Towards a Parallel Framework of Grid-based Numerical Algorithms on DAGs

Zeyao Mo Aiqing Zhang Xiaolin Cao
Institute of Applied Physics and Computational Mathematics, P.O.Box, Beijing, 100088.
{zeyao_mo, aiqing_zhang, xiaolincao} @iapcm.ac.cn

Abstract†

This paper presents a parallel framework of grid-based numerical algorithms where data dependencies between grid zones can be modeled by a directed acyclic graph (DAG). It consists of three parts on how to partition, order and calculate the vertices of digraph. Numerical results using hundreds of processors on two parallel machines show the efficiencies and moderate scalability of this framework.

1. Introduction

In recent two decades, various parallel algorithms have been successfully designed for a wide range of numerical computations arising from the grid-based simulations of partial differential equation where a discrete solution is defined on a grid [6]. The grid consists of a set of disjoint polyhedrons called ***zones*** [5]. Dependencies between zones are often symmetric and can be depicted by undirected graphs [8]. In sequences, the parallel framework of such numerical algorithms can be designed in the concept of the Bulk Synchronous Parallel (BSP) programming model [20].

However, such symmetric dependencies don't hold for another type of grid-based numerical computations where zones should be calculated in the downstream style of dataflow modeled by DAG. For example, in the field of high energy density plasma physics [5], radiation transport should be mathematically modeled in the form of discrete ordinates of the Boltzmann equation [5] as follows

$$\frac{1}{v_g}\frac{\partial I_{mg}}{\partial t}+\Omega_m \bullet \nabla I_{mg}=R(I,t)\equiv S_{mg}-(\sigma_A+\sigma_S)I_{mg}+\sum_{\hat{g}}\sum_{\hat{m}} w_m \sigma_{S,\hat{g}\to g,\hat{m}\to m} I_{\hat{m}\hat{g}} \qquad m=1,2,....,M;\ g=1,2,..,G \tag{1}$$

Here, $I_{mg}(x,y,z,t)$ is the radiation flux where g shows the energy group distinguished with the velocity of v_g and m shows the angular direction Ω_m scaled with the weight of w_m. $\Omega_m \bullet \nabla I_{mg}$ is the transport term. $R(I,t)$ denotes the source term including loss factors due to absorption σ_A and scattering σ_S, an additive term due to the production of radiation flux from material, and an in-scattering term from various directions and energies.

Implicit stencil for temporal discretization is essential for above equation. It can be written as

$$\frac{1}{v_g}\frac{I_{mg}^{l+1}-I_{mg}^{l}}{\Delta t}+\Omega_m \bullet \nabla I_{mg}^{l+1}=R(I^{l+1},t^{l+1}) \tag{2}$$

This system is often solved with the following algorithm.

Algorithm 1 [13, 22]. *Iterative solution of a time step.*

(1) $v=0, I_{mg}^{l,v}\equiv I_{mg}^{l}$;

(2) FOR (m=1,2,…,M) DO {

$$I_{mg}^{l,v+1}+v_g\Delta t\Omega_m \bullet \nabla I_{mg}^{l,v+1}=I_{mg}^{l,v}+v_g\Delta t R(I^{l,v},t^{l+1}) \qquad g=1,2,...,G \tag{3}$$

} ENDDO

(3) Update the source term using new flux.

(4) Source term converges? If no, v=v+1, goto step (2).

Downstream sweeping is an exact solver for equation (3). However, as showed in the left of figure 1, zones should be downstream swept for the angular direction

† This work is under the auspices of NSF for DYS (No. 604252 05) ,National Basic Key Research Special Fund (2005CB321 702), Chinese NSF (No.60533020) and Funds of CAEP.

