Portable and Scalable Parallel Applications with VCluster

Joohan Lee, Hua Zhang, and Ratan Guha
School of Computer Science
University of Central Florida
Orlando, FL 32816, USA
E-mails: {jlee, hzhang, guha}@cs.ucf.edu

KEYWORDS
Middleware, Cluster Computing, Parallel Processing, Message Passing.

ABSTRACT

Message passing based parallel programming paradigm and associated libraries such as MPI and PVM have proven its novelty and efficiency by successful applications in many diverse areas ranging from scientific computation, simulation, graphics, machine learning, to data mining. Those tools have been de facto parallel programming libraries for cluster computing where a cluster consists of usually homogeneous uni-processor machines. Recent advances in processor technologies made it possible to build a cluster of multi-processor machines more economically. In addition, a new programming language, Java, and its associated technologies opened a door to flourish of new and more efficient development of distributed computing software. Virtual Cluster Computing (VCluster) library is a portable parallel runtime system we have developed to support parallel programs that run on the cluster of heterogeneous uni- and multi-processors. In this paper, we briefly introduce the features of VCluster system and describe how parallel programs can be developed efficiently with VCluster. We also present the experimental results showing the performance of the developed system on two exemplary applications, heat diffusion and back propagation neural network, and compare them with other systems.

INTRODUCTION

With the availability of high performance microprocessors, high speed networks, and the associated distributed computing tools, cluster computing has become widely used in many diverse areas from fluid dynamics, bioinformatics, simulation, and data mining. Cluster computing provides a low cost parallel computing platform consisting of multiple interconnected PCs or workstations through commodity network technology as a replacement for an expensive parallel computer.

Developing a parallel program over a cluster requires an associated parallel programming model, a language, and a software library to realize that. The most widely used parallel programming model in cluster computing is message passing where each processor executes a different stream of instructions and exchanges messages when they need to share data or coordinate with other processors. Message Passing Interface (MPI) [Gropp 1995] has been used as a de facto standard for message passing based parallel computing. MPI specifies the necessary point-to-point and advanced collective communication primitives for message passing. MPI and other message passing libraries such as Parallel Virtual Machine (PVM) [Sunderam 1990] and P4 [Butler 1994] have been widely used in developing parallel applications proving its effectiveness due to simplicity and portability over various parallel computing platforms.

However, in cluster computing, lower network speed has been the main bottleneck that impedes the scalability of a cluster. One solution to overcome this obstacle is to improve the bandwidth using faster networking technologies such as Gigabit Ethernet, Myrinet, or ATM. Another solution is to use a cluster of multiprocessors instead of uniprocessor machines. Recently, processor vendors started producing relatively low cost Symmetric Multiprocessor (SMP) machines with usually up to eight processors that share the same memory, I/O devices, and other resources and run different copy of the operating system on each processor. Emergence of low cost SMPs made it possible to build a cluster of multiprocessors economically. With use of multiprocessors, fewer number of machines need to be connected through a slower network and the communication among the processors within the same machine can take place using a much faster bus or an interconnection network. This reduces the amount of communication over the network and improves the scalability of the cluster.

With this configuration, a cluster of multiprocessors, current parallel programming models don't support this new type of cluster computing architecture effectively. With a multiprocessor architecture, shared memory programming models must be incorporated into the programming model while message passing model is still needed for a cluster based parallel computing. A parallel programming model that supports this architecture must address two different parallel computing paradigms in a single framework. In a shared memory multiprocessors, multithreading and synchronization based on shared variables are common techniques. In message passing, communication is defined between two processes rather than threads and explicit message exchanges are used for work

Proceedings 19th European Conference on Modelling and Simulation
Yuri Merkuryev, Richard Zobel, Eugène Kerckhoffs © ECMS, 2005
ISBN 1-84233-112-4 (Set) / ISBN 1-84233-113-2 (CD)

distribution, data sharing, coordination, and synchronization.

