SIAM J. SCI. COMPUT.
Vol. 31, No. 5, pp. 3626–3642
© 2009 Society for Industrial and Applied Mathematics

SCALABLE HEURISTIC ALGORITHMS FOR THE PARALLEL EXECUTION OF DATA FLOW ACYCLIC DIGRAPHS*

ZEYAO MO†, AIQING ZHANG‡, AND GABRIEL WITTUM§

Abstract. Data flow acyclic directed graphs (digraphs) can be applied to accurately describe the data dependency for a wide range of grid-based scientific computing applications ranging from numerical algebra to realistic applications of radiation or neutron transport. The parallel computing of these applications is equivalent to the parallel execution of digraphs. This paper presents a framework of scalable heuristic algorithms for the parallel execution of digraphs. This framework consists of three components: the heuristic partitioning method of a digraph, the parallel sweeping algorithm for a partitioned digraph, and the heuristic strategy for vertex scheduling and vertex packing. Evaluation rules of heuristic algorithms are presented for better theoretical understanding and performance optimization. Parallel benchmarks for the multigroup neutron or radiation S_n transport using processors from 100 to 2048 on two massively parallel machines show that these heuristic algorithms scale well.

Key words. parallel algorithm, acyclic digraph, grid-based scientific computing, Sn transport

AMS subject classifications. 68W10, 68R10

DOI. 10.1137/050634554

1. Introduction.

In the past two decades, parallel algorithms have been designed for a wide range of grid-based scientific computing [9], [11], [23], especially for the solution of partial differential equations. The grid is the discrete formulation of the computational geometry and consists of a set of disjoint polyhedrons called zones [6], [22]. The data dependency among neighboring zones can be accurately depicted by the undirected graphs [13] during the period of parallel execution. After these graphs are partitioned among processors, parallel algorithms can be designed using the concept of `supersteps` as defined in the bulk synchronous parallel (BSP) programming model [5], [34].

However, the BSP model is not suitable for another type of grid-based scientific computing where zones should be calculated in the behavior of data flow [19] and the data dependency must be depicted by the directed graph (digraph) [13]. For example, for the high energy density plasma physics [8], particle transport should be coupled to shock hydrodynamics for the material responses which can be mathematically well modeled in the form of the multigroup discrete ordinates (S_n *transport*) of the Boltzmann equations [6]

$$\frac{1}{v_g}\frac{\partial I_{mg}}{\partial t} + \Omega_m \bullet \nabla I_{mg} = R(I,t), \qquad m = 1,2,\ldots,M; g = 1,2,\ldots,G. \tag{1.1}$$

*Received by the editors June 27, 2005; accepted for publication (in revised form) July 9, 2009; published electronically September 25, 2009. This work was carried out under the auspices of the Chinese NSF (90718029, 60533020, 10701015, 60603050) and the National Basic Key Research Special Fund (2005CB321702).
http://www.siam.org/journals/sisc/31-5/63455.html

†Corresponding author. Institute of Applied Physics and Computational Mathematics, Laboratory of Computational Physics, Beijing, 100088, China (zeyao_mo@iapcm.ac.cn).
‡Institute of Applied Physics and Computational Mathematics, Laboratory of Computational Physics, Beijing, 100088, China (Aiqing_Zhang@iapcm.ac.cn).
§Technical Simulation Group, IWR, University of Heidelberg, D-69120 Heidelberg, Germany and Goethe Center for Scientific Computing, University of Frankfurt, D-60325 Frankfurt, Germany (wittum@gcsc.uni-frankfurt.de).

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Here, $I_{mg} \equiv I_{mg}(x, y, z, t)$ represents the particle flux in the gth energy group and the mth discrete ordinate (S_n) angle. Ω_m denotes the unit vector of the mth angle. M is the number of angles, and G is the number of energy groups. $\Omega_m \bullet \bigtriangledown I_{mg}$ is the S_n transport term. $R(I, t)$ is the source term of the form

$$R(I,t) \equiv S_{mg} - (\sigma_A + \sigma_S) I_{mg} + \sum_{\hat{m}} \sum_{\hat{g}} w_m \sigma_{S,\hat{m}\to m,\hat{g}\to g} I_{\hat{m}\hat{g}}, \tag{1.2}$$

where S_{mg} represents the flux source of material, σ_A and σ_S denote the coefficients of absorption and scattering, respectively, the last term on the right-hand side of (1.2) represents the flux emitting from itself and received from others, and w_m is the constant unit weight of angle Ω_m.

Usually, for the numerical solution of the S_n transport equations (1.1), the fully implicit stencil is used to achieve an acceptable time step size. It can be written as

$$\frac{1}{v_g}\frac{I_{mg}^{l+1} - I_{mg}^{l}}{\triangle t} + \Omega_m \bullet \nabla I_{mg}^{l+1} = R(I^{l+1}, t^{l+1}), \quad m = 1, 2, \ldots, M; g = 1, 2, \ldots, G \tag{1.3}$$

and can be solved by the source iteration method below.

