A Runtime System for Object-Oriented Programming Environment with Multithreaded Computational Model

Jonghoon Song[1], Heunghwan Kim[2], Sangyong Han[1]

[1] Dept. of Computer Science, Seoul National Univ. Seoul, 151-742, Korea
{song, syhan}@pplab.snu.ac.kr
[2] Dept. of Computer Science, Konkuk Univ., Chungju, Chungcheng-Do, 380-701, Korea
khh@kcucc.cj.konkuk.ac.kr

Abstract. We suggest a runtime system for parallel object-oriented programming environment. This system is using the multithreaded computation model and can be run on general parallel computer architectures. We developed several techniques for an efficient runtime system. : thread merging, message merging and frame sharing. The system is implemented in IBM SP2 with MPI message interface, and its performance is analyzed with executing benchmark programs. The results show that the suggested parallel model has a good performance and the runtime techniques can critically affect the performance of execution.

1 Introduction

Parallel computer system has the limitation of performance and scalability because of memory latency and synchronization and it is known as the fundamental issues in multiprocessing[1]. To overcome these problems, multithreaded computational model has been studied in various ways. Multithreaded computational model combines the instruction-level context switching and sequential scheduling, and can efficiently solve the latency and synchronization problems by overlapping the computational time with communication time. Multithreaded architectures based on dataflow model[2] exploits enough parallelism using the dataflow computational model among threads, while it exploits the locality of computation using the von Neumann computational model within a thread.

Because the multithreaded computational model can have the advantage of von Neumann architecture and dataflow architecture, it has been the subject of many research efforts[3]. But, No commercially successful products have been produced yet. The reason can be summarized as follows.

First, programming in multithreaded computational model is difficult and has the limitation in the expressive power. Traditionally, the programming language that uses functional semantics is used in order to exploit the maximum parallelism in multithreaded computational model[4]. But, the functional programming environment is not friendly with the most programmers and has a shortcoming in making real application programs. Especially, in the application to represent a real world that has the concept of state, it shows the limitation of programming model, Second, the multithreaded architecture is strange and complex to the existing programmers. The most multithreaded computational model is experimented and implemented on the basis of special architecture. The reason is that the context switching of thread is frequent and the special hardware that can support fast context switching is needed[5]. The existing programmers and users that are familiar with the typical architecture may not easily accept a new architecture.

In this paper, to solve the above problems and use the advantage of multithreaded computational model, we suggest a new runtime model. The new model provides an object-oriented programming environment and is constructed on general parallel architecture. Actually, the object-oriented paradigm is suitable for parallel programming, because users can consider each object as a message-passing unit[6]. And using data abstraction, we can make private member not directly accessed. This technique simplifies the parallel control by treating each object as a monitor[7].

To implement a multithreaded computational model in general architecture efficiently, it is needed to reduce the overhead of context switching. We suggest several run-time techniques to reduce the number of context switching and its overhead. Using these techniques, efficient parallel object-oriented model can be constructed.

2 Related Work

There are many parallel object-oriented models. Mentat[8] is the dynamic object-oriented parallel system using MPL language, Concert System[9] is the model that can efficiently support the fine-grained parallelism, etc. But, we observe the OO system based on the multithreaded architecture : ABCL/EM4 and DAVRID.

ABCL/EM-4[10] is an object-oriented system that executes ABCL program on EM-4[11]. EM-4 is a kind of multithreaded architecture based on macro dataflow model and uses the concept of strongly connected block and direct matching. But, the implementation of efficient languages on EM-4 has not been as successful. The problem was that the overhead from message reception to dynamic method dispatching consumes a significant amount of execution time. So, the new software/hardware architecture for ABCL/EM-4 was proposed to overcome the problems using a combination of software/hardware technologies[14]. This architecture tries to solve the problem with the methods of address-specifiable reactive packet-driven architecture, zero-overhead context switching, and packet-driven allocation of message boxes. But, because ABCL/EM-4 calls the member function using a data structure called message box, it requires complicated exchanges of packets to manage it.

On DAVRID[12], the object-oriented programming model[13] was implemented. This model chose the object-oriented language OOId, which is the extension of functional language Id. This model tried to solve the problem that the object-oriented environment in multithreaded architecture has more overhead of message exchange than in other architectures. But, this model has the various limitation of performance because the functional model is extended to object-oriented model, and OOId is a difficult programming language to program.

