Parallelization of Radiation Transport on Unstructured Triangular Grids with Spatial Decomposition and OpenMP

Eric E. Aubanel
Faculty of Computer Science
University of New Brunswick
Fredericton, New Brunswick
E3B 5A3 Canada
aubanel@unb.ca

Faysal El Khettabi
Department of Mechanical Engineering
University of New Brunswick
Fredericton, New Brunswick
E3B 5A3 Canada
faysalek@unb.ca

Abstract

Parallel computing can be of critical importance to the deterministic solution of radiation transport equations, especially with respect to solving the transport process within highly variable media with complex geometries. We propose an approach that extends the Alternating Direction of Transport Sweeps (ADTS) method of M. Yavus and E.D. Larson to unstructured triangular grids, using the Recursive Spectral Bisection (RSB) algorithm to partition our domain and OpenMP to implement the parallelism on shared memory computers. Our results, using a neutron transport test case, show that the ADTS method with the RSB algorithm leads to a significant increase in the parallel convergence rate, resulting in improved parallel efficiency, and thus improved turnaround time. We emphasize that excellent parallel efficiency is possible using domain decomposition and an SPMD-OpenMP implementation.

1 Introduction

The S_n transport codes have been widely used in neutronics calculations, including shielding calculations and reactor core calculations. They are also being considered for treatment planning in Boron Neutron Capture Therapy (BNCT) [1].

In S_n transport codes, the Boltzmann equation is solved as a difference equation using the discrete ordinates methods to describe the position and direction of particles. These codes can be very time and resource consuming, particularly for realistic solutions in domains with complex geometries such as are found in BNCT dose calculations, which are based on neutron flux calculations.

Several studies on parallelization have been performed for S_n transport calculations by applying angular and spatial domain decomposition [2]-[8]. Since the transport sweeps over discrete ordinate angles are independent of each other, parallelization over these angles is trivial. This approach may not be ideal, since the number of angles is much less than the number of finite elements, and may be less than the number of processors available.

Until recently, parallelization based on spatial domain decomposition has been restricted to rectangular meshes. In the past few years there have been several applications to unstructured meshes: Nowak and Nemanick [7] used a Hybrid MPI/OpenMP implementation of a method using Jacobi iteration, Plimpton *et al.* [8] developed an asynchronous message passing algorithm, and one of the present authors (F.E. Khettabi) extended the ideas of Yavuz and Larson [2] to unstructured triangular meshes together with an implementation in Cray's High Performance Fortran (HPF/CRAFT) [9]. In the present work, we have implemented the latter work using OpenMP, resulting in more portable and highly scalable code. We outline the scalar iteration method in Section 2. Our application of the parallel source iteration method to unstructured grids using OpenMP is given in Section 3. Results illustrating convergence rates and speedups are presented and discussed in Section 4, and we conclude in Section 5.

2 Scalar Source Iteration

The one-group $X - Y$ geometry S_n equations can be written:

$$\mu_m \frac{\partial \psi_m}{\partial x}(r) + \nu_m \frac{\partial \psi_m}{\partial y}(r) + \sigma_t(r)\psi_m(r) = \frac{1}{4\pi}[\sigma_s(r)\phi_0(r) + S(r)], \quad r \in \Omega, \tag{1}$$

Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'02)
1530-2075/02 $17.00 © 2002 IEEE

where $r = (x, y)$ is the position, Ω is the spatial domain, $\psi_m(r)$ is the angular neutron flux at position r in the discrete ordinates direction (μ_m, ν_m), and $\phi_0(r)$ is the scalar flux

$$\phi_0(r) = \sum_{m=1}^{N} \omega_m \psi_m(r). \quad (2)$$

The boundary conditions for this equation are

$$\psi_m = 0, \quad r \in \partial\Omega, \quad (\mu_m, \nu_m) \cdot n(r) < 0, \quad (3)$$

where $n(r)$ is the unit outer normal on the boundary $\partial\Omega$ at point r.

The discretized-in-space versions of this equation can be solved by the source iteration scheme, in which one iterates on the scattering source $\sigma_s\phi_0$. Thus, introducing an iteration superscript, the last equation for the source iteration method become

$$\mu_m \frac{\partial \psi_m^{n+1}}{\partial x}(r) + \nu_m \frac{\partial \psi_m^{n+1}}{\partial y}(r) + \sigma_t(r)\psi_m^{n+1}(r) = \frac{1}{4\pi}[\sigma_s(r)\phi_0^n(r) + S(r)], \quad r \in \Omega, \quad (4)$$

$$\psi_m^{n+1} = 0, \quad r \in \partial\Omega, \quad (\mu_m, \nu_m) \cdot n(r) < 0, \quad (5)$$

$$\phi_0^{n+1}(r) = \sum_{m=1}^{N} \omega_m \psi_m^{n+1}(r). \quad (6)$$

The solution algorithm of equation involves transport sweeps across the domain Ω in each direction of neutron travel. As a result, each iteration on the scattering source consists of transport sweeps along each of the directions of the angular quadrature. In each spatial cell, the exiting angular fluxes are used as incident angular fluxes for the "downwind" cells, which in turn are used for obtaining the exiting flux in those cells. Thus, the calculation for any given cell depends on the calculations in all "upwind" cells. We call this the scalar source iteration method. Because each cell calculation depends on all the "upwind" cell calculations, this scalar source iteration method cannot be geometrically decomposed for use on parallel computers. However, a geometrically decomposed modification of this method is described in the following section.

3 Parallel Source Iteration

A triangulation $\mathcal{T}_h$ is established over the domain Ω, i.e, the set Ω is subdivided into a finite number of triangles T and h is the mesh size. An efficient parallel source iteration method centers about four related considerations: 1) ordering of transport sweeps; 2) domain decomposition; 3) asynchronization iteration; 4) implementation of the algorithm.

3.1 Transport Sweeps

For each direction (μ_m, ν_m), we partition the mesh into layers $S_0^m, S_2^m, ..., S_{end}^m$

$$S_0^m \equiv \{T \in \mathcal{T}_h : \partial_{in}^m T \subset \partial_{in}^m \Omega\}, \quad (7)$$

$$S_{i+1}^m \equiv \{T \in \mathcal{T}_h : \partial_{in}^m T \subset \partial_{in}^m (\Omega - \bigcup_{j \le i} S_j^m)\}, \quad i = 0, 1, ..., end, \quad (8)$$

where $\partial_{in}^m \Omega$ is the inflow boundary of Ω and $\partial_{in}^m T$ is the inflow boundary of the triangle T [10].

With this partition of $\mathcal{T}_h$, the approximate solution may be obtained in an explicit manner, first in S_0^m, then in S_1^m, etc. The updated exiting fluxes in S_i^m are transferred to the neighboring layer S_{i+1}^m for use as updated incident fluxes. This approach represents an inherently sequential procedure which does not lend itself to parallel processing. In order to overcome this obstacle, we use spatial domain decomposition and asynchronous iteration that we describe in the following sections.

3.2 Domain Decomposition

Many large scale computational transport equations are based on unstructured computational domains. One good method for decomposing such domains is Recursive Spectral Bisection (RSB) [11, 12], which is based on the computation of an eigenvector of the Laplacian matrix associated with the graph(mesh). This method produces good load balancing of the subdomains, and minimizes communication between subdomains by minimizing the number of edges cut between them.

Each processor is assigned a subdomain. The general problem described by equation 1 for P processors can be written as:

$$\mu_m \frac{\partial \psi_{m,p}}{\partial x}(r) + \nu_m \frac{\partial \psi_{m,p}}{\partial y}(r) + \sigma_t(r)\psi_{m,p}(r) = \frac{1}{4\pi}[\sigma_s(r)\phi_{0,p}(r) + S(r)], \quad r \in \Omega_p, \quad (9)$$

$$p = 0, 1, ..., P-1,$$

$$\psi_{m,p}(r) = \begin{cases} 0 & \text{if } r \in \Gamma_p, (\mu_m, \nu_m) \cdot n(r) < 0, \\ \psi_{m,p'}(r) & \text{if } r \in \Gamma_{p,p'}, \end{cases} \quad (10)$$

$$\phi_{0,p}(r) = \sum_{m=1}^{N} \omega_m \psi_{m,p}(r), \quad (11)$$

where Γ_p is the part of the boundary of the subdomain Ω_p that coincides with the outer boundary of Ω and $\Gamma_{p,p'}$ is the interface between subdomains Ω_p and $\Omega_{p'}$.

