042 Rennes Cedex, France
{dmargery,gvallee,rlottiau,cmorin}@irisa.fr,jy.berthou@edf.fr

Abstract

Writing parallel programs for clusters of workstations is still a challenging task. In this paper, we present Kerrighed, a Single System Image (SSI) operating system giving the illusion of an SMP machine, and providing the standard posix thread interface to developers. It is therefore possible to use Kerrighed to run OpenMP programs compiled for SMP-machines using the posix thread interface. In this paper, we explain how we managed to achieve that goal, and present the benefits of providing OpenMP support through the SSI approach as opposed to a dedicated run-time environment.

1 Introduction

Clusters of SMPs are attractive for executing shared memory parallel applications but reconciling high performance and ease of programming remains an open issue. A possible approach is to provide a dedicated run-time environment implementing a common parallel programming environment such as MPI, OpenMP, Corba or one of their extensions. The problem with this approach lies in the coexistence on a same cluster of different environments that might take conflicting decisions in respect to the shared resources. An other problem is the management of the different levels of parallelism available : at the node level between the different processors and at the cluster level between the different nodes.

Providing an efficient Single System Image (SSI) operating system giving the illusion of an SMP machine is an interesting alternative to this approach, because programming for such a target is one of the simplest form of parallel programming. Moreover, it is rather simple to provide implementations of the common run-time environments used in parallel programming as implementations targeting SMP machines are often available. All these run-time environments will use the same shared resources provided by the system,

[1] IRISA/INRIA

[2] EDF, Research and development division, 1 av du Général de Gaulle, 92141 Clamart cedex, France

and therefore be globally managed in a non-conflicting manner at the cluster level.

In this paper, we focus on the way OpenMP programs are supported on Kerrighed, our attempt at building such an SSI operating system. We believe that using OpenMP to program parallel applications on Kerrighed is a promising development path, as usage of OpenMP enables the production of a working parallel program in relatively short times. Of course, this first working version will certainly not be very efficient, but applying pertinent transformations (such as those suggested in [5]) to the OpenMP program could enable the production of a version efficient enough if high performance is not a high priority or necessity for the considered program. Moreover, the cost of distributed operations on a cluster being an order of magnitude higher than on a genuine SMP, deployment of the parallel application programmed using OpenMP on a cluster could help identify the portions of code that need tuning. Here usage of a cluster giving the illusion of an SMP is used as a way of exaggerating problems encountered when parallelizing programs, either for profiling or in a pedagogic environment.

The paper is structured in the following way. In Section 2, we present the process of running an OpenMP program on Kerrighed, from source to system concepts involved in correct execution. In Section 3, results are presented demonstrating correct execution of OpenMP programs and giving an idea of the performance obtained. Section 4 compares our approach to related work and Section 5 concludes.

2 Running an OpenMP Program on Kerrighed

2.1 Compilation

The first step of running an OpenMP application on Kerrighed is to compile it. For our purposes, we need to use a compiler producing code targeted at an SMP machine using posix threads. We choose to experiment with Omni, the compiler produced by RCWP[8]. This compiler was modified to link the program resulting of compilation with our implementation of posix threads, a library called *gthreads*. This library implements the posix thread interface using the system services offered by Kerrighed, which is presented more completely in the following section.

2.2 Overview of Kerrighed

Kerrighed is composed of a set of distributed services (see figure 1) providing global management of different logical resources in the cluster as described in more details in [9] and [13]. Gandalf module is in charge of global memory management, Aragorn module is in charge of global process management, Elrond module provides a set of synchronization tools for parallel applica-

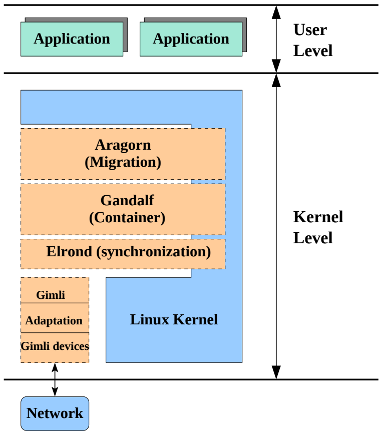

Figure 1: *Kerrighed architecture*

tions (atomic counters, locks, barriers...). All these services are based on Gimli for their communications. Gimli is a service providing high performance communication inside the cluster and a complete abstraction of the communication system to Kerrighed that makes it portable on various interconnection networks. A generic Gimli device has been implemented to make the system completely independent from the network technology.

All Kerrighed services are implemented outside the Linux kernel using the standard module mechanism.