1-4244-0054-6/06/$20.00 ©2006 IEEE

from the left upper to the right lower in the following sequence {1,6} -> {2,11} -> {3} -> {4} -> {5, 7} -> {8} -> {9,12} -> {10,13,14} -> {15} -> {16}. Here, bracket denotes zones can be concurrently swept there and an arrow denotes the data dependencies. In fact, such dependencies can be accurately defined by a DAG as showed in the right of figure 1 where each vertex relates to a zone, and each arc shows that the zone located at the tail must be swept after the zone located at the head.

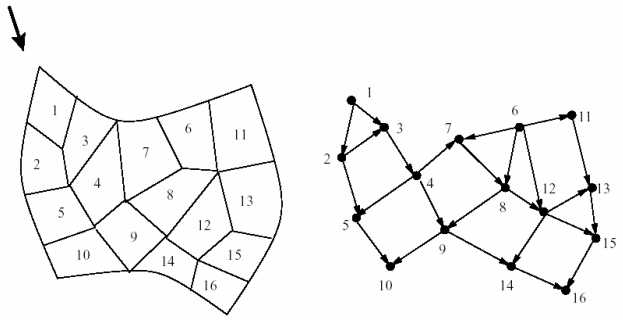

Figure 1 [17] Left: direction sweeping across unstructured grid. Right: DAG.

Transport equation is a great challenge for larger scale computations in the space of seven dimensions [13]. Many parallel realizations have been presented for algorithm 1 based on the well-known parallel pipelining techniques [23]. On the rectangular grid, the realization is trivial because regular pipelines can be well predefined [2,3,19]. However, on the unstructured grid, it is more difficult because of the irregular dependencies. Plimpton et.al.[17] and Mo et.al.[15, 16] addressed this problem in different coordinate systems for various applications.

Besides from transport equation, the downstream sweeping are used for realization of many other numerical cores for convection dominated or Navier-Stokes equation on rectangular grid [14,24] and on unstructured grid [4, 21]. All these realizations must be carefully designed according to the characteristics of zone shapes, discrete stencils, downstream directions, and so on. Is a parallel framework possible for downstream sweeping independent of applications? This paper tries to find a solution.

In section 2, the model of DAG is constructed for accurate description of data dependencies for a wide range of grid-based numerical computations. In section 3, a parallel framework is presented to compute the DAG. Lastly, numerical results are reported using hundreds of processors on two parallel machines in section 4.

2. The Model of DAG

We use the basic terminologies and notations introduced in monograph [8]. A digraph D consists of a non-empty finite set $V(D)$ of elements called ***vertices*** and a finite set $A(D)$ of ordered pairs of distinct vertices called ***arcs***. We write $D=(V, A)$ which means that V and A are the vertex set and arc set of D, respectively. The ***order*** (***size***) of D is the number of vertices (arcs) in D; the order of D is denoted by $|D|$.

Rewrite the downstream sweeping in equation (3) as a more general formulation suitable for other numerical cores as follows

$$I_m^{v+1} + a\Omega_m \bullet \nabla I_m^{v+1} = W \tag{4}$$

Here, Ω_m is perhaps a constant, a linear or a nonlinear function for the grid-based numerical algorithms varying from transport equation, linear convection equation to the nonlinear convection equation. So, we refer them as constantly, linearly or nonlinearly downstream sweeping. In this paper, we mainly consider the former two cases because the digraph can be accurately predefined there.

2.1 DAG for single constant sweeping

Consider a digraph $D\equiv D(\Omega_m)$ constructed from constant direction Ω_m across the computational grid composed of N zones denoted by $Z=\{z_i|i=1,2,\ldots,N\}$. Assign $V(D)=\{v_i \mid i=1,2,\ldots,N\}$. Vertex v_i corresponds to zone z_i. We refer each face of a zone be the ***inflow face*** or ***outflow face*** if and only if the inner product $\theta_{mi} = \Omega_m \bullet \vec{\chi}_i$ is negative or positive. Here, $\vec{\chi}_i$ is the outer normal vector of this face. In the left of figure 2, we depict a zone with four faces, the inflow and outflow faces are denoted by $\partial^- z$ and $\partial^+ z$ respectively. Inflow face means that the flux should enter this zone from its neighboring zone called ***upstream zone*** across this face. Moreover, each zone ***depends on*** its

upstream zones. The right in figure 2 depicts the data dependencies among three zones where zone z depends on both zone z_0 and zone z_1.