3 Compiler

The multithreaded code in our model can be generated from parallel compiler. The source program used as input of parallel compiler is written in the extension of C++. The extension of C++ can exploit task parallelism and data parallelism. Data parallelism is represented by the extended reserved words, and task parallelism can be automatically detected by parallel compiler.

The parallel compiler generates thread code as object code. There are four steps in the compilation. The first step is the analysis of member and method. This step is to detect externally visible variables and extent constant variables and to perform the commutative analysis between methods. The studies of [14] can be used in this step. The second step is the construction of program dependence graph(PDG)[15]. PDG is the representation for showing the dependence among all operations. For detecting the dependence that the method calling is involved with, the analysis results of the first step can be used. The third step is the partition of PDG. Considering the long latency operation, PDG is partitioned into threads, There should be no dependence including long-latency operation within thread. The final step is the generation of thread code.

4 Runtime System

Runtime system executes the object code that is compiled in the multithreaded style, using the MPI message protocol. Each method in the source code is transformed into multithreaded code including many threads by the parallel compiler. Each thread can be executed without interruption, because all operations within a thread don't have any long-latency dependence. The basic function of runtime system is to fire an executable thread one by one through the handling of messages. The system also supports the efficient execution, reducing the number of context-switching and minimizing the overhead.

4.1 Basic Structure

Our runtime system is based on multithreaded computation model. It provides the environment that many frames are activated in a time and many threads are concurrently executed. There are two important concepts in runtime system : thread and frame. Thread is the unit composing thread code. A thread have the unique synchronization counter(SC). In view of runtime system, SC is the number of message receptions to execute the thread. That is, "The SC of thread T1 is 2" means "Thread T1 needs to receive 2 messages in order to execute."

Frame is the data structure to be allocated in the unit of objects or methods. That is, whenever one object is created, one object frame is allocated, and whenever one method is called, one method frame is allocated. The object frame stores and manages values of member variables in object and the synchronization information of threads in constructor method. And the method frame stores and manages the synchronization information of threads in its own method.

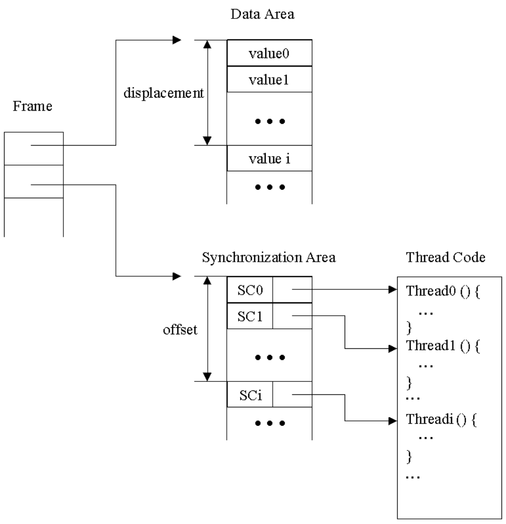


Fig. 1. Data Area and Synchronization Area

Frames have the data area and the synchronization area, which play an important part in supporting the multithreaded computational environment. The data area is the place where the result of each thread execution is stored, and each data area can be considered as the unit of data parallelism. And the synchronization area is where thread pointers and synchronization counters of threads are stored, and this area is considered as the area that controls the task parallelism because each thread is the unit of task parallelism. Fig. 1 shows the data area and the synchronization area in frame.

In order to send values to the frame and execute the thread to process the values, the runtime system uses the tuple (displacement, offset). That is, first, using "displacement", it stores the value in the data area. And then, using "offset", it decreases the synchronization counter by 1 in the synchronization area. If the synchronization counter becomes 0, it executes the thread using thread pointer, otherwise, it must wait another signals to arrive. This process is called "thread synchronization". Thread synchronization is the basic operation to fire threads, and is requested by messages exchanged among threads.

4.2 Runtime Model

The runtime system exists in form of library program and becomes the complete executable code after it is linked to the thread code that is output from compiling the source program. All nodes that participate in computation have the runtime system and the thread code, and Fig. 2 shows the runtime model.