2.3 Support for Shared Variables

Running OpenMP programs compiled for SMP machines implies some support for shared variables, usually through shared memory. With Kerrighed, the Gandalf module performs global memory management at the system level. More precisely, each node executes its own operating system kernel (*host operating system*), which can be coarsely divided into two parts: (1) *system services* and (2) *device managers*. For global memory management, Kerrighed uses a generic service inserted between the system services and the device managers layers called *container* [10]. Containers are integrated in the core kernel thanks to *linkers*, which are software pieces inserted between containers and existing device managers and system services. The key idea is that containers give the illusion to system services that the cluster physical memory is shared as in an SMP machine. They therefore implement a Distributed Shared Memory (DSM) inside the host operating system. This DSM implements sequential coherence with a page granularity ; any memory allocated for a process is naturally shared between different threads of that process running on different nodes, as memory allocation is a system service that allocates memory from containers. For more details please refer to [13].

Therefore, Gandalf enables the sharing of the complete address space of a process, including the stack of each of its threads. This differs from most DSM systems where explicit memory allocation must be used to enable sharing, and greatly simplifies the implementation of an OpenMP compiler, or, as we have done, enables the use of an OpenMP compiler targeted for SMP systems.

2.4 Support for Private Variables

Nevertheless, sharing the complete address space between all the threads of a process isn't always a good property, especially when implementing private variables for parallel sections. For OpenMP programs such variables can be detected at the compilation level and thus redeclared for each thread, or they can be implemented using the thread_private data mechanism offered by posix threads. Despite the fact that Omni uses the former method, we choose to implement *thread_private data* in order to support any OpenMP compiler that would choose to use that facility. As threads are distributed throughout the cluster by Kerrighed, they are implemented as processes of the host operating system (in a manner similar to LinuxThreads, the standard posix thread implementation of Linux). Therefore, explicit work has to be performed at the operating system level so that memory mapped in one host process can be seen by all other host processes that implement the threads of the running program. By extending extending the *mmap* system call to support local memory (by adding a `MAP_LOCAL` flag), it it possible to ensure that memory will be allocated in the virtual address space of each thread of the process (each host process) without linking it to a container, thus ensuring that no coherence will be maintained between the different copies of that memory segment across the different threads. It becomes therefore trivial to implement thread specific memory regions, because the same virtual address will designate a different memory region on each thread.

2.5 Support for Synchronization Mechanisms

All the synchronization mechanisms necessary for support of the posix thread synchronization primitives are directly provided by the Elrond module. This module provides distributed locks, barriers, semaphores, and wait condition to processes in a implementation that supports process checkpointing as well as migration between nodes. These mechanisms are implemented at the process level and not at the node level.

3 Results

3.1 Correctness

With the current version of our prototype (Kerrighed 0.65) and its associated posix thread library (*libgthread*), we were able to run, on four nodes and using 4 OpenMP threads, the 288 tests provided with the Omni compiler to make sure it's installation was correctly completed.

We believe correct execution of these tests is sufficient proof that the mechanisms provided by Kerrighed to support execution of OpenMP programs are mature and provide the requested service.

Of course, initial performance measurements are of great interest to the OpenMP community. Before presenting these results, we would like to stress once more that high-performance execution of OpenMP programs hasn't been the focus of the work presented here. It should be evident to the reader that using a general purpose OpenMP compiler targeting SMP machines using a posix compliant thread library won't result in efficient code when executed on a SSI operating system. Indeed, specific work should be done to tailor the compiler to take into account the specifics of an SSI operating system for clusters before significant results can be produced. We nevertheless believe that providing such results demonstrates that Kerrighed can already be used as a platform to support research of better OpenMP compilation algorithms targeting SSI clusters.

3.2 Performance

For performance measurements, we have used the HRM1D code developed by Electricité De France Research and Development division. This code computes critical flashing water flows using the homogeneous relaxation model, and its parallelization is described in [3]. The measurements where done on a PC cluster of 4 nodes with 512 MB of memory and a Pentium III 500Mhz connected throught standard Ethernet 100 technology. The results are given in Table 1 and can be compared to those obtained on a 4-way SMP running on Pentium 3 at 550Mhz presented in Table 2

number of OpenMP threads	execution time in seconds
1	19,78
2	31,57
4	42,71

Table 1: Execution times for HRM1D using Kerrighed

As expected using an OpenMP compiler not tailored to the specifics of cluster systems, these result show a significant slowdown and the same kind

number of OpenMP threads	execution time in seconds
1	24,19
2	12,24
4	6.68

Table 2: Execution times for HRM1D using a genuine SMP

of results can be measured on the NAS benchmarks. Nevertheless, these tests produce correct results, and as Kerrighed and the compiler are tied in no particular fashion, other optimized compilers could be used. This will be the focus of our future work.

4 Related Work

A few people have explored running OpenMP programs on clusters. Most notable efforts include work targeting a cluster using an extended version of TreadMarks [1, 6, 4] as a DSM and the work of M. Sato that has led to different versions of the Omni [8, 11] compiler we used. These works focus on efficient execution of OpenMP programs on clusters and both conclude that the OpenMP standard must be extended to enable such high performance execution.