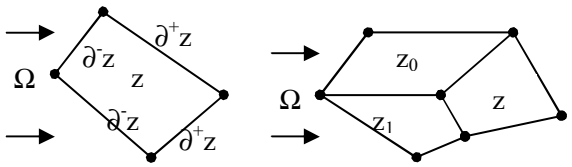

Figure 2. Data dependencies among three zones.

We generate an arc denoted by $e_{ij}=(v_i,v_j)$ means that vertex v_i depends on vertex v_j if and only if zone z_i depends on zone z_j. We assign the weight of each vertex v_i the local computations of zone z_i and that of each arc the data size on which the downstream zone depends. So, the ***computation of digraph*** can be defined as algorithm 2 modified from monograph on digraph theory [8].

Algorithm 2: *The computation of digraph*

(1) FOR (i=1,2,…,N) DO {$f(v_i)$= $d^-(v_i)$: in-degree of vertex v_i }, list $\psi=\{v_k \in V(D): f(v_k)=0\}$, list $\psi_0=\phi$.

(2) DO WHILE {$|\psi_0|\neq|D|$} {

(2.1) Compute the head v_k of ψ, let $\psi_0=\psi_0\cup\{v_k\}$, $\psi=\psi\setminus\{v_k\}$.

(2.2) FOR (each downstream v_j of v_k) DO {

(2.2.1) $f(v_j)=f(v_j)$-1.

(2.2.2) if ($f(v_j)$=0) then { $\psi=\psi\cup\{v_j\}$;

} else { transfer data from v_k to v_j.}

} ENDDO for step (2.2)

} ENDDO for step (2).

List ψ always maintain vertices ready for computations. A vertex can be joined into ψ if and only if all its upstream vertices have been calculated. A digraph is ***computable*** if and only if step (2) terminates within $|D|$ iterations. In fact, we have the trivial conclusions.

Theorem 1 *A digraph is computable if and only if it has no cycle.*

Property 1. *A diagraph is computable on rectangle grid.*

Property 2 *A digraph is acyclic on unstructured grid in two-dimensional geometry only if each zone is convex.*

Proof: Assume the digraph has a cycle denoted by $\Theta=v_{i,1}v_{i,2}\cdots v_{i,m}v_{i,1}$ where the double subscripts indicating the indices of vertices. For each zone represented by vertex $v_{i,j}$, consider the normal vector of its outflow face to its neighbor represented by vertex $v_{i,j+1}$, then the vector must positively intersects with the sweeping direction. So, two faces are existed such that their normal vectors negatively intersect with each other. This contradicts with the convex property of unstructured grid.

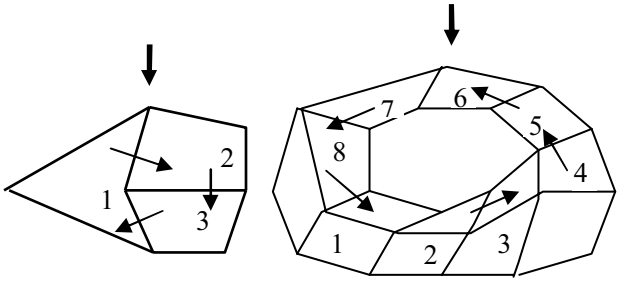

Figure 3. Left: three zones form a cycle. Right: a ring of eight hexahedrons form a cycle

A digraph may include a cycle for concave zones in the case of two-dimensional geometry as showed in left of figure 3. Fortunately, such zones are usually forbidden for numerical simulations of realistic applications [5]. However, a digraph may include a cycle in the case of three-dimensional geometry even if each zone is convex. In fact, as showed in the right of figure 3, the hexahedrons will constitute a cycle if we cut a ring along with the direction of downstream sweeping.

