THESIS

AUTONOMIC COMPUTING EXTENSION USING SELF HEALING TECHNIQUE

SUKSAN LERTPAIBULPANYA

GRADUATE SCHOOL, KASETSART UNIVERSITY

2006

THESIS APPROVAL

GRADUATE SCHOOL, KASETSART UNIVERSITY

Master of Engineering (Computer Engineering)
DEGREE

Computer Engineering
FIELD

Computer Engineering
DEPARTMENT

TITLE: Autonomic Computing Extension Using Self Healing Technique

NAME: Mr. Suksan Lertpaibulpanya

THIS THESIS HAS BEEN ACCEPTED BY

THESIS ADVISOR
(Associate Professor Putchong Uthayopas, Ph.D.)

COMMITTEE MEMBER
(Associate Professor Arnon Rungsawang, Ph.D.)

COMMITTEE MEMBER
(Associate Professor Kitsana Waiyamai, Ph.D.)

DEPARTMENT HEAD
(Associate Professor Kemathat Vibhatavanij, Ph.D.)

APPROVED BY THE GRADUATE SCHOOL ON _____

DEAN
(Associate Professor Vinai Arkongharn, M.A.)

THESIS

AUTONOMIC COMPUTING EXTENSION USING SELF HEALING TECHNIQUE

SUKSAN LERTPAIBULPANYA

A Thesis Submitted in Partial Fulfillment of
the Requirements for the Degree of
Master of Engineering (Computer Engineering)
Graduate School, Kasetsart University
2006

ISBN 974-16-2759-9

Suksan Lertpaibulpanya 2006: Autonomic Computing Extension Using Self Healing Technique. Master of Engineering (Computer Engineering),
Major Field: Computer Engineering, Department of Computer Engineering.
Thesis Advisor: Associate Professor Putchong Uthayopas, Ph.D. 50 pages.
ISBN 974-16-2759-9

In a large scale system, the percentage of node fault will increase as the number of computation nodes increases. Thus, it is hard to control or recover from the fault. The natural fault tolerance, a method that relies on a self-recovery property of the application, can be used for certain class of algorithm to allow process to recover from fault. This approach can potentially require much less storage space to keep the backup data compared to the checkpoint base algorithm.

In this work, a performance improvement of the recovery process in a natural fault tolerance method is discussed. We propose the new method calls SUSADS which is a hybrid algorithm between checkpoint base algorithm and natural fault tolerance. SUSADS balances the network capability and the computing power consumption by sending more information to neighbor node. In this work, we use parallel iterative algorithm to introduce SUSADS and compare the performance of the proposed method with a traditional checkpoint and natural fault tolerance method. It has been found that the propose method is faster and more efficient.

_____ _____ __ / __ / __

Student's signature Thesis Advisor's signature

ACKNOWLEDGEMENT

Firstly, I would like to express my thanks and appreciation to my advisor, Associate Professor Putchong Uthayopas, for his great help to this project and also teaches me a lot on work and life. He is more than my professor. I still remember every word that he said to me about the responsibility of who is the professor. He said that *"… Professor is not only teaching you to gain knowledge or something in a book but Professor is responsible to you to make you be a good man ..."*

Thank you very much to my family for their love and warmhearted support. The other one that I would like to offer my thanks is Ms. Rasamee Methamontri, who always understands and walks beside me. Finally, many thanks to all my friends and my colleagues, who always give me a big hug and joyful world. Hopefully, I have not forgotten anyone, but if I have, it was an oversight and please let it be my fault.

Suksan Lertpaibulpanya

August 2006

TABLE OF CONTENTS

Page

TABLE OF CONTENTS (Cont'd)

Page

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES (Cont'd)

LIST OF ABBREVIATIONS

AC	=	Autonomic Computing
CFD	=	Computational Fluid Dynamics
DCP	=	Diskless Checkpoint
MPI	=	Message Passing Interface
PDE	=	Partial Differential Equation
PSUSADS	=	Parallel Speed Up Self-healing Algorithm with Domain Splitting
NFTM	=	Natural Fault Tolerance Model
SUSADS	=	Speed Up Self-healing Algorithm with Domain Splitting
VBF	=	Virtual Boundary Forecast

AUTONOMIC COMPUTING EXTENSION USING SELF HEALING TECHNIQUE

INTRODUCTION

Autonomic Computing (AC) is a new research topic on the building of a computing system that is capable of managing themselves. This approach has been inspired by the human autonomic nervous system that has the ability to self-configure, self-tune and even repair themselves without any human conscience involvement. The concept of developing the next era of computing systems is driven by the convergence between the biological systems and the digital computing systems. Since the system is growing very fast, the high performance system as BlueGene, which contains the number of processor up to 65,536 processor chips in the 2005 time frame, has more probability to face the system fault problem. The system which has a self-healing property will help an admin and user to maintain a system to successively work under inconsistency environment.

The iterative method is the classical method used to solve many important scientific and mathematical problems. The iterative method is implemented under the concept of self-healing property. Due to the very long computation times for large problem, most of researchers depend on parallel computing technique and platform to increase the performance. Al Giest (2002) pointed out that although the computing power is increasing rapidly, the fact that numerous numbers of processors has to be employed will create a problem of maintaining system reliability as well. When the number of computing nodes is increasing, the checkpoint method which requires the huge storage to backup the system may not be efficient or even practical. A natural fault tolerance based algorithm has been proposed to address the problem. In this approach, some redundant data will be distributed to neighboring node. Then, when some of the computation faults, the recovery can be done using that redundant data.

To increase the performance of the recovery process in the iterative method, we propose the hybrid method between Natural fault tolerance algorithm and Checkpoint method. The hybrid method works on the sub domain concept and adapts the step to create and distribute all sub domains to all computation nodes in the system. An analytical comparison of the standard checkpoint method, natural fault tolerance algorithm and hybrid natural algorithm in term of computation time and recovery time are provided. The Heat Transfer problem is used as the case study. The classical Jacobi method and steady state heat transfer problem are also used to evaluate the performance of our model.

OBJECTIVE

The purpose of this work is separated in two areas as follows:

1. Improving a performance of Natural fault tolerance, iterative method, which is the one main function of the Autonomic Computing concept. To achieve the objective so the new method should improve in the both two areas as follows:

   1.1 Improving on the recovery performance: The new method provides better performance when run on both dedicate and non-dedicate system.

   2.1 Reducing the memory usage: The new method does not use the huge resource to backup the data for recovery when a node fault.

2. Comparing a performance of basic natural fault tolerance method, classical checkpoint method and the proposed model.

LITERATURE REVIEW

IBM Corporation (2001) presented a new idea about a new intelligent system which is able to manage and maintain it self and called autonomic computing system. This system is implemented on the basic approach which is to build computing systems that are capable of managing themselves. This approach has been inspired by the human autonomic nervous system that has the ability to self-configure, self-tune and even repair themselves without any human conscience involvement. The concept of developing the next era of computing systems is driven by the convergence between the biological systems and the digital computing systems. To drive the automation operation in the system, IBM identifies the specific set of functions.
Eight key features characterize any AC system, cf. [IBM Corporation, 2001]:

1. An AC system possesses *system identity*, i.e., it has knowledge of its components, current status, functions, and interactions with the environment.

2. An AC system has the ability of *self-configuration and reconfiguration*, i.e., it can automatically perform dynamic adjustments to itself in varying and unpredictable environments.

3. An AC system performs *constant self-optimization*, i.e., it monitors its constituent parts and adapts its behavior to achieve predetermined system goals.

4. An AC system is *self-healing*, i.e., it is able to discover the causes of failures and then finds alternative ways of using resources or reconfiguring the system to keep functioning smoothly.

5. An AC system is capable of *self-protection*, i.e., it detects, identifies, and protects itself against various types of attacks to maintain overall system security and integrity.

6. An AC system uses *self-adaptation* to find ways to best interact with neighboring systems, i.e., it can describe itself to other systems and discover those systems in the environment.

7. An AC system is a *non-proprietary open solution based on standards* that provide the basis for interoperability across system boundaries.

8. An AC system has *hidden complexity*, i.e., it automates IT infrastructure tasks and relieves users of administrative tasks.

Regarding the area of self-healing system, Al Giest (2002) created a lab to test a performance of a normal operation to make a system more durable, fast recovery and adaptive on the high performance parallel computation environment such as BlueGene. These work shown that the natural fault tolerance, which is implemented by using an iterative method, more effective than classical fault tolerance method like Diskless Checkpoint method. These work also shown the number of concurrent node fault in the system which are not making the operation fail.

Barrett *et al* (1995) group iterative methods to two types which are Stationary methods and Nonstationary methods. The Stationary iterative method is the iterative method that performs in each iteration the same operations on the current iteration vectors and the Nonstationary iterative method is the iterative method that has iteration-dependent coefficients. The iterative method is used to solve a partial differential equation (PDE), which is an equation containing derivatives of function of two or more variables, Many phenomena studied by scientists and engineers can be modeled by PDEs such as airflow over and aircraft wing, blood circulation in the human body, water circulation in an ocean, heat transfer problem and etc. Those problems are solved by mapping a problem into mesh and using the computer to solve the problem but when a problem is more and more complex, one computation node will be not suitable.