With Kerrighed, we have taken an other approach. Our goal is to provide the user of a cluster the illusion he is using an SMP machine as well as providing the tools useful for analyzing the performance bottlenecks of parallel programs running on the cluster. Indeed, we believe high performance execution of parallel programs can not be achieved without devoting significant time to fine-tune the parallelization algorithms used. Therefore, we believe that it is important to provide a programming environment where is is relatively simple to get a parallel application up and running and the tools to tune it. In that respect, we believe OpenMP is a good candidate as it provides a high level of abstraction.

Different OpenMP compiler techniques targeted to clusters have also been studied [12, 7]. Sven Karlsson *et al.* focus on reducing the cost of using a software DSM by minimizing the synchronization points produced by their compiler, while still providing a compliant compiler. Seung Jai Min *et al* focus on optimizations involving access patterns to the shared memory, especially *page-aware optimizations*. These works are very important to us as we believe a cluster aware compiler would produce much more efficient code than the one we have used as proof of concept.

5 Conclusion and Future Work

Our primary goal in developing the Kerrighed SSI operating system for clusters is to give the illusion of an SMP machine to programs running on the cluster as well as providing mechanisms for fault-tolerance. It is thus possible to checkpoint applications running on Kerrighed [2], as well as to migrate threads or processes between nodes according to a global scheduling policy [14].

Therefore, OpenMP programs running on Kerrighed automatically gain these capabilities, adding to the interest of our approach to running OpenMP programs. It is also possible to run several OpenMP programs in a concurrent way, thus increasing the global use of the computing resources. Resource management at the operating system level can then make decisions based on the state of the whole system and not only on the state of a single program.

Future work on OpenMP on Kerrighed should explore compilation methods than produce more efficient code for OpenMP programs as well as providing OpenMP developers with tools to better understand the performance bottlenecks of their applications so that they can tune the parallelization algorithms used.

References

[1] C. Amza, A. L. Cox, S. Dwarkadas, P. Keleher, H. Lu, R. Rajamony, W. Yu, and W. Zwaenepoel. TreadMarks: Shared memory computing on networks of workstations. *IEEE Computer*, 29(2):18–28, 1996.

[2] R. Badrinath, C. Morin, and G. Vallée. Checkpointing and recovery of shared memory parallel applications in a cluster. In *Proc. Intl. Workshop on Distributed Shared Memory on Clusters (DSM 2003)*, pages 471–477, Tokyo, May 2003. Held in conjunction with CCGrid 2003. IEEE TFCC.

[3] J.-Y. Berthou and E. Fayolle. Comparing OpenMP, HPF and MPI programming a studycase. *The International Journal of High Performance Computing Applications*, 15(3):297 – 309, August 2001.

[4] Y. C. Hu, H. Lu, A. Cox, and W. Zwaenepoel. OpenMP for networks of SMPs. In *Proc. of the Second Merged Symp. IPPS/SPDP 1999*, 1999.

[5] G. Krawezik and F. Cappello. Performance comparison of MPI and three OpenMP programming styles on shared memory multiprocessors. In *ACM SPAA 2003*, San Diego, USA, June 2003.

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

[7] S. J. Min, A. Basumallik, and R. Eigenmann. Supporting realistic OpenMP applications on a commodity cluster of workstations. In *OpenMP Shared Memory Parallel Programming: International Workshop on OpenMP Applications and Tools, WOMPAT 2003, Toronto, Canada, June 26-27, 2003. Proceedings*, pages 170 – 179.

[8] Mitsuhisa Sato, Shigehisa Satoh, Kazuhiro Kusano, and Yoshio Tanaka. Design of OpenMP compiler for an SMP cluster. In *EWOMP'99*, Lund, Sweden, october 1999.

[9] C. Morin, P. Gallard, R. Lottiaux, and G. Valle. Towards an efficient single system image cluster operating system. In *Proc. of International Conference on Architecture and Algorithms for Parallel Processing (ICA3PP 2002)*, pages 370–377, Beijing, China, Oct. 2002. IEEE Computer Society.

[10] R.Lottiaux and C.Morin. Containers : A sound basis for a true single system image. In *Proceeding of IEEE International Symposium on Cluster Computing and the Grid*, pages 66–73, May 2001.

[11] M. Sato, H. Harada, and Y. Ishikawa. OpenMP compiler for a software distributed shared memory system SCASH. In *WOMPAT2000*, July 2000.

[12] M. B. Sven Karlsson, Sung-Woo Lee. A fully compliant OpenMP implementation on software distributed shared memory. In *High Performance Computing - HiPC 2002: 9th International Conference Bangalore, India, December 18-21, 2002. Proceedings*, pages 195 – 206.

[13] G. Valle, R. Lottiaux, L. Rilling, J.-Y. Berthou, I. Dutka-Malhen, and C. Morin. A case for single system image cluster operating systems: Kerrighed approach. *Parallel Processing Letters*, 13(2), June 2003.

[14] G. Vallée, C. Morin, J.-Y. Berthou, and L. Rilling. A new approach to configurable dynamic scheduling in clusters based on single system image technologies. In *Proc. of the 17th International Parallel and Distributed Processing Symposium (IPDPS 2003)*, Nice, Apr. 2003. IEEE.