ALGORITHM 1.1. *One source iteration for a time step.*

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

(2) For ($m = 1, 2, \ldots, M$) DO {

$$I_{mg}^{l,v+1} + v_g \triangle t \Omega_m \bullet \nabla I_{mg}^{l,v+1} = I_{mg}^{l,v} + v_g \triangle t R(I^{l,v}, t^{l+1}), \qquad g = 1, 2, \ldots, G \tag{1.4}$$

} ENDFOR($m = 1, 2, \ldots, M$).

(3) Update the source term (1.2) using $I_{mg}^{l,v+1}$ by some acceleration techniques [10].

(4) Source term converges? If yes, $I_{mg}^{l+1} = I_{mg}^{l,v+1}$, stop; otherwise $v = v + 1$; go to step (2).

The discretization of the transport term often depends on the type of grid. For example, the diamond finite difference methods (DFDM) [22] are often used on a rectangular grid, while the finite volume methods (FVM) or the discontinuous Galerkin finite element methods (DG-FEM) [36] are often used on a deforming structured or unstructured grid. No matter which method is used, a lower triangular sparse linear system will be formed and can be accurately solved if we use the well-known downstream sweeping in accordance with the data dependency given by angle Ω_m. So, the data flow directed graph (digraph) should be used to depict the data dependency.

On the left in Figure 1, 16 quadrilateral zones of an unstructured grid are given. The downstream sweeping for an angle in the upper left corner leads to the following sequence:

```
{1,6}->{2,11}->{3}->{4}->{5,7}->{8}->{9,12}->{10,13,14}->{15}->{16},
```

where the bracket means that the zones included can be concurrently swept and the arrow means that its tail bracket must wait for the head. This sequence leads to an acyclic digraph shown on the right in the same figure, where a vertex denotes a zone and an arc shows the data dependency of two neighboring zones.

The S_n transport is computationally challenging because O(10^{10}) zones, groups, and angles are often required. Usually, massively parallel computers (MPP) are used to solve such problems and message passing interface (MPI) [12] is used to realize the parallel downstream sweeping.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

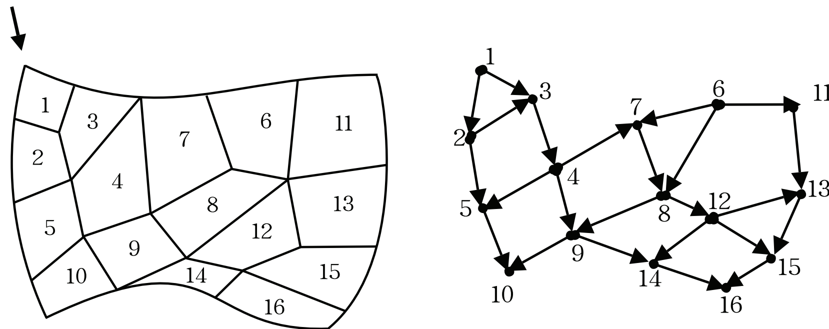

Fig. 1. *Left: An unstructured grid and a sweeping angle. Right: An acyclic digraph shows the data dependency.*

Many parallel algorithms have been proposed for the downstream sweeping using the well-known pipelining techniques [37]. On a rectangular grid, the MPI realization is natural because pipelines can be well defined before sweeping. The first is tested by Baker and Alcouffe on Cray-T3D [2] and by Baker and Koch on CM-200 [3], the second is given in the ASCI SWEEP3D benchmark code [33], the third is designed for four thousand processors [1]. On a deforming structured or unstructured grid, however, the MPI realizations are more complicated because pipelines are hard to predefine owing to the irregular data flow. Plimpton et al. [29] first addressed such irregular problems in Cartesian geometry, where all angles are independent of each other. Mo and Fu [27] extended that work to problems in cylindrical geometry, where data dependency exists among angles. Mo in [28] studies the parallel concatenation between radiation hydrodynamics and S_n transport.

In addition to S_n transport, pipelining techniques are also essential for many subroutines of numerical algebra arising from grid-based scientific computing. On the rectangular grids, Mo and Li [26] studied the robust downstream smoothers of multigrid solvers for convection diffusion equations. Brown Falgout, and Jones [7], Zuo and Mo [39], and Zhang [38] solved a group of tridiagonal sparse linear systems, respectively, arising from the semicoarsening smoother of an algebraic multigrid solver (AMG), the alternative direction implicit discretization (ADI) of multigroup heat diffusion equations, and the compact finite difference discretization of incompressible Navier–Stokes equations. Hackbush and Wittum [14], Mo and Li [25], and Saad [30] addressed the Gauss–Seidel or ILU smoothers.