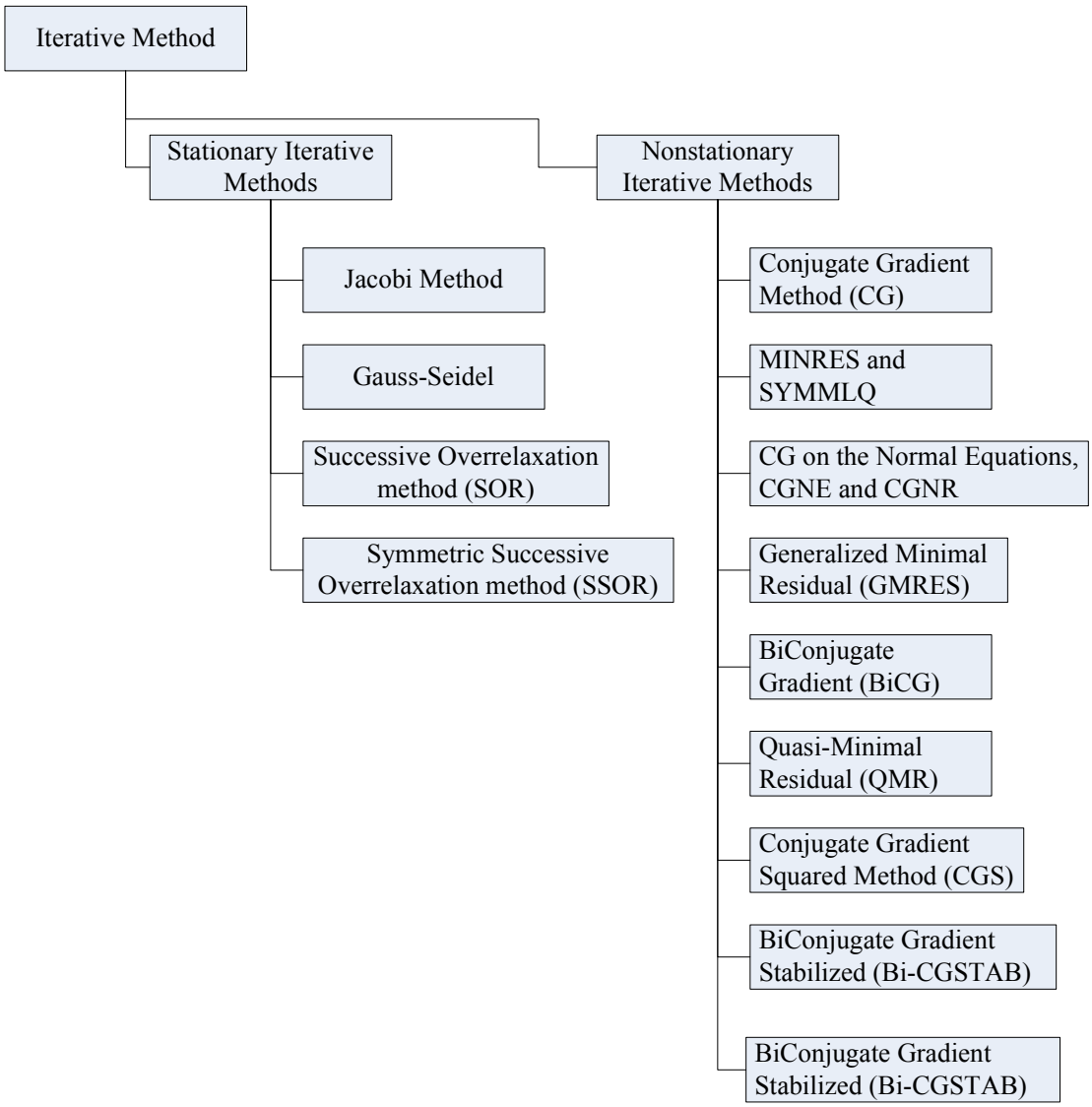


Figure 1 The hierarchical graph of the Iterative method.

To reduce the execution time in the heat transfer problem, Zhengquan (1996) discussed domain decomposition method of multi-grid distributed computing and pointed out that a good initialize value is very important when the equation is solved by using iterative method. Qingping *et al* (2000) pointed out that communication costs are crucial factors in network computing. Qingping *et al* (2000) proposed the VBF (Virtual Boundary Forecast) method to reduce the communication time to exchange the data between neighbor nodes by forecast the of boundary of data in each node to calculate the data for a few iteration times in local node and use the few

iteration to do the global smoothing to reduce the errors are caused by the sub-domain independent. Xin (2005) used the Jacobi Iterative method, which is classic method, to solve the heat transfer problem base on the PDE model to find the new temperature that span to the new area by using the neighbor areas. When parallelizing these algorithms, scientists mostly use MPI (Message Passing Library) as a tool.

In the MPI level, most of the fault tolerance methods are based on the checkpoint and roll-back technologies. The checkpoint method keeps the data in the stable storage (i.e. disk). Geist *et al* (1988) pointed the number of checkpoint times (ϕ) helps the system to keep the most updating data but it also increase the overhead significantly and make the overall of computation times much slower. So, they proposed the mechanism to select the appropriate checkpoint interval. Stellner (1996) proposed the CoCheck method which uses Condor library for check-pointing and entire MPI application. James *et al* (1998) proposed the idea to keep the checkpoint data in the memory to reduce the overhead of the I/O access times. Batchu (2001) introduced the MPI/FT which is a central co-coordinator and MPI process replication with voting algorithm among the replicates. MPICH-V, Bosilca *et al* (2002), from Universite'e de Paris Sud, France used the mixed of the uncoordinated check-pointing and distributed message logging techniques to improve the consistent state in any local set of process checkpoints. However, this method increases the resource requirement substantially which may led to a partial performance degradation. FT-MPI is much lower overhead implementation of the same approach. Fagg *et al* (2005) proposed the new recovery algorithm base on FT-MPI which aimed at being both scalable and latency tolerant. Since implementing fault tolerance in MPI level may consume a lot of resources, fault tolerance at the application level seems to be an attractive solution if applicable.

To improve the performance of the recovery process on MPI level, most researchers implement the recovery process on application level such as natural fault tolerance method instead. Although the benefit of fault tolerance on the MPI level is hiding a complexity but the natural fault tolerance is also hiding the complexity under the property of the method and condition of the math model. To prove the concept of

proposed model, both the Jacobi method and heat transfer problem, which is a one famous problem, are selected.

MATERIALS AND METHODS

Materials

Machine

1. Pentium IV 2.8 GHz with Memory 512 Mb
2. Intel (R) Xeon (TM) 1.4 GHz with Memory 256 Mb

Operating System

1. Linux Advance Server 3
2. Linux SUSE 9

Software

1. LAM/MPI
2. Java SDK 1.4

Methods

Introduction of Parallel Iterative Method

Most of the iterative algorithms, such as PDE solver and CFD solver, use mesh based data mapping for the calculation. The domain decomposition method is frequently used to break the data into sub-mesh is called domain.

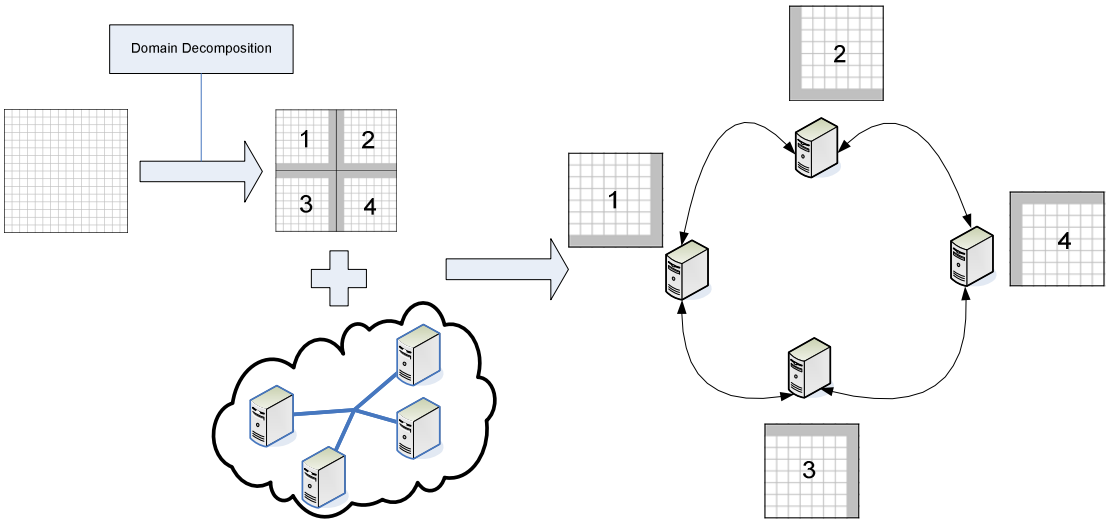


Figure 2 The work flow of the iterative method (for heat transfer problem).

Then, the domains are distributed to all computation nodes in the system for computation in parallel as Figure 3. Every time that the computation nodes have processed all the cells in the domain, we call as iteration. For each iteration, every node will exchange the cell values or ghost cells, which close to the neighbor node, between neighbor nodes. When a computation node is fault, the ghost cells, which are the most basic backup data, will be used to recovery the hold data from the fault node.

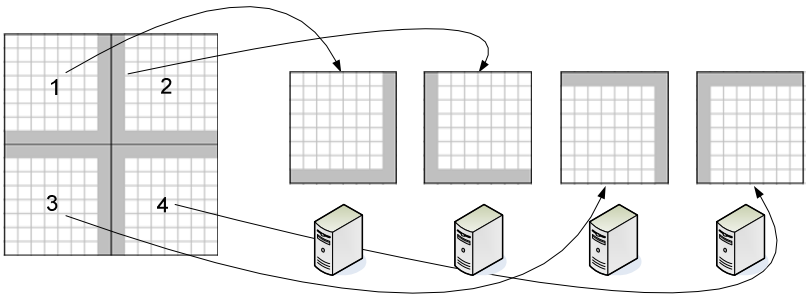


Figure 3 The domain is distributed to all computation nodes in the system.

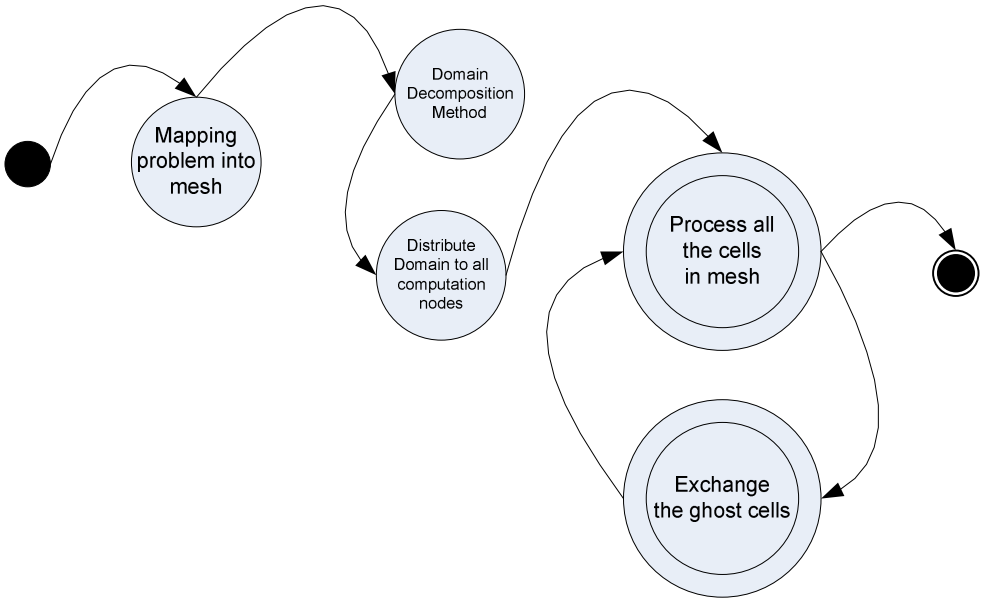


Figure 4 The work flow diagram of the iterative method.