In this parallel source iteration method, one iterates on the scattering source as well as the interface fluxes. At the beginning of a transport sweep, processor p has estimates for both the incident interface fluxes and the scattering source for subdomain Ω_p. At the end of a transport sweep, new estimate have been calculated for both the scattering source within Ω_p and the exiting fluxes from Ω_p. The updated exiting fluxes are transferred to neighboring processors p' for use as updated incident fluxes.

3.3 Asynchronization Iteration

In the parallel method described above, for a given iteration in a particular subdomain, incident fluxes from adjacent subdomains are calculated in the previous iteration. This results in an increase in the number of iterations required until convergence is achieved. This problem can be mitigated by using updated incident fluxes as they become available, as proposed by Yavuz and Larson [2] and described below.

Let $\oplus$ show the need for incident boundary fluxes and let $\otimes$ show the availability of new outgoing fluxes from a neighboring subdomain. Thus, if $\otimes \equiv \oplus$, then we use the new available information; otherwise, we use the older estimates. Starting with the initial guess for the scalar and interface boundary fluxes in processor p, we solve

$$\mu_m \frac{\partial \psi^{n+1}_{m,p}}{\partial x}(r) + \nu_m \frac{\partial \psi^{n+1}_{m,p}}{\partial y}(r) + \sigma_t(r)\psi^{n+1}_{m,p}(r) = \frac{1}{4\pi}[\sigma_s(r)\phi^n_{0,p}(r) + S(r)], \quad r \in \Omega, \tag{12}$$

$$p = 0, 2, ..., P-1,$$

$$\psi^{n+1}_{m,p}(r) = \begin{cases} 0 & \text{if } r \in \Gamma_p, (\mu_m, \nu_m) \cdot n < 0, \\ \psi^{n+1}_{m,p'}(r) & \text{if } r \in \Gamma_{p,p'}, \otimes \equiv \oplus, \\ \psi^{n}_{m,p'}(r) & \text{if } r \in \Gamma_{p,p'}, \otimes \neq \oplus. \end{cases} \tag{13}$$

Then, we update the scalar flux by

$$\phi^{n+1}_{0,p}(r) = \sum_{m=1}^{N} \omega_m \psi^{n+1}_{m,p}(r). \tag{14}$$

Yavuz and Larson's Alternating Direction of Transport Sweeps (ADTS) source iteration scheme uses this equation and an alternating order of transport sweeps. In its original form, this meant that while one swept in a specified direction in processor p, one swept in a direction in a processor with a neighbouring subdomain p' so that both processors were able to use the new outgoing interface boundary fluxes from their neighbors as soon as possible. In the ideal case of a square domain divided into four subdomains this works perfectly, and incident interface fluxes are always calculated before they are needed. In general, updated fluxes are not always available. We optimize the ordering of transport sweeps as follows. For a given subdomain D_p, we find the first layer in the subdomain for each direction, that is the smallest $n_{i,m}$ such that $S^m_{n_{i,m}} \in D_p$ (see Eqs. 7,8); we then order the directions $m_1, m_2, ...$, such that $n_{i,m_1} \leq n_{i,m_2} ....$ This means, for instance, that sweeping in subdomains on the outer boundary of the domain will be done first for directions that come from outside.

3.4 Implementation for Unstructured Grids

Implementation of the ADTS method for irregular grids requires nontrivial domain decomposition (see section 3.2) and a communication pattern that is much more complicated than for regular grids. We have used the observation that problems with asynchronous communication are best implemented on shared memory computers [13], because each processor can access a global address space without the participation of other processors, and have implemented the ADTS scheme using OpenMP, an industry-wide standard for threads-based shared memory parallelization.

The computational kernel of the code, which contains the source iteration, was placed in one OpenMP parallel region and parallelized using the Single Program Multiple Data (SPMD) approach. That is, we did not use the typical approach taken with OpenMP of parallelizing individual loops. An optimized order of sweeps was used to maximize availability of updated incident fluxes from adjacent subdomains. When such incident fluxes are required they are obtained (updated or not) transparently from global memory. This may require access to distributed shared memory, which can have performance implications, as will be discussed below.

4 Results

We initially implemented our parallel source iteration algorithm in Fortran 90 on a single IBM Winterhawk-II node, containing four 375 MHz Power3 processors and 1 GB memory. The code was then moved to an SGI Origin 2400 with 64 MIPS R12000 400 MHz processors and 16 GB memory. The Origin 2400 is a distributed-shared-memory machine, with a "first touch" data placement policy. This means that pages are allocated to memory close to the processor that runs the code. Therefore, shared arrays were initialized in parallel to maximize local memory references. All the processors initiate their work simultaneously after a particular processor initializes the input data for others. At the end of every iteration a global L_1 relative error norm is calculated for the scalar fluxes:

$$E = \frac{\|\phi^{n+1}_0 - \phi^n_0\|_1}{\|\phi^n_0\|_1}. \tag{15}$$

This is the only serial part of our algorithm, in addition to the initialization described above. If the global convergence criterion is satisfied, the program prints out the results and terminates. Otherwise, all processors continue to perform transport sweeps until the entire problem converges.

4.1 Model Problem

This problem consist of an $X-Y$ geometry system with a vacuum boundary. The disk Ω was triangulated by means of a quasi-uniform mesh of 169,366 triangles and decomposed into between 2 and 64 subdomains, depending on the number processors to be used. Constant values of the cross sections were used: $S(x,y) = 1.0$, $\sigma_t(x,y) = 1.0$, $\sigma_s(x,y) = 0.5$ for all x, y, unless otherwise specified. An S_8 angular quadrature approximation was used, which spans 40 angles in the plane. The convergence criteria was that the relative change in the average flux between source iterations (Eq. 15) not exceed 1.0×10^{-5} in magnitude.

4.2 Convergence Rate

First, we examine how the asynchronous parallel source iteration method affects the convergence rate. Figure 1 shows the number of iterations required to converge as a function of the number of processors, for $\sigma_s = 0.5$ (equal scattering and absorption) and $\sigma_s = 0.8$ (more scattering than absorption). For $\sigma_s = 0.5$ the number of iterations rises rapidly at first from 16 for the serial calculation, but then levels off to 33 for the 64-processor parallel calculation. As expected, the number of iterations is greater for the optically thin case, and the number of iterations increases at a slightly greater rate.

The parallel source iteration method involves both the use of updated incident fluxes from adjacent subdomains (Eq. 13) and the ordering of transport sweeps to maximize the availability of the fluxes. We examined the influence of these two factors on the convergence rate. We modified the algorithm so that only incident fluxes from adjacent suddomains from the previous iteration were available, leaving only synchronous communication once every iteration. Figure 2 shows that the synchronous algorithm takes almost twice as many iterations as does the asynchronous algorithm. The importance of the ordering of transport sweeps was determined by performing calculations where transport sweeps were done in the same order in each subdomain, in contrast to the original algorithm where the ordering is optimized for each subdomain. The result, indicated in Figure 2, shows that there is a small increase of only a few iterations when the same ordering of transport sweeps is used.

Clearly, the asynchronous algorithm converges much faster than the synchronous algorithm, which results from the use of updated incident fluxes. The fraction of triangles that lie on subdomain interfaces, require incident fluxes from adjacent subdomains, and get updated values ($\otimes \equiv \oplus$ in the notation of Eq. 13) is shown in Table 1. Also shown is the same fraction for the case where an identical ordering of transport sweeps is used in each subdomain. As many as 66% of triangles satisfy $\otimes \equiv \oplus$ in the asynchronous case, whereas there are none in the synchronous case. This results in the significant increase in the convergence rate of the asynchronous method seen in Fig. 2. Using an order of transport sweeps optmized for each subdomain yields more triangles that satisfy $\otimes \equiv \oplus$ and therefore increases the convergence rate, but the improvement is modest. Curiously, for 64 processors, both single and optimized ordering of transport sweeps yield the same fraction of updated incident fluxes, while the latter method still converges faster. Evidently this measure is insufficient, as it doesn't take into account the number of dowwind triangles that depend on the triangle that is receiving fluxes from another subdomain.

# Procs	Optimized dir.	Identical dir.
2	0.64	0.34
4	0.66	0.49
8	0.64	0.48
16	0.61	0.47
32	0.57	0.52
64	0.59	0.58

Table 1. Fraction of subdomain interface triangles satisfying $\otimes \equiv \oplus$.

4.3 Parallel Speedup

The speedup per iteration of the asynchronous parallel program is shown in Figure 3, and is defined as $S_p = t_s/t_p$, where t_s and t_p are the execution times per iteration of the sequential and parallel (using p processors) programs respectively, averaged over all iterations. The speedup is superlinear, a result of the program data fitting into the 8 MB L2 cache of the MIPS R12000 processor when it is divided into more than eight processors. Figure 3 also shows the speedup for a finer mesh of 677,464 triangles; again, the speedup becomes superlinear when the data used by each processor fits into its cache, which occurs here after 48 processors.

