# A Fine-grained Fault Recovery Strategy for Task Pipeline

Jiantao He
College of Computer Science National University of Defense Technology
Changsha, China
Jiantao_he@yeah.net

Hong Ning
College of Computer Science National University of Defense Technology
Changsha, China
hning@nudt.edu.cn

***Abstract*—With the development of space technology, data processing and computational grow exponentially, leading to future spaceflight hardware platform structure changed from the traditional monolithic processor to multicore platforms. Thus the high-performance computing in space should be considered. Flowing parallel as an effective way to achieve high-performance computing is widely used in the field of space image processing. However, due to the complexity of the space environment, computing tasks easier fails due to failure during running. The traditional recovery policy of simply rebooting the entire system costs excessive computation time although it ensures the reliability of the calculation. This paper presents a fine-grained recovery strategy for the task pipeline model and it ensures the security of the tasks in pipeline with less performance cost.**

***Keywords-failure ; recovery ;task pipeline ; fine-grained***

## I. INTRODUCTION

With the development of the hardware architecture, multiprocessor platforms have become a trend [1]. While computation of the data in space is exponential times growth，such as the development of space remote sensing technology lead the resolution of spatial images been increasing. Thus it makes more requests for high-performance in space. Currently, parallel computing in multiprocessor architecture is the main research direction of high performance computing. Flow parallel as one of the parallel calculations is widely used in the aerospace field and is an effective way to improve computing performance. Meanwhile there is a large number of high-energy charged particles, such as electrons, protons, particles and heavy ions in outer space [2], these energetic particle bombardment aerospace equipment will occur Single Event Upset (SEU)[3,4] which perhaps leads the task to fail. Currently, the traditional fault recovery strategy is simple to restart the system, this coarse-grained recovery strategy to ensure reliability while offering high performance loss.

In this paper, we propose a granular recovery policy for flowing parallel task model; it would ensure the reliability of computing tasks on the basis of reducing the overhead of system for recovering.

## II. PARALLEL TASKS OF PIPELINE

Dividing a whole computing task into several pipeline parallel tasks are popularly available in consumer electronics systems such as multimedia processing and wireless communications [5]. For example, the upper portion of Figure 1 shows the JPEG image compression process [6]. The jpeg image compression algorithms can be divided to four mutually independent sub-tasks: image segmentation, Discrete Cosine Transform(DCT), zigzag scan sorting and coding. These sub-tasks compute in parallel pipeline. Each sub-task to get input data from the previous task, and send it after processing to the next task for processing until this data is finally out of the pipeline. At the same time, we have to assume that the output of each sub-task in the pipeline depends only on the current input data. The operation on the current data does not affect the processing of the following data，That is, data are mutually independent. The general flow parallel model is shown in the lower part of Figure 1

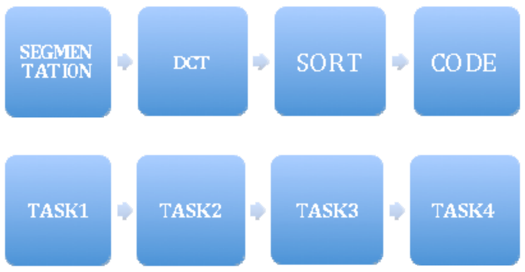


Figure 1. Flowing parallel model

## III. ARCHITECTURE PLATFORM

According to storage, Multiprocessor architecture platforms can be divided into shared memory and distributed memory two basic parallel computing platforms. As Shown in Figure 1 (a), in the shared memory parallel computer [7], Each processing unit exchanges information and coordinates parallel tasks on each processor through accessing to the shared memory. As Shown in Figure 1 (b), in distributed-memory parallel computer [8], each processor has its own independent local memory. Since there is no common storage units are available, data exchange, coordination and control of the execution of each processor implement through the messaging.

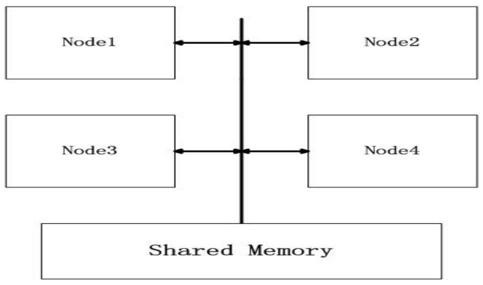


Figure 2(a). Shared MEM architecture

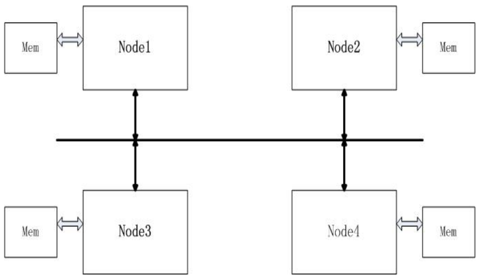


Figure 2(b). Distributed MEM architecture

The recovery strategy that this article discusses is based on a shared memory parallel architecture platform. Assuming each task distributions on the different processors in pipeline for data processing, while adjacent tasks pass data through the shared memory. As Shown in Figure 3, we need to create a buff pool in the shared memory for every two adjacent tasks in pipeline and hanging in the list of corresponding head node. There are three tasks in the pipeline: task A, task B and task C. Task A shares a buffer pool buff in the shared memory with Task B. Task A will put processed data into shared buff and task B obtains the input data from the buff, other so on.