Although the iterative method has a self-healing capacity but this model face a recovery problem when the problem size is huger size than the computing power of the recovery node. Because these model recoveries the data by reprocessing the problem form zero state so if the recovery process is take a lot of time to recovery the data, the problem of system fault will be face with. To improve the performance of recovery process, next section will present techniques are used to implement and also compare the pro and cons of each models.

Conventional Fault Tolerance Model

Natural Fault Tolerance Model

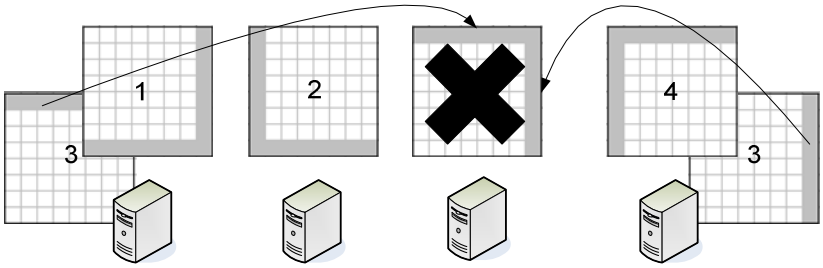


Figure 5 The recovery process of the NFTM model.

The NFTM model is the basic characteristic of iterative method which is able to recovery the node fault. This model uses the computing power to recovery the data by using the most updated data, which is a ghost cell, from the neighbor nodes to reprocess the data from zero state. The data recovered is the approximately value.

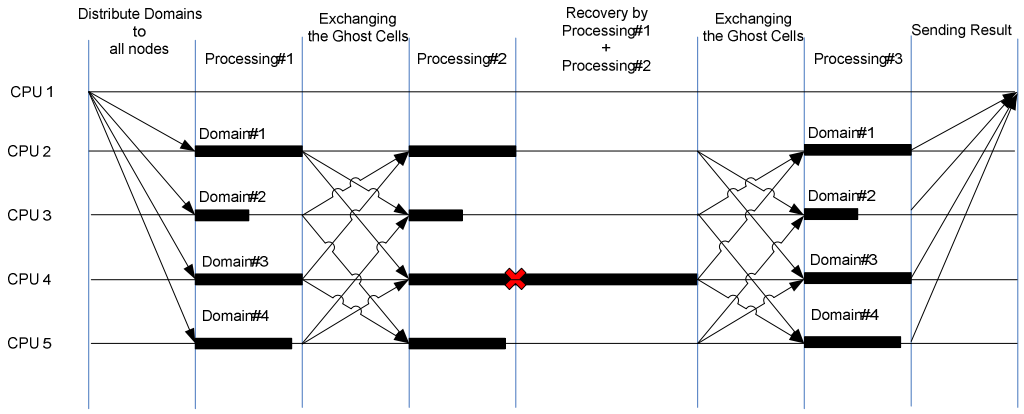


Figure 6 The execution time frame of NFTM model.

Diskless Checkpoint Model (DCP)

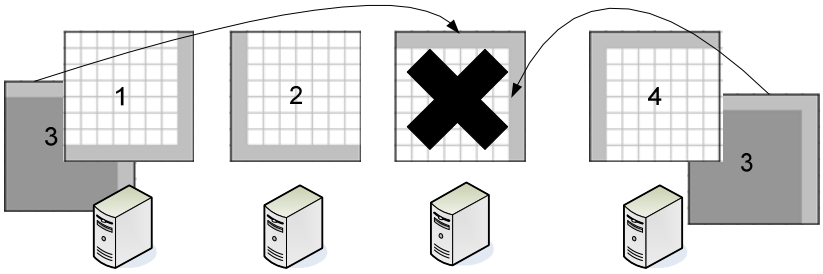


Figure 7 The recovery process of the DCP model.

The checkpoint model is a very well known mechanism of fault tolerance model. The performance of DCP depend on both the appropriate a number of checkpoint interval times (ϕ) and the problem size. For every checkpoint interval, every nodes send the hold data to the neighbor nodes so as to back-up and when a node is fault, the backup node be able to use the backup data from neighbor nodes and do not need to reprocess the data from zero state. This model also has a weak point which is this model need to have a memory or space to keep the backup data so this problem will limit the system computation spec or the size of the domain which is distributed to all computation nodes.

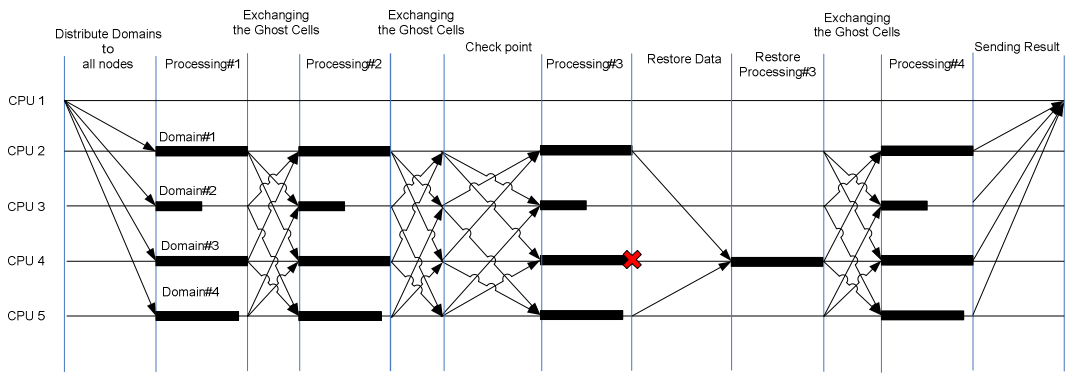


Figure 8 The execution time frame of DCP model.

Proposed SUSADS and PSUSADS Model

Speed Up Self-healing Algorithm with Domain Splitting (SUSADS)

We proposed the model names *Speed Up Self-healing Algorithm with Domain Splitting (SUSADS)* (2006), present the idea which relies on the most updated data. If the most updated data has been kept as much as possible without reducing the processing time, the size of the updated data increased will increase all of the system performance to recovery the data.

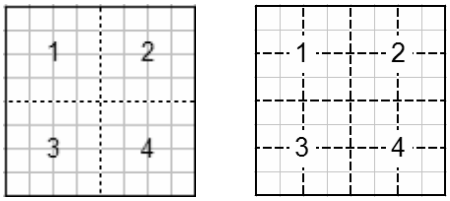


Figure 9 Domain Splitting model.
(a) Original domain pattern.
(b) Sub domain in the original domain.

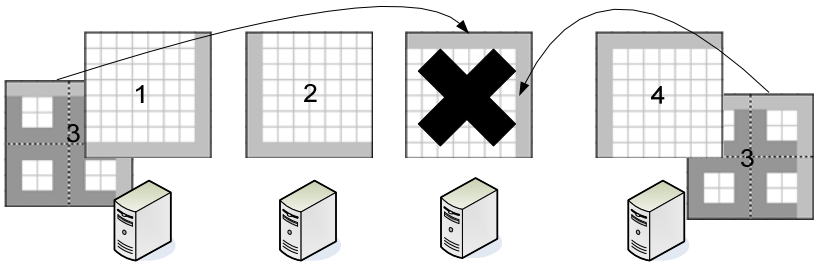


Figure 10 The recovery process for SUSADS model.

To achieve this idea, the domains have to be broken to smaller size, *Domain Splitting*, and the appropriate number of sub domain will need to be fined. Regarding the model (a) and model (b) in the Figure 9, it shows that both the processing times and the ghost cells of two models are not difference because of the number of cells in

each nodes are equal but the number of ghost cells of sub domain which are needed to be backup in the neighbor node increased. If one node in the system is fault, the recovery time of model (b) will be faster than model (a) so the appropriate value of the increasing number of sub domain in k times will speed up the recovery times.

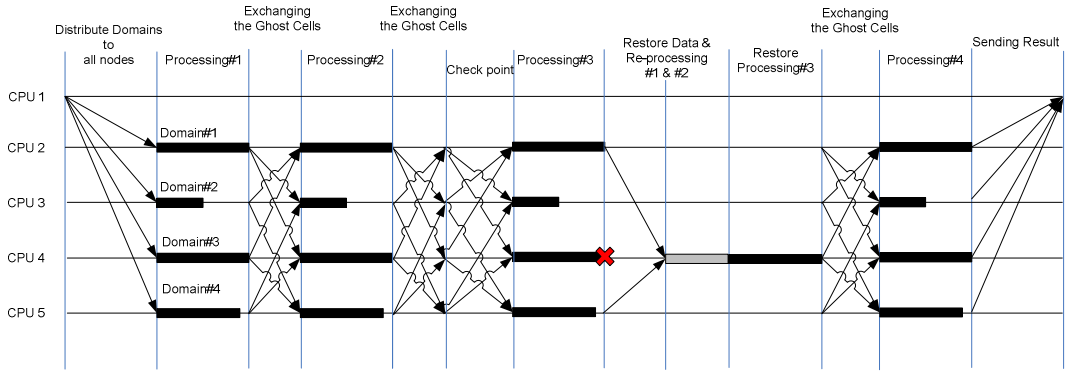


Figure 11 The execution time frame of SUSADS model.

(a) (b)

Figure 12 Recovery pattern of domain splitting model.

(a) Original domain pattern.

(b) Sub domain pattern.

