Multi Failure Analysis In Parallel Processing Using Gift Tool

KR.Senthil Murugan[1] , V.Karpagam[2]
Department of Computer Science and Engineering
[1]Assistant Professor,Sri Krishna College Of Engineering and Technology, Coimbatore,Tamilnadu,India
[2]Assistant Professor(Sr.Gr.),KLNCEMadurai, TamilNadu, Inida

Abstract-*As the size of large-scale computer systems increases, their mean-time-between-failures are significantly shorter than the execution time of many current scientific applications. To complete the execution of scientific applications, they must tolerate hardware failures. Conventional rollback-recovery protocols redo the computation of the crashed process since the last checkpoint on a single processor. As a result, the recovery time of all protocols is no less than the time between the last checkpoint and the crash. In this paper, we propose a new application-level fault-tolerant approach for parallel applications called the Fault-Tolerant Parallel Algorithm (FTPA), which provides fast self-recovery. When fail-stop failures occur and are detected, all surviving processes recompute the workload of failed processes in parallel. FTPA requires the user to be involved in fault tolerance. Get it Fault-Tolerant (GiFT), a source-to-source precompiler tool to automate the FTPA implementation. The experimental results show that the performance of FTPA is better than the performance of the traditional check pointing approach.*

Keywords-Fault tolerance, fault-tolerant parallel algorithm, fast self-recovery, parallel recomputing.

I. INTRODUCTION

The size of high-performance computers from thousands to tens of thousands and even to hundreds of thousands of processors. Now, the fastest computer system in the world, IBM's Roadrunner, has 6,562 dual-core AMD Opteron chips, as well as 12,240 Cell chips [1]. However, as the complexity of a computer system increases, its mean-time between- failure (MTBF) is drastically decreased. The Google Cluster, using about 8,000 nodes, experiences a node failure rate of 2 percent-3 percent per year. This can be translated to a node failure every 36 hours [3]. On the other hand, many scientific applications are designed to run for weeks or even months. Therefore, the MTBF of these computers is becoming significantly shorter than the execution time of many current scientific applications. To complete the execution of such applications, they must tolerate hardware failures.

Check pointing is widely used in the domain of large scale systems, which periodically saves the state of a computation to a stable storage [4], [5]. Check pointing requires a cold restart of the entire parallel job when a process failed. In cold restart, a complete reload of all processes in the parallel job is conducted. Then, all processes have to roll back to the last checkpoint to restart the computation from there. Current fault-tolerant protocols redo the computation of the crashed process since the last checkpointon a single processor [6]. As a result, the recovery time of all protocols is no less than the time between the last checkpoint and the crash. To avoid the cold restart and to speed up the recovery procedure, we propose a new application-level fault tolerant approach called the Fault-Tolerant Parallel Algorithm(FTPA) [7]. FTPA is a parallel algorithm that provides fast self-recovery. When a process failure is detected, FTPA redistributes the workload of the failed process to the surviving processes, which then recompute the workload in parallel. Parallel recomputing speeds up the recovery
procedure.

II. RELATED WORK

Rollback recovery is a popular mechanism to incorporate fault tolerance into large-scale scientific applications [4], [6],[8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18].

Check pointing and message logging are two main rollback recovery techniques. Check pointing techniques can be classified into system level and application level. System-Level Check pointing (SLC) requires that all processes periodically checkpoint themselves by saving the content of their address space (including all values in the stack,heap, and global variables), registers, and the state of communication library to stable storage. In a system including tens and even hundreds of thousands of processors, terabytes of data might be transmitted to the stable storage through I/O components on one checkpoint.

In message-logging-based techniques [3], only the failed process needs to roll back to the last checkpoint, and surviving processes need not roll back but replay the messages

sent before failure to help failure recovery. This technique requires all processes to either save a copy of each message it sends or regenerate the messages on demand using approaches like reversible computation. overhead of writing checkpoint data. This method introduces Application-level check pointing(ALC)[7], aims at reducing the checkpoint size.

