1000-9825/2001/12(03)0334-06 ©2001 Journal of Software 软件学报 Vol. 12, No. 3

Completely Debugging Indeterminate MPI/PVM Programs*

WANG Feng, AN Hong, CHEN Zhi-hui, CHEN Guo-liang

(National High Performance Computing Center, Department of Computer Science and Technology, University of Science and Technology of China, Hefei 230027, China)

E-mail: {wangf,zhchen}@mail. ustc. edu. cn; {han,glchen}@ustc. edu. cn

http://www. nhpcc. ustc. edu. cn

Received October 15, 1999; accepted January 25, 2000

Abstract: This paper discusses how to completely debug indeterminate MPI/PVM parallel programs. Due to the indeterminacy, the previous bugs may be non-repeatable in successive executions during a cyclic debugging session. Based on the FIFO communication model of MPI/PVM, an implementation of record and replay technique is presented. Moreover, users are provided with an easy way to completely debug their programs by covering all possible execution paths through controllable replay. Comparied with other solutions, the proposed method produces much less temporal and spatial overhead. The implementation has been completed on two kinds of message passing architectures: one is Dawning-2000 super server (that was developed by the National Research Center for Intelligent Computing Systems of China) with single-processor (PowerPC) nodes which are interconnected by a custom-built wormhole mesh network; the other is a cluster of workstations (PowerPC/AIX) which has been built in National High Performance Computing Center at Hefei.

Key words: parallel debugger; record and replay; message passing; MPI; PVM

Debugging indeterminate parallel programs is much more difficult than debugging sequential ones. The first issue is the repeatability of bugs. Traditional method of cyclic debugging relies on the determinacy of the execution procedures to locate the bugs, which means, to the same input, not only the results but also the execution paths are the same in successive runs. For parallel programs, this determinacy is destroyed by the message racing which is unpredictable due to many factors such as network delay, load balance, etc. In order to debug such indeterminate parallel programs, there comes the record and replay technique. A number of papers discussed about it and a few parallel debuggers implement it[1~3].

Furthermore, indeterminacy produces another issue——users can not ensure the correctness of their programs

* This project is supported by the National High Technology Development Program of China under Grant No. 863-306-ZD01-02-3 (国家 863 高科技发展计划基金) and the Youth Science Foundation of the University of Science and Technology of China under Grant No. 98-1101 (中国科学技术大学青年科学基金). **WANG Feng** was born in 1977. He is a M. S. student at Department of Computer Science and Technology, University of Science and Technology of China. He received his B. S. degree in computer science from University of Science and Technology of China in 1997. His research interests include parallel computing, parallel programming environments, design and analysis of algorithms. **AN Hong** was born in 1963. She received her Ph. D. degree in 2000 from University of Science and Technology of China. Her current research interests are parallel and distributed programming environments and tools, high performance computing. **CHEN Zhi-hui** was born in 1976. He is a M. S. student at Department of Computer Science and Technology, University of Science and Technology of China. His research interests include distributed computing and computer network. **CHEN Guo-liang** was born in 1938. He is a professor and doctoral supervisor of the Department of Computer Science and Technology, University of Science and Technology of China. His current research areas include parallel algorithms, high performance computing.

even though they have debugged them many times, because they cannot ensure they have covered all possible execution paths. This is the main shortcoming of passively debugging—users can only correct the bugs they can find. Till now, few debuggers have solved this problem[4,5].

In this paper, we present an implementation of record and replay technique taking advantage of the FIFO property of communication in MPI/PVM. Particularly, we provide an easy way to actively and completely debug parallel programs by covering all possible execution paths through controllable replay. We have implemented our approach in the DCDB[6], a portable parallel debugger developed at the National High Performance Computing Center (NHPCC) at Hefei and the National Research Center for Intelligent Computing Systems (NCIC) of P.R. China. The DCDB was designed to debug parallel programs that are distributed across a large number of processors. It supports debugging of MPI/PVM programs. Currently, it is implemented on Dawning-2000 cluster at NCIC and a cluster of AIX workstations at NHPCC at Hefei. Our approach produces little extra time and space overhead, and is applicable to single threaded MPI/PVM programs written in Fortran or C.

We organize this paper as follows. The next section will briefly introduce the message passing communication models. In Section 2, we describe our implementation of record and replay technique. Section 3 focuses on how to completely debug parallel programs with the DCDB. Section 4 is a brief description of the related work. Lastly, we give some conclusions in Section 5.

1 Communication Models

There are two basic message passing communication models[7]: synchronous and asynchronous.