While many of the existing parallel programming libraries have been targeted for C/C++ and Fortran programming languages, a new programming language, Java, and its associated technologies opened a door to flourish of more efficient development of distributed computing software due to the platform neutral byte codes, concurrent programming model based on *monitor* concept, object oriented, and inter-process communication mechanisms such as TCP/IP sockets and Remote Method Invocation (RMI). Recently, Java has also strengthened its viability as a distributed computing tool by incorporating Java cryptography and security packages as a part of recent JDKs.

Motivated by these observations, we have developed a new parallel programming model and a parallel runtime system that can address both tightly coupled shared memory multiprocessors and loosely coupled distributed memory multiprocessors, a cluster, in a single framework. VCluster is a prototype realization of the proposed framework. In this paper, we discuss the architecture and features of VCluster and compare them with other related approaches. We present how parallel programs can be developed using VCluster and show experimental results with two parallel applications.

RELATED RESEARCH

In this section, we discuss the various aspects of high performance cluster computing at different levels and the associated issues. They include communication networks, cluster architecture, parallel programming models and libraries.

In cluster computing, communication latency is the main bottleneck that impedes the scalability of the parallel programs when commodity networking technology such as Fast Ethernet is used. In order to reduce the communication latency, use of faster networking technologies such as ATM network or Myrinet can improve the network performance. However, one critical issue is that they usually use a particular network protocol to utilize the underlying faster network fully rather than using a general TCP/IP protocol, which results in reducing the portability and causing the developed parallel programs to be bound to a specific networking technology.

Recent clusters began to use the Gigabit Ethernet as an alternative networking technology to connect the computers on the cluster. However, the performance of the Gigabit Ethernet doesn't provide much faster networking speed as the name implies. Gigabit Ethernet technology simply increased the network speed to Gigabit per second (Gbps) but was still built on top of the same Ether protocol [Park 1998]. Even with the use of faster networking technologies, it cannot provide enough latency and bandwidth compared with the dedicated interconnection network used in the shared and distributed memory multiprocessors.

As an alternative solution to address scalability, building a cluster using multiprocessors is attractive. It can reduce the communication latency among the processors in the same machine and only the communication between the processors on the different machines needs to take place over the slower network. With the emergence of low cost SMPs and the low cost operating systems running on top of those machines such as LINUX and FreeBSD, SMPs have gained more attention in cluster computing to substitute uni-processor machines. SMP has multiple processors sharing the same system resources. It is symmetric; the processors have the same capability and can access the memory, I/O, and other resources at the same level. And, they are usually connected by a bus with a few processors.

Successful development of parallel programs requires efficient parallel programming models, languages, and libraries. Depending upon the underlying parallel and distributed computing platforms, there can be many different approaches. In a shared memory machine like SMP, multithreading and associated synchronization primitives such as semaphore, mutex, and monitor, are used for developing parallel programs. High level programming languages such as OpenMP provide parallel language constructs that automatically and dynamically create processes and execute some part of the program in parallel such as parallel *for* loops [Chandra 2000].

In cluster computing environment where computers have their own private memories and are loosely connected through a slower network, message passing has been the dominating parallel programming paradigm. Data sharing, coordination, and synchronization are done through explicit message exchanges. In this parlance, MPI and PVM were the most successful implementations and used in developing many different types of parallel applications. Message passing architectures favors coarse-grained parallelism because send and receive operations can be designed to amortize latency over the long messages that are sent infrequently in coarse-grained programs [Jordan 2000].

Implementations of message passing libraries were geared towards a cluster of homogeneous uni-processor machines. Even though multithreading can be used with those message passing libraries, however, communication in the message passing model was defined for inter-process communication and it's up to the users how to incorporate threads into MPI programs. While Scalability and better speedup have been the main driving motivations of parallel and distributed processing, we cannot emphasize the importance of portability and maintainability enough in more recent cluster computing environments where a cluster is composed of heterogeneous machines rather than homogeneous ones. The number representation formats can be different, big endian and little endian. Once again, it can also be handled by the user who is aware of

the heterogeneity and converts the number representations accordingly, however, heterogeneity is not hidden from the users and reduces the portability of the developed parallel programs. Several portable message passing libraries to address these issues based on Java have been developed such as JPVM and mpiJava utilizing the platform neural features of the Java byte codes [Baker 1998, Ferrari 1998]. mpiJava [7] wraps the MPI library with Java using Java Native Interface (JNI). JPVM [Ferrari 1998] implemented the PVM specifications purely in JAVA.