Parallel Speed Up Self-healing Algorithm with Domain Splitting (PSUSADS)

Regarding the concept of the Domain Splitting, the Figure 12 shows that the domain splitting model can solve the none-converting problem on the recovery process. The ghost cells of sub domain are a controller is used to control the temperature of the cells which are covered by them. That make the data recovered is more trustable than the original domain pattern.

Following the Figure 11, a recovery node use own computing power to recovery the losing data. Although the reprocessing time reduced as a result of the number of backup data increased but it cannot guarantee that the performance of this model is better than the classical checkpoint method. To reduce the reprocessing time, the neighbor nodes are used to help the recovery node to recovery the data. Regarding the domain splitting and backup concept, the parallelization of a reprocessing of the neighbor nodes will increase the performance of recovery process to 2 or 4 times. This new model names *Parallel Speed Up Self-healing Algorithm with Domain Splitting (PSUSADS).*

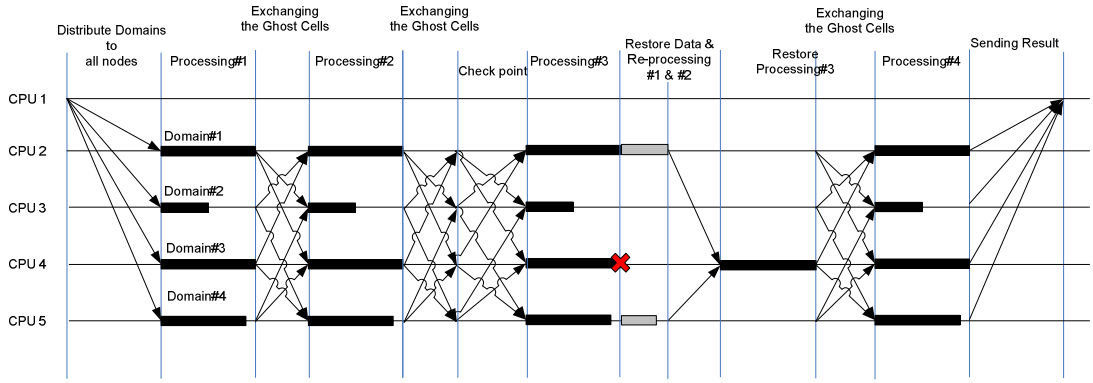


Figure 13 The execution time frame of PSUSADS model.

Mathematical Analysis of Parallel Iteration Model

Let T_{TOTAL} be the total of computation time. That will combine with three parameters. T_{DCOMP} be the computation time for each domain, T_{DCOMM} be the communication time between domains and β be the total number of iteration. The value of T_{TOTAL} is as given in Eq. 1.

$$T_{TOTAL} = \beta\left(T_{DCOMP} + T_{DCOMM}\right) \tag{1}$$

Let T_{CELL} be the execution time for each mesh point or cells in the domain. The value of T_{DCOMP} will be computed from the summation of the execution time of each cell in the domain as depicted in Eq. 2

$$T_{DCOMP} = \sum_{i=1}^{n} T_{CELL} \tag{2}$$

If the mesh size nxn is break into Ω domains, the number of cell will equal n^2/Ω. Then, the value of T_{DCOMP} is as given in Eq. 3.

$$T_{DCOMP} = \sum_{i=1}^{n^2/\Omega} T_{CELL} \tag{3}$$

The value of T_{DCOMM} will depend on the size of the data that need to be exchanged and network bandwidth (B). The data size can calculated by multiply the size of data type (D_{SIZE}) with number of ghost cells. The Eq. 4 gives the communication time between domains in square shape.

$$T_{DCOMM} \approx \frac{D_{SIZE}\left(\frac{\Omega}{P}\right)\left(\frac{4n}{\sqrt{\Omega}}\right)}{B} \approx \left(\frac{\Omega}{P}\right)\left(\frac{4nD_{SIZE}}{B\sqrt{\Omega}}\right) \tag{4}$$

There are two communication types which are implemented in MPI; point-to-point communication and collective communication (broadcast). Duncan A. Grove (2003) [12] tested the communication benchmark and showed that collective communication is the approximately $\log_2 P$ times of a point-to-point communication.

$$T_{DCOMM\_BC} \approx T_{DCOMM} \log_2 P \tag{5}$$

If the number of domain in each computation node is balanced, the number of sub domain (Ω) will equal the number of node (P) and T_{TOTAL} will be as given in Eq. 6.

$$T_{TOTAL} = \beta\left(\frac{\Omega T_{DCOMP}}{P} + T_{DCOMM}\right) \approx \beta\left(T_{DCOMP} + T_{DCOMM}\right) \tag{6}$$

Mathematical Analysis of Fault Recovery Using Natural Fault Tolerance Model (NFTM)

The simple model is used to represent the processing time is as given is Eq. 7.

$$T_{TOTAL\_NFTM} = T_{TOTAL} + T_{RECOVERY} \tag{7}$$

Let N_{FAULT} be the number of fault times that takes place during the execution time and $\beta_{RECOVERY}$ be the time used in recovery. Then, the recovery time is as given in Eq. 8.

$$T_{RECOVERY} = N_{FAULT}\beta_{RECOVERY}\left(T_{D\_RECOVERY}\right) \tag{8}$$

Where

$$T_{D\_RECOVERY} = \begin{cases} 0 & , N_{DOMAIN} \le G_{CELL} \\ \sum_{i=1}^{N_{DOMAIN}-G_{CELL}} T_{CELL} & , N_{DOMAIN} > G_{CELL} \end{cases}$$

$$N_{DOMAIN} = \frac{n^2}{P}$$

$$G_{CELL} = \frac{4n}{\sqrt{\Omega}} = \frac{4n}{\sqrt{P}}$$

When nodes process the large problem size, the number of the iteration is used to recover the data in this model will increase ($\beta_{RECOVERY}$) to equals the number of the iteration is used to find the result (β). That is caused by the number of N_{DOMAIN} is more huge than the number of G_{CELL} . The total execution time for this algorithm will be changed as given in Eq. 9.

$$T_{TOTAL\_NFTM} = T_{TOTAL} + N_{FAULT}T_{DCOMP} \tag{9}$$

Mathematical Analysis of Fault Recovery Using Diskless Checkpoint Model (DCP)

The simple model is used to represent the processing time of the DCP model will be showed as in Eq. 10 and Eq. 11 Let N_{FAULT} be the number of fault times that takes place during the execution time and ϕ be the number of the checkpoint interval times.

$$T_{TOTAL\_DCP} = T_{TOTAL} + T_{RECOVERY} \tag{10}$$

$$T_{RECOVERY} = T_{CHECK\_POINT} + N_{FAULT} T_{RESTORE} \tag{11}$$

Where

$$\begin{aligned} T_{CHECK\_POINT} &= \frac{\phi n^2 D_{SIZE}}{PB} \\ &= \frac{\phi n T_{DCOMM}}{4\sqrt{\Omega}} \\ &= \frac{\phi n T_{DCOMM}}{4\sqrt{P}} \end{aligned}$$

$$\begin{aligned} T_{RESTORE} &= \frac{T_{TOTAL}}{\phi} + \frac{T_{CHECK\_POINT}}{\phi} \\ &= \frac{T_{TOTAL}}{\phi} + \frac{n T_{DCOMM}}{4\sqrt{P}} \end{aligned}$$

Following the Figure 8, $T_{RESTORE}$ will be combined with two parameter are the data restore ($\frac{T_{CHECK\_POINT}}{\phi}$) and time is using to reprocess the data after node fault ($\frac{T_{TOTAL}}{\phi}$).

Mathematical Analysis of Fault Recovery Using Speed Up Self-Healing Algorithm with Domain Splitting (SUSADS)

Regarding the domain splitting model, the recovery process of SUSADS model will give a better performance than NFTM model and the checkpoint time of this model is also less than the DCP model. The processing time will be represented as given is Eq. 12.

$$T_{TOTAL\_SUSADS} = T_{TOTAL} + T_{RECOVERY} \tag{12}$$

The processing time equation will not difference with equation (7) but the $T_{RECOVERY}$ will reduce, if we increase the number of ghost cells in $\sqrt{k}$ times.

$$T_{RECOVERY} = T_{CHECK\_POINT} + N_{FAULT} T_{RESTORE} \tag{13}$$

$$\begin{aligned} T_{CHECK\_POINT} &= \frac{4n\phi\sqrt{k}D_{SIZE}}{B\sqrt{P}} \\ &= \phi\sqrt{k}T_{DCOMM} \end{aligned} \tag{14}$$

$$T_{RESTORE} = \beta_{RECOVERY} T_{D\_RECOVERY} + \frac{T_{TOTAL}}{\phi} \tag{15}$$

$$T_{D\_RECOVERY} = \begin{cases} 0 & , N_{DOMAIN} \le G_{CELL} \\ \sum_{i=1}^{N_{DOMAIN}-G_{CELL}} T_{CELL} & , N_{DOMAIN} > G_{CELL} \end{cases}$$

$$N_{DOMAIN} = \frac{n^2}{P}$$

$$T_{TOTAL\_SUSADS} = T_{TOTAL} + \phi\sqrt{k}T_{DCOMM} + N_{FAULT} T_{RESTORE} \tag{16}$$

Mathematical Analysis of Fault Recovery Using Parallel Speed Up Self-healing Algorithm with Domain Splitting (PSUSADS)

The math model of PSUSADS model is not difference with SUSADS model but only the value of $T_{D\_RECOVERY}$ have to divide by the number of neighbor node. The average number of neighbor node is 4 so the new model of PSUSADS will represent as in the Eq. 17.

$$T_{TOTAL\_PSUSADS} = T_{TOTAL} + \phi\sqrt{k}T_{DCOMM} + N_{FAULT}T_{RESTORE} \tag{17}$$

When

$$T_{RESTORE} = \frac{\beta_{RECOVERY}T_{D\_RECOVERY}}{4} + \frac{T_{TOTAL}}{\phi} \tag{18}$$

RESULTS AND DISCUSSION

System Parameters

To study the performance of our approach, the heat transfer problem is used as the sample application to find the steady state of the thin steel plate is surrounded on three sides by condensing steam bath (temperature 100 degrees Celsius). The fourth side touches an ice bath (temperature 0 degrees Celsius). An insulating blanket covers the top and the bottom of the plate. The value of threshold is set to 0.01 to find the parameter values. Following the equations in the mathematical analysis section, the parameters are listed in the Table 1.