For each digraph constructed from the grid-based numerical algorithms, the cycle should be detected and then be broken before the digraph is computed. Some efficient algorithms can detect a cycle [8], Hackbush et.al [9,10] present a algorithm for convection dominated fluids, Plimpton and Hendrickson [17] present an application-specific strategy. In this paper, we always assume the digraph is acyclic.

2.2 DAG for multiple constant directions

A vertex is a pair between zone z_i and sweeping direction

Ω_m denoted by $u_{im}=(z_i,\Omega_m)$, an arc is an ordered pair of vertices denoted by $e_{im,,op}=(u_{im},u_{op})$ if and only if

(C1) $m=p$ and z_o depends on z_i for Ω_m, or,

(C2) $i=o$, $p>m$ and Ω_p depends on Ω_m.

In the case of Cartesian geometry [17], the second condition is no useful since all directions are independent from each other. But in the case of cylindrical geometry [15], or in the sphere geometry [13], these two conditions should be simultaneously considered.

Property 3. *The digraph for multiple directions is acyclic if and only if each digraph for single direction is acyclic.*

Proof. The sufficiency is trivial because each digraph for single direction is an induced subdigraph. If each digraph for single direction is acyclic, arcs introduced by condition (C1) will never form a cycle. Based on these arcs, condition (C2) always generates arcs pointing from lower direction to upper direction, so new cycles will never be introduced.

2.3 DAG for linearly downstream sweeping

Linearly downstream sweeping usually occurs in the robust solver for convection-dominated fluids where the convection direction Ω_m is a function independent of solution. The DAG can be similarly defined since a new arc can be constructed from the direction Ω_m for two neighboring zones independent of solution.

2.4 The model of DAG

We construct another digraph $\Re(D)$ from the digraph D by substituting the set of vertices $\{u_{im}:m=1, 2,\ldots,M\}$ predefined on zone z_i to a single vertex denoted by u_i, and denote the underlying graph of D and $\Re(D)$ by $\underline{D}$ and $\underline{\Re}(D)$ respectively. For $\underline{\Re}(D)$, we contract the parallel arcs and delete the loops, then a simple digraph will be resulted, denote its underlying graph by $U(D)$. In order to distinguished from the original underlying graph $\underline{D}$, we refer $U(D)$ as the super underlying graph of D, each vertex as the super vertex and each arc as the super arc. Obviously, $U(D)$ represents the zones connectivity of grid-based numerical algorithms. For the case of single direction, $U(D)$ is the same as $\underline{D}$.

By above definitions, we can consider the following model of DAG

$$D \equiv (V(D), A(D), U(D)) \tag{5}$$

Here, $V(D)$,$A(D)$ and U(D) are the set of vertices, arcs and the super underlying graph. It accurately represents the data dependencies independent of the characteristics of grid-based numerical algorithms no matter what types of grid zones, coordinate system, or equation coefficients are used.

3. Parallel framework

In this section, we present a parallel framework for the computation of DAG. It consists of three parts such that we firstly partition the digraph for the distribution of vertices among processors, secondly design the parallel algorithm for downstream sweeping and thirdly present the priority strategies for ordering each of vertices.

The natural method to partition the DAG is the application of many undirected graph partitioning methods [18] on the super underlying graph. The set of vertices predefined on a super vertex is distributed to the same processor. For these methods, loads can be well balanced. We don't discuss them here.

Assume D has been partitioned into P subdigraphs denoted by D^k distributed to processor p_k (k=1,2,…,P), then the downstream sweeping in algorithm 2 can be substituted by the following parallel version.

Algorithm 3. *Parallel downstream sweeping.*

FOR (k=1,2,…,P) *processor* p_k DO *in parallel* {

(1) $f(v_i)= d^-(v_i) \quad \forall v_i \in V(D^k)$; *list* $\psi_k=\{v_i \in V(D^k): f(v_i)=0\}$; *list* $\psi_0=\phi$.