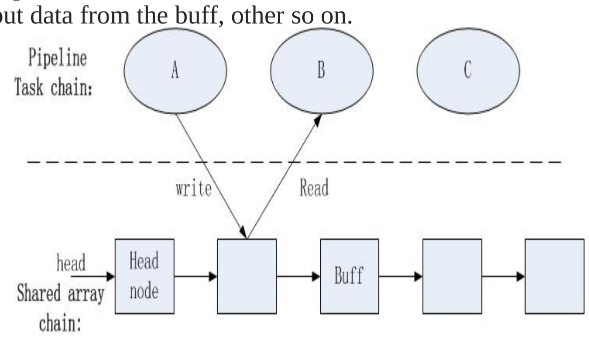


Figure 4. The shared array chain

## IV. FAULT RECOVERY STRATEGY

Before considering recovery, we should assume that the store is reliable. That is to say, the fault tolerant mechanisms for the shared memory will ensure that the data stored in the shared memory is correct. Now assuming a certain task in the pipeline failing, the processes for this certain task from failure to failure recovery as shown in Figure 4.

In the above four stages, the key is the third stage. When the task is restarted due to a fault, how to recover the communications between the failed task and the adjacent tasks is an important question. So we need to do the following aspects of works for answering this question.

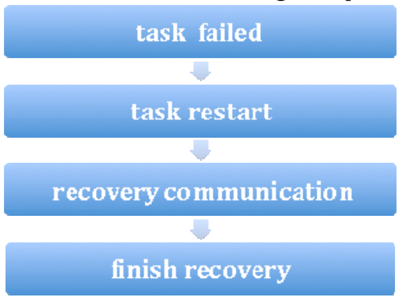


Figure 3. Failure recovery process

- Backup data

If the task in the pipeline restarts due to failure during computing, then the current data processed by this task will be lost. So we need to backup the data flowing in the pipeline. The following section describes the methods of data backup. Buffi still keeps the data which had been read by taski+1 until taski+1 has processed this data and send it to next task in the pipeline. In this case, the currently processed data will not be lost even the corresponding task restarts.

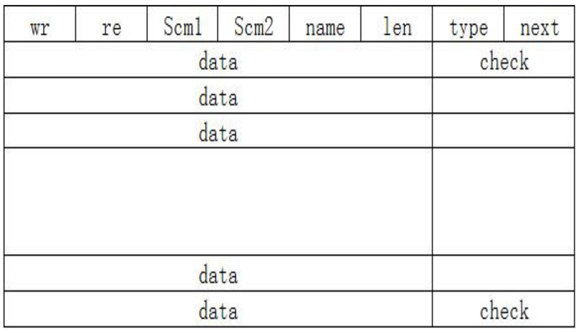
- Recover read and write pointers

When the task restarting due to failure, it need to know which data should been dealt with. To solve this problem, it needs to ensure that the restarted task must know where it will read the data from and write the processed data to. So every time the task to read from the buffer pool and write data to the buff pool, we must save or update the read and write pointers of this task.

- Recover semaphores

A task communicates with the adjacent tasks via the shared memory. But it needs to use semaphore mechanism to control synchronous communication between tasks. Assuming sem is one of semaphores in the program. If the task has do the operation of semTake(sem) but not do the operation of semGive(sem) when the task will be restarted, then sem will be incorrect after this task being restarted. So before every time to do the operation of semTake(sem), it should save the current value of sem to the shared memory for recovery and when the task is restarted it should reread the value of sem from the shared memory firstly.

The adjacent tasks in the pipeline communicate with each other via the buff pool in the shared memory. The design of the buff pool in the shared memory should ensure that the buff pool could help to backup data, recover semaphores and recover read and write pointers. So the structure of the buff pool is shown in Figure 5. Buffer pool structure contains two parts, identification and data. Identification section holds the buffer name, semaphores, the pointers that positioned reading and writing location after related task restart, as well as a pointer to point to the next buffer pool.



| wr | re | Scm1 | Scm2 | name | len | type | next |
|---|---|---|---|---|---|---|---|
| data | | | | | | check | |
| data | | | | | | | |
| data | | | | | | | |
| | | | | | | | |
| data | | | | | | | |
| data | | | | | | check | |

Figure 5. Structure of shared buff pool

## V. Experiment and Conclusion

The experiment uses the JPEG image compression program as an example and the system platform is Vxworks. The experiment divides the whole task of image compression into four pipelined sub-tasks firstly. These subtasks respectively responsible for image segmentation, Discrete Cosine Transform(DCT), zigzag scan sorting and coding.

The first step, the recovery strategy of this paper can recover it or not when failure occurs in the pipeline. The whole pipeline will stop when one of the sub-tasks in the pipeline fails if not using any recovery mechanism. Then the pipelines can recovery from the failure if using the recovery method proposed by this paper.