ALC provides the opportunity for users to save the minimum amount of data necessary to recover the program state. For applications on most platforms such as the IBM Blue Gene and the ASCI machines, ALC is the default approach for tolerating hardware failures. ALC complicates the coding of application programs, and it requires a user to guarantee the consistency of the global state and to decide what state needs to be saved. In automating ALC and designed the semiautomatic system C3. C3 is a coordination protocol that guarantees checkpoint consistency for the application-level coordinated non blocking check pointing of MPI programs. C3 saves the entire state when it makes a checkpoint. ALEC determines what state needs to be saved at each checkpoint and inserts code to save the state and to restore it during recovery.

There are usually two aspects in rollback-recovery protocols requiring improvement: 1) The current protocol requires a cold restart of the entire parallel job, which results in a long response time for users, and 2) The workload of the crashed process since the last checkpoint is recomputed on a single processor.

III. THE FAULT-TOLERANT PARALLEL ALGORITHM

This section describes the basic idea and design methodology of FTPA. In this paper, A scientific application is an SPMD-style program where all processes use the same program operating on a different part of the same data structure, and it coordinates and synchronizes execution through explicit message passing. A scientific application has a good load balance and regular communication patterns.

3.1 Basic Idea

FTPA is a parallel algorithm that can achieve fast self recovery.FTPA saves data at data-saving points for correct recovery during its execution. When a process fails, the failure will be detected by all surviving processes, which will re-execute the work lost on the failed process in parallel.

The definition of FTPA. Logically, a parallel program has a certain number of program sections, which are code fragments of the parallel program. parallel algorithm has ***program sections*** S0; S1; _ _ _ ; Sn. The design of FTPA allows the manipulation of each program section into a fault-tolerant program section with the insertion of a data saving section, a failure detectionsection, and a recovery section, as shown in Fig. 1.

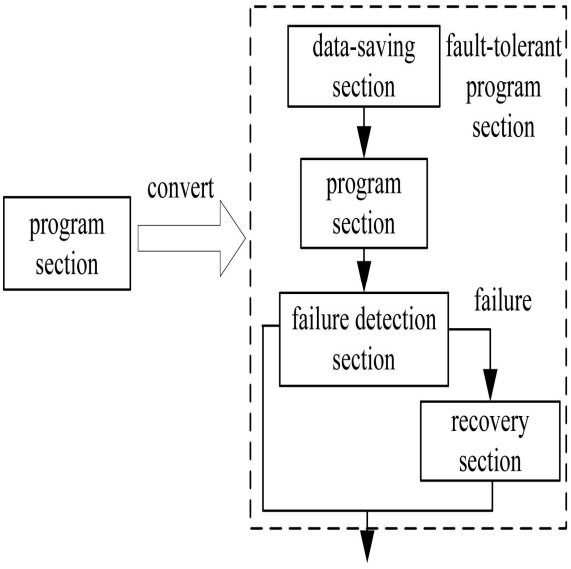

Fig. 1. Skeleton of a fault-tolerant program section.

The ***data-saving section*** needs to save data, which is a set of variables involved in the execution of the application to guarantee correct recovery for a parallel application.

The ***failure detection section*** checks the system failure vector FV to make it aware which process has failed. Let N denote the number of processes participating in the execution of an application; then, FV is an N-tuple, <F0; F1; _ _ _ ; FN_1> , where Fi represents the failure type of the process Pi.

The ***recovery section*** is implemented by transforming, following the SPMD programming paradigm, the original program section. Let Wj Sk be the workload of the failed process Pj that executes the program section Sk, and Wi RSk be the workload on every surviving process that executes the recovery section RSk corresponding to Sk.