Table 1 List of the parameters is used to evaluate the performance.

No.	List of parameters
1	CPU
2	Memory
3	The average of the execution time for one cell.
4	The total number of the iteration is used to solve the problem.
5	The number of the iteration is used to recovery the data.
6	The appropriate checkpoint time.
7	The best value of *k*.

Following Table 2, the Linux AS3, 2.8 GHz Pentium IV and Memory 512 Mb is used to execute a heat transfer code to find the steady-state temperature. The mesh size start from 64x64 to 10,000x10,000 cells is used to find the number of iteration and average execution time of one cell (0.0874 microsecond). The result point that the number of iteration is going to the steady state when we process the problem size is larger than 100x100 cells.

Table 2 The average execution time of cell in each problem size.

Problem size	Number of Iteration	Execution Time (seconds)	Execution Time per cell (microseconds)
64x64	2184	1	0.112
100x100	3602	2	0.056
256x256	3602	14	0.059
1000x1000	3602	436	0.121
1024x1024	3602	998	0.264
10000x10000	3602	31544	0.088

Table 3 The number of iterations is used to recovery the data.

(a) A node is faulted when running the application to 25%. (b) A node is faulted when running the application to 50%.

(c) A node is faulted when running the application to 75%. (d) A node is faulted when running the application to 90%.

Problem Size	Number of sub domain					
	4	**16**	**64**	**256**	**512**	**1024**
16x16(253)	33	9				
64x64(2184)	390	162	52			
256x256(3602)	832	806	495	196		
1024x1024(3602)	832	832	832	806	691	495

(a)

Problem Size	Number of sub domain					
	4	**16**	**64**	**256**	**512**	**1024**
16x16(253)	48	11				
64x64(2184)	591	207	60			
256x256(3602)	1661	1403	637	230		
1024x1024(3602)	1663	1663	1161	1403	1055	637

(b)

Problem Size	Number of sub domain					
	4	**16**	**64**	**256**	**512**	**1024**
16x16(253)	62	13				
64x64(2184)	748	244	67			
256x256(3602)	2468	1788	743	250		
1024x1024(3602)	2493	2493	2466	1789	1232	743

(c)

Problem Size	Number of sub domain					
	4	**16**	**64**	**256**	**512**	**1024**
16x16(253)	70	15				
64x64(2184)	834	264	71			
256x256(3602)	2930	1926	794	258		
1024x1024(3602)	2991	2991	2928	1927	1304	794

(d)

Following the Table 3, the same machine spec of the machine in Table 2 is used to execute the heat transfer problem by using multithread which is implemented by Java under the assumption of one fault node when running the application to 25%, 50%, 75% and 90% and set the value of threshold to 0.01. The results show that whatever how long the processing time is used, the sub domain size 16x16 cells can be solved by using the number iteration less than 253 iterations, the size 64x64 cells can be solved by using the number of iterations less than 2184 iterations and the size more than 256x256 cells can be solved by using the number of iterations less than 3602 iterations.

Figure 14 The procedure is used to find the number of iteration by using thread model.

```
1   Initial the value of each cell in the mesh size nxn.
2   Define EPSILON equals 0.01
3   Use domain decomposition technique to separate the problem to sub-domain.
4   Create the thread equals the number of sub-domains and running the following
    steps:
5       Assign the value of diff= 0.1 and iteration=0
6       Create loop while until diff equal or less than EPSILON value
7          for i equals 0 to n
8              for j equals 0 to n
9                 bk_cell[i, j] = (cell[i+1, j] + cell[i-1, j] + cell[i, j+1] +  cell[i, j-1])/4
10                if  (fabs(bk_cell[i, j] – cell[i, j]) > diff)
11                   diff = fabs(bk_cell[i, j] – cell[i, j]);
12         iteration=iteration+1
13         if  ((job = fault_job) && (iteration = 90% of total iteration)
14             Reset the value of cell follow the condition
15             Processing the recovery process.
```

This model also uses the LAM/MPI to implement the heat transfer problem to find the number of iterations is used to find the result and recovery the data. The heat transfer code is running on the LAM/MPI to cross check with the result of thread

model. We use two difference machine specs, which are the Linux AS3 on 2.8 GHz Pentium IV with memory 512 Mb and SUSE 9 on Intel Xeon 1.4 GHz with memory 256 Mb, to create the parallel environment. These two machines are set as five CPUs to simulate as five computation nodes for executing the code.

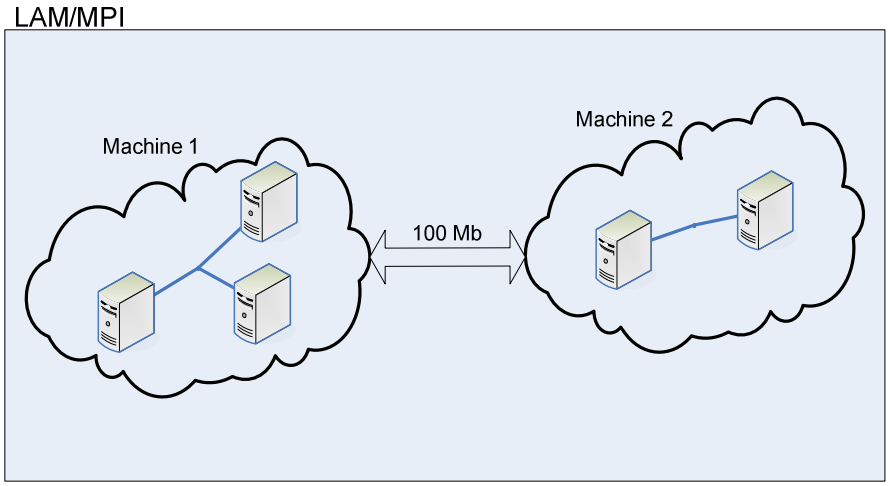


Figure 15 The network diagram of the sample parallel computing on LAM/MPI.

The pseudo code in the Figure 16 and 17 show the working steps of the MPI code. The number of iteration is used to recovery the data is not difference with the result in the thread model. The number of iteration, which is used to recovery the data, is showing in the Figure 18.

Figure 16 The procedure is used to find the number of iteration by using MPI model (Master Node).

```
1   Initial the value of each cell in the mesh size nxn.
2   Define EPSILON equals 0.01
3   Use domain decomposition technique to separate the problem to sub-domain.
4   Send the worker id, offset, number of rows, neighbor#1 and neighbor#2 to all
    computation nodes.
5   Sending the cells in sub-domain to all computation nodes.
6   While until all node return the diff value less than EPSILON.
7       Receiving the result of diff value from all nodes.
8     Sending the result of diff value to all nodes.
9   Waiting for the result from all computation nodes.
```

Figure 17 The procedure is used to find the number of iteration by using MPI model (Slave Node).

```
1   Define EPSILON equals 0.01
2   Receive the worker id, offset, number of rows, neighbor#1 and neighbor#2 from
    master node.
3   Receive the cells in sub-domain from master node.
4   Assign the value of diff= 0.1 and iteration=0
5   Create loop while until diff value from master equal or less than EPSILON value.
6       Sending the ghost cells to neighbor#1.
7     Receive the ghost cells from neighbor#1.
8     Sending the ghost cells to neighbor#2.
9     Receive the ghost cells from neighbor#2.
10    for i equals 0 to n
11          for j equals 0 to n
12              bk_cell[i, j] = (cell[i+1, j] + cell[i-1, j] + cell[i, j+1] +
     cell[i, j-1])/4
13              if  (fabs(bk_cell[i, j] – cell[i, j]) > diff)
14                  diff = fabs(bk_cell[i, j] – cell[i, j]);
15      iteration=iteration+1
16      if  ((job = fault_job) && (iteration = 90% of total iteration)
17          Reset the value of cell follow the condition.
18          Processing the recovery process.
19    Sending the diff value to master node.
20    Receiving the diff result from master node.
21  Sending all the cells in sub-domain to the master node.
```

Table 4 The number of iteration is used to recovery the data from one node fault when running the application to 90%.

	Sub-Domain Size 1024x1024				
	16	**32**	**64**	**256**	**512**
Number of Iteration	200	900	2100	2300	2500

Figure 18 Example out put of MPI program.

```
/home/song/Thesis/MPI/C++:> mpirun C heat 1024 16 200 1
=========================================
 Number of arg : 4
 Problem size  : 1024
 Sub domian size : 16
 Number of Recovery Iteration : 200
=========================================
=========================================
 Number of arg : 4
 Problem size  : 1024
 Sub domian size : 16
 Number of Recovery Iteration : 200
=========================================
=========================================
 Number of arg : 4
 Problem size  : 1024
 Sub domian size : 16
 Number of Recovery Iteration : 200
=========================================
=========================================
 Number of arg : 4
 Problem size  : 1024
 Sub domian size : 16
 Number of Recovery Iteration : 200
```

Figure 18 (Cont'd).

```
========================================
 Number of arg : 4
 Problem size  : 1024
 Sub domian size : 16
 Number of Recovery Iteration : 200
========================================
Grid size: X= 1024 Y= 1024
Initializing grid and writing initial.dat file...
Initializing grid and writing initial.dat file[Slave]...
Initializing grid and writing initial.dat file[Slave]...
Initializing grid and writing initial.dat file[Slave]...
Initializing grid and writing initial.dat file[Slave]...
Finished writing initial.dat file...
Sent to= 1 offset= 0 number_rows= 256 neighbor1= 0 neighbor2=2
worker number = 1 offset= 0 number_rows= 256 end =255
Sent to= 2 offset= 256 number_rows= 256 neighbor1= 1 neighbor2=3
worker number = 2 offset= 256 number_rows= 256 end =511
Sent to= 3 offset= 512 number_rows= 256 neighbor1= 2 neighbor2=4
worker number = 3 offset= 512 number_rows= 256 end =767
Sent to= 4 offset= 768 number_rows= 256 neighbor1= 3 neighbor2=0
worker number = 4 offset= 768 number_rows= 256 end =1023
========================================
 Iteration before reset : 3241
 Value of K size : 16
 Number of Recovery Rounds : 200
========================================
 NUMBER OF ITERATION : 1 : 3602
 NUMBER OF ITERATION : 2 : 3602
 NUMBER OF ITERATION : 3 : 361
 NUMBER OF ITERATION : 4 : 3602
```

Following the Table 5 and 6, the equations in the mathematical analysis section will be used to find the value of checkpoint time and the 4 numbers of computing node is used to solve the problem size 16,384 x 16,384 cells on the assumption that the node is faulted when run to 90%. The best checkpoint time value is 4% (25 times) of total running times and the best sub domain size is 256 (the value of k be 262,144).