However, simply applying JAVA technologies to the existing message passing models is not an efficient solution. Those message passing models were designed to augment the existing sequential programming languages with additional message passing functionalities. Message passing model was not designed towards object oriented paradigm. Java is an object oriented programming language and the corresponding message passing model has to take that into consideration in its design to be more efficient.

VIRTUAL CLUSTER COMPUTING (VCLUSTER) LIBRARY

VCluster is our effort to address the issues raised in the previous section. VCluster provides a framework to support a cluster of heterogeneous uni- or multi-processor machines using a portable parallel programming library. This combines the shared memory programming paradigm and the message passing in a single framework. Initial design of VCluster was inspired by SCPlib and CRlib that are both multithread message passing libraries with richer set of advanced functionalities such as dynamic load balancing, thread mobility, heterogeneous computing, replication, and fault tolerance [Watts 1998, Lee 2002]. They were primarily developed for parallel C programs over a variety of parallel computing platforms.

VCluster is a parallel run time system implemented using 100% Java codes to support better portability exploiting Java's platform neutral byte code technologies. A basic computing unit in the VCluster is a communicating virtual thread that can migrate across the Java Virtual Machines (JVM) and computers. Each thread has a list of associated named states and named communication channels as shown in Figure 1. Unlike MPI message passing model where communication resources are bound to a process, in VCluster, communication resources are associated with an individual thread. A thread creates a set of uni-directional communication channels to communicate with other threads. A thread also associates itself with a set of states of interest that are shared data and methods. The same state can be shared by multiple threads and the atomic and mutually exclusive access to the states is guaranteed.

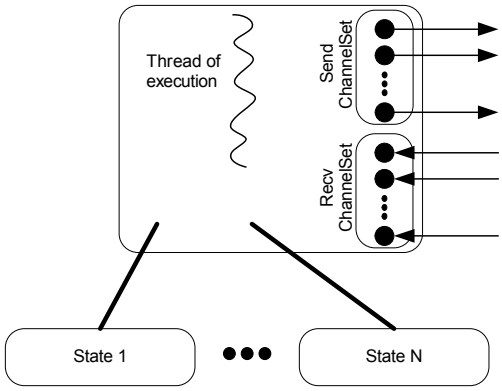

Figure 1. Virtual Thread Structure in VCluster

Java's object oriented paradigm and polymorphism provides useful structure to hide complexity of the software structure and make the parallel program modular and layered software structure more effective. Programmers write the parallel program by extending VCluster thread, state, channel, and channel set classes and inheriting their properties, which hides the internal complexity of those abstract concepts. This architectural features of VCluster make a virtual thread migrate to another machine.

VCluster provides thread to thread message passing functionality. Those communicating threads can be on the same machine or on remote machines. If they are on the same machine, the message will be just copied from one thread buffer to another thread buffer. If they are on the different machines, then the message is actually transmitted over the network. In order to support shared memory programming paradigm without using direct message passing, the threads on the same machine can be bound to the same states and use monitor based coordination and synchronization schemes. In VCluster, a thread is virtual in that the thread can migrate within the system by detaching itself from one computer and move to the remote location with the current status.

Prototype implementation of the proposed VCluster was done using JDK 1.4.2 and used to develop several parallel and distributed applications.