On a deforming structured or unstructured grid, parallel pipelining techniques are seldom reported in the literature. However, there are some articles which are tightly related. Bey and Wittum [4] presented a robust multigrid solver by reordering zones along the downstream direction of flow, Hackbush and Probst [15] proposed a downwind block Gauss–Seidel smoother for the cyclic data dependency, and Wang and Xu [35] presented a crosswind strip block iterative method.

Summarily, the parallel realization of the pipelining techniques has been recently addressed and is widely required for grid-based scientific computing; however, they are different from the types of grids or applications. Is it possible to design a framework of parallel algorithms independent of grids or applications? This paper tries to find a solution.

First, computing models are crucial for the framework of parallel algorithms, and so acyclic digraphs are presented to accurately describe the data flow of downstream sweeping on the structured or unstructured grids in section 2. Based on these models,

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

the parallel downstream sweeping is transformed into the parallel execution of acyclic digraphs. Unfortunately, the optimal parallel execution of an acyclic digraph is NP-hard; only heuristic algorithms are possible. In section 3, we give a framework of scalable heuristic parallel algorithms. In section 4, numerical benchmarks for typical neutron or radiation S_n transport are performed using processors from 100 to 2048 on two parallel machines. The performance results show that these algorithms scale well. Lastly, a more efficient digraph partitioning method is proposed.

2. Acyclic digraph as the computing model.

We use the basic terminology and notation introduced in the monograph [18]. A digraph D consists of a nonempty 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 set of vertices and the set of arcs of D, respectively. The size of D, denoted by $|D|$, is the number of vertices.

Without loss of generality, we omit the subscript g and the superscript l and rewrite the downstream sweeping in (1.4) as

$$I_m^{v+1} + a_m \nabla \bullet I_m^{v+1} = Q_m, \quad m = 1, \ldots, M, \tag{2.1}$$

where a_m is the downstream coefficient and can be a constant, a linear function, or a nonlinear function, and formula (2.1) can be called the constant, linear, or nonlinear downstream sweeping, respectively. For the constant and the linear cases, the digraphs can be constructed before sweeping; however, for the nonlinear case, the digraph needs to be simultaneously constructed while the sweeping is performed. In this paper, we mainly consider the constant and the linear sweeping.

2.1. Acyclic digraph for the constant downstream sweeping.

Take S_n transport as an example. Consider the digraph $D \equiv D(\Omega_m)$ constructed from angle Ω_m across the computational grid. The grid consists of N zones denoted by $\{z_i | i = 1, 2, \ldots, N\}$. Assign $V(D) = \{v_i | i = 1, 2, \ldots, N\}$, where the vertex v_i represents zone z_i and its weight is the calculation cost of zone z_i.

In order to construct arcs, we introduce some basic concepts. The face of a zone is called the inflow (outflow) face if and only if the inner product $\theta_{mi} = \Omega_m \bullet \vec{\chi}_i$ is less (larger) than zero. Here, $\vec{\chi}_i$ is the outer normal vector to the face. In Figure 2, the left depicts a deforming quadrilateral zone Z; $\partial^- Z$ and $\partial^+ Z$, respectively, denote the inflow and outflow faces across which the flow enters or exits. As shown on the right in Figure 2, zone Z is located at the downstream and will not be computed until zone Z_0 and zone Z_1 have been computed.

An arc $e_{i \to j} = (v_i, v_j)$ is constructed if zone z_j depends on zone z_i. Assign the weight of arc $e_{i \to j}$ the data volume of vertex v_i on which vertex v_j depends.

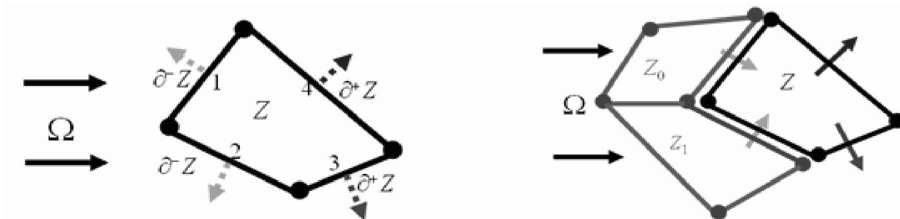

Fig. 2. *Left: Outer normal vectors of a zone. Right: Data dependency.*

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

After the digraph is constructed, the serial execution of a digraph can be described by Algorithm 2.1 below, where we say that a vertex is *computable* if all its upstream vertices have been calculated. List Ψ maintains all currently computable vertices.

ALGORITHM 2.1. *The serial execution of a digraph. ϕ denotes the null set.*

(1) FOR $(i = 1, 2, \ldots, N)$ 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 list Ψ; $\Psi_0 = \Psi_0 \cup v_k, \Psi = \Psi \setminus v_k$.
    (2.2) FOR (each downstream neighborhood v_j of vertex v_k) DO $\{$
            (2.2.1) $f(v_j) = f(v_j) - 1$;
            (2.2.2) IF $(f(v_j) = 0)$ $\{$Insert v_j into list Ψ; $\}$
            (2.2.3) Transfer vertex v_k to vertex v_j.
        $\}$ ENDDO for step (2.2).
    $\}$ ENDDO for step (2).