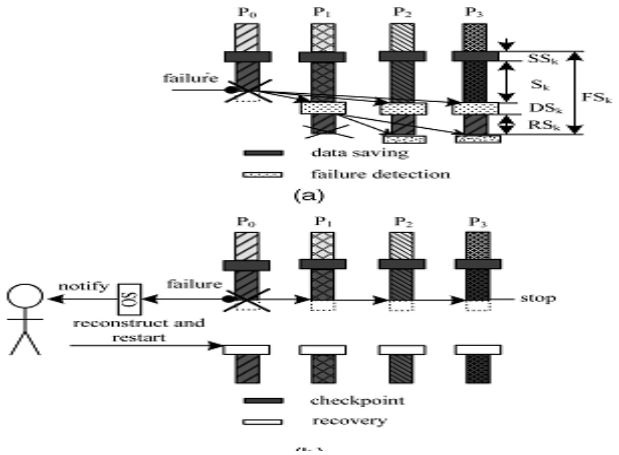

Fig. 2a represents how FTPA works.

FSk denotes the fault-tolerant program section related to Sk, while the data saving section and failure detection section in FSk are SSk and DSk, respectively. The FTPA application saves data in its data-saving section. If process Pj fails when executing the program section Sk, all other processes will detect the failure when they finish the execution of Sk.

The FTPA application's execution transfers to the recovery section to perform *parallel recomputing*. Thus, FTPA is an automatic fault-tolerant approach through algorithm design. It automatically detects possible failures and performs recovery upon failures. FTPA versus checkpoint/restart. To assist in a comprehensive understanding of FTPA's principle,
The primary difference between FTPA and check pointing is in how the failure is recovered, with the trade-off being that check pointing is simpler and FTPA is faster.

Fig. 2b shows how check pointing works. During the execution of an application, each process in the system saves its local computational state. If Pj fails, the application is restarted from a recently stored computational state, meaning all processes roll back and restart from the computational state. Relative to this, FTPA has a fast self recovery.

3.2 Partitioning a Program into Program Sections

A communication statements to partition a parallel program into program sections. In the partitioning method, if a branch structure contains communication statements and its conditional expression is related to the process rank, the branch structure is treated as a single communication statement. The following method can be used to partition a parallel program into program sections:

1. Determine the set of leaders, which are the first statements of program sections. The rules are given as follows:
    a. The first statement of a program is a leader.
    b. The statement that immediately follows a communication statement is a leader.
2. For each leader, it defines a program section that consists of the leader and all statements up to but not including the next leader or the end of the program.

3.3 Failure Detection Section Design Methodology

The failure detection section consists of routines perceiving possible failures according to the status of FV, which is determined by the parallel runtime environment. The failure detection routine has barrier synchronization semantics. A failure detection routine is inserted before the communication routine in each program section or prior to the termination statement of a parallel program, where the failure detection routine exploits the natural synchronization of scientific applications.

3.4 Data-Saving Section Design Methodology

To guarantee the correctness of the recovery, some variables required during recovery need to be saved in the data saving section. The design of the data-saving section is such that it can choose the variables. The variables to be saved in the data-saving section are used to recover the local computation of the failed processThese variables are defined prior to the program section. The definition of each variable, there is an element located at the point within or following the program section, i.e., the variables are live at the point immediately before the program section. Every process has to save live variables on the disk at the entrance of the program section.

3.5 Recovery Section Design Methodology

The recovery section consists of the data recovery code and the parallel recomputing code. The former is used to restore the saved data in the data-saving section. The latter is used to recompute the workload of a program section executed on the failed process. The parallel recomputing code only parallelizes the loops in the original program
section, while the remaining parts of the program section are recomputed serially.

IV. A TOOL FOR AUTOMATING FTPA IMPLEMENTATION

4.1 Overview of the Tool

FTPA is an application-level fault-tolerant approach. The requirements for users are that they choose program sections and design failure detection sections, data-savingsections, and recovery sections. The inclusion of the three sections of code in the application increases the user burden and reduces productivity. In order to ease the FTPA implementation, we develop *GiFT*, a source-to-source precompiler tool to transform an MPI/Fortran or MPI/C program with user-instrumented compiler directives into its FTPA version. The framework of GiFT is shown in Fig.3.