(2) WHILE $\{|\psi_0|\neq|D^k|\}$ DO {

(2.1) WHILE ($|\psi_k|\neq\phi$) DO {

(2.1.1) Compute the head v_l of ψ_k,

$\psi_0=\psi_0\cup\{v_l\}$, $\psi_k=\psi_k\backslash\{v_l\}$;

(2.1.2) FOR (each downstream v_j of v_l) DO {

+ IF ($v_j \in V(D^k)$) {$f(v_j)=f(v_j)$-1; }
+ IF ($f(v_j)$=0) {$\psi=\psi\cup\{v_j\}$;} ELSE {
send a message including v_l to
the processor owning vertex v_j.}
} ENDDO for step (2.1.2)
(2.1.3) Receive messages from processors;
(2.1.4) FOR (each received message) DO {
+ unpack this message for v_l;
+ FOR(each downstream v_j of v_l) DO{
IF ($v_j \in V(D^k)$) {$f(v_j)=f(v_j)$-1;}
IF ($f(v_j)$=0) {$\psi=\psi\cup\{v_j\}$;
} ENDDO for step (2.1.4)
} ENDDO for step (2.1)
ENDDO for step (2)

The priority strategy for ordering of vertices decides the operation in step (2.1.2) and step (2.1.4) on how to join a computable vertex into the list. In fact, it is a key factor to affect parallel performance. Some geometrical strategies are presented in [15,17]. Here, we present a new strategy suitable for general model of DAG.

Algorithm 4: *priority strategy using length of shorted path away from processor boundry.*

(1) Let set of vertices $\Lambda=\Lambda_1=\Lambda_0=\phi$;
(2) FOR (each vertex v_i whose out-degree $d^+(v_i)$=0) DO
{ $r(v_i)=Q$, let $\Lambda=\Lambda\cup\{v_i\}$; };
(3) FOR (each vertex v_j is the upstream of at least one vertex in set Λ) DO {
(3.1) IF (there is one downstream vertex of v_j belongs to neighboring processors) { $r(v_j)$=1; } ELSE {$r(v_j)=\min\left(\min_{k\in S(j)}\{r(v_k)\}+1, Q\right)$;};
(3.2) $\Lambda_0=\Lambda\cup\{v_j\}$;
} ENDDO for step (3)
(4) $\Lambda_1=\Lambda_1\cup\Lambda_0$;
(5) IF ($|\Lambda_1|\neq|V(D)|$) {$\Lambda=\Lambda_0$; $\Lambda_0=\phi$; goto step (3)}.

Here, Q denotes the length of critical path, $r(v_i)$ is the priority, $S(j)$ represents the set of indices for downstream vertices of vertex v_j. In fact, the priority is equal to the length of the shortest path away from the processor boundaries. This is coincident with the philosophy of that a vertex should be inserted into the list satisfying that the current vertex located at the head is most welcomed for the release of downstream vertices located at neighboring processors.

Now, we introduce some new concepts for better evaluation of our parallel algorithm. We refer T_1 as the sequential time for the computation of DAG, denote O_P or O_∞ by the optimal or shortest time for parallel execution under the assumption of zero overhead for each message passing using P or unlimited number of processors respectively, and denote T_P by the elapsed time for parallel execution using P processors on parallel computer. Then, we define the ***optimal speedup***, ***algorithm speedup*** or ***realistic speedup*** denoted by S_∞ , S_A and S_P respectively. They are defined as the quotient of O_∞, O_P or T_P over T_1. Obviously, S_∞ represents the parallelism in the digraph, S_A represents the parallelism extracted using our algorithm 3, and S_P represents the realistic performance on parallel computer. Obviously, we wish S_A is close to S_∞ and as well as S_P to S_A.