Obviously, a digraph is computable if and only if $|D|$ iterations are executed in step (2). It is also obvious to have the following conclusion.

THEOREM 2.1. *A digraph is computable if and only if it has no cycle.*

From Theorem 2.1, we have the following properties.

Property 1. A digraph is computable on a rectangular grid.

Property 2. On a two-dimensional grid, a digraph is acyclic if and only if each zone is convex.

Proof. Assume $\Theta = v_{i_1} v_{i_2} \cdots v_{i_m} v_{i_1}$ is a cycle and $a_{i_1}, a_{i_2}, \ldots, a_{i_m}$ are m arcs of this cycle. Denote by s_{i_k} the outflow side of zone v_{i_k} corresponding to arc a_{i_k} and by h_{i_k} the outer normal vector. The projection of vector h_{i_k} to the sweeping angle must be larger than zero. So, as shown on the left in Figure 3, a cycle means that there is one zone whose inflow face and outflow face intersect and the intersection inner corner is larger than π. This contradicts the convex property.

In the two-dimensional case, concave zones are seldom used for the numerical discretization of realistic applications, and so no cycle exists. However, in the three-dimensional case, a cycle may exist even if each zone is convex. In fact, as shown on the right in Figure 3, a ring cut by a series of planes along the sweeping angle will result in a cycle.

Many algorithms are proposed to detect and break cycles in a digraph. Jørgen and Gregory [18] present a general detection algorithm, Hackbush and Probst [15], [16] present an efficient detection and breaking algorithm for convection-dominant flows, and Plimpton et al. [29] give a strategy on unstructured grids. In this paper, we do not address such algorithms and assume the digraphs are always acyclic. ∎

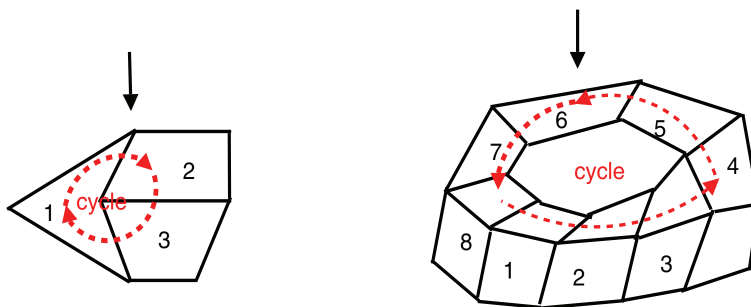

FIG. 3. *Left: Three zones form a cycle. Right: Eight hexahedrons form a cycle.*

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

2.2. Acyclic digraph for multiple constant sweeping. For the multiple constant sweeping, the digraph can be constructed in general. A vertex is defined as a zone-angle pair such that $u_{im} = (z_i, \Omega_m)$, where z_i is the zone and Ω_m is the angle.

In the Cartesian geometry, an arc is defined as a pair of vertices such that $e_{im \to op} = (u_{im}, u_{op})$ if the following conditions are satisfied:

C_1 *$m = p$ and zone z_o is the downstream of zone z_i for angle Ω_m.*

C_2 *$i = o$, $m < p$, and Ω_p depends on Ω_m in the cylindrically or spherically symmetric geometry* [22], [27].

Property 3. The digraph for the multiple constant sweeping is acyclic if and only if each subdigraph for a single sweeping is acyclic.

Proof. If the digraph is acyclic, then, naturally, each subdigraph is acyclic. In return, if each subdigraph is acyclic, then the arcs introduced by condition C_1 cannot form a cycle, the arcs introduced by condition C_2 point only to the larger subscript angle from the smaller subscript angle, and new cycles will never be introduced. ☐

2.3. Acyclic digraph for the linear downstream sweeping. The linear downstream sweeping usually acts as a robust solver for the linear convection-dominant flows. For example, the downwind Gauss–Seidel smoother [15] is such a typical solver for which each iteration can be transformed to the execution of a digraph.

2.4. The model of an acyclic digraph. Given a digraph D, denote by $\underline{D}$ the underlying graph, by $\Re(D)$ the multidigraph, and by $U(D)$ the underlying supergraph. $\Re(D)$ comes from that all vertices $\{u_{im} : m = 1, 2, \ldots, M\}$ in D are substituted by a supervertex u_i, and $U(D)$ comes from that all parallel edges connecting two zones in $\underline{\Re(D)}$ are substituted by a superedge. Obviously, $U(D)$ depicts the geometric connectivity of zones and can be simultaneously constructed in the same manner as the digraph. In the case of a single sweeping, $U(D)$ is the same as $\underline{D}$.