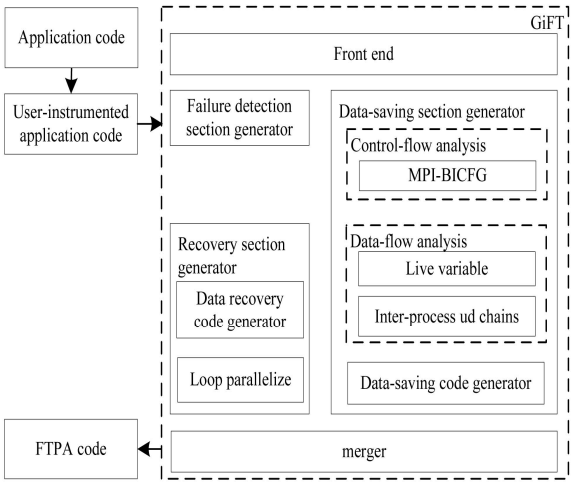

Fig 3 - Framework of the precompiler

GiFT is comprised of five components:

a front end, a failure detection section generator, a data-saving section generator, a recovery section generator, and a merger.

Front end.

The front end of GiFT is a modified version of gcc _ 4:2:1. It is used to identify the instrumented compiler directives and generate the symbol table and syntax tree for FORTRAN and C languages.

Failure detection section generator.

The failure detection section generator is used to produce the code of failure detection.

Data-saving section generator.

The data-saving section generator is used to figure out the variables necessary to be saved through control-flow analysis and data-flow analysis. It is also used to generate the code for saving them.

Recovery section generator.

The recovery section generator produces two code fragments: the data recovery code and the parallel recomputing code.

Merger.

The merger inserts the code of the above three sections into the proper places in a program section and turns it into a fault-tolerant program section.

4.2 Determining Program Sections

Program sections significantly determine the performance of the program's FTPA version. Theoretically, a program section may be the whole program or as little as a single statement. Should a program have several large program sections, its corresponding recovery section has a heavywork load, An MPI program is partitioned according to where some communication routines naturally occur in the program.

These communication routines include blocking point-to point communication routines MPI_Send/MPI_Recv and the completion routines MPI_Wait/MPI_Waitall which are used to complete non blocking send and receive, and all collective communication routines. To achieve an optimal program section size, the partitionedprogram sections can be split and combined. The program sections finally used by FTPA are produced by partitioning the original program or by splitting and combining the partitioned program sections:

1. Splitting a program section.

To reduce the workload of its corresponding recovery section, a large program section whose size is larger than the optimal program section size can be split into smaller ones.

2. Combining program sections.

To reduce the overhead of data saving, adjacent small program sections can be combined into a larger one, and the size of the combined one cannot exceed the optimal program section size.

A compiler directive ***CKPT HERE*** denotes an entry of a program section and also marks a state-saving point to save the live variables.

4.3 The Failure Detection Section Generator

The failure detection routine in GiFT is detect error, which detects the failure by the parallel runtime environment and includes barrier synchronization semantics.

4.4 The Data-Saving Section Generator

The generator consists of control-flow analysis and data flow analysis followed by a data-saving code generator.

The control-flow analysis phase provides the foundation for data-flow analysis and constructs control-flow graph representations of MPI programs.

The data-flow analysis phase aims at obtaining the variables that need to be saved in the data-saving section.

To perform data-flow analysis for an MPI program, the inter procedural data flow is a concern. Subroutines and functions present in an MPI program can be divided into three categories: user-defined, intrinsic, and MPI calls.

A data-saving code generator produces the code for saving variables obtained in the data-flow analysis.