The overall speedup, given by $(N_{iter,s}t_s)/(N_{iter,p}t_p)$, where $N_{iter,s}$ and $N_{iter,p}$ are the number of iterations required to converge for the serial and parallel calculations respectively, is shown in Figure 4 for the two mesh sizes. The decrease in the convergence rate of the parallel programs offsets to some extent the speedups obtained per iteration. However, the speedup still increases with the number of processors, even before the cache effects takes over.

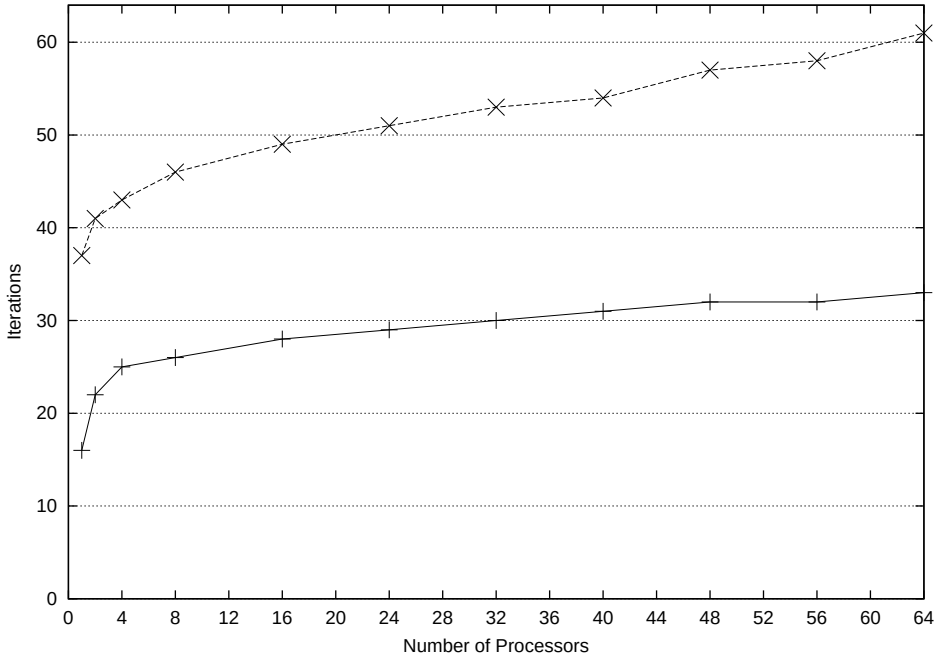

Figure 1. Convergence rates for $\sigma_s = 0.5(+)$ and $\sigma_s = 0.8(\times)$.

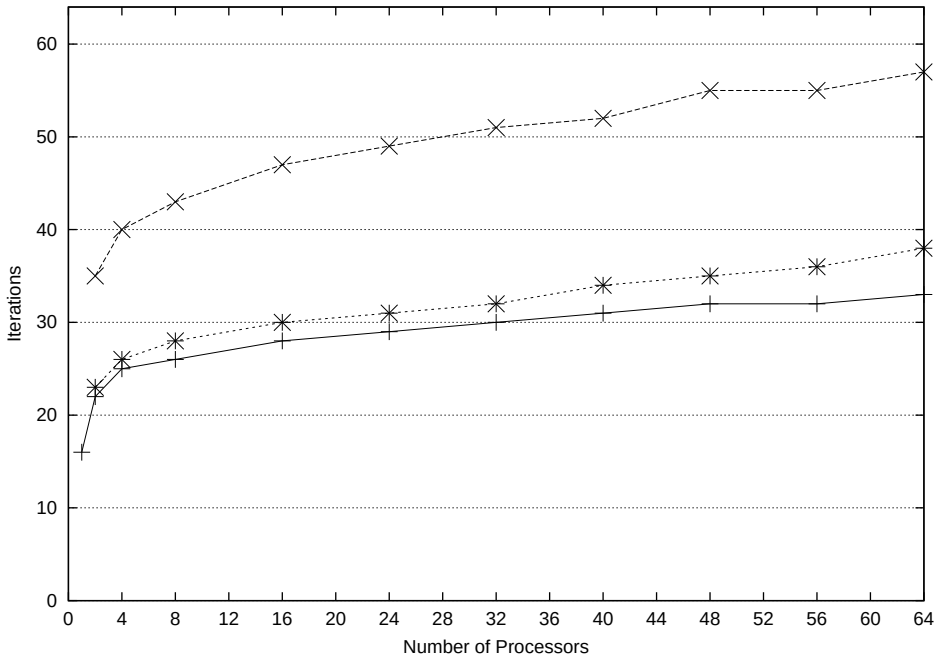

Figure 2. Convergence rates for asynchronous method with optimized directions ($+$), identical order of directions in each subdomain($*$) and for the synchronous method ($\times$).

Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'02)
1530-2075/02 $17.00 © 2002 **IEEE**

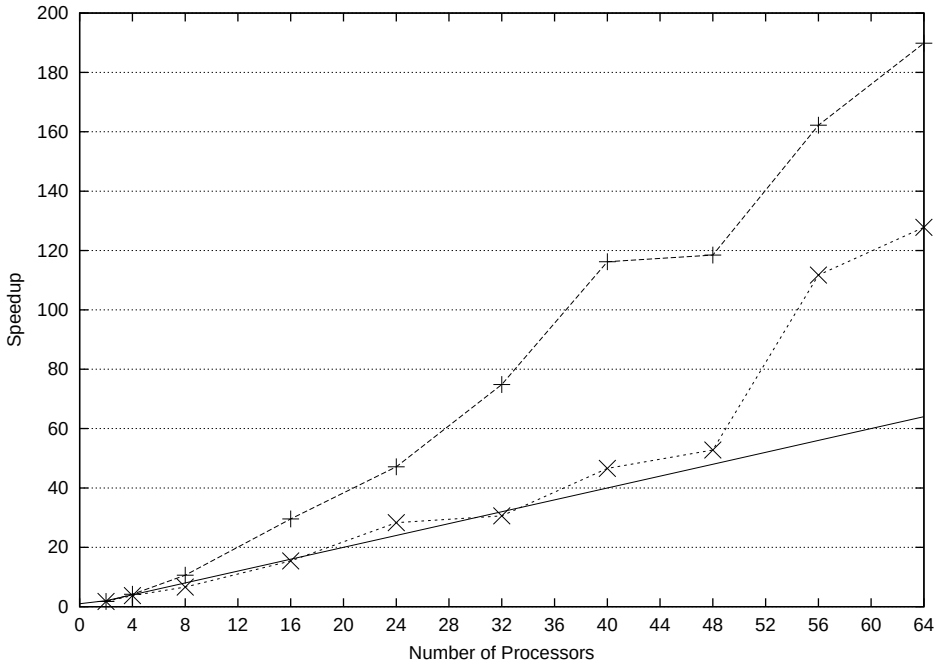

Figure 3. Speedup per iteration for 169,366 (+) and 677,464 (×) triangles; linear speedup (–) is also shown.

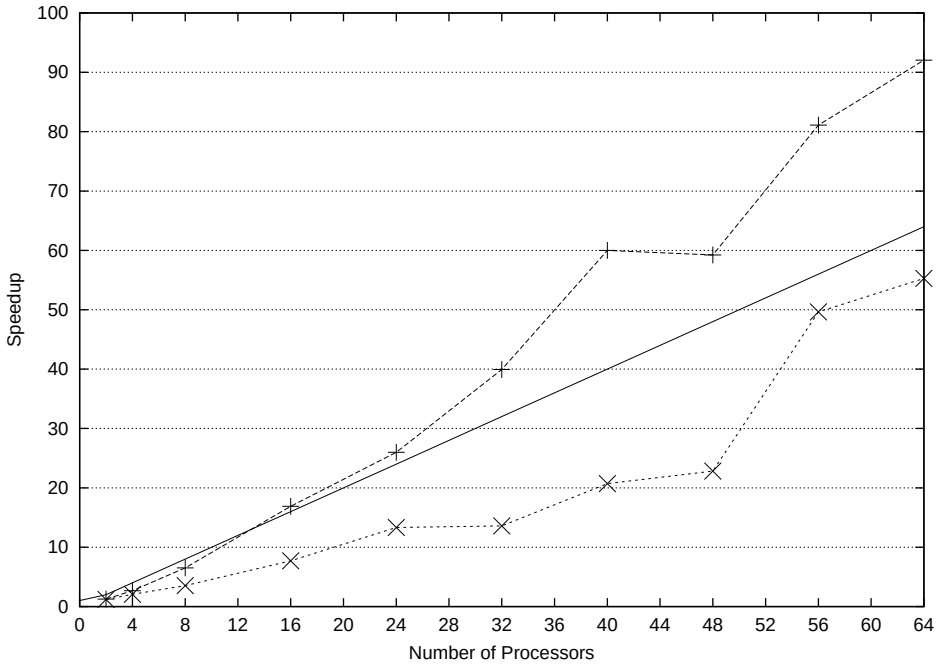

Figure 4. Overall speedup for 169,366 (+) and 677,464 (×) triangles; linear speedup (–) is also shown.