In summary, consider the following model of an acyclic digraph D:

$$D = (V(D), A(D), U(D)), \tag{2.2}$$

where $V(D)$ and $A(D)$ are the set of vertices and arcs, respectively, and $U(D)$ is the underlying supergraph.

This model accurately describes the geometrical neighbors of a zone and the data dependency for acyclic sweeping. Obviously, digraphs are universal and are independent of grid-based applications after they are constructed. Of course, such models can be easily built on structured or unstructured grids.

3. Scalable heuristic algorithms for the parallel execution of an acyclic digraph. The parallel algorithms for the execution of a digraph are equivalent to the parallel tasks scheduling and are typically NP-hard [18]. In this section, we present a framework of scalable heuristic algorithms. It consists of three components. The first is the digraph partitioning methods spawning the vertices among processors, the second is the parallel sweeping for the execution of a digraph, and the third is the priority strategies for vertices scheduling and vertices packing.

3.1. Digraph partitioning methods. If the grid is rectangular, many authors [1], [2], [3] have shown that one-dimensional strip decomposition in two-dimensional geometry or two-dimensional strip decomposition in three-dimensional geometry is most efficient. Therefore, the framework in this paper simply uses strip decomposition.

If the grid's structure is deformed or the grid is unstructured, two types of methods exist. One method uses the well-known graph partitioning method [9], [31] for

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

the underlying supergraph and assigns the processor all vertices defined on the same zone, and another method partitions the digraph using the knowledge of downstream sweeping. The first always has minimal communication surface and better load balance efficiency; the second usually has the shortest sweeping time. In this section, only the first type is used. In section 5, the second type will be introduced.

3.2. Parallel sweeping algorithms. Assume that the digraph D is decomposed into P subdigraphs denoted by $\{D^k\}_{k=1}^P$. Each subdigraph is assigned to an individual processor. Then, the parallel sweeping algorithm for the execution of a partitioned digraphs can be written as follows.

ALGORITHM 3.1. *The parallel downstream sweeping of a partitioned acyclic digraph.* $\mathbb{I}_{j\to k}$ *represents the operator inserting vertex* v_j *into the vertices list* Ψ_k*, and* $\mathbb{S}_{i\to j}$ *represents the message passing operator by which vertex* v_i *is sent to the processor owning vertex* v_j*. Other notation is the same as that in Algorithm* 2.1.

FOR $(k = 1, 2, \ldots, P)$ processor p_k DO in parallel $\{$
  (1) $f(v_i) = d^-(v_i)\ \forall v_i \in V(D^k)$; 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 vertex v_i of list Ψ_k,
        let $\Psi_0 = \Psi_0 \cup \{v_i\}, \Psi_k = \Psi_k \setminus \{v_i\}$.
      (2.1.2) FOR (*each downstream vertex* v_j *of vertex* v_i) DO $\{$
          IF $(v_j \in V(D^k))$ $\{$
            $f(v_j) = f(v_j) - 1$;
            IF $(f(v_j) == 0)$ $\mathbb{I}_{j\to k}$;
          $\}$ ELSE $\{$ $\mathbb{S}_{i\to j}$; $\}$
        $\}$ ENDDO for step (2.1.2).
      (2.1.3) Receive all messages from other processors.
      (2.1.4) FOR (*each received message*) DO $\{$
          unpack this message to get vertex v_i.
          FOR (*each downstream vertex* $v_j \in D^k$ *of* v_i) DO $\{$
            $f(v_j) = f(v_j) - 1$; IF $(f(v_j) == 0)$ $\mathbb{I}_{j\to k}$;
          $\}$
        $\}$ ENDDO for step (2.1.4).
    $\}$ ENDDO for step (2.1).
  $\}$ ENDDO for step (2).

Theoretically, each processor does not need to wait for other processors until no computable vertices are available. So, the insert strategies for operator $\mathbb{I}_{j\to k}$ will dominate the performance if the communication overheads of message passing operator $\mathbb{S}_{i\to j}$ are very small. Unfortunately, the optimal insert strategy is the same problem as the optimal vertices scheduling and is NP-hard; only heuristic algorithms are available.

The message passing operator $\mathbb{S}_{i\to j}$ may issue many short messages. If MPI latency is large on parallel machines, the performance will be greatly damaged. So, some optimization strategies should also be used.

3.3. Heuristic priority strategies and packing strategies. The usual strategy for operator $\mathbb{I}_{j\to k}$ assigns each vertex v_j a priority and then inserts each vertex into the computable list Ψ_k according to its priority. In order to suppress the number

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

of operators $\mathbb{S}_{i\to j}$, several messages should be grouped. This section introduces these strategies.