4.4.1 Control-Flow Analysis

Most MPI programs do not have all of their MPI statements in one subroutine. A constructing an MPI-Inter procedural Control-Flow Graph (MPI-ICFG). They built the MPI-ICFG by first constructing an Interprocedural Control-Flow Graph (ICFG) and then adding communication edges between the MPI communication routines. A more effective strategy is to copy the control-flow graph for each process, provideeach process with its own variable namespace, model communication with global shared variables, and propagate data-flow information over communication edges. However, this approach is not scalable.

We constructed an MPI-Branch-based ICFG (MPI-BICFG) according to process-rank-based conditional statements.

4.4.3 The Data-Saving Code Generator

The generator generates the data saving code that is used to save the following two categories of variables:

- State-saving points.
- Definition points

4.5 The Recovery Section Generator

The code of the recovery section consists of the data recovery code and the parallel recomputing code. In the implementation of the recovery section, assume that when a processPjfails, a new process named recovered process is restarted to replace the failed one, and all surviving processes keep their old rank numbers. The processes involved in parallel recomputing are named recomputing processes.

4.5.1 The Data Recovery Code

The data recovery code is used to save the two categories of variables. The first is the live variables at state-saving points, and the second is the variables defined at definitionpoints that are on the inter process ud chains of the uses. The first is used to recover the local computation of the failed process, and the second is used to recover the data received by that process.

A code template is inserted at the beginning of every subroutine or function that can reach a CKPT_HERE. This code template will check whether the execution is restarted and read the saved live variables to recover the local computation if the processes fail.

V. CONCLUSIONS

In this paper, the concept of FTPA, which is a parallel algorithm, to achieve fast self-recovery. FTPA achieves fast failure recovery by using multiple surviving processes to re-execute the work lost on the failed process in parallel. However, it requires the user to be involved in fault tolerance. In order to ease the FTPA implementation, we developed GiFT, a source-to-source precompiler tool to automate the FTPA implementation. Through rank-based control-flow analysis and data-flow analysis, GiFT reduces the overhead of data saving.

REFERENCES

[1] IBM Roadrunner, http://www.ibm.com/, 2008.
[2] D.A. Reed, C. da Lu, and C.L. Mendes, “Reliability Challenges in Large Systems,” Future Generation Computer Systems, vol. 22, no. 3, pp. 293-302, 2006.
[3] G. Bosilca, A. Bouteiller, F. Cappello, S. Djilali, G. Fedak, C.Germain, T. Herault, P. Lemarinier, O. Lodygensky, F. Magniette,
[4] V. Neri, and A. Selikhov, “MPICH-V: Toward a Scalable Fault Tolerant MPI for Volatile Nodes,” Proc. ACM/IEEE Conf. Supercomputing (Supercomputing ’02), pp. 1-18, 2002.
[5] E.N. Elnozahy, L. Alvisi, Y.M. Wang, and D.B. Johnson, “A Survey of Rollback-Recovery Protocols in Message-Passing Systems,” ACM Computing Surveys, vol. 34, no. 3, pp. 375-408, 2002.
[6] A.N. Norman, C. Lin, and S.-E. Choi, “Compiler-Generated Staggered Check pointing,” Proc. Seventh ACM Workshop Languages, Compilers, and Runtime

Support for Scalable Systems (LCR ’04), pp. 1-8, Oct. 2004.

[7] S. Chakravorty and L.V. Kale, “A Fault Tolerance Protocol with Fast Fault Recovery,” Proc. 21st IEEE Int’l Parallel and Distributed Processing Symp. (IPDPS ’07), pp. 120-128, Mar. 2007.

[8] X. Yang, Y. Du, P. Wang, H. Fu, J. Jia, Z. Wang, and G. Suo, “The Fault Tolerant Parallel Algorithm: The Parallel Recomputing Based Failure Recovery,” Proc. 16th Int’l Conf. Parallel Architectures and Compilation Techniques (PACT ’07), pp. 199-209, 2007.