1.1 Synchronous model

In synchronous model, the message can be communicated only if the sender and receiver are both ready. The send and receive cannot return until the message is completely sent and received. There is no indeterminacy in synchronous communication, because one send corresponds to one receive. In each execution, the same receive will get the message from the same send. It is easy to debug such programs. But the parallelism is quite poor in communication intensive programs.

1.2 Asynchronous model

Asynchronous model is much more commonly used for cluster system. The sender need not wait for the receiver to be ready. The send event can return if the message is sent out (blocking) or even not (non-blocking), while receive perhaps does not begin yet. The parallelism is highly improved. But the determinacy is destroyed, because the one-to-one correspondence between the send and receive does not exist any longer.

1.2.1 FIFO communication model

Let's put some restriction on the above asynchronous model, that is, the messages should pass though a specific channel between every two processes[7]. This channel is FIFO. From process P_1 to process P_2, the first message sent should be received first. Then, there's no racing among the messages sent from P_1 to P_2. Racing exists only among the messages sent from different processes to a same receiver. The indeterminacy is reduced greatly.

Fortunately, MPI and PVM provide such model. There is a so-called "non-overtaking" rule[5] of point-to-point communication in MPI/PVM: if task 1 sends message A to task 2, then task 1 sends again message B to task 2, message A will arrive at task 2 before message B. Moreover, if both messages arrive before task 2 does a receive, then a wildcard receive will always return message A.

Here, a wildcard receive means a receive operation that does not designate the source process or the message tag.

2 Record and Replay in DCDB

2.1 Strategy

The basic idea of record and replay technique is that: record the execution traces during the first run, then replay it for cyclic debugging as many times as needed according to the traces. Then, the bugs appearing in the recording phase can be repeated when replaying.

The key issue is what should be recorded. There are mainly three kinds of recording. (1) Record the program statuses during the execution. In order to replay precisely, users should record as many statuses as possible. But the overhead in time and space is too much. (2) Record the content of the message received. This needs too much time and space when the messages are long. (3) Record the message sequence. The overhead is smaller than that of the first two methods. But it needs an extra clock, and its overhead is still a problem in the communication intensive programs.

In fact, from section 1.2.1, we can see that: firstly, it is not necessary to record information of all messages. Only those possibly racing messages are responsible for the indeterminacy. If there is no wildcard receive, then there will be no message racing. Secondly, there is no need to record all contents of the message. In MPI and PVM, the indeterminacy is mainly caused by wildcard receives. And because of the non-overtaking rule, the wildcard parameter of message tag can be neglected[5]. So we only need to record the identifiers of the processes whose sends are matched by receives having no designated source. That is enough for replaying.

2.2 Implementation

In the DCDB, we implemented record and replay technique by wrapping calls to the communication library with instrumentation reading or writing trace files. Besides the normal message passing functions, the new wrapped library for record gets the actual senders of wildcard receives from status information and writes the ids into a trace file. The new library for replay reads the trace file and simply replaces the wildcard parameter sender with the actual one. Thus it seems that there are no wildcard receives in the parallel programs.

The debugging process of indeterminate MPI/PVM programs is shown as Fig. 1.

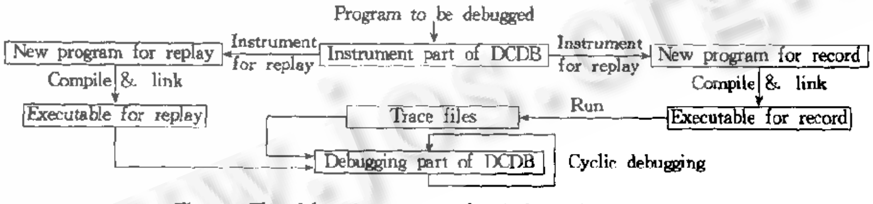

Fig. 1 The debugging process for indeterminate programs

Firstly, the DCDB instruments the original source files by replacing the MPI/PVM calls with wrapped ones. Then users compile the instrumented programs for record and replay. After running the new program for record, users will get some trace files. Lastly, by repeatedly executing the program for replay, users can cyclically debug their programs.

2.3 Overhead

A large fraction of the parallel debugging community believes that perturbations to the program and the sheer volume of trace data generated make tracing and determinate replay an impractical alternative[8]. For those traditional methods described in section 2.1, that is true. But for ours, it is not the case. Because of the FIFO property of communication between two processes, we can simplify record and replay greatly. Therefore, the extra overhead is highly reduced. Furthermore, the calls to wildcard receives are not too many in most MPI/PVM programs, so the temporal overhead can almost be ignored considering the influence of other factors. The spatial overhead is just the total size of trace files that are generally very small. Some data of overhead in time are shown in Table 1.