PARALLEL PROGRAMS WITH VCLUSTER

In this section, we describe how parallel programs are developed using the VCluster library. Figure 2 shows the basic structure of a parallel program. It illustrates how two virtual threads are created, set up the communication links, create states, and exchange messages. Note that two processes on each computer execute the same code but with a different process ID number that tells who is a sender and a receiver. A user creates a virtual thread by extending a VC_Thread class and needs to implement the run method.

```
public class Application
{
  public static void main(String args[])
```

```
  {
    VCluster vc;
    VC_ChannelSet  vchannelSet;
    VC_Channel vchannel;
    myThread vthr;

    // Initialize
    vc = new VCluster(args);

    // Create two virtual threads and bind them to VCluster
    vthr = new myThread();
    vc.addThread(vthr);

    // Create the states and bind them to the vthreads
    mystate = new myState();
    vthr.addState(mystate);

    // Setup the communication links
    if (vc.getMyPid() == 0) {
      vchannelSet = new VC_ChannelSet("sendChannelSet");
      vchannel = new VC_Channel(1,"ch", vc.VC_WRITE);
    }
    else {
      vchannelSet = new VC_ChannelSet("recvChannelSet");
      vchannel = new VC_Channel(0,"ch", vc.VC_READ);
    }
    // Bind the communication links to virtual threads
     vchannelSet.addChannel(vchannel);
     vthr.addChannelSet(vchannelSet);

    // Start the program
    vc.start();

    // Finalize and exit the program
    vc.finalize();
  }
}

class myThread extends VC_Thread
{
  // User need to implement this run method
  public void run()
  {
    myState mystate;
    VC_ChannelSet vchannelSet;
    int myTid, myPid;

    myPid = getPid();
    myTid = getTid();
    mystate = (myState)getState();

    // Send/Recv a single integer between tow vtheads
    if (myPid == 0) {
      vchannelSet = getChannelSet("sendChannelSet");
      VC_Channel vchannel = vchannelSet.getChannel("ch");
      vchannel.writeInt(888);
    }
    else if (myPid == 1) {
      vchannelSet = getChannelSet("recvChannelSet");
      VC_Channel vchannel = vchannelSet.getChannel("ch");
      int   a = vchannel.readInt();
    }

    finalize();
  }
}
```

Figure 2. An Simple Send/Recv Program with VCluster

EXPERIMENTAL RESULTS

In this section, we examine the performance of the developed system and compare it with other relevant systems. Three relevant systems were compared with VCluster; MPICH, mpiJava, and JPVM. MPICH is a C implementation of MPI by Argonne National Laboratory [MPICH 2004], mpiJava by [Baker 1998], and JPVM by [Ferrari 1998]. MPICH was chosen as the base case that extends the C programming language with message passing functions. It is most widely used and would yield the best performance compared with other Java based libraries. mipJava is simply wrapping the MPICH with Java interface and is expected to be slower than MPICH but faster than JPVM and VCluster that were implemented purely in JAVA.

Our performance analysis was done in two aspects. First we measure the communication overhead. Since JAVA uses the neural byte format for the communication, the communication overhead was expected to be higher than that of C. Fig 3 shows the round trip communication overhead with respect to the varying message size from one byte to one mega bytes. As expected, Java based message passing libraries are slower than C based MPICH. For a larger message sizes closer to one mega bytes, VCluster performed better than the JPVM. We believe that it is because of the VCluster's sending and receiving mechanisms based on separate send/recv threads and polling. Despite the higher communication overhead in VCluster and other Java based systems, it doesn't discourage the use of Java for cluster computing in that coarse grain parallelism is mostly used in cluster computing and the communication overhead can be compensated by the larger computation part of the application.

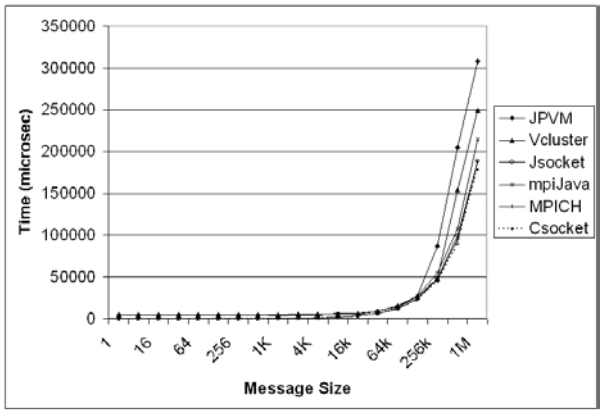

Fig 3. Communication Overhead

Next two experimentations were performed to analyze the real application performance that includes both communication and computation. First application is a