The second step, the experiment compares the efficiency of the recovery strategy of this paper with the traditional method that reboots system for recovery. The comparison is mainly from two aspects.

- In the case of a single fault, we compare the running time of compressing images of different sizes using two recovery strategies for recovery respectively, as shown in Table 1. From Table 1, it tells us that the efficiency of recovery increases greater with the increase of the size of the image when comparing the recovery strategy proposed by this paper with the traditional method. Now the size of aerial image is larger and larger, so this recovery strategy is more appropriate.
- Now the size of the tested image is 2048*2048, and the compression programs will suffer different times of failures. The running time of compression program using two recovery strategies for recovery respectively, as shown in Table 2. Table 2 shows that the efficiency of recovery increases greater with the increase of the number of the faults when comparing the recovery strategy proposed by this paper with the traditional method.

TABLE I. THE RUNNING TIME SUFFERED FROM SINGLE FAULT

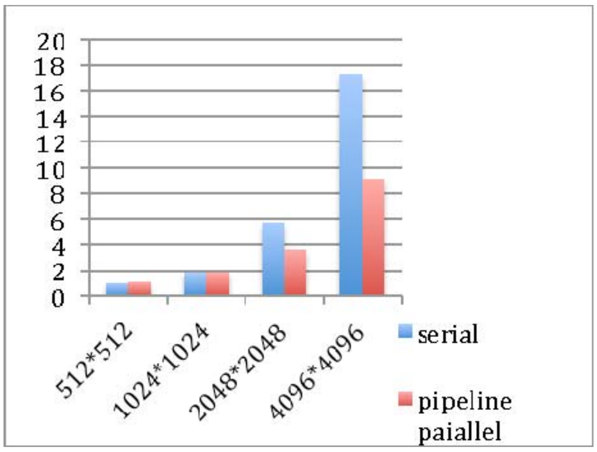
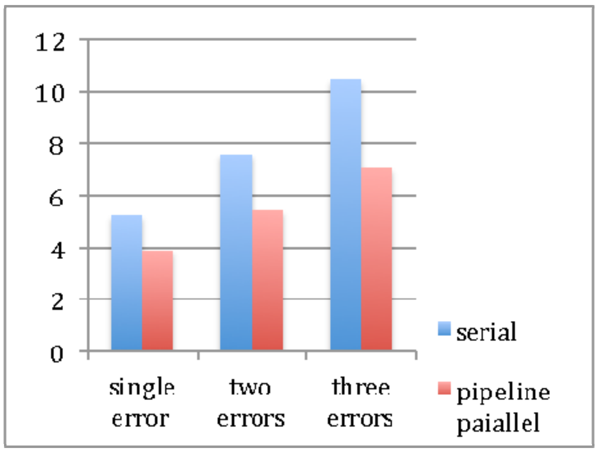


TABLE II. THE RUNNING TIME SUFFERED FROM MULTIPLE FAULTS



Though the efficiency is not remarkably improved, the pipelining of compression program of jpeg pictures does effectively shortened the compression of high-resolution pictures. Moreover more efficiency would be achieved when dealing with bigger pictures. The sub-tasks in the pipeline and related synchronized communication, whenever any fault occurs, could be restored. And the efficiency of fault recovery improves with the size of the picture, thus greatly reducing the time consumption on recovery. Considering pictures needed in the astronomy fields are to be compressed, our approach is hugely timesaving when compressing pictures and recovering from possible faults.

## Acknowledgment

This work was supported by National University of Defense Techonology in Changsha, China..

## References

[1] Javaid H, Parameswaran S. Synthesis of heterogeneous pipelined multiprocessor systems using ILP: jpeg case study[C]//Proceedings of

the 6th IEEE/ACM/IFIP international conference on Hardware/Software codesign and system synthesis. ACM, 2008: 1-6.

[2] J. H. Adams. The Natural Radiation Environment inside a Spacecraft. IEEE Trans. on Nuclear Science. 29(6):2095-2100,1982.

[3] Karlsson, J.; Liden, P.; Dahlgren, P.; Johansson, R.; Gunneflo, U.;Using heavy-ion radiation to validate fault-handling mechanisms,Micro, IEEE Volume 14, Issue 1, Feb. 1994 Page(s):8 – 23.

[4] R. Koga, S. D. Pinkerton, T. J. Lie, et al. Single-word multiple-bit upsets in static random access devices, IEEE Trans. on Nuclear Science. 40(6):1941-1946,1993.

[5] Yu Z, Shi Z, Zeng X. Fault tolerant computing for stream DSP applications using GALS multi-core processors[C]//Circuits and Systems (ISCAS), 2011 IEEE International Symposium on. IEEE, 2011: 2305-2308.

[6] Wallace, Gregory K. "The JPEG still picture compression standard." Communications of the ACM 34.4 (1991): 30-44.

[7] Gottlieb, Allan, et al. "The NYU ultracomputer—Designing an MIMD shared memory parallel computer." Computers, IEEE Transactions on 100.2 (1983): 175-189.

[8] Kumar V, Grama A, Gupta A, et al. Introduction to parallel computing[M]. Redwood City: Benjamin/Cummings, 1994.