The runtime system controls the main flow of execution. It receives messages from other node and fires the execution of thread in form of calling the procedure during handling the messages. The runtime system and the thread code share the frame area and message buffer. Message buffer consists of transmission buffer and reception

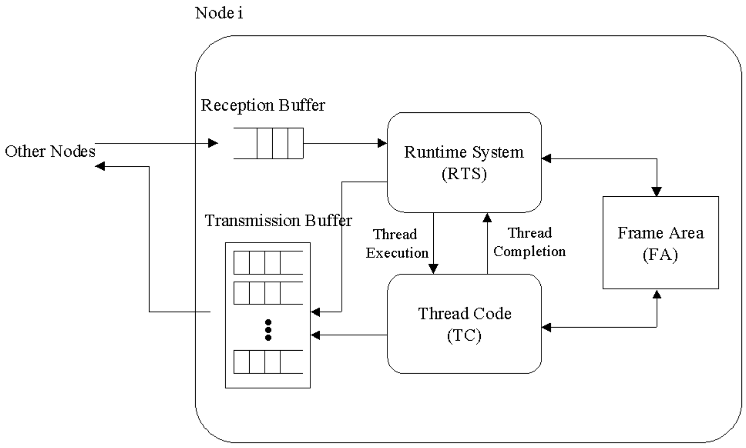


Fig. 2. Runtime Model

buffer. Reception buffer is a single buffer but transmission buffer has as many buffers as the number of node that participate in computation. Frame area is the place for keeping frames. Frames are dynamically allocated and released to store the inter-thread synchronization information and values.

The basic algorithm of the runtime system is simple : reading a message from reception buffer and handling the message. Message handling is to execute the proper routine according to the message type. Ten kinds of message type used in runtime system are used and can be classified as Table 1. During handling messages, the runtime system can allocate a frame using class table and method table, synchronize and execute a thread or transmit a message to other node.

4.3 The Feature of Runtime System

Thread merging and message merging. In order to execute thread codes efficiently, the runtime system uses the merging techniques. These techniques can be reduce the message transmission overhead, using only runtime information without the support of compiler.

Table 1. Messages in Runrime System

Frame management messages	OAlloc, OsAlloc, MCall, Odealloc
Data access messages	RRead, RWrite, Rtake, RPut
Synchronization messages	RReply, Tsync

Thread merging provides the way that the overhead of context switching can be reduced using the concept of local message. The messages exchanged in run-time can be classified into remote message and local message according to its destination node. That is, if the source node and the destination node are different, the message is called remote message, otherwise, the message is called local message. In local message, the actual messages are not exchanged among threads, but message handling routine within node is directly executed to reduce the network traffic. If another thread is fired during handling a local message, we can gain the effect that the length of code that is executed without the long latency operation is longer. Through this technique, we can execute the longer code without context switching than the length of thread that compiler generated, and it actually results that threads are merged in run-time.

Message merging is the technique to transmit many messages at a time as possible by merging the messages with the same destinations. In case of sending small messages, the message transmission time is very larger than message setup time. So, message merging technique can give the performance improvement by reducing the setup time. Merging techniques must be performed in run-time because the destination node is determined by load balancing algorithm in run-time. If message merging is used together with thread merging, we can gain the synergic effect because messages in many threads can be merged.

Inter-frame data sharing within node. Each node has frame directory and supports the way that data can be shared among frames. This technique is for reducing the message exchange among nodes, and it is very profitable when there are many extent constant variables in program.

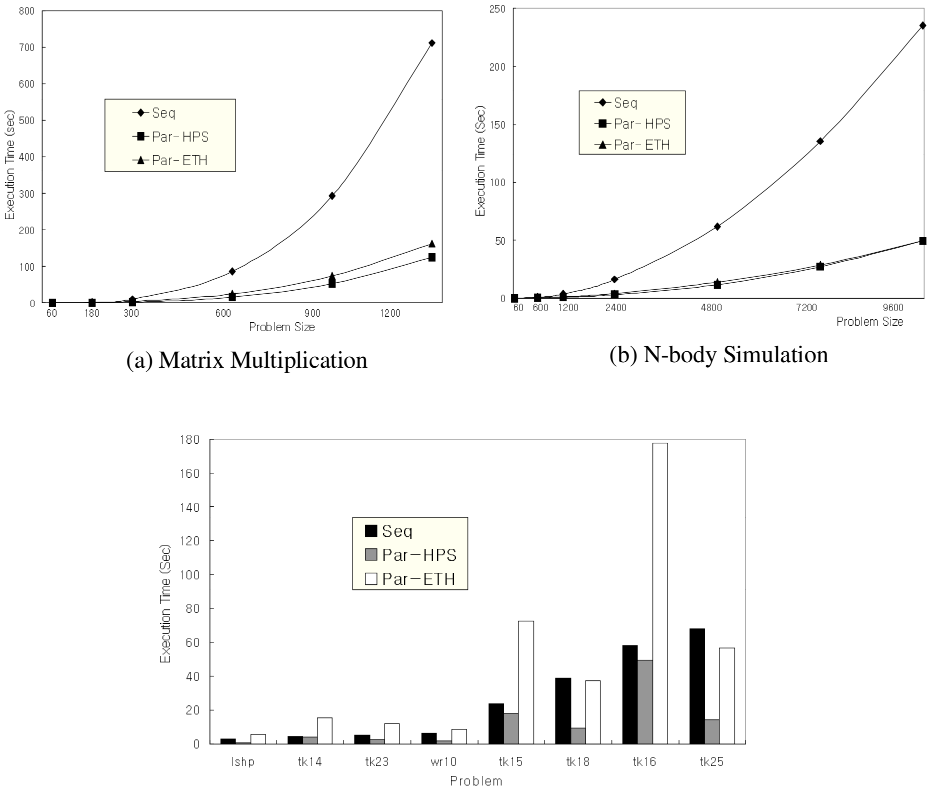
That is, when compiler recognized some variable as extent constant variable, if the variable is remotely accessed in runtime system, the frame directory can be accessed before remote read message is transmitted. If the information of the data does not exist in frame directory, data is read to its frame by remote read message and the address of the read data is recorded in frame directory. But, if the information of the data exists in frame directory, it is needless to transmit the remote read message and can use data in other frame just like in its own frame, using the address of data in frame directory.