fluid dynamics application, Dirichlet boundary problem. The Dirichlet boundary problem is a simple numerical simulation problem on a two dimensional grid. Each point on the grid has a (x, y) location and a value representing temperature of some material. At each time step, each point's temperature is averaged with its neighbor's temperatures to find the point's temperature at the end of the time step. This operates for all grid points that are not on the boundary. Boundary grid points are assumed to have a constant value. The workload is uniform in the Dirichlet problem. This allows the domain decomposition technique to be used in dividing up the workload among processes. We have implemented the Dirichlet boundary problem using four message passing libraries introduced above. Our cluster computing environment was composed of a cluster of 32 LINUX PCs equipped with 900MHz AMD Athlon processor, 1 GB memory, and 100BT networking. Figure 4 shows their performance on this cluster with varying number of processors. Performance of three Java based message passing libraries was comparable and the MPICH outperformed them. With a small number of processors, the execution time difference was noticeable, however, with larger number of processors, for example 32, the difference became ignorable. This experimentation results indicate that with enough number of processors Java based message passing libraries can mitigate the associated overhead compared with C based libraries and be an attractive tool for a large scale parallel application development. mpiJava was expected be faster than VCluster and JPVM as it still uses C codes in it, however, its performance was close to the other Java implemented libraries.

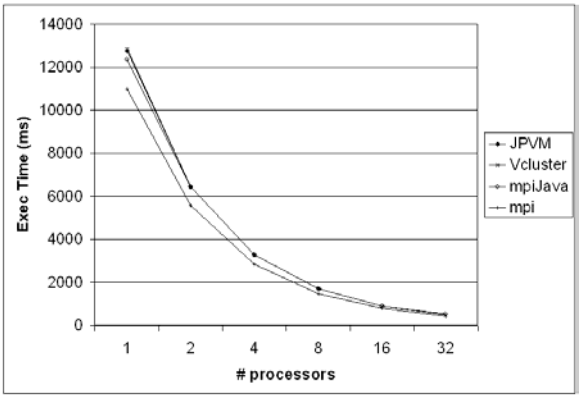

Fig 4. Parallel Dirichlet Problem Performance

The second application was a parallel Back Propagation Neural Network (BPNN) algorithm for network based intrusion detection. BPNN [Rumelhart 1986] is one of the most popular neural network training algorithms and has shown robust performance in many diverse applications. However, computational complexity of the BPNN makes its use challenging especially when the training data set size is huge. BPNN can be parallelized in many different ways depending on the underlying parallel computing architecture and programming models [Zickenheiner 1994, Nordstrom 1992]. We used a training set partitioning method that reduces the amount of communication needed for distributing data and synchronization. BPNN is trained iteratively until an acceptable mean squared error rate is achieved. In the beginning, the partitions of the data set are distributed to each worker thread. Each iteration is called an epoch. In each epoch, each worker threads work on the fraction of the computation corresponding to the distributed partition and the master thread aggregates the partial computation results. Then, the updated weight vectors are redistributed to all the worker threads for the next epoch.

Figure 5 shows the performance of the parallel PBNN on the same cluster. Once again, three Java based libraries showed the comparable performance. Note that the gap between VCluster and MPICH converges as the number of processors increases. The required amount of computation is much higher than that of Dirichlet boundary problem and it is expected that with use of more processors the gap will further converges as the previous application did.

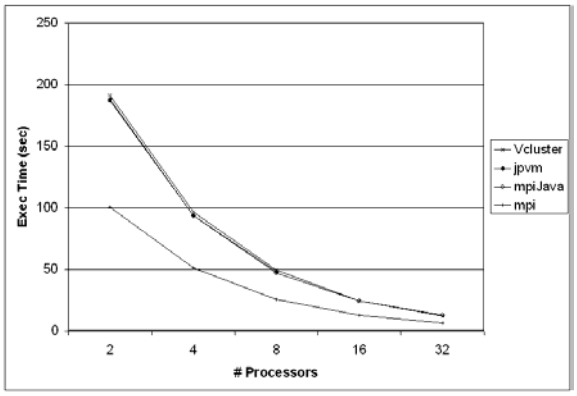

Fig 5. Parallel Neural Network Performance

CONCLUSION AND FUTURE RESEARCH

In this paper, we have introduced a new portable cluster computing library called VCluster and its features. Preliminary experimentation results showed that the performance of VCluster is comparable to the relevant message passing libraries such as mpiJava and JPVM that are based on Java. However, VCluster provided more convenient and efficient application development environment due to its unique programming model that combines multithreading with communication. Even though Java based parallel programming library has disadvantages with respect to the performance compared with C based implementation, when enough number of processors are available this problem could be mitigated significantly. Currently, we are working on implementing dynamic load balancing using VCsluter's thread migration capability.