3.3.1. Heuristic priority strategies. It is obvious that, for the strip decompositions on a rectangular grid, the optimal priority strategy assigns each vertex its projection coordinates in the direction of strip decomposition. However, for the partitioning of a digraph, the optimal priority strategy is NP-hard, so heuristic strategies should be used.

Some application-specific priority strategies on unstructured grids are given in [27], [29]. Here, we present a new priority strategy suitable for the acyclic digraph which is independent of applications.

ALGORITHM 3.2. *Locally shortest path priority strategy. Q is the length of critical path in digraph, and $S(j)$ represents the set of subscript indices for direct successors of vertex v_j.*

FOR $(k = 1, 2, \dots, P)$ processor p_k DO in parallel $\{$
  (1) Assume $\wedge = \wedge_1 = \wedge_0 = \phi$.
  (2) FOR (each vertex v_i with out-degree $d^+(v_i) = 0$) DO $\{$
      $r(v_i) = Q; \wedge = \wedge \cup \{v_i\};\ \}$
  (3) FOR (each vertex v_j dominates at least one vertex of $\wedge$) DO $\{$
    (3.1) IF (one upstream of v_j is spawned to other processor) $\{$
        $r(v_j) = 1; \}$
      ELSE $\{r(v_j) = \min\left(\min_{k\in S(j)}\{r(v_k)\} + 1, Q\right); \}$
    (3.2) $\wedge_0 = \wedge_0 \cup \{v_j\}$;
    ENDDO for step (3).
  (4) $\wedge_1 = \wedge_1 \cup \wedge_0$;
  (5) IF $(|\wedge_1| \neq |V(D^k)|)$ $\{\wedge = \wedge_0; \wedge_0 = \phi;$ goto step $(3); \}$ ELSE $\{$*stop*$; \}$.
ENDFOR $(k = 1, 2, \dots, P)$

In fact, the priority of a vertex given by Algorithm 3.2 is equal to the length of the shortest path away from the processor boundaries. This means that a vertex required by other processors should be superior to other vertices.

3.3.2. Heuristic packing strategies for message passing. In this section, we introduce the most simple packing strategy described in Algorithm 3.3. We rewrite step (2.1.2) of Algorithm 3.1 as follows.

ALGORITHM 3.3. *Heuristic packing strategy for message passing. N_b is a parameter.*

(2.1.2) FOR (*each downstream vertex v_j of vertex v_i*) DO $\{$
    IF $(v_j \in V(D^k))$ $\{$
      $f(v_j) = f(v_j) - 1$;
      IF $(f(v_j) = 0)$ $\{\ \mathbb{I}_{j\to k};\ \}$
    ELSE $\{$
      buffer message for vertex v_j for destination processor;
      IF (N_b messages are buffered or no new messages)$\{$
        send messages buffered to destination processor;$\}$
    ENDIF
  ENDDO for step (2.1.2).

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Parameter N_b depends on the MPI latency on a parallel computer. Usually, the larger the latency is, the larger N_b should be.

3.4. Scalability evaluation of parallel sweeping. Let T_1 be the sequential execution time of an acyclic digraph. Denote by O_P and by O_∞ the ideal execution time, respectively, using P and an infinite number of processors provided that the communication overheads of message passing operators $\mathbb{S}_{i\to j}$ are zero. Denote by T_P the elapsed time for the execution using P processors on a parallel computer.

Let the *Optimal Speedup* S_∞, the *Algorithm Speedup* S_A, and the *Realistic Speedup* S_P be the quotient of O_∞, O_P, and T_P over T_1, respectively. S_∞ represents the maximally potential parallelism of the digraph, S_A is the ideal speedup of the parallel algorithm, and S_P is the realistic performance on a parallel computer. Of course, S_A and S_P are, respectively, desired to be close to S_∞ and S_A.

If the weights of vertices are equal to each other, then $S_\infty = |V(D)|/Q$, where Q is the length of the critical path; S_A can be calculated using Algorithm 3.4 below.

ALGORITHM 3.4. *Calculate the algorithm speedup of Algorithm* 3.1.

(1) $Y_0 = 0$; `execute step` (1) `of Algorithm 3.1;`
    $X_0 =$ `sum of` $|\Psi_0|$ `over processors.`
(2) `WHILE` $\{X_0 \neq |V(D)|\}$ `DO {`
  (2.1) `IF` $(\Psi_k \neq \phi)$ `{execute steps` (2.1.1)–(2.1.2) `of Algorithm 3.1;}`
  (2.2) `Update` X_0; $Y_0 = Y_0 + 1$;
  (2.3) `Execute steps` (2.1.3)–(2.1.4) `of Algorithm 3.1;`
   }
(3) $S_A = |V(D)|/Y_0$.