If vertices have equal weights, S_∞ is equal to the parameter Q of the length of critical path, and S_A can be evaluated by the following algorithm.

Algorithm 5. *Computation of algorithm speedup.*

(1) Y_0 = 0; algorithm 3:(1); X_0 = sum reduction of $|\psi_0|$;
(2) WHILE {$X_0\neq|D|$} DO {
(2.1) IF ($|\psi_k|\neq\phi$) {algorithm 3 : (2.1.1)~(2.1.2) };
(2.2) Update X_0; $Y_0=Y_0+1$;
(2.3) Algorithm 3: (2.1.3)~(2.1.4);}
(3) $S_A=|D|/Y_0$.

In the case such that vertices have dynamic weights, the algorithm speedup is difficult to calculate. However, for many grid-based numerical algorithms including transport equation and convection-dominated fluids, we can assume vertices have equal weights. So, we can use algorithm speedup to evaluate the performance of implementation of algorithm 3.

4. Performance results

In this section, we will list some performance results on the parallel solution of neutron or radiation transport equation using our parallel framework. Particularly, two parallel computers are used for testing. One is a massively distributed memory machine called **GX0** with 600 processors for which the MPI message latency is about 10 microseconds. Another is a distributed shared memory machines called **DX0** with 100 processors for which the MPI message latency is less than 2 microseconds. Each processor has the peak performance of 1.0 Gflops.

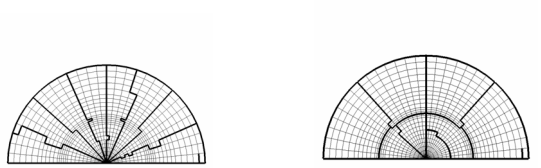

Figure 4. Left: ***GP1***; Right: ***GP3***.

We firstly take the realistic application introduced in [15] for comparison. A 44-groups neutron transport equation is solved using implicitly discrete ordinates stencil S_4 (16 directions) based on discontinuous finite element discretization for cylindrical geometry on two-dimensional unstructured grid. The grid consists of 750 conforming quadrilateral zones. As depicted in figure 1, the super underlying graph is partitioned by two methods called ***GP1*** and ***GP3*** respectively. Method GP1 partition the grid by sorting the radial coordinate and horizontal coordinate, method GP3 partition the grid using the Inertial Kemighan-Lin (IKL) method implemented in Chaco [11].

In table 1, we list the algorithm speedup for parallel downstream sweeping using algorithm 3. The algorithm speedup presented in [15] is also listed for comparison. In essence, the difference mainly comes from the priority strategy. In work [15], geometrical strategies are used, but here, algorithm 4 is used. Results show that the new priority strategy has improved performance by 10% for 64 processors. Compared with the optimal speedup S_∞=168, the algorithm speedup is satisfying.

Table 1. Algorithm speedup for two priority strategies.

#processors	*P*=16		*P*=64	
Strategies	GP1	GP3	GP1	GP3
[15]	*11.9*	*6.8*	*40.7*	*30.9*
Algorithm 4	13.4	7.4	44.3	32.3

In table 2, we list the realistic speedup for 64 processors on machine DX0 and GX0. The results on DX0 show that the small network latency can be ignored for such applications because they are almost the same as algorithm speedup. On machine GX0, the larger network latency still likes the partitioning method GP3 because it has the smallest surface-to-volume ratio though it has the smallest algorithm speedup. However, we can find that the new priority strategy has obviously improved the performance especially on machine GX0.

Table 2. Realistic speedup on 64 processors.

Machines	DX0		GX0	
Strategies	GP1	GP3	GP1	GP3
[15]	*40.5*	*36.0*	*14.2*	*22.2*
Algorithm 4	44.2	31.7	20.2	26.4

Secondly, we consider another realistic application for 24-groups neutron transport equation with implicitly discrete ordinates stencil S_8 (40 directions) on machine GX0. This grid consists of 13214 quadrilateral zones moving in accordance with hydrodynamics. Only method GP3 is used to partition the domain. Table 3 lists the elapsed time (seconds) for each time step and the realistic speedup for parallel downstream sweeping on processors scaling from 1 up to 512. These results show that the parallel framework is moderately scalable.