4.4 Scalability of OpenMP

Parallelization with OpenMP is not normally associated with domain decomposition (see *e.g.* [14]). The usual approach is to parallelize individual time-consuming loops, which can yield poor performance due to Amdahl's Law and inefficient placement of data on a distributed shared memory platform. One soultion to the latter problem is to dynamically repartition the data [15]. We believe that a better approach is to use SPMD parallelization together with OpenMP, which in our case yields excellent speedup. Maximixation of local memory references is ensured by domain decomposition and parallel initialization of data, together with SGI's "first touch" data placement policy. This does require more code modification than the loop-level approach, but it is still far easier than a message-passing implementation.

5 Conclusions

In this paper, we have presented a parallel source iteraction method for the tranport equation, using spatial domain decomposition and an optimized ordering of transport sweeps. We have observed no significant degradation in the convergence rates as has been reported [4, 5, 6]. Parallelization with OpenMP on a 64-processor SGI Origin 2400 results in excellent speedups, even superlinear speedups due to cache effects. We suggest that excellent parallel efficiency is possible in general with OpenMP, provided the SPMD programming model is used.

Acknowledgements

Calculations were performed on an SGI Origin 2400 at the University of Alberta, Canada, part of the Multimedia Advanced Computational Infrastructure (MACI).

References

[1] International Atomic Energy Agency, "Current Status of Neutron Capture Therapy", IAEA-TECDOC-1223 (May 2001).

[2] M. Yavuz and E. Larson,"Iterative Methods for Solving $X-Y$ Geometry S_n Problems on Parallel Architecture Computers", *Nucl. Sci. Eng.*, **111**, 46, (1992).

[3] Y.Y Azmy,"Performance and Performance Modeling of a Parallel Algorithm for Solving the Neutron Transport Equation", *Journal of Supercomputing*, **6**, 211 (1992).

[4] R. Mattis and A. Haghighat,"Domain Decomposition of a Two-Dimensional S_n Method" , *Nucl. Sci. Eng.*, **111**, 180, (1992).

[5] S.P. Burns and M.A. Christon, "Spatial Domain-Based Parallelism in Large-Scale, Participating Media, Radiative Transport Applications", *Numerical Heat Transfer*, Part B, **31**, 401 (1997).

[6] J. Goncalves and P.J. Coelho, "Parallelization of the Discrete Ordinates Method", *Numerical Heat Transfer*, Part B, **32**, 151 (1997).

[7] P. Nowak and M.K. Nemanic, "Radiation Transport Calculations on Unstructured Grids using a Spatially Decomposed and Threaded Algorithm", *Proc. ANS Conf. on Math. and Computation, Reactor Physics and Environmental Analysis in Nuclear Applications*, 379 (1999).

[8] S. Plimpton, B. Hendrickson, S. Burns, W. McLendon III, "Parallel Algorithms for Radiation Transport on Unstructured Grids", *Proc. SuperComputing 2000*, IEEE (2000).

[9] F. E. Khettabi, "Parallel Source Iteration For Solving $X-Y$ Geometry S_n Problems with Unstructured Triangular Mesh", *Proc. Int. Topl. Mtg. Radiation Protection and Shielding*, Nashville, Tennessee, April 19-13, 1998, **1**, pp. 488-495, American Nuclear Society, La Grange Park (1998).

[10] F.E. Khettabi and C. Lecot, "Characteristic Methods for Discretizing the Two-Dimensional Transport Equation on an Unstructured Grid of Triangular Cells", *Proc. Joint International Conference on Mathematical Methods and Supercomputing for Nuclear Applications*, **2**, pp. 975-984, American Nuclear Society, La Grange Park (1997).

[11] A. Pothen, H.D. Simon and K.P. Liou "Partitioning Sparse Matrices with Eigenvectors Of Graphs", *SIAM Journal of Matrix Analysis and Applications* , **11**, 430 (1990).

[12] H.D. Simon,"Partitioning of Unstructured Problems for Paralle Processing", *Computing Systems in Engineering*, **2**, 135 (1991).

[13] S.M. Pancake, "Is Parallelism for You?", *Computational Science and Engineering* Vol. 3, No. 2, 18 (Summer, 1996).

[14] J. Hoeflinger, P. Alavilli, T. Jackson, and B. Kuhn, "Producing Scalable Performance with OpenMP: Experiments with Two CFD Applications", *Parallel Computing*, **27** 391 (2001).

[15] D.S. Nikolopoulos and E. Ayguade, “Scaling Irregular Parallel Codes with Minimal Programming Effort”, *Proc. SuperComputing 2001* (November 2001).

A Limited-Global Fault Information Model for Dynamic Routing in 2-D Meshes *

Zhen Jiang and Jie Wu
Department of Computer Science and Engineering
Florida Atlantic University
Boca Raton, FL 33431
{zjiang, jie}@cse.fau.edu

ABSTRACT
In this paper, a fault-tolerant routing in 2-D meshes with dynamic faults is provided. It is based on an early work on minimal routing in 2-D meshes with static faults. Unlike many traditional models that assume all the nodes know global fault information, our approach is based on the concept of limited global fault information. First, a fault model called faulty block is used in which all faulty nodes in the system are contained in a set of disjoint faulty blocks. Then, the information of faulty block needs to be distributed to a limited number of nodes at the boundary lines of block to avoid a message entering a detour area. We study the limited distribution of fault information in a dynamic network where faults occur during a routing process. Our study shows that fault information can be distributed quickly to help the routing process. In addition, the performance of routing process degrades gracefully in such a dynamic system. PCS routing scheme used in this paper and its experimental results show that certain levels of fault tolerance can be offered.

KEY WORDS
Dynamic faults, fault tolerance, 2-D meshes, routing, safety levels

1. Introduction

In a multicomputer system, a collection of processors (also called nodes) works together to solve large application problems. These nodes communicate and coordinate their efforts by sending and receiving messages through the underlying communication network. Thus, the performance of such a multicomputer system is dependent on the end-to-end cost of communication mechanisms. Routing is the process of finding a path from the source node to the destination node in a given system. Routing in mesh-connected networks, such as 2-D meshes, has been commonly discussed due to the structural regularity for easy construction and the high potential legibility for variety of algorithms. Some multicomputers are built based on 2-D meshes [3, 4, 6, 7].

As the number of nodes in a mesh-connected multicomputer increases, the chance of failure also increases. The complex nature of networks also makes them vulnerable to disturbances which can be either deliberate or accidental. Therefore, the ability to tolerate failure is becoming increasingly important, especially in the communication subsystems. Several studies have been conducted which achieve fault tolerance by adding (or deleting) extra components of the system [1, 8]. However, adding and deleting nodes and/or links require modification of network topologies which may be expensive and difficult. We focus here on achieving fault tolerance using the inherent redundancy present in the mesh-connected multicomputer, without adding spare nodes and/or links.

Recently, a routing switching technique known as *pipelined circuit switching* (PCS) is developed by Gaughan and Yalamanichili [2]. Unlike wormhole routing, PCS allows backtracking during the path setup phase. Backtracking is a key element in providing fault tolerance in a system with dynamic faults. However, without fault information, routing process may enter a region where all minimal paths to the destination are blocked by faulty nodes. Thus, PCS routing needs either detour or backtracking and causes routing difficulty which will increase routing delay and cause traffic congestion. The routing process here refers to the path setup phase. In PCS, the actual message sending occurs after a routing path is set up. Dynamic faults refer to ones appeared in the set-up phase only.

An optimal routing algorithm using faulty block information, which is a special form of limited distribution of fault information, is presented in [10]. Comparing with other fault information such as a routing table associated with each node, the update of faulty block information converges quickly. It reduces oscillation update caused by unstable information (also called inconsistent information). First, all faulty nodes are contained in disjointed faulty blocks by applying a labeling process. Routing is based on block information distributed at the nodes along the boundary lines of faulty blocks to avoid routing difficulties. Compared with the routing-table-based routing, this approach reduces the memory requirement to sto
information in the whole network. When a disturba
curs, only those affected nodes need to update fau

* This work was supported in part by NSF grant CCR 9900646.

Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'02)
1530-2075/02$17.00 © 2002 IEEE

mation. However, the approach in [10] uses a static fault model; that is, it is assumed that no new fault will occur during a routing process.

When dynamic faults occur, faulty blocks need to be reconstructed and their fault information needs to be redistributed. In this case, the update of fault information and the routing process proceed hand-in-hand. During the converging period, the routing process may experience more detours with inconsistent information. Most of routing techniques are not suitable for networks with dynamic faults. In addition, a good analytical model is lacking while we resort mostly simulations.

This paper is our first attempt to study the effect of dynamic faults on routing in 2-D meshes. We provide a collection and distribution process of fault information which exhibits desirable properties of self-stabilizing, self-optimizing, and self-healing. In a 2-D mesh, based on the extended safety level (a distance vector to closest faulty blocks along different dimensions), a safe source node can determine a minimal routing path. A detour may occur only if new faults occur during the routing process. Unlike a safe source, a routing message from an unsafe source needs one or more detours to reach its destination. We propose a fault-information-based PCS routing which keeps certain levels of fault-tolerance and adaptivity for any routing from either a safe source or an unsafe source.

The paper is organized as follow: In Section 2, the limited-global information model and its relevant properties are introduced. Some related research work are discussed. A collection and distribution process of the information model is presented in Section 3. In Section 4, a PCS routing based on fault information is provided. In Section 5, a dynamic fault model for the PCS routing is introduced. In Section 6, we discuss some important properties in term of the number of routing detours in 2-D meshes with dynamic faults. Section 7 shows simulation results. Section 8 concludes the paper and provides ideas for future research. Throughout the paper, proofs to theorems are omitted and they can be found in [5].

2. Preliminaries

2-D meshes. Each node u in a 2-D mesh is labeled as (x_u, y_u). Two nodes (x_v, y_v) and (x_u, y_u) are connected if their addresses differ in one and only one dimension. Basically, nodes along each dimension are connected as a linear array. The distance between two nodes s and d, $D(s,d)$, is equal to $\mid x_d - x_s \mid + \mid y_d - y_s \mid$. Assume node u is the current node, d is the destination node, and v is a neighbor of node u. v is called a *preferred neighbor* if $D(v,d) < D(u,d)$; otherwise, it is called a *spare neighbor*. Respectively, the corresponding connecting directions are called *preferred direction* and *spare direction*.

Block fault model. Most existing literatures on fault-tolerant routing use disjoint rectangular blocks to model and identify routing difficulties in meshes. A

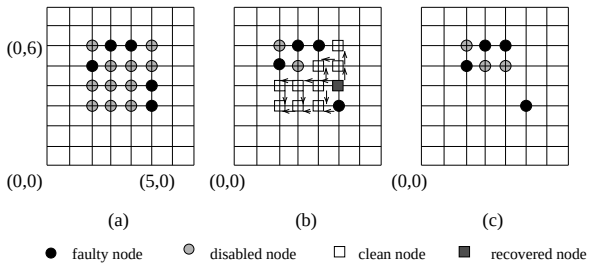

Figure 1. (a) A faulty block consisting of disabled and faulty nodes. (b) A clean process triggered by the recovery of (5,4). (c) Stabilized faulty blocks after the recovery.

node-labeling scheme that identifies nodes is as follows:

Definition 1: *In a 2-D mesh, a non-faulty node is initially labeled enabled; however, its status is changed to disabled if there are two or more disabled or faulty neighbors in different dimensions.*

In a 2-D mesh, an enabled node is an *adjacent node* of a faulty block if it has one faulty or disabled neighbor in that faulty block. And its connecting direction to that faulty or disabled node is *blocked* direction of such an adjacent node. A *corner* of a faulty block is defined as an enabled node with two adjacent nodes of that faulty block in different dimensions [9]; that is, it's connecting direction to one is the blocked direction of another. It is noted that an enabled node can be a corner for more than one faulty block. Connected disabled and faulty nodes form a faulty block. In Figure 1 (a), five faults (2,5), (3,6), (4,6), (5,4), and (5,3) form a rectangle [2:5, 3:6]. In general, $[x_{min} : x_{max}, y_{min} : y_{max}]$ represents a rectangle with four corners: $(x_{min}-1, y_{min}-1)$, $(x_{min}-1, y_{max}+1)$, $(x_{max}+1, y_{max}+1)$, and $(x_{max}+1, y_{min}-1)$. To simplify the discussion, it is assumed that souce s and destination d of a routing message are out of any faulty block.

Extended safety level. The extended safety level [10] of a node in a given 2-D mesh is a 4-tuple: (E, S, W, N), where E stands for the distance from this node to the closet faulty block to its east. S, W, and N are defined in a similar way. To ensure a minimal path from source node, Wu [10] provided a safe node definition: *Assume that source node s:$(0,0)$ has an extended safety level (E,S,W,N) and destination node is (x_d, y_d), with $x_d, y_d \geq 0$. The source node is safe (to the routing) if $x_d \leq E$ and $y_d \leq N$; otherwise, it is unsafe.* If a node is safe, a minimal path is guaranteed from source $(0,0)$ to (x_d, y_d) as long as no new fault occurs during the routing process.

Faulty-block-information used in minimal routing. A safe source of a routing message can ensure the existence of a minimal path. In [10], the notion of *region of minimal paths* (RMP) is introduced that includes all the inter nodes of minimal paths for a given source and des pair. That is, nodes and only nodes in this region a

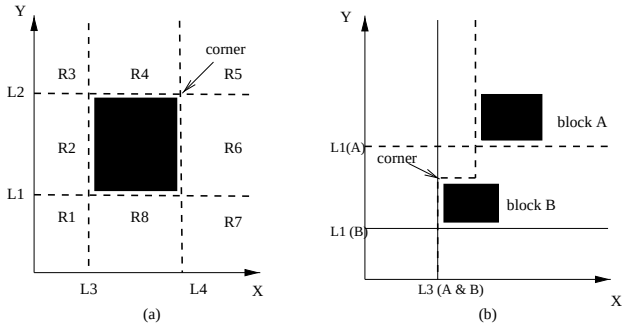

Figure 2. Boundaries of faulty blocks.

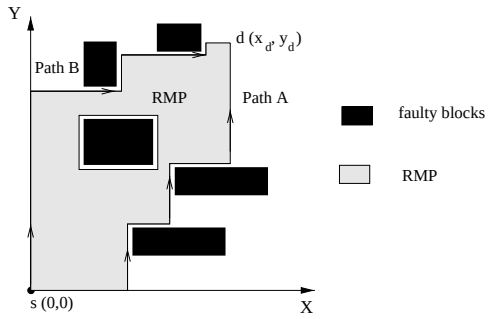

Figure 3. A sample RMP.

to construct a minimal path. The task of routing process for a minimal path is to ensure that each forwarding node along the path is inside RMP. Once the boundary of RMP is known we can construct any minimal path to support fully adaptive routing.

In [10], Wu presented a *faulty-block-information* model and such information is associated with nodes in the adjacent lines of a faulty block. These lines (L_1, L_2, L_3, and L_4 in Figure 2 (a)) are called *boundaries* of that faulty block. Based on these boundaries, RMP is easy to derive. Figure 2 (b) shows an example of boundaries of multiple faulty blocks. The boundaries start from two corners (NE-corner $(x_{max}+1, y_{max}+1)$ and SW-corner $(x_{min}-1, y_{min}-1)$, or NW-corner $(x_{min}+1, y_{max}+1)$ and SE-corner $(x_{max}-1, y_{min}-1)$) of each faulty block and go forward along with each direction of X and Y dimensions. Without any other faulty block, the propagation of boundary information is forwarded node by node in each direction until it reaches an edge of the mesh. If a boundary L_i intersects with another faulty block, a turn is made towards L_i of the second faulty block. Another turn is made at the corner of the second faulty block to join L_i (see Figure 2 (b) where L_3 of block A joins L_3 of block B).

For a message from source s: (0,0) to destination d: (x_d, y_d) with $x_d, y_d \geq 0$, a construction of RMP boundary from destination is provided in [10] and its reverse procedure from source is provided as following: If the source $(0,0)$ is safe, RMP is enclosed by two paths, Path A and Path B (see Figure 3). Faulty blocks inside RMP are excluded. Starting from source $(0,0)$, Path A is constructed by going east (positive X) until reaching the line $x = x_d$ and then by going north (positive Y) to reach destination (x_d, y_d). A turn from east to north is made if (a) the path hits a faulty block or (b) it crosses the lower section of boundary L_3 of a faulty block and the destination is in the area of R_4 which is one of regions divided by the boundaries of that faulty block (see Figure 2 (a)). After that, a turn from north to east is made at the NW corner (x_{min}, y_{max}) of the faulty block and the process continues. Path B is constructed in a similar way. In a regular mesh without faults, the corresponding RMP is a rectangle $[0 : x_d, 0 : y_d]$. The RMP of a source s and a destination d in a 2-D mesh with multiple faulty blocks is shown in Figure 3.