Table 5 The performance of DCP and NFTM models when the number of checkpoint time changed.

Number of CP times	Execution Time (minutes)				% Performance of DCP compare with NFTM	
	Without Node fault		Node fault		Without Node fault	Node fault
	NFTM	DCP	NFTM	DCP	NFTM	NFTM
1	21.19	21.22	42.31	42.44	-0.16%	-0.30%
2	21.19	21.26	42.31	31.88	-0.32%	32.71%
5	21.19	21.36	42.31	25.63	-0.80%	65.09%
10	21.19	21.53	42.31	23.68	-1.60%	78.66%
11	21.19	21.57	42.31	23.51	-1.77%	79.98%
13	21.19	21.62	42.31	23.35	-1.99%	81.25%
14	21.19	21.68	42.31	23.20	-2.26%	82.42%
17	21.19	21.76	42.31	23.07	-2.63%	83.45%
20	21.19	21.87	42.31	22.97	-3.14%	84.23%
25	21.19	22.05	42.31	22.93	-3.90%	84.55%
33	21.19	22.33	42.31	23.00	-5.13%	83.95%
50	21.19	22.91	42.31	23.36	-7.50%	81.11%
100	21.19	24.62	42.31	24.87	-13.95%	70.14%
500	21.19	38.37	42.31	38.44	-44.78%	10.07%
1000	21.19	55.55	42.31	55.60	-61.86%	-23.90%
2000	21.19	89.91	42.31	89.95	-76.43%	-52.96%
3602	21.19	144.95	42.31	144.99	-85.38%	-70.82%

Table 6 Comparing the performance between SUSADS and other models when the sub domain size changed.

Sub Domain Size	Execution Time (minutes)					
	Without Node fault			Node fault		
	NFTM	DCP	SUSADS	NFTM	DCP	SUSADS
16	21.19	22.05	22.05	42.31	22.93	22.93
256	21.19	22.05	21.40	42.31	22.93	23.40
4096	21.19	22.05	21.19	42.31	22.93	42.31
65536	21.19	22.05	21.20	42.31	22.93	42.31

Performance Comparison

The goal of this work is to compare the performance of each model in both processing time and memory usage areas.

Processing Time

The number of computing nodes and the problem size are the main parameters to evaluate our model and compare with other models. So the evaluate pattern will base on the two assumptions as follows:

- Fix the number of computation nodes but vary the problem size.
- Fix the problem size but vary the number of computation nodes.

Following the Eq. 7, the performance of each model is base on the processing time and recovery time. The NFTM model is using the computation power to recovery the data. This model works under the idea that the computing power is more powerful than the network speed. The DCP model is the most popular model that more powerful when running in the very fast network environment that reduces the transfer time to backup the data. When compare the performance of the PSUSADS, SUSADS and DCP models, the Eq. 11 and 16 show that the value of T_{TOTAL} for both models are not difference so the performance of both models will base on the recovery time.

$$T_{DCP\_RECOVERY} = \left(\frac{\phi n}{4\sqrt{P}} + \frac{nN_{FAULT}}{4\sqrt{P}} \right) T_{DCOMM} + \frac{N_{FAULT} T_{TOTAL}}{\phi} \quad (19)$$

$$T_{SUSADS\_RECOVERY} = \phi\sqrt{k} T_{DCOMM} + N_{FAULT} \left(\beta_{RECOVERY} T_{D\_RECOVERY} + \frac{T_{TOTAL}}{\phi} \right) \quad (20)$$

$$T_{PSUSADS\_RECOVERY} = \phi\sqrt{k} T_{DCOMM} + N_{FAULT} \left(\frac{\beta_{RECOVERY} T_{D\_RECOVERY}}{4} + \frac{T_{TOTAL}}{\phi} \right) \quad (21)$$

Table 7 The result of the performance comparison value of NFTM and other models when the problem size changed.

Number of Node	PB Size ($x10^6$)	PB Size / Number of Nodes	Execution Time (Seconds)							
			P2P				Broadcast			
			NFTM	DCP	SUSADS	PSUSADS	NFTM	DCP	SUSADS	PSUSADS
4096	1.048576	256	0.01	0.02	0.01	0.01	0.11	0.14	0.11	0.11
4096	16.777216	4096	1.76	1.81	1.77	1.77	6.95	7.58	7.11	7.11
4096	268.435456	65536	22.52	23.36	22.73	22.73	43.29	53.36	45.81	45.81
4096	4294.967296	1048576	337.66	351.08	341.02	341.02	420.75	581.82	461.02	461.02
4096	68719.47674	16777216	5311.93	5526.68	5365.62	5365.62	5644.31	8221.29	6288.55	6288.55
4096	1099511.628	268435456	84628.32	88064.29	85487.31	85487.31	85957.81	127189.50	96265.73	96265.73

Table 8 The percentage of the performance comparison value of NFTM and other models when the problem size changed.

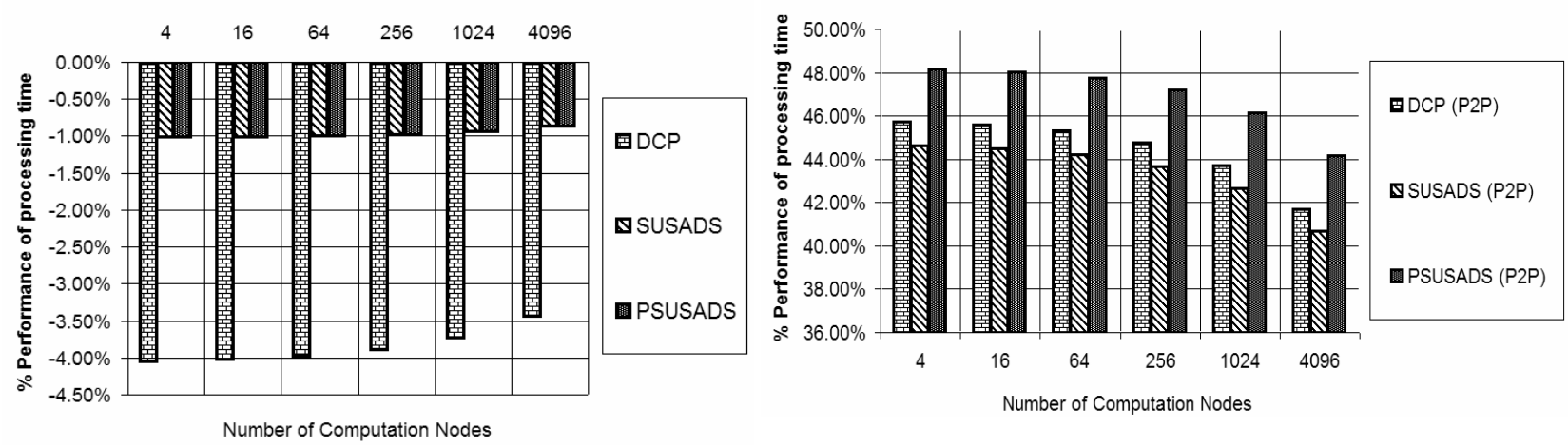
Number of Node	PB Size ($x10^6$)	PB Size / Number of Nodes	% Performance improvement of every models compare with NFTM model					
			P2P			Broadcast		
			DCP	SUSADS	PSUSADS	DCP	SUSADS	PSUSADS
4096	1.048576	256	-23.49%	0.00%	0.00%	-37.40%	0.00%	0.00%
4096	16.777216	4096	-2.98%	-0.74%	-0.74%	-9.05%	-2.26%	-2.26%
4096	268.435456	65536	-3.72%	-0.93%	-0.93%	-23.25%	-5.81%	-5.81%
4096	4294.967296	1048576	-3.97%	-0.99%	-0.99%	-38.28%	-9.57%	-9.57%
4096	68719.47674	16777216	-4.04%	-1.01%	-1.01%	-45.66%	-11.41%	-11.41%
4096	1099511.628	268435456	-4.06%	-1.02%	-1.02%	-47.97%	-11.99%	-11.99%

Table 9 The result of the performance comparison value of NFTM model and other models when node faulted and the problem size changed.

Number of Node	PB Size ($x10^6$)	PB Size / Number of Nodes	Execution Time (Seconds)							
			P2P				Broadcast			
			NFTM	DCP	SUSADS	PSUSADS	NFTM	DCP	SUSADS	PSUSADS
4096	1.048576	256	0.02	0.02	0.02	0.02	0.11	0.15	0.11	0.11
4096	16.777216	4096	3.05	1.89	1.92	1.81	8.24	7.89	7.48	7.21
4096	268.435456	65536	43.15	24.29	24.75	23.24	63.93	55.49	49.03	46.62
4096	4294.967296	1048576	667.77	365.13	372.45	348.87	750.86	605.09	501.68	471.19
4096	68719.47674	16777216	10593.65	5747.75	5864.92	5490.44	10926.02	8550.14	6895.64	6440.32
4096	1099511.628	268435456	169135.77	91586.86	93461.66	87480.90	170465.27	132277.08	105805.09	98650.57

Table 10 The percentage of the performance comparison value of NFTM model and other models when node faulted and the problem size changed.