ACKNOWLEDGEMENTS

This work was partially supported by ARO under grants DAAD19-01-1-0502,W911NF04110100 and NSF under Grant EIA 0086251. The views and conclusions herein are those of the authors and do not represent the official policies of the funding agencies or the University of Central Florida.

REFERENCES

Baker, M., B. Carpenter, S. Ko, and X. Li. 1998. "mpiJava: A Java Interface to MPI", *UK Workshop on Java for High Performance Network Computing* (Sep). Europar.

Butler, R. and E. Lusk. 1994. "Monitors, Message, and Clusters: The P4 Parallel Programming System", *Parallel Computing*, No. 20, (Apr), 547-564.

Chandra, R., R. Menon, L. Dagum, D. Kohr, D. Maydan, J. McDonald. 2000. *Parallel Programming in OpenMP*, Morgan Kaufmann.

Ferrari, A.J. 1998. "JPVM: Network Parallel Computing in Java". *ACM Workshop on Java for High-Performance Network Computing* (Palo Alto, CA, Feb).

Gropp, W.; E. Lusk; and A. Skjellum. 1995. *Using MPI: Portable Parallel Programming with the Message Passing Interface*. MPI Press.

Jordan, H.F. and G. Alaghband. 2003. *Fundamentals of Parallel Processing*. Prentice Hall.

Lee, J., S. Chapin, and S. Taylor. 2002. "Computational Resiliency". *Journal of Quality and reliability Engineering International*, No. 18, Vol. 3, 185-199.

MPICH-A Portable Implementation of MPI. 2004. http://www-unix.mcs.anl.gov/mpi/mpich.

Nordstrom, T. and B. Svensson. 1992. "Using and designing massively parallel computers for artificial neural networks", *Journal of Parallel and Distributed Computing*, Vol. 14, No. 3, 260-285.

Park, S., J. Lee, and S. Hariri. 1998. "Performance Evaluation of ATM and Gigabit Networks", *IEEE Information Technology Workshop* (Syracuse, NY, August).

Rumelhart, D. E., G. E. Hinton, and R. J. Williams. 1986. "Learning Representations By Back-propagating Errors", *Nature*, Vol. 323, 533-536.

Sunderam, V.S. 1990. "PVM: A Framework for Parallel Distributed Computing", *Concurrency: Practice and Experience*, Vol. 2, No. 4, (Dec), 315-339.

Watts, J., S. Taylor, and S. Nilpanich. 1998. "SCPlib: A Concurrent Programming Library for Programming Heterogeneous Networks of Computers". *IEEE Information Technology Conference* (Syracuse, NY, August). 153-156.

Zickenheiner, S., M. Wendt, B. Klauer, and K. Waldschmidt. 1994. "Pipelining and parallel training of neural networks on distributed-memory multiprocessors", *IEEE International Conference on Neural Network* (Orlando, FL, June). Vol. 4, No. 27, 2052-2057.

AUTHOR BIOGRAPHIES

JOOHAN LEE is an assistant professor in the School of Computer Science at the University of Central Florida in Orlando, Florida. He received a Ph.d in Computer Science from Syracuse University in 2002. His research interests include parallel and distributed computing, computer and network security, and high performance data mining.

HUA ZHANG received a BSc in Computer Science from Shanghai Jiaotong University, Shanghai, China in 1999. He is currently a Ph.d student in the School of Computer Science at the University of Central Florida. His research interests are high performance parallel and distributed computing.

RATAN GUHA is a professor at the department of Computer Science, University of Central Florida, with primary interests in Computer Networks, Distributed Computing, Distributed Simulation, and Computer Graphics. He received his Ph.D from the University of Texas at Austin, and his M.Sc. in Applied Mathematics, and B.Sc. with honors in Mathematics from the Calcutta University, India.