The above construction of RMP boundaries can be used to prevent the routing message from moving out of RMP. In [10], Wu proposed a unicast algorithm based on the faulty block information for any routing from a safe source s:$(0,0)$ to a destination d:(x_d, y_d) with $x_d, y_d \geq 0$. The safety status of the source is determined from the extended safety level associated with the source. The routing starts from a safe source and uses any adaptive minimal routing until the boundary of any faulty block is met. If the selection of any preferred neighbor does not affect the minimal routing, the path is *non-critical*; otherwise, it is *critical*. In case of a critical path, one of preferred directions cannot be selected in a minimal routing due to the effect of faulty blocks. Such a direction is called *preferred but detour direction*. The selection should be done at current node u:(x_u, y_u) based on the relative location of the destination:

- (u is on the left section of L_1 of any faulty block): If the destination is in the area of R_6 divided by the boundaries of that faulty block (see in Figure 2 (a)), the routing message should stay on L_1 until reaching the intersection of L_1 and L_4 (that is, positive X is a preferred direction and positive Y is a preferred but detour direction); otherwise, the next hop can be randomly selected.

- (u is on the lower section of L_3 of any faulty block): If the destination is in the area of R_4 divided by the boundaries of that faulty block the routing message should stay on L_3 until reaching the intersection of L_3 and L_2 (that is, positive Y is a preferred direction and positive X is a preferred but detour direction); otherwise, the next hop can be randomly selected.

For example, as shown in Figure 2 (b), when the routing from $(0,0)$ meets the lower section of L_3 of faulty block B, it also meets L_3 of faulty block A. If the destination is not in R_4 of A or R_4 of B, the routing is still non-critical and any of two preferred directions (positive X and positive Y) can be selected randomly; otherwise, the routing is critical and the message cannot be for

to positive X. In this case, positive X is the preferred but detour direction and positive Y is the only preferred direction that can be selected to construct a minimal path.

3. Faulty Block Information

In 2-D meshes, the shape of a faulty block may change by the occurrence of new faults or by the recovery of nodes from faulty status to healthy one. To identify the shape of new faulty blocks and propagate their block information along the boundaries, three procedures are used to exchange and update related information among neighbors: *block construction*, *identification process*, and *boundary construction*. Here we use a *reactive approach* where information update at a node is done only when there are status changes among its neighbors.

First, a new labeling scheme is proposed as follows:

Definition 2: *In a 2-D mesh, if any new fault occurs, Definition 1 is applied. If any node is recovered from faulty status, it is labeled clean. A disabled node is labeled clean if it has a clean neighbor and has not two faults in different dimensions. Once all its neighbors have known its clean status in the clean process, each clean node is labeled enabled. Each enabled node applies Definition 1 until there is no status change.*

Specifically, a recovered node is set to *clean*. The change will be propagated to its disabled neighbors and contribute further changes. In Figure 1 (b), node (5,4) is recovered from faulty status. First, (5,4) is labeled clean and it triggers the change of status in its disabled neighbors (4,4) and (5,5) to clean. The procedure continues until there is no more status change. The stabilized faulty blocks are shown in Figure 1 (c). It is noted that the clean status of (4,4) will trigger the change of status in (4,5) and (3,4). When (3,5) knows the status changes of (4,5) and (3,4), it does not change its status to clean since it has two faulty neighbors in different dimensions. (4,5) changes to enabled once all its neighbors have known its clean status. In the next round, it has one faulty neighbor (4,6) and one disabled neighbor (3,5). And then, this new enabled node will change to disabled when Definition 1 is applied.

The new enabled/disabled labeling scheme can quickly identify those non-faulty nodes that may cause routing difficulty by labeling them disabled. For each occurrence of a new fault or a new recovered node, the new node status can be easily determined through rounds of status exchanges among neighbors. Only these affected nodes update their status. Such a procedure is called *block construction*.

After the block construction incurred by some new faults, a faulty block may enlarge its size, and even a new faulty block may appear in the network. On the other hand, after the block construction incurred by some nodes recovered from faulty status, a faulty block may shrink its size or be divided into several small faulty blocks. If a corner finds its existing condition cannot

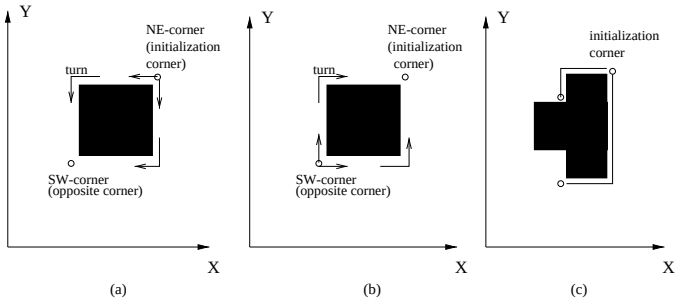

Figure 4. (a) Identification process activated at NE-corner and SW-corner. (b) Identified information re-sending. (c) Exceptions in identifying propagation.

be satisfied, the existing boundaries of such a block is outdated. The deletion process starts and will propagate along those old boundaries. To identify a new faulty block, information of all the adjacent nodes of this block is identified by a procedure called *identification process*. For each new faulty block in the network that needs identification (if any), it has at least one new corner of this faulty block. The identification process is activated at such a new corner. Since the system is distributed and dynamic, no corner knows if the block construction is completed. Thus, this procedure starts whenever a *new* corner is formed. For each initialization corner, two identification messages (one clockwise and one counter-clockwise) are initiated (see Figure 4 (a)). Each message initiated at that corner C:(x_c, y_c) carries partial block information: (x_c, y_c) and the blocked direction of the neighbor is also sent. First, they will be sent to those two neighbors adjacent to the new block. Such propagation will continue until the message traverses all the enabled nodes adjacent to the new faulty block. The clockwise and counter-clockwise messages from one corner will reach the opposite corner of that initialization corner. With the position information of pair of corners (the initialization corner and its opposite corner), the new faulty block is identified and the faulty block information $[x_{min} : x_{max}, y_{min} : y_{max}]$ is formed at that meeting corners. After that, these two messages will continue their propagation (see Figure 4 (b)) and carry this identified faulty block information back to the initialization corner.

To guide the routing process, the faulty block information is transferred along the boundary of the new faulty block from the initialization corner and its opposite corner when they get the identified information (see in Figure 2). In our reactive model, if any corner has already had the new faulty block information, there is no need to start a new boundary propagation. This propagation may also incur a deletion of out of date boundaries and update the boundaries of other faulty blocks. Such a procedure is called *boundary construction*. All these proced
shown in Algorithm 1.

Note that each identification message (cloc

Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'02)
1530-2075/02 $17.00 © 2002 IEEE

Algorithm 1: Block construction and information distribution

1. Block construction by applying Definition 2.
2. Identification of adjacent nodes and corners.
3. Identification process: (a) Two identification messages (one clockwise and one counter-clockwise) are sent along the enabled nodes adjacent to the new block from a new corner, until they reach the opposite corner. (b) Partial block information from the initialization corner is transferred to form faulty block information at the opposite corner. (c) The faulty block information is sent along the adjacent nodes back to the initialization corner by these two messages.
4. A boundary construction is activated at initialization corner and its opposite corner that receives consistent faulty block information.

counter-clockwise) sent from its initialization corner is expected to make one and only one *specific turn* (90° right/left turn) before reaching the opposite corner. If there is a faulty or disabled neighbor in the forwarding direction, it is ensured that the new block is not stable. The message is discarded at current node to avoid generating incorrect faulty block information. If only one message from the initialization corner is received at the opposite corner, the other is discarded in the propagation procedure or has a wrong turn. That is, the shape of block may not be the exact rectangle indicated by the positions of the initialization corner and its opposite corner. Normally, a TTL (time-to-live) is associated with each identification message and the corresponding message will be discarded once the time expires (see Figure 4 (c)). After these two messages from the same initialization corner meet at the opposite corner, the propagation continues. But this time, the stable block ensures that they can go back to their initialization corner.

4. Faulty-block-information-based Routing

The PCS routing in [2] needs a detour when its preferred neighbors are all faulty and needs a backtracking when all its outward directions have been tried or blocked by faulty neighbors. The routing based on fault information is an adaptive routing in 2-D meshes. Like a regular minimal routing, at each step it tries to forward the message to a preferred neighbor. The difference is that the selected preferred neighbor can ensure the remaining path is minimal in the fault-information-based routing (if there is no occurrence of a new fault).