Frame directory consists of P hash tables. P is the number of all nodes in system. The key of hash function is the address that the member variable is located at and hash function computes the hash address with the method of shift folding. The slots of hash table have pointers to the entry that keeps the state information of member variable.

5 Performance Evaluation

In this section, we evaluate the performance of model suggested in this paper. Because the runtime model is based on message passing mechanism and uses the primitive operation like message sending and receiving, there is little restriction in the environment of implementation. The implementation is possible in any parallel environment that supports the massage passing programming model.

Our experimental model is constructed on the basis of MPI[16]. Recently, MPI is expected to be the standard of message passing protocol, and most parallel system supports MPI message protocol interface. We executed various benchmarks in IBM



(a) Matrix Multiplication

(b) N-body Simulation

(c) Cholecky Factorization

Fig. 3. Comparison Between Sequential Execution and Parallel Execution

SP2 system[17] and analyzed the results. Interconnection network of SP2 can be selected to be high performance switch(HPS) or Ethernet switch.

Benchmark programs are matrix multiplication, N-body simulation and Cholesky factorization for sparse matrix. Fig. 3 shows the comparison between the sequential execution in one node and the parallel execution using the model suggested in this paper. In matrix multiplication and N-body simulation, we experimented with various problem sizes by changing the matrix size and the number of body respectively, and in Cholesky factorization, we chose 8 input matrices and executed them because this program needs the positive-definite symmetric matrix as input. In the legend, SEQ, Par-HPS and Par-ETH mean the sequential execution, high performance switch execution and Ethernet switch execution respectively.

In the experimental results of matrix multiplication and N-body simulation, parallel execution is superior to sequential execution. Because the increasing rate of execution time in parallel execution according to the problem size shows a gentle slope, the execution time of Par-HPS and Par-ETH is much smaller than that of SEQ, especially in large problem size. Actually, the speedup of Par-HPS in matrix multiplication is 2.01 when the problem size is 60 and 5.69 when the problem size is 1200. It is also remarkable that the difference between the execution time of Par-HPS and Par-ETH is slight though the transmission rates of high performance switch and Ethernet switch are 320Mb/s and 10Mb/s respectively. It means that the message traffic is reduced by runtime techniques.

In Cholesky factorization, the patterns of message exchange is dependent on input matrices and Fig. 3 (c) show various results according to input matrices. The execution of Par-HPS is faster than that of SEQ in all cases. Especially, the performance improvement is remarkable in lshp, tk18 and tk25. But, the execution of Par-ETH does not catch up with that of SEQ and it shows better results in tk18 and tk25 only. It means that the message overhead exceeds the advantage of parallelism because the pattern of message exchange is irregular and many small messages are continuously exchanged.

6 Conclusion

In this paper, we suggested an object-oriented system using multithreaded computational model on general parallel architecture. And we present several runtime techniques to implement the system efficiently : message merging, thread merging and frame sharing. The performance of the suggested system model was experimented by executing various benchmarks on IBM SP2 system. The results of experiment show that the suggested parallel model has a good performance and the runtime techniques can critically affect the performance of execution.

