A Heterogeneous Environment for Distributed Computing

Karen Barett *Christopher Carothers* *Ron Chadwick* *Vernard Martin*

College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332
e-mail: {karen, chrisc, rlc, vernard}@cc.gatech.edu

Abstract

This paper focuses on the design and implementation of a Heterogeneous Parallel Computing environment. The environment's main goals is to provide a consistent interface for programming distributed applications using user level threads that will allow the user to exploit the capabilities of each individual node of the distributed network. The hardware architecture is kept hidden from the user whenever possible. We combined the existing PVM system with the Ga Tech version of the CMU Cthreads package as the basis for our environment.

Keywords PVM, threads, heterogeneous computing, multiprocessor.

1 Introduction

One of the logical steps in the struggle to achieve linear speedup using parallel systems is to use a collection of heterogeneous existing systems and have them communicate in some fashion that is hopefully transparent to the end user. The PVM software package does this fairly effectively. However, there are some limitations and features that the basic PVM package does not provide. In particular, PVM does not address the concept of threaded applications directly. In addition, PVM version 2.4.2 uses sockets for communication purposes even when two communicating applications are on the same node. We attempted to solve the above two problems by adding a threads package to the PVM system and using shared memory for communications respectively. Our solution also sustains the portability of the original PVM package.

The article is structured as follows: First we present a brief overview of the Ga Tech Cthreads package. Next we explain how PVM and Cthreads interact We then address the modifications made to replace PVM's socket communication interface with shared memory interface. Finally, we discuss our findings, unresolved problems with the current system, and future work followed by a short summary of the results.

2 What are user level threads?

A user level thread, hereafter referred to as a thread, is an independent flow of control within a process. Threads may be dynamically created and terminated such that multiple threads may exist within the same process. Threads within the same process share that process's global data (global variables, files,etc.) but maintain their own stacks ,local variables and program counters. Threads are referred to as light-weight because their context is smaller than the context of processes. Therefore, context switches between threads are considerably cheaper relative to process context switches. Threads are a good tool for exploiting parallelism in shared memory multiprocessor environments such as the Sequent, KSR, SGI, or BBN Butterfly series of computers [Mue93].

The Ga Tech Cthreads package is essentially a C library extension that provides routines to properly implement threads at the user level. By letting the code run in user space instead of kernel space, the overhead of context switches is lessened.

The Ga Tech Cthreads package is functionally compatible to the CMU definition of Cthreads that was implemented in the Mach OS. In addition, the Ga Tech Cthreads offers some enhancement to allow for custom thread schedulers and monitoring of code [SFG+91].

3 Efficient Inter Node Communications

The PVM 2.4.2 software package provides a framework that allows users to use a set of networked UNIX workstations as a virtual parallel machine [BDG+92]. Each workstation (henceforth called nodes) has a daemon that arbitrated communications between applications. Each application is a single process. For processes to communicate, they first have to register themselves with the daemon on which they execute. All of the daemons on each node share information with each other about enrolled processes and process locations so that messages can be sent between applications on any of the nodes. The actual communication between nodes is done via sockets even when two applications reside on the same node. We modified the daemons to use a shared memory buffer to hold the message data that will be sent to enrolled applications. The daemon then checks to see if the destination application is currently on the same node as the messages' source. If it is not, then the normal socket method is used to send the message across the network. Otherwise, the message is simply copied from the shared memory buffer by the destination application, thereby eliminating the socket overheads and taking advantage of the speed of shared memory.

4 Threadizing PVM

The Ga Tech Cthreads package is similar to PVM in that each Cthreads application is represented on single or multiple processes. When Cthreads are used with unmodified PVM, each application can be compiled with both the Cthreads and PVM libraries and ran in its own address space. Due to the addition of shared memory as discussed in the section above, the modified PVM Cthreads package requires that each application is compiled with the PVM daemon so that both the daemon and the applications can access the shared memory buffer used for inter process communication. This necessitates that all applications that will access the PVM system are known at compile time.

The Ga Tech Cthreads environment is started up with one master thread. That thread spawns other threads as needed but requires that those thread's code be declared as C functions. This also implies that all applications running with the modified Cthreads/PVM system must be declared as functions in the actual **pvmd** main application. Each application must also be explicitly added to a list of application names so that these applications can be spawned on different nodes from within other threads. This necessitates re-compilation of the PVM daemon every time an application is added.

There is one master thread per node that runs the **pvmd**. That thread then spawns the thread that is the actual application to run on that particular node. On a multiprocessor machine such as the Sequent, each processor would have its own process and one thread per process.

Due to re-entrancy problems with the combined Cthreads and PVM library, only one thread per node can actually enroll. This does not present a significant problem on single processor nodes, and on multiprocessor nodes each processor can also be considered its own node and thus communication can occur between any individual processor on a multiprocessor machine and any node that consists of a single processor.