Algorithm 2 shows a fault-information-based PCS routing from s:(x_s, y_s) to d:(x_d, y_d) in a 2-D mesh with dynamic faults. For the current node u:(x_u, y_u) $(\neq d)$ with

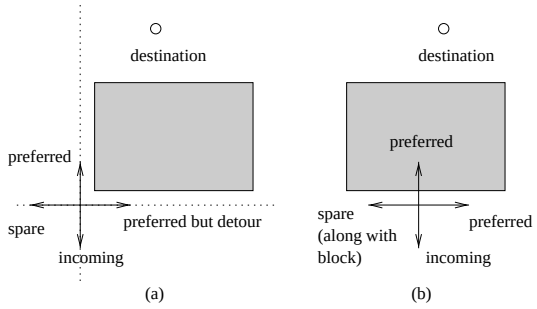

Figure 5. Routing directions.

Algorithm 2: Fault-information-based PCS routing

1. If the current node u is disabled, backtrack; otherwise,
2. pick an unused outgoing direction with the highest priority. The address of u and the direction selected is recorded in the message header.
3. If there is no unused outgoing direction, backtrack.
4. If the message is backtracked to the source, the destination is unreachable.

tions, the routing selects one of directions as the forwarding direction in the priority order of preferred, spare (along with block), preferred but detour, and incoming directions (see in Figure 5). Normally, we have the following cases for intermediate node u: (a) If $x_u = x_d$ or $y_u = y_d$, there is one preferred direction and two spare directions. (b) If $x_u \neq x_d$ and $y_u \neq y_d$, there are two preferred directions and one spare direction. (c) At a boundary line, if it is critical, one preferred direction changes to preferred but detour direction. If it is not critical, there is no preferred but detour direction.

It is noted that each forwarding direction at a participant node cannot be used again. Thus, each routing header in our PCS routing includes destination address and a list of used-directions for each forwarding node along the path. This is because that the system is dynamic and the priority of directions may also change. Theorem 1 ensures the effectiveness of our fault information model when faults are recovered in the networks.

Theorem 1: *The constructions of the fault recovery do not affect the optimal routing.*

5. Dynamic Fault Model

In general, it is impossible to design an optimal fault-tolerant routing algorithm when node failure and r
can occur dynamically at any time. Under such gen
namic failure assumptions, it is not even possible t

Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'02)
1530-2075/02 $17.00 © 2002 IEEE

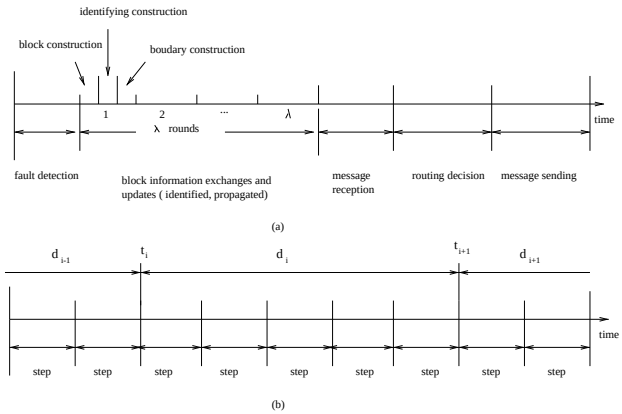

Figure 6. (a) Actions within a step. (b) Interval d_i.

antee an upper bound on the routing distance since no algorithm can predict when and where faults will occur. It is therefore necessary to impose some failure conditions and restrictions.

We adopt the following model for activities in a node. At each step, every node in an $m \times m$ mesh starts with fault detection of adjacent links and nodes, and then collects and distributes three kinds of fault information: block information, identifying information, and boundary information through λ rounds of exchanges and updates. The disabled/enabled status propagation, identifying message, block information updating advance one hop further at each round. Before the end of each step, based on the fault information, a routing decision selects a forwarding node to forward the routing message and, then, the message is sent to this forwarding node. Therefore, every routing message advances one hop along with its routing path at each step. The actions within a step are shown in Figure 6 (a).

To simplify the discussion, it is assumed that any adjacent faults, links and nodes, are detected at fault detection phase (any faults occur after the fault detection phase will be detected at the next step). During the update of fault information, each node can also receive one incoming message (if any). It is also assumed that the action "message receive" occurs right before the "routing decision" as shown in Figure 6 (a). The model used represents a reactive approach where information update is done only when there is a change of information of at least one neighbor.

We assume there are at most F faulty nodes in a 2-D mesh network, including dynamically generated faults. Faults f_1, f_2, ..., f_F occur at time $t_1, t_2, ..., t_F$; respectively, where $t_{i+1} - t_i = d_i$ $(1 \leq i < F)$. d_i is the interval between two consecutive fault occurrences (see in Figure 6 (b)). To simplify our discussion, it is assumed that the fault information updating in the mesh is already stabilized before the occurrence of the next fault and there is no fault that occurs at the edge of mesh. Based on the properties discussed in [10], there is no disconnected area in 2-D mesh. It means that there is always a path between

F	number of faults in a given $m \times m$ mesh and $F \leq 2m$
f_i	i^{th} fault occurrence where $i \in \{1, 2, ..., F\}$
t_i	occurrence time of f_i
d_i	the interval between two consecutive fault occurrences f_i and f_{i+1}; i.e., $d_i = t_{i+1} - t_i$
t	start time of a routing process
p	number of fault occurrences before the routing starts
D	distance from source to destination
$D(i)$	distance from the current node to the destination at t_i
a_i	total rounds that the stabilizing block construction for f_i converges
a_{max}	$\max\{a_i\}$
e_{max}	maximum of all length or width of fault blocks
b_i	total rounds that the stabilizing identification process for f_i converges
c_i	total rounds that the stabilizing boundary construction for f_i converges
λ	number of rounds of fault block construction and information distribution at each step

Table 1. List of notations used in the discussion.

the source and the destination. Before a routing message is initiated at time t, it is assumed that the first p fault occurrences have already occurred; that is, $p = \max\{l | t_l \leq t\}$. $D(i)$ represents the distance from the current node (u) to the destination (d) at time t_i when f_i occurs ($1 \leq i \leq F$) and D represents the distance from source to the destination. Before the start time t, the routing message is at its source and $D(i) = D = \mid x_s - x_d \mid + \mid y_s - y_d \mid$ $(i \leq p)$. For the i^{th} fault occurrence, the block construction will be stabilized in $\lceil \frac{a_i}{\lambda} \rceil$ steps, the identifying construction will be stabilized in $\lceil \frac{b_i}{\lambda} \rceil$ steps, and the boundary construction will be stabilized in $\lceil \frac{c_i}{\lambda} \rceil$ steps. We assume that $d_i > \max\{\frac{a_i + b_i + c_i}{\lambda}\}$. Therefore, before the next occurrence of fault (t_{i+1}), the new boundaries incurred by the fault occurrence at t_i are already stabilized. To simplify the discussion, Table 1 summarizes the notation used in this paper.

6. Detour Analysis

Based on the definition of safe source, the corresponding PCS routing does not need any detour if there is no new fault. If a new fault occurs, before the new information distribution is stabilized, a routing message may use inconsistent information and enter a detour area. If the size of a faulty block is limited by e_{max}, the number of detours is limited. Enlarging the interval will increase the number of optimal steps in it. Since the distance from s to d is limited in a 2-D mesh, the maximum number of intervals before the routing message reaches its destination can be reduced by enlarging the interval; in other words, the number of total detours is reduced.

Theorem 2: *For any fault-information-based routi*
a safe source s to an enabled destination d, if D(i+

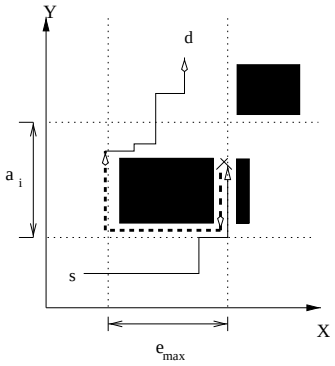

Figure 7. The maximum number of detours by a new fault.

$$\begin{cases} D(i) & = D & i \le p \\ D(i+1) & \le D - (d_i - t + t_p - 2a_i - 2e_{max}) & i = p \\ D(i+1) & \le D(i) - (d_i - 2a_i - 2e_{max}) & p < i < F \end{cases}$$

Assume that $p + q - 1$ is the largest index for fault such that $D(p+q-1) > 0$. That is, f_{p+q-1} is the last fault occurrence that could affect the routing process. Actually, q is the maximum number of intervals in which the routing message detours at least once.