Number of Node	PB Size ($x10^6$)	PB Size / Number of Nodes	% Performance improvement of every models compare with NFTM model					
			P2P			Broadcast		
			DCP	SUSADS	PSUSADS	DCP	SUSADS	PSUSADS
4096	1.048576	256	8.64%	0.00%	0.00%	-35.59%	0.00%	0.00%
4096	16.777216	4096	38.17%	37.23%	40.68%	4.33%	9.23%	12.61%
4096	268.435456	65536	43.70%	42.64%	46.15%	13.19%	23.30%	27.08%
4096	4294.967296	1048576	45.32%	44.22%	47.76%	19.41%	33.19%	37.25%
4096	68719.47674	16777216	45.74%	44.64%	48.17%	21.75%	36.89%	41.06%
4096	1099511.628	268435456	45.85%	44.74%	48.28%	22.40%	37.93%	42.13%

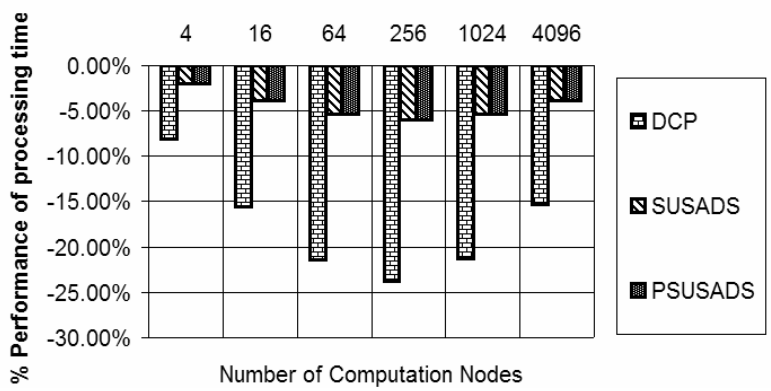


(a) (b)

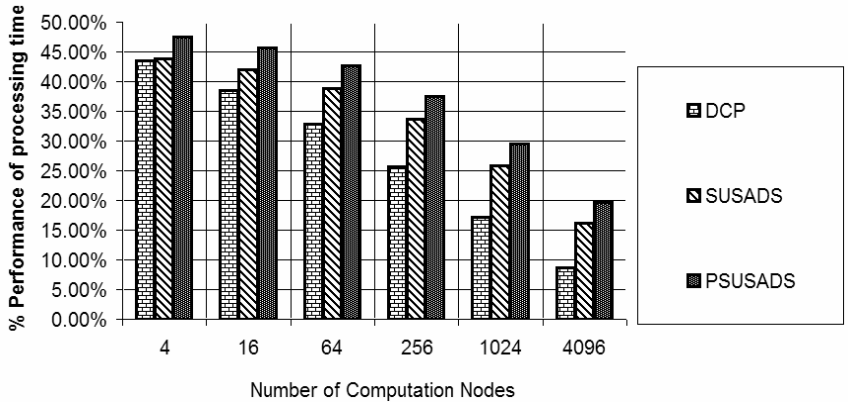
Figure 19 The percentage of the performance comparison on processing time of NFTM model and other models when using the 4,096 nodes and vary the problem size (point-to-point communication).

(a) The performance of point-to-point communication without node faulted.

(b) The performance of point-to-point communication when node faulted.



(c)



(d)

Figure 20 The percentage of the performance comparison on processing time of NFTM model and other models when using the 4,096 nodes and vary the problem size (Broadcasting communication).

(c) The performance of broadcasting communication without node faulted.

(d) The performance of broadcasting communication when node faulted.

Table 11 The result of the performance comparison value of NFTM model and other models when the number of computing nodes changed.

Number of Node	PB Size ($x10^6$)	PB Size / Number of Nodes	Execution Time (Seconds)							
			P2P				Broadcast			
			NFTM	DCP	SUSADS	PSUSADS	NFTM	DCP	SUSADS	PSUSADS
4	67.108864	16777216	5311.93	5526.68	5365.62	5365.62	5342.15	5771.64	5449.52	5449.52
16	67.108864	4194304	1335.54	1389.22	1348.96	1348.96	1380.86	1595.61	1434.55	1434.55
64	67.108864	1048576	337.66	351.08	341.02	341.02	375.43	455.96	395.56	395.56
256	67.108864	262144	86.30	89.66	87.14	87.14	112.74	139.59	119.45	119.45
1024	67.108864	65536	22.52	23.36	22.73	22.73	39.52	47.91	41.61	41.61
4096	67.108864	16384	6.10	6.31	6.15	6.15	16.49	19.01	17.12	17.12

Table 12 The percentage of the performance comparison value of NFTM model and other models when the number of computing nodes changed.

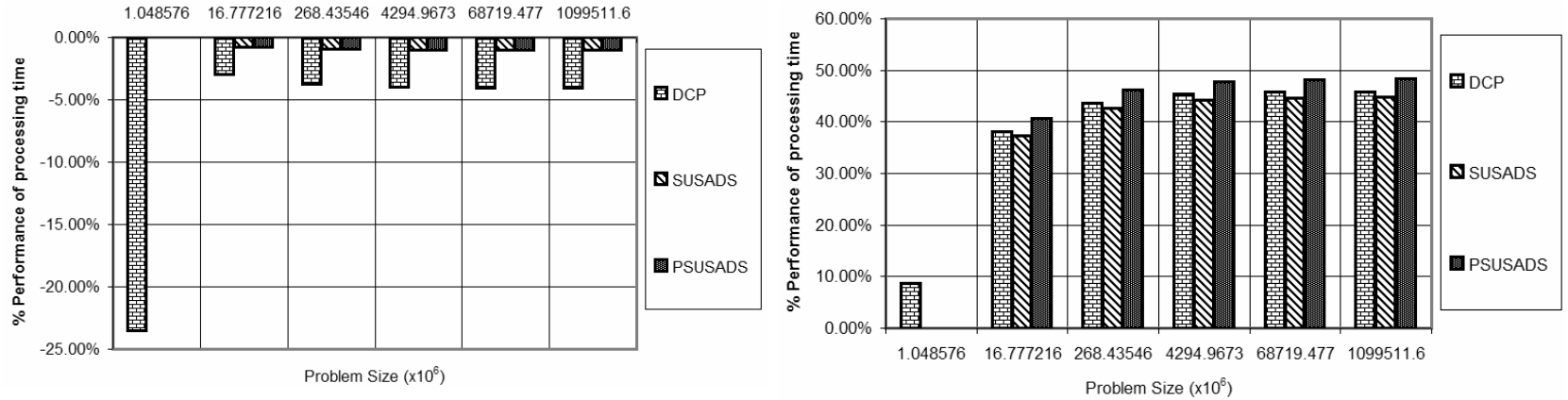
Number of Node	PB Size ($x10^6$)	PB Size / Number of Nodes	% Performance improvement of every models compare with NFTM model					
			P2P			Broadcast		
			DCP	SUSADS	PSUSADS	DCP	SUSADS	PSUSADS
4	67.108864	16777216	-4.04%	-1.01%	-1.01%	-8.04%	-2.01%	-2.01%
16	67.108864	4194304	-4.02%	-1.00%	-1.00%	-15.55%	-3.89%	-3.89%
64	67.108864	1048576	-3.97%	-0.99%	-0.99%	-21.45%	-5.36%	-5.36%
256	67.108864	262144	-3.89%	-0.97%	-0.97%	-23.81%	-5.95%	-5.95%
1024	67.108864	65536	-3.72%	-0.93%	-0.93%	-21.23%	-5.31%	-5.31%
4096	67.108864	16384	-3.44%	-0.86%	-0.86%	-15.26%	-3.82%	-3.82%

Table 13 The result of the performance comparison value of NFTM and other models when node faulted and the number of computing nodes changed.

Number of Node	PB Size ($x10^6$)	PB Size / Number of Nodes	Execution Time (Seconds)							
			P2P				Broadcast			
			NFTM	DCP	SUSADS	PSUSADS	NFTM	DCP	SUSADS	PSUSADS
4	67.108864	16777216	10593.65	5747.75	5864.92	5490.44	10623.86	6002.51	5958.62	5576.80
16	67.108864	4194304	2655.97	1444.79	1474.09	1380.24	2701.29	1659.43	1567.93	1467.89
64	67.108864	1048576	667.77	365.13	372.45	348.87	705.54	474.20	431.19	404.47
256	67.108864	262144	168.83	93.25	95.08	89.13	195.27	145.17	129.38	121.94
1024	67.108864	65536	43.15	24.29	24.75	23.24	60.15	49.82	44.62	42.36
4096	67.108864	16384	11.26	6.56	6.68	6.29	21.65	19.77	18.15	17.38

Table 14 The percentage of the performance comparison value of NFTM and other models when node faulted and the number of computing nodes changed.

Number of Node	PB Size ($x10^6$)	PB Size / Number of Nodes	% Performance improvement of every models compare with NFTM model					
			P2P			Broadcast		
			DCP	SUSADS	PSUSADS	DCP	SUSADS	PSUSADS
4	67.108864	16777216	45.74%	44.64%	48.17%	43.50%	43.91%	47.51%
16	67.108864	4194304	45.60%	44.50%	48.03%	38.57%	41.96%	45.66%
64	67.108864	1048576	45.32%	44.22%	47.76%	32.79%	38.88%	42.67%
256	67.108864	262144	44.77%	43.69%	47.21%	25.66%	33.74%	37.55%
1024	67.108864	65536	43.70%	42.64%	46.15%	17.17%	25.82%	29.57%
4096	67.108864	16384	41.70%	40.69%	44.18%	8.69%	16.15%	19.73%

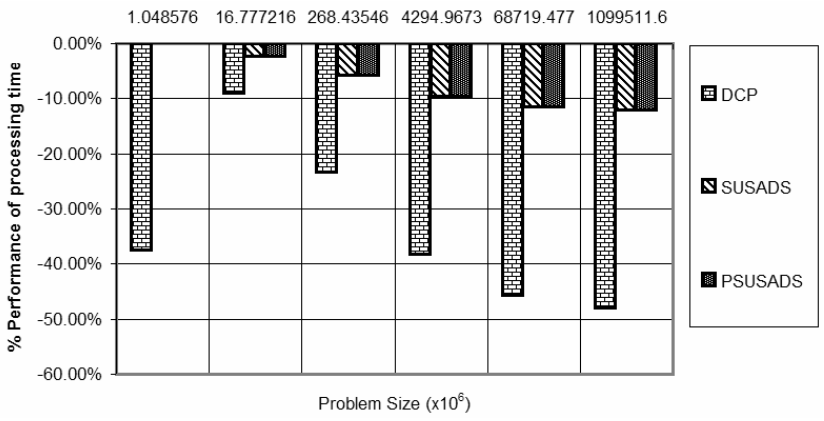


(a) (b)

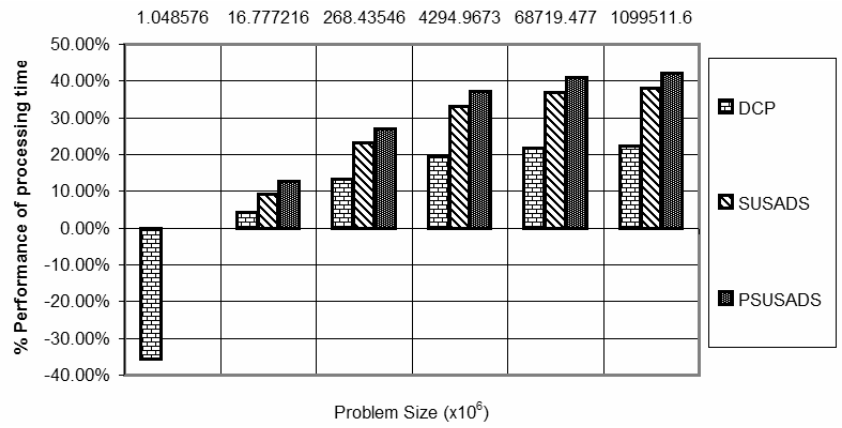
Figure 21 The percentage of the performance comparison on the processing time of NFTM model and other models when fix the problem size and vary the nodes (point-to-point communication).