For the performance improvement of our system, the following research is progressed. First, the new strategy of load balancing is being studied. In current model, the basis of load balancing is only the number of allocated frame. That may be the problem in the application that has the irregular communication pattern. Though many frames are allocated, the number of active frame may be small. So, the new load balancing scheme must include the way that the number of current active frames is considered. Second, the use of Active Message[18] is being considered. Active message is a low-latency communication mechanism for multiprocessors that emphasize the overlap of communication and computation. Messages are in the form of requests and matching replies and there is no unnecessary transfer of data because messages contain the address of a handler that is invoked on receipt. It is expected to reduce the overhead in the application that small messages are continuously exchanged and to support our computational model efficiently.

References

1. Arvind, R. and A. Iannucci, "Two Fundamental Issues in Multiprocessing," MIT CSG Memo 226-5, 1986.
2. R. S. Nikhil, "Multithreaded Architecture," Tutorial, *16th Int'l Sympo. on Computer Architecture*, May 1992.
3. Ben Lee and A.R. Hurson, "Dataflow Architectures and Multithreading," *IEEE Computer*, Vol. 27, No. 8, pp. 27-39, Aug. 1994.

4. J. Backus, "Can Programming be Liberated from the von Neumann Style? A Functional Style and its Algebra of Programs", *Communication of ACM*, Vol. 21, No. 8, pp. 613-641, Aug. 1978.
5. R.A. Iannucci, *Multithreaded Computer Architecture: A Summary of the State of the Art*, Kluwer Academic Publisher, 1994.
6. Andrew S. Grimshaw, "Easy to Use Object Oriented Parallel Processing with Mentat," *IEEE Computer*, Vol. 26, No. 5, pp.39-51, May 1993.
7. J. Sargeant, Uniting Functional ad Object-Oriented Programming, Invited Paper, *Int'l Sympo. on Object Technologies for Advanced Software*, Kanazawa, Japan, LNCS 742, pp.1-26, November 1993.
8. Andrew S. Grimshaw, W.Timothy Strayer, and Padmini Narayan, "Dynamic, Object-Oriented Parallel Processing," *IEEE Parallel & Distributed Technology*, May, 1993.
9. Andrew Chien, Vijay Karamcheti, and John Plevyak, "The Concert System - compiler and runtime support for efficient fine-grained concurrent obejct-oriented programs," Technical Report UIUCDCS-R-93-1815, Department of Computer Science, University of Illinois, Urbana, Illinois, June 1993.
10. A. Yonezawa, S. Matsuoka, M. Yasugi, and K. Taura, "Implementing Concurrent Object-Oriented Language on Multicomputers", *IEEE Parallel & Distributed Technology*, May, 1993.
11. S. Sakai, Y. Kodama, and Y. Yamaguchi, "Prototype Implementation of a Highly Dataflow Machine EM-4", *In Proc. Parallel Processing Symposium*, 1991.
12. E. H. Rho, S. H. Ha, S. Y. Han, H. H. Kim and D. J. Hwang, "Compilation of a Functional Language for the Multithreaded Architecture : DAVRID", *In Proc. of 1994 International Conference on Parallel Processing*, Vol. 2. pp.239-242. Aug 1994.
13. Jonghoon Song, Juno Chang, Heunghwan Kim, Sangyoung Han, "Exploiting Locality of Array Data with Parallel Object-Oriented Model for Multithreaded Computation", *Proc. of the 6th Int'l Conf. on Advanced Computing(ADCOMP'98)*, Pune, India, pp.45-52, December 14-16, 1998.
14. Martin C. Rinard and Pedro C. Diniz, "Commutativity Analysis : A New Analysis Technique for Parallelizing Compilers," *ACM Transaction on Programming Language and Systems*, Vol.19, No.6, November, pp.942-991, 1997.
15. Michael Wolfe, *High Performance Compilers For Parallel Computing*, Addison-Wesley Publishing Company, 1996
16. Peter S. Pacheco, *Parallel Programming with MPI*, Morgan Kaufmann Publishers, 1997
17. T. Agerwala, J.L. Martin, J.H. Mirza, D.C. Sadler, D.M. Dias and M. Snir, "SP2 System Architecture," *IBM Systems Journal*, Vol. 34, No. 2, 1995.
18. T. von Eicken, D. E. Culler, S. C. Goldstein and K. E. Schauser, "Active Messages: A Mechanism for Integrated Communication and Composition," *In Proc. of the 19th Int'l Symposium on Computer Architecture*, Gold Coast, Australia, May 1992.