Table 3. Performance results on machine GX0.

P	*1*	32	64	128	256	512
Time	*2850*	102	54.0	29.8	18.5	14.7
S_P	*1.00*	28.0	52.8	95.6	154	194

Now, we consider the third realistic application for 20-groups radiation transport equation with stencil S_8 (40 directions) on the rectangular Cartesian grid. This grid consists of 128×50 zones. It is noting that these zones vertically move as well as the hydrodynamics, the grid is often nonconforming. An example is depicted in the left of figure 5. Such nonconforming grids will complicate data dependencies for downstream sweeping. We only consider the horizontal stripe decomposition for the super underlying graph. In the right of figure 5, we give an example for two processors. In table 4, we list the algorithm speedup at the second line on processors scaling from 1 up to 128. We can find that the parallel framework is almost the optimal. On machine DX0, we again achieve the realistic speedup close to the algorithm speedup. However, similar cases don't hold on machine GX0. In the worst case, only speedup 13.5 is gained on 64 processors. The worse performance mainly comes from too many message issued in algorithm 3. So, we buffer multiple short messages into a longer message in order to reduce the number of communications in step (2.1.2) of algorithm 3 towards larger network latency. Take the number of short messages buffered together be four, the speedup is listed in the last line of table 4, the performance is significantly improved.

5. Conclusions and Prospects

The model of directed acyclic graph (DAG) presented here is suitable for general description of data dependencies for grid-based numerical algorithm varying from the complex transport equation on the unstructured grid to the simple numerical cores on the rectangular grid. The parallel framework for solution of DAG is moderately scalable on hundreds of processors, so it is also suitable for parallel implementation of such kinds of grid-based algorithms and their realistic applications.

We look forward to more efficient partitioning methods for DAG, more clever techniques for latency tolerance besides from the simplest method presented at the end of last section, and also more efficient methods for cycle detection and tie breaking. Most of all, we prospect more and more realistic applications.

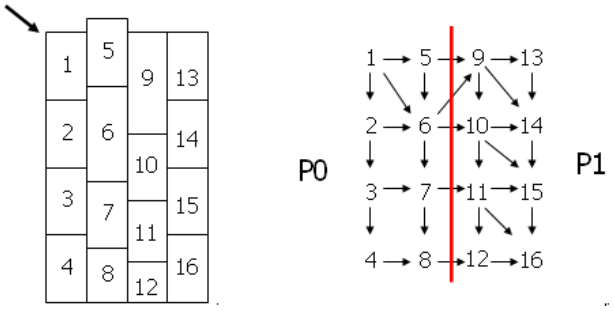

Figure 5. Nonconforming grid and its stripe decomposition.

Table 4. Performance results on machine GX0.

P	4	8	16	32	64	128
A_P	4	8	16	31	62	121
S_P/DX0	4.1	7.7	15.6	29.1	54.6	-
S_P/GX0	3.8	7.4	13.3	19.2	13.5	-
S_P/GX0	3.8	7.7	15.2	27.2	41.6	66.1

In this paper, we only consider DAG for both constantly and linearly downstream sweeping, however, the nonlinear cases should also be addressed where dynamical digraph should be constructed. In fact, our parallel framework can be generalized to be suitable for such digraphs. However, the priority strategy should be modified.

References

1. Ardra: Scalable parallel code system to perform neutron and radiation transport calculations, http://www.llnl.gov/casc/ardra.
2. R.S.Baker, R.E.Alcouffe, Parallel 3-d Sn Performance for MPI on Cray-T3D, in Proc. Joint Intl. Conference on Mathematics Methods and Supercomputing for Nuclear Applications, Vol. 1, pp.377-393, 1997.
3. R.S.Baker, K.R.Koch, An S_n algorithm for the massively parallel CM-200 computer, Nucl.Sci.Eng., 128:312-320,1998.
4. J.Bey, G.Wittum,On the robust and efficient solution

of convection diffusion problems on unstructured grids in two and three space dimensions, Appl. Numer. Math. 23:177-192,1997.