Table 1 Instrumentation overhead (second)

Benchmark	NAS Benchmark EP	NAS Benchmark IS
Processes	8	8
Problem size	$2^{26}=67108864$	$2^{20}=1048576$
Calls to wildcard receives	14	218
Time (uninstr.)	53.683	46.898
Time (instr. for record)	55.026	47.188
Time (instr. for replay)	55.527	47.595

3 Completely Debugging

There is a rule about testing in software engineering, that is, in order to ensure the correctness of a program, testing should adequately cover program logic and all conditions in the procedural design should be exercised[9]. That is to say, every possible branch of the program should be tested. This rule can be extended to the debugging of indeterminate parallel programs——the correctness can not be ensured until all the possible execution paths have been debugged. But without the control to the message sequence, it is nearly impossible to exercise all possible cases because the running environment, including number of processors, assignment of processes, system load and so on, is relatively fixed during a debugging session. Even users have corrected all the errors he found, there are probably some potential errors that will appear under different environments.

Most of the currently available debuggers have not considered this problem. The few tools that tried to solve it generated too much overhead in both time and space because of the tremendous number of all possible execution paths. In DCDB, we only record the identifiers of some message senders into the trace files. So we can easily control the message sequence by changing the sequence of the ids and then "replaying" the execution according to the new trace files. Except the simply changing of the trace files, our method does not cause other extra overhead.

Let's see an example. Fig. 2 shows a fraction of MPI source code srtest.c written in C:

```
......
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
MPI_Comm_rank(MPI_COMM_WORLD,&myid);
......
if (myid==0){
        ......
    for (i=0;i<numprocs-1; i++) {
        MPI_recv(rbuffer,BUFLEN,MPI_CHAR,MPI_ANY_SOURCE,99,MPI_COMM_WORLD, &status);
        .....
        }
    }
else{
    ......
    MPI_Send(sbuffer,strlen(sbuffer)+1,MPI_CHAR,0,99,MPI_COMM_WORLD);
    ....}
......
MPI_Finalize();
```

Fig. 2 An example for record and replay

We get the programs RC_srtest.c and RP_srtest.c for record and replay respectively by replacing the MPI functions with the new wrapped ones. Firstly we run RC_srtest in 4 processes to get the trace files. The content of the file for process ranking 0 is:

```
2
1
3
```

Other files are empty because the corresponding processes have no wildcard receives. In order to completely debug srtest. c, we only need to change the order of the process ranks into other 3!−1=5 possible combinations and "replay" according to them. However, there comes another problem: if the number of messages that are possibly racing is larger, e. g. the number of messages is 7, we must try 7!=5040 cases! Debugging will be impossible when the parallel program needs so long time to run. In fact, users can greatly reduce this number after analyzing their programs. If the contents of two racing messages are identical, the order of them will not influence the execution path. Furthermore, if the contents are different, but the difference does not involve the values on which the program will rely to determine the branch to go on, e. g., the sender id that only influences the printing results, so the order could be neglected. Once more, we reduce the overhead by only exercising the possible combinations of those racing messages whose contents are essentially different.

4 Related Work

Although many parallel debugging techniques and tools have been developed, only a few of them emphasize the importance of record and replay. Fewer consider the completeness of debugging.

Instant replay technique[1] gives a conceptual solution to how to re-execute parallel programs. It was used in an integrated toolkit for debugging and performance analysis of shared memory parallel programs. This toolkit supports program trace collection and trace visualization. Event resolution in this toolkit was limited to the accessing of shared variables, process communication events, and synchronization events.

The optimal tracing and replay[2,10,11] method is applicable to message passing and shared memory programs. It records the message sequence in a preliminary execution and then re-executes the program by forcing each message to be delivered as the recorded sequence. The method allows on-the-fly detection of racing messages. It does not consider the completeness of debugging either.

Mdb[4] is a tool for on-the-fly analysis. It will test all the possible orders in message passing programs. The racing messages are sent in different orders to see if they cause different executions.

NOPE[5] is a NOndeterministic Program Evaluator. It uses the non-overtaking rule to simplify the record and replay. With NOPE it is possible to uncover all possible execution paths of an indeterminate program. Firstly, all possible race conditions are computed. Secondly, event manipulation is used to exchange the order of events and replay steps are initiated for each combination of messages arriving at the wildcard receives. NOPE tries to solve the problem caused by the indeterminacy. Its drawback is that the overhead is too much to be tolerable.