4. Performance results. In this section, the heuristic algorithms are benchmarked for the parallel solution of the multigroup neutron or radiation transport equations. Two parallel computers are used. One is a massively distributed memory machine named **GX0** with thousands of microprocessors interconnected by a high performance network for which the MPI message latency is equal to 5 microseconds or so. Another is a distributed shared memory machine named **DX0** with hundreds of processors; its MPI message latency is equal to 2 microseconds. For these two machines, the peak performance of microprocessors is equal to 6.4 GFlops.

4.1. Benchmarks for neutron transport applications. We first take the neutron transport applications introduced in [27] for comparison. 24 groups of neutron transport equations are solved by the implicitly S_4 discontinuous finite element methods on a two-dimensional unstructured grid in the cylindrically symmetric geometry, and 16 angles are simultaneously swept.

The grid consists of 3600 quadrilateral zones. As shown in Figure 4, three methods, i.e., **GP1, GP2, and GP3**, are used to partition the underlying supergraph. **GP1** and **GP2** partition the grid by sorting the radial and the horizontal coordinates, respectively, and **GP3** partitions the grid using the inertial Kemighan–Lin (IKL) method implemented in the tool Chaco [17].

In Table 1, the third row lists the algorithm speedup S_A using Algorithm 3.2, and the last row lists a similar speedup using the strategy in [27]. These results show that Algorithm 3.2 is superior by 10% when 256 processors are used and is satisfactory in comparison to the optimal speedup $S_\infty = 318$.

Table 2 lists the realistic speedup for 64 processors on **DX0** and **GX0**. The speedup on the machine **DX0** is almost the same as the algorithm speedup listed in Table 1. This shows that the MPI message latency of the machine **DX0** is small

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

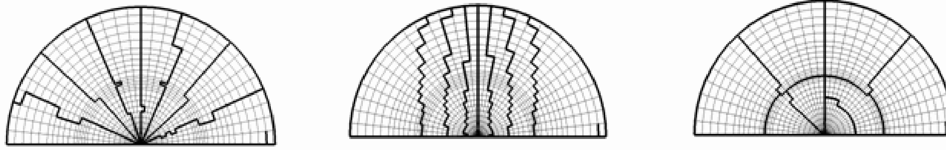

FIG. 4. *A quadrilateral grid is partitioned into eight subdomains using three methods, i.e.,* **GP**1*,* **GP**2*, and* **GP**3 *from the left to the right.*

TABLE 1
The algorithm speedup S_A of Algorithm 3.1*. P is the number of processors.*

P	16			64			256		
Strategies	GP1	GP2	GP3	GP1	GP2	GP3	GP1	GP2	GP3
Algorithm 3.2	14.9	13.1	9.4	58.3	59.5	38.3	159.2	157.4	105.4
In [27]	13.1	11.3	8.6	48.3	51.5	36.3	140.2	141.4	94.2

enough for the parallel execution of Algorithm 3.1. However, the speedup on the machine **GX0** is obviously smaller than that on the machine **DX0** because of its larger MPI message latency.

In order to reduce the infection of the MPI latency on the machine **GX0**, we use the heuristic packing strategy given in Algorithm 3.3. Table 3 lists the speedup using 16, 64, and 256 processors, respectively, where the parameter $N_b = 4$. These results are close to the algorithm speedup listed in Table 1.

4.2. Benchmarks for radiation transport applications. Consider another application of 20 groups of radiation transport equations. It is implicitly discretized using S_8 or 48 angles and the finite difference stencils. The grid consists of 512×50 horizontally nonconforming and rectangular zones with Cartesian geometry shown on the left in Figure 5. Such nonconforming dependency complicates the downstream sweeping digraph as shown on the right in Figure 5. The horizontal strip decomposition is used to partition the underlying supergraph. The algorithm speedup of Algorithm 3.1 using the locally shortest priority strategy is given in Table 4 when scaling the number of processors from 4, 16, 64, and 128 to 256. The last row in this table lists the realistic speedup on the machine **GX0** using the heuristic packing strategy.

4.3. Benchmarks for large scale simulations. For the above benchmarks, 20 groups obviously improve the calculation granularity of each vertex. If we continuously increase the number of groups, more scalable performance will be achieved. However, it is embarrassing for parallel scalability. So, we increase only the number of zones or angles while scaling up the number of processors.

We consider again the neutron transport applications discussed in subsection 4.1. Increasing the number of zones from 3600 to 57600, the second row in Table 5 lists the elapsed time per 100 time steps of Algorithm 3.1 using both the locally shortest priority strategy and the heuristic packing strategy ($N_b = 4$) on the machine **GX0**. Moreover, **GP1** and **GP3** are used to partition the grid when $P \leq 64$ and $P \geq 128$, respectively. The next two rows list the realistic speedup and efficiency. It is obvious that good scalability is achieved for $P \leq 512$.

Scale up the number of angles from 16 (S_4) to 48(S_8). The three bottom rows in Table 5 list the elapsed time and realistic performance. Good scalability is achieved

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

TABLE 2
The realistic speedup S_P of Algorithm 3.1 using 64 processors.