5 Current Research Evaluation and Results

We are using the Sun Sparcstation as our platform for development. There are two reasons for this: 1) The Sparc platform has been the main platform for development of the Ga Tech Cthreads environment and thus the version of Cthreads with the latest enhancements runs stably on this platform. 2) The Sparc platform is a fairly common platform for both academic and industry environment and therefore is more likely to be available for actual use than other architectures. In addition the Sparc version of Cthreads is able to simulate a multiprocessor machine. Thus, code that is developed on the Sparc platform will port to multiprocessor machines without modification.

Timings were taken moving various sized packets of information between two applications running on two separate nodes with the unmodified PVM system. For comparison, the same timings were repeated with the new PVM environment that uses the Cthreads library and the shared memory interface. The timing results for the Cthreads version of PVM were slightly slower than the unmodified PVM. Profiling shows that the majority of the slowdown is due to the shared memory interface. The shared memory buffer routines used in the modified PVM environment have been chosen for simplicity rather than efficiency. By improving the internal workings of the shared memory buffer, we believe it is possible to gain a significant improvement in performance by reducing the contention for the shared memory buffer. This can be achieved by having a separate shared memory buffer for each enrolled application. Currently, there exist only one shared memory buffer for all applications enrolled on the same node. We are in the process of implementing those changes.

Further examination shows that the SYS V shared memory subsystem that is used by the Suns is also fairly slow. We point out that a platform offering hardware support for shared memory (such as the Sequent) will result in better performance. Steps to port the modified PVM with Cthreads to the Sequent and to other shared memory platform like the SGI and KSR machines are underway.

6 Future Work

- Modify the shared memory interface to be more efficient so as to reduce contention and therefore increase throughput.
- The combined PVM and Cthreads library needs to be made reentrant. This will help in removing the restriction of having only one thread per node enrolled in the PVM network.
- We plan to finish the port of the modified environment to the Sequent, KSR, and SGI.
- Both the unmodified PVM code and the latest Ga Tech threads package have been ported to the Kendall Squared Research KSR architecture as the first steps in porting the new environment to the KSR platform

7 Conclusion

The PVM package version 2.4.2 already allows for a group of networked machines (possibly of different architectures) to send and receive information transparent of their architectures. The system uses sockets for communication across nodes even when the two communicating processes were on the same node. We modified the system to use a shared memory buffer instead to improve performance.

By using the Ga Tech Cthreads package we also allow user level multi-threaded applications to exist on each of the nodes. This allows each of the nodes to exploit parallel capabilities of that node if they are present. Each thread can send and receive messages normally although messages to threads that are not on the same node must be routed. Each node has one master thread which routes for that node. This routing is done transparently to the user.

Both the Ga Tech Cthreads package and the PVM package are portable to multiple architectures, and they work on single processor and multiple processor machines. It is possible to develop code with both single and multiple processor nodes on just a single node machine and port the code to the actual multiple processor machine with only re-compilation.

In summary, the modified PVM environment gives users a stable portable platform for users to write threaded applications which can exploit the capabilities of multiprocessor nodes.

8 Acknowledgements

Many thanks to Vaidy Sunderam at Emory University for some very helpful comments on the internals of the standard PVM package. Thanks to Dr.Karsten Schwan for the initial idea for the project and his guidance throughout.

9 Bibliography

References

[BDG+92] Adam Beguelin, Jack Dongarra, Al Geist, Robert Manchek, and Vaidy Sunderam. A users's guide to pvm paralle virtual machine. Technical Report ORNL/TM-11826, Oak Ridge National Laboratory, P.O.Box 2008, Bldg. 6012 Oak Ridge, TN 37831-6367, July 1992.

[Mue93] Frank Mueller. A library implementation of posix threads under unix. In *1993 Winter USENIX Proceedings*, 1993.

[SFG+91] Karsten Schwan, Harold Forbes, Ahmed Gheith, Bodhisattwa Mukherjee, and Yiannis Samiotakis. A c thread library for multiprocessors. Technical Report GIT-ICS-91/02, Georgia Institute of Technology, College of Computing Georgia Institute of Technology Atlanta, GA 30332-0280, January 1991.

10 Author information

Karen Barrett is a Masters Student in Computer Science at the Georgia Institute of Technology's College of Computing. Her interests are Parallel Systems and Software Engineering. Email: karenb@cc.gatech.edu.

Christopher Carothers is a Masters Student in Computer Science at the Georgia Institute of Technology's College of Computing. His main interest is Optimistic Distributed Parallel Simulations. Email: chrisc@cc.gatech.edu

Ron Chadwick is a Masters Student in Computer Science at the Georgia Institute of Technology's College of Computing, a Research Scientist at the Georgia Tech Research Institute, and a part time instructor for the college. His interests are Parallel Systems, Networking Technologies, and Software Engineering. Email: rlc@cc.gatech.edu

Vernard Martin is a PhD Student in Computer Science at the Georgia Institute of Technology's College of Computing and part time instructor with the college as well. His interests are Parallel Computing Systems and Computer Graphics. Email: vernard@cc.gatech.edu