5 Conclusions

As presented in this paper, the DCDB is a debugger for effectively solving the problem of completely debugging indeterminate MPI/PVM programs written in Fortran or C. Completely debugging has a significant beneficial effect on the development of parallel software. Nothing other than it can ensure the correctness of the programs. In comparison with other debuggers, our method produces much less overhead in time and space, and is easy to be controlled by users.

Now, the permutation of message orders that adequately cover all possible execution paths is generated by users. It is somewhat difficult and complicated for the users. Our further goals are concerned with generating automatically the permutations by analyzing the programs with our tools.

References:

[1] Leblanc, T. J., Mellor-Crummey, J. M. Debugging parallel programs with instant replay. IEEE Transactions on Comput-

ers, 1987,36(4):471～482.

[2] Netzer, R. H. B., Miller, B. P. Optimal tracing and replay for debugging message-passing parallel programs. In: Robert Werner ed. Proceedings of the Supercomputing'92. Los Alamitos: IEEE Computer Society Press, 1992. 502～511.

[3] Hicks, L., Berman, F. Debugging heterogeneous applications with pangaea. In: SIGMETRICS ed. Proceedings of the 1st Symposium on Parallel and Distributed Tools. New York: ACM Press, 1996. 41～50.

[4] Damodaran-Kamal, S. K., Francioni, J. M. Nondeterminacy: testing and debugging in message passing parallel programs. In: Barton, P. M., McDowell, C., eds. Proceedings of the ACM/ONR Workshop on Parallel and Distributed Debugging. New York: ACM Press, 1993,28(12):118～128.

[5] Dieter, Kranzlmueller, Jens Volkert. Debugging point-to-point communication in MPI and PVM. In: Alexandrow, V., Dongarra, J., eds. Proceedings of the EURO PVM/MPI'98 International Conference. Berlin, Heidelberg: Springer-Verlag, 1998. 265～272.

[6] Wang, Feng, Zheng, Qi-long, An, Hong, *et al.* A parallel and distributed debugger implemented with Java. In: Jian Chen, Jian Lu, Bertrand Meyer, eds. Proceedings of the 31st International Conference on Technology of Object Oriented Languages and Systems (TOOLS Asia'99). Los Alamitos: IEEE Computer Society Press, 1999. 342～348.

[7] Chen, Qing-ping. Research and implementation of parallel debugging techniques and tool for cluster system [MS. Thesis]. University of Science and Technology of China, 1998.

[8] Lumetta, S. S., Culler, David E. The mantis parallel debugger. In: SIGMETRICS ed. Proceedings of the 1st Symposium on Parallel and Distributed Tools. New York: ACM Press, 1996. 118～126.

[9] Pressman, R. S. Software Engineering——a Practitioner's Approach. Fourth edition, New York: McGraw-Hill, 1999.

[10] Netzer, R. H. B., Brennan, T. W., Damodaran-Kamal, S. K. Debugging race conditions in message passing programs. In: SIGMETRICS ed. Proceedings of the 1st Symposium on Parallel and Distributed Tools. New York: ACM Press, 1996. 31～40.

[11] Netzer, R. H. B. Optimal tracing and replay for debugging shared-memory parallel programs. In: Barton, P. M., McDowell, C., eds. Proceedings of the ACM/ONR Workshop on Parallel and Distributed Debugging. New York: ACM Press, 1993. 1～11.

不确定性 MPI/PVM 程序的完全调试

王 锋, 安 虹, 陈志辉, 陈国良

(中国科学技术大学 计算机科学技术系 国家高性能计算中心,安徽 合肥 230027)

摘要: 讨论如何完全地调试不确定性 MPI/PVM 并行程序. 在循环调试过程中,不确定性导致前次遇到的错误在以后的执行中很可能无法再现. 基于 MPI/PVM 的 FIFO 通信模型,给出一种记录-重放技术的实现. 通过可控制的重放,用户可以覆盖所有可能的程序执行路径,从而达到完全调试的目的. 和其它方法相比,所提供的方法所需时空开销要小得多. 此技术已在两种消息传递体系结构上得到实现:一种是曙光-2000超级服务器(由国家智能计算机研究中心开发),它由单处理器(PowerPC)结点经 MESH 网互联而成;另一种是国家高性能计算中心(合肥)的工作站(PowerPC/AIX)机群系统.

关键词: 并行调试器;记录-重放;消息传递;MPI;PVM

中图法分类号: TP311 **文献标识码**: A