Theorem 3: *For a routing message from a safe source s to a destination d in an $m \times m$ 2-D mesh, the routing process will end in the following q intervals and*

$$q \le \max\{l | D + t - t_p - \sum_{i=p}^{p+l-2} (d_i - 2a_i - 2e_{max}) > 0\}.$$

Let us consider several cases for a routing from a safe source, i.e., an optimal path exists before it starts:

- The routing message will get closer to the destination between two occurrences of consecutive faults as long as these two faults are separated by more than $2a_i + 2e_{max}$ time steps.
- When d_i's are uniform, i.e., $d_i = c$, $q = \max\{l | (l-1)(d_i - 2a_i - 2e_{max}) < D + t - t_p \le D + c\}$. Then $q \le \lfloor \frac{D+c}{c-2a_{max}-2e_{max}} \rfloor$.
- The maximum number of detours for a message from a safe source is $(a_{max} + e_{max}) \times \lfloor \frac{D+c}{c-2a_{max}-2e_{max}} \rfloor$.

Theorem 2 shows the maximum number of detours in each interval for a routing message from a safe source. Based on this, we get an upper bound of the maximum number of detours for such a routing message in Theorem 3. The following theorem extends the above result for any routing message including the one from an unsafe source (see in Figure 8).

Theorem 4: *For a routing message from a source s to a destination d in an $m \times m$ 2-D mesh, if there is a routing*

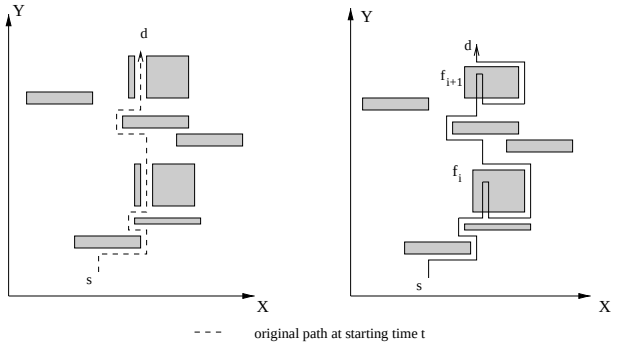

Figure 8. The maximum number of detours for a routing with an unsafe source.

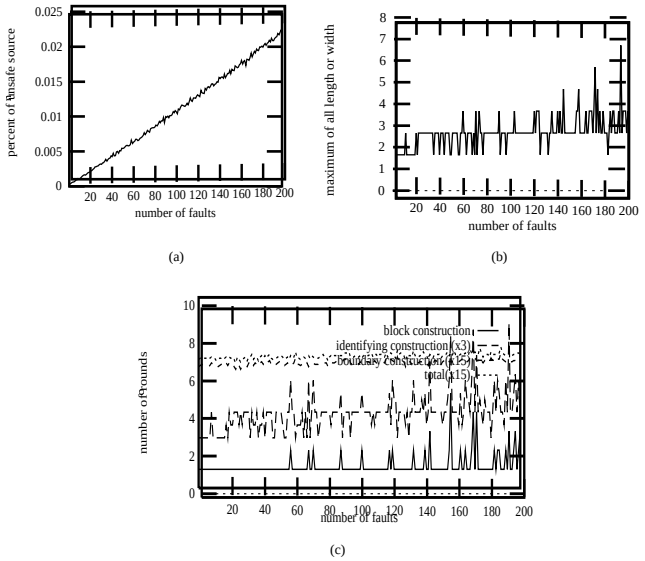

Figure 9. a_i, b_i, c_i, e_i, and $a_i + b_i + c_i$ in a 100×100 mesh with at most 200 faults (intervals).

path of length L at start time t, the routing process will end in the following k intervals and

$$k \le \max\{l | L + t - t_p - \sum_{i=p}^{p+l-2} (d_i - 2a_i - 2e_{max}) > 0\}.$$

The probability of the maximum number of detours in a routing is no more than

$$\prod_{i=p}^{p+k-1} (\frac{1}{m^2}).$$

7. Simulation

A simulation has been conducted on a 100×100 r
test the estimated maximum number of detours us
PCS routing based on fault information.

λ	maximum number of detours					average detours /message		
	analysis results		experiment results					
	(S)	(G)	(S)	(G)	(W)	(S)	(G)	(W)
200	42	177	24	48	184	0.066	0.067	0.508
110	42	177	24	48	168	0.068	0.069	0.593
30	42	177	32	78	175	0.132	0.134	0.520
7	42	177	32	78	173	0.136	0.139	0.511
1	42	177	42	85	158	0.202	0.231	0.604

Table 2. Comparison of the maximum number of detours among routing from a safe source (S) (or from an unsafe source (G)) with fault information and routing without fault information (W).

We randomly generate faults, source and destination. Figure 9 (a) shows that only few source nodes are unsafe. It means that almost all the routings fall under the cases discussed in Theorem 3. From experimental results shown in Figure 9 (b), $e_{max} = 7$. Figure 9 (c) shows the number of rounds needed for a new block construction and its information distribution. From experimental result, $a_{max} = 7$ and $\max\{a_i + b_i + c_i\} = 110$.

Table 2 shows the maximum number of detours of the routing with fault information at different information distribution speed (λ =200, 110, 30, 7, and 1) in a 100×100 2-D mesh with up to 200 dynamic faults when $d_i = 110$. (S) is the maximum number of detours for routing from safe source, (G) is for routing from unsafe source, and (W) is for routing without fault information. We make the following observations from the comparison shown in Table 2.

- The analytical result on the maximum number of detours is rather accurate as an upper bound.
- The experimental results of PCS routing using fault information are lower than those of analytical results. It is because that the worst fault configuration is difficult to occur when faulty nodes and source and destination pair are all randomly generated. Even when the worst case occurs, based on Theorem 4, the probability of the maximum number of detours is very small.
- Based on different requirements, we can adopt different information distribution speeds. If we can accept several more detours, the best choice is to select $\lambda \geq \max\{a_i\}$. It does not need a fast information distribution. If the propagation of fault information is processed as the same speed as transmission of routing message ($\lambda = 1$), although it is the worst case of all, the upper bound of the maximum number of detours is still accurate.

In summary, the experimental results of routing algorithm using fault information are close to those analytical results, even when the routing starts from an unsafe source. As an upper bound of the maximum number of detours, the analytical result is rather accurate.

8. Conclusions

We have studied an upper bound of the maximum number of detours in a 2D mesh with dynamic faults using fault-tolerant routing algorithm based on limited global information. The concept of fault information associated with each node at the boundary lines of faulty blocks has been used to represent limited global information. Our study shows that such limited global information can be collected and distributed quickly to help the routing process. Simulation results show the accuracy of our analytical upper bound of detour number. Applying this approach to other fault models are interesting problems for future research.

References

[1] S. Dutt and J. P. Hayes. Some practical issues in the design of fault-tolerant multiprocessors. *IEEE Trans. on Computers*. May 1992, 588-598.

[2] P. T. Gaughan and S. Yalamanchili. A family of fault-tolerant routing protocols for direct multiprocessor networks. *IEEE Transactions on Parallel and Distributed Systems*. 6, (5), May, 1995, 482-497.

[3] INTEL. *A Touchstone DELTA System Description*. Intel Corp., Santa Clara, CA, 1990.

[4] INTEL. *Paragon XP/S Product Overview*. Intel Corp., Santa Clara, CA, 1991.

[5] Z. Jiang and J. Wu. Dynamic routing in 2-d meshes. TR-CSE-02-01, Dept. of Computer Science and Engineering, Florida Atlantic University.

[6] S. L. Lillevik. The touchstone 30 gigaflop delta prototype. *Proc. of the 6th Distributed Memory Computing Conference*, pages 671–677, 1991.

[7] C. L. Seitz. The architecture and programming of the ametek series 2010 multi computer. *Proc. of the 3rd Conference on Hyperucbe Concurrent Computers an d Applications*, pages 176–182, 1988.

[8] N. F. Tzeng and G. Lin. Maximum reconfiguration of 2-d mesh systems with faults. *Proc. of 1996 International Conference on Parallel Processing*. 1996, 77-84.

[9] J. Wu. A distributed formation of orthogonal convex polygons in mesh-connected multicomputers. *Proc. of International Parallel and Distributed Processing Symposium*. April, 2001.

[10] J. Wu. Fault-tolerant adaptive and minimal routing in mesh-connected multicomputers using extended safety levels. *IEEE Trans. Parallel and Distributed Systems*. 2, (11), Feb., 2000, 149-159.