5. R.L.Bowers, J.R.Wilson, Numerical modeling in applied physics and astrophysics, Jones and Bartlett publishers, 1991.
6. J.Dongarra,K.Kennedy,L.Torczon,et.al, Sourcebook of parallel computing, Morgan Kaufmann Publisher, California, 2003.
7. W.Gropp,E.Lusk and A.Skjellum,*Using MPI: portable parallel programming with the message-passing interface*, 2nd edition, MIT Press, MA, 1999.
8. J.L.Gross, J.Yellen, eds., Handbook of Graph Theory, Series: Discrete Mathematics and Its Applications Volume: 25, CRC Press, December, 2003.
9. W.Hackbush, T.Probst, Downwind Gauss-Seidel smoothing for convection dominated problems, Numer. Linear Alg. Appl., 4:85-102,1997.
10. W.Hackbusch, On the feedback vertex set problem for a planar graph, Computing, 58:129-155,1997.
11. B.Hendrickson, R.Leland, The Chaco user's guide: Version 2.0, Technical Report, SAND94 -2692,Sandia National Laboratories, Albuquerque, NM,1994.
12. C.Koutsougeras, C.A.Papachristou, Data flow graph partitioning to reduce communication cost, in Proc. of 19th annual workshop on mircoprogramming, pages 82-91,1986.
13. E.E.Lewis, W.F.Miller, Computational Methods of Neutron Transport, John Wiley & Sons Publisher, 1984.
14. Z.Mo,X.Li, Parallel multigrid computation for anisotropic diffusion problems, Chinese J. Numer. Math. & Appl., 20:31-43,1998.
15. Z.Mo,L.Fu, Parallel flux sweeping algorithm for neutron transport on unstructured grid, Journal of Supercomputing, 30(1):5-17,2004.
16. Z.Mo, Concatenation algorithm for parallel numerical simulation of hydrodynamics coupled with neutron transport, Intern. J. Parallel Programming, 33:57-71, 2005.
17. S.Plimpton, B.Hendrickson, S.Burns, W.McLendon, Parallel algorithms for radiation transport on unstructured grids, in Proc. of SuperComputing'2000, Dallas, Nov.4-10, 2000.
18. K.Schloegel,G.Karypis,V.Kumar, Graph partitioning techniques for high performance scientific simulations, Chapter 18, Sourcebook of Parallel Computing, edited by J.Dongarra, K.Kennedy, L.Torczon, et.al., Morgan Kaufmann Publisher, California, 2003.
19. SWEEP3D:3D Discrete Ordinates Neutron Transport Benchmark Codes, http://www.llnl.gov/asci_benchmarks /asci/limited/sweep3d/sweep3d_readme.html.
20. L.G.Valiant, A bridging model for parallel computation, Communications of the ACM, 33(8): 108-111,1990.
21. F.Wang, J.Xu, A cross-wind strip block iterative method for convection-dominated problems, SIAM J. Comput., 21:646-665,1999.
22. T.A.Wareing, J.M.McGhee, J.E.Morel and S.D.Pautz, Discontinuous Finite Element Sn Methods on 3-D unstructured Grids, in Proceeding of International Conference on Mathematics and Computation, Reactor Physics and Environment Analysis in Nuclear Applications, Madrid,1999.
23. B.Wilkinson, M.Allen, Parallel programming: techniques and applications using networked works -tations and parallel computers, Prentice Hall, 2002.
24. L.Zhang, Pipelining parallelization of a cluster of recursive formulae, Chinese J. Numer. Math. & Comput. Appl., 20(3):184-191,1999.