(a) The performance of point-to-point communication without node faulted.

(b) The performance of point-to-point communication when node faulted.



(c)



(d)

Figure 22 The percentage of the performance comparison on the processing time of NFTM model and other models when fix the problem size and vary the nodes (broadcasting communication).

(c) The performance of broadcasting communication without node faulted.

(d) The performance of broadcasting communication when node faulted.

Regarding the results in the Table 7 to 14, the PSUSADS model gives a better performance than SUSADS, DCP and NFTM models. When compare times is used to backup the data and times is used to recovery the data, both PSUSADS and SUSADS be able to balance the usage time to backup the data better than DCP and also provides the good performance to recovery the data than NFTM. Regarding the high performance system such as BlueGene, NFTM will give a better performance than DCP but it may not give a big difference performance with SUSADS and PSUSADS – it could be the same performance. Because both SUSADS and PSUSADS also use the computing power to recovery the losing data and back up only the ghost cells of sub domain and also the network performance of BlueGene system also faster than the environment is used to compare in this work. Both SUSADS and PSUSADS also do not use the storage to backup the data as much as DCP that why a Non-dedicate systems, which is a heterogeneous system, are possible to use SUSADS and PSUSADS than DCP.

Memory Usage

In term of memory usage, NFTM will use less of memory than SUSADS, PSUSADS and DCP model because NFTM uses only the ghost cells as the backup data to recovery the fault node. In the opposite of NFTM is DCP. That uses a lot memory to keep the hold backup data from neighbor node. We propose both SUSADS and PSUSADS which are a new model to balance the memory usage of both NFTM and DCP models.

$$M_{NFTM} = \frac{4n}{\sqrt{P}} \tag{22}$$

$$\begin{aligned} M_{PSUSADS} = M_{SUSADS} &= \sqrt{k} M_{NFTM} \\ &= \frac{4n\sqrt{k}}{\sqrt{P}} \end{aligned} \tag{23}$$

$$M_{DCP} = \frac{n^2}{\sqrt{P}} \tag{24}$$

Following the Eq. 22 and 23, both SUSADS and PSUSADS use the memory than NFTM in $\sqrt{k}$ times which is depended on the number of computation node (P). That means if the number of computation node is higher enough to break the problem to smaller size, the value of k will be reduce to equals 1. The performance between DCP and PSUSADS will be showing in the Eq. 25.

$$\begin{aligned} Performance &= \frac{M_{DCP}}{M_{PSUSADS}} \\ &= \frac{n^2}{\sqrt{P}} \times \frac{\sqrt{P}}{4n\sqrt{k}} \\ &= \frac{n}{4\sqrt{k}} \end{aligned} \tag{25}$$

Following the Table 9, the best value of sub domain size is 16x16 (256) so the best value of k is $\frac{n}{16}$ so the Eq. 25 will be changed as follows:

$$\begin{aligned} Performance &= \frac{n}{4\sqrt{k}} \\ &= \frac{n}{4} \times \sqrt{\frac{16}{n}} \\ &= \sqrt{n} \end{aligned} \tag{26}$$

That means both SUSADS and PSUSADS can reduce the memory usage of DCP in $\sqrt{n}$ times.

CONCLUSION

We propose the fault tolerance model base on the iterative model. SUSADS uses the idea of both checkpoint algorithm and natural fault tolerance to implement. SUSADS can balance the network performance and computation power better than DCP and NTFM that improve the performance of normal process and recovery process. SUSADS also use the resource to back up the data in the neighbor nodes less than DCP that make SUSADS be able to run on various environment that DCP.

We enhance SUSADS model to make this model give a higher performance. The benefit of domain splitting model gives a new recovery method on the SUSADS model by using the reprocessing on the parallelism of the neighbor nodes. That increases the performance of reprocessing process of SUSADS to 2 or 4 times which depend on the number of neighbor node. We name the new model as PSUSADS.

Regarding our future goals for this work, we try to map this model to the multiple Grid environment systems. That is very challenge for us to manage it on the heterogeneous environment such as the difference network performance between Grid system and inside the Grid system, the difference in the number of computation node in each Grid system and etc.

LITERATURE CITED

Barrett, R., M. Berry, T. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine and H. Vorst. 1995. **Template for the Solution of Linear Systems: Building blocks for Iterative Methods**. Available Source: http://www.netlib.org/linalg/html_templates/report.html

Batchu, R., J. Neelamegam, Z. Cui, M. Beddhua, A. Skjellum, Y. Dandass and M. Apte. MPI/FT: Architecture and Taxonomies for Fault-Tolerance, Message-Passing Middleware for Perforance-Portable Parallel Computing. 2001. pp. 16. **In Proceedings of the 1st IEEE International Symposium of Cluster Computing and the Grid held in Melbourne**, IEEE Computer Society, Washington, DC, USA

Bosilca, G., A. Boteiller, F. Cappello, S. Djilali, G. Fedak, C. Germain, T. Herault, P. Lemarinier, O. Lodygensky, F. Magniette, V. Neri, and A. Selikhov. 2002. MPICH-V: Toward a Callable Fault Tolerant MPI for Volatile Nodes. pp. 1-18. **Proceedings of the 2002 ACM/IEEE conference on Supercomputing.** IEEE Computer Society Press, Los Alamitos, CA, USA.

Charng-da Lu and Daniel A. 2004. Assessing Fault Sensitivity in MPI Applications. pp. 37. **Proceedings SC2004 High Performance Computing, Networking, and Storage Conference**. IEEE Computer Society Washington, DC, USA

Fagg, G., E. Gabriel, Z. Chen, T. Angskun, G. Bosilca, J. Pjesivac-Grbovic and J. Dongarra. 2005. vol. 19. pp. 465-477. Process Fault-Tolerance: Semantics, Design and Applications for High Performance Computing. **International Journal of High Performance Computing Applications.** SAGE Publications.

Foster, I. and C. Kesselman. 1998. Globus: A metacomputing infrastructure toolkit. pp. 115-129. **International Journal of Supercomputer Application**.

Geist, G. A. and C. Engelmann. 2002. Development of Naturally Fault Tolerant Algorithms for Computing on 100,000 Processors. **Journal of Parallel and Distributed Computing, submitted.**

Geist, R., R. Reynolds and J. Westall. 1988. Selection of a Checkpoint Interval in a Critical-Task Environment. vol. 37. pp. 395-400. **IEEE Transactions on Reliability.**

Grove, D.A. 2003. **Performance modeling of message-passing parallel programs.** Ph.D. Thesis, University of Adelaide, Department of Computer Science.

IBM Corporation. 2001. **AUTONOMIC COMPUTING: IBM's Perspective on the State of Information Technology**. 22.

James, S., L. Kai and A. Michael. 1998. Diskless Checkpointing. pp. 972-986. **IEEE Transactions on Parallel and Distributed Systems**. IEEE Press, Piscataway, NJ, USA.

Koehler, J., G. Chirs, G. Dieter and R. Hauser. 2003. On Autonomic Computing Architectures. **IBM Research. Technical Report RZ 3487**. 10.

Lertpaibulpanya, S. 2006. Fast Self-healing Parallel Iterative Algorithm with Domain Splitting. pp. 85-90. **The Joint Conference on Computer Science and Software Engineering.** Thailand.

Qingping, G., Y. Paker and D. Parkinson. 2000. **Network Computing Performance Evaluation of PVM on PC-LAN Distributed Computing**.

Qingping, G., Y. Paker, Z. Shesheng, D. Parkinson and W. Jialin. 2000. Parallel Multi Grid Algorithm with Virtual Boundary Forecast Domain Decomposition Method for Solving Non-linear Heat Transfer Equation. pp. 568-571. **HPCN Europe 2000**. Amsterdam, The Netherlands.

Stellner, G. 1996. Cocheck: Checkpointing and process migration for MPI, pp. 526-531. **In Proceeding of the 10th International Parallel Processing Symposium (IPPS'96)**. IEEE Computer Society, Washington, DC, USA.

Wei, J., S. Zhang, Q. Guo, Y. Paker and D. Parkinson. 2001. A Multi-grid Parallel Model of Two-Material Heat Transfer. **Third AFOSR International Conference on Direct Numerical Simulation and Large Eddy Simulation.** University of Texas at Arlington.

Xin, H. 2005. A Numerical Study on Heat Transfer, **ICS 691: Parallel & Distributed Computing Seminar.** Department of Mechanics and Engineering Science. Peking University.

Zhengquan, X. 1996. Domain Decomposition Method of Multi Grid Distributed Computing. pp. 1-5. **Journal of Numerical Computation**.

CURRICULUM VITAE

NAME	:	Mr. Suksan Lertpaibulpanya
BIRTH DATE	:	December 16, 1976
BIRTH PLACE	:	Bangkok, Thailand
EDUCATION	:	YEAR INSTITUTION DEGREE/DIPLOMA
		1999 King Mongkut Institute of Technology North Bangkok B.S. Secd Hons. (Computer Science)
POSITION/TITLE	:	Group Leader
WORK PLACE	:	Reuters Software (Thailand) Co. Ltd.
SCHOLARSHIP/AWARDS	:	