Parallel machines	$DX0$			$GX0$		
Priority strategies	GP1	GP2	GP3	GP1	GP2	GP3
Algorithm 3.2	57.6	57.2	38.1	39.8	38.2	30.3
Strategy in [27]	47.2	49.7	33.4	38.6	38.5	24.7

TABLE 3
The realistic speedup S_P on machine **GX0** *using the heuristic packing strategy.*

P	16			64			256		
Realistic speedup	13.2	12.4	7.4	51.3	52.1	34.3	135.2	133.4	86.5

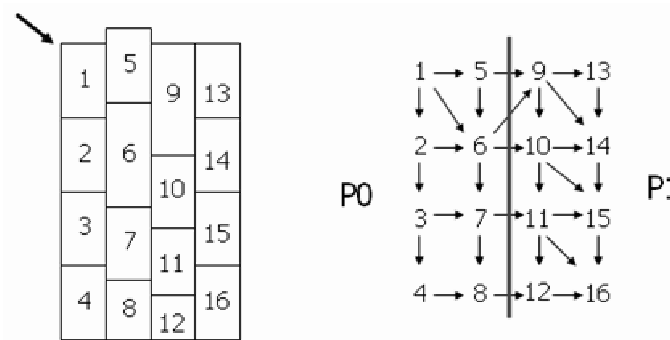

FIG. 5. *Left: A horizontally nonconforming rectangular grid. Right: Digraph; P_0 and P_1 represent two processors.*

for $P \leq 2048$. Moreover, the superlinear speedup occurs around $P = 256$ because of the reduction of memory size and the increment of cache hit ratio per processor.

5. Heuristic partitioning methods for digraphs. The grid partitioning methods used above depend on the grid geometry or the undirected underlying supergraph; they may introduce worse decompositions which are not suitable for the downstream sweeping. More efficient partitioning methods may be designed if the data flow of a digraph is utilized.

Some authors [19], [20], [21], [32] have addressed similar problems. Particularly, Lee, Hurson, and Feng [20] present an efficient method by which vertices are grouped into horizontal layers and then are partitioned into vertical strips. In each layer, vertices are independent of each other and can be executed in parallel, and in each strip, vertices belonging to different layers should be executed in sequence. However, these works often focus on the shortest execution of data flow of an acyclic digraph on shared memory parallel computers without consideration of the load balancing and geometry connectivity of zones. These two factors are very important for large scale grid-based scientific computing. In fact, for the multigroup neutron or radiation transport equations, Algorithm 3.1 covers only 40% of the CPU time; load balancing and geometry connectivity are crucial for the other 60%.

In this section, we present a heuristic partitioning method for digraphs toward algorithm speedup without serious loss of load balancing efficiency and loss of geometry connectivity. In this section, only one sweeping angle is considered.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Table 4
The algorithm and the realistic speedup on the machine **GX0**.

P	4	16	64	128	256
Algorithm speedup	4.0	16.0	62.0	124.2	248.3
Realistic speedup	3.8	15.2	54.6	82.3	105.2

Table 5
The realistic performance for large scale simulation on machine **GX0**.

	P	32	64	128	256	512	1024	2048
S_4	Elapsed time (sec.)	531	270	169	117	78	65	-
	Realistic speedup	1.0	1.97	3.17	5.87	6.87	8.25	-
	Realistic efficiency	1.0	1.00	0.79	0.73	0.43	0.26	-
S_8	Elapsed time (sec.)	1781	908	413	193	103	58	41
	Realistic speedup	1.0	1.87	4.31	9.22	17.24	30.70	43.41
	Realistic efficiency	1.0	0.94	1.08	1.15	1.08	0.96	0.68

Denote by

$$\eta_P = \frac{\sum_{k=1}^{P} W(D^k)}{P \times \max_{k=1}^{P}\{W(D^k)\}} \quad \text{and} \quad \gamma_P = \max_{k=1}^{P}\{\vartheta(D^k)\} \tag{5.1}$$

the load balancing efficiency and the communication complexity, respectively. Here, $W(D^k)$ is the sum of vertices weights and $\vartheta(D^k)$ is the number of arcs around subdigraph D^k. Obviously, the three partitioning methods mentioned above have the optimal balancing efficiency and the method **GP3** has the smallest communication complexity.

Given a vertex u_i, we refer to its *dominator processors* as those processors which own at least one of its dominator vertices, and its *optimal executable time* $o(u_i)$ as the ideal earliest executable time without consideration of MPI message passing overheads, and its *distribution expectation*

$$\delta(u_i) = \begin{cases} 1, & d^-(v_j) = 0, \\ \sum_{j \in S(i)} \frac{\delta(v_j)}{d^-(v_j)} + 1, & d^-(v_j) > 0, \end{cases} \tag{5.2}$$

as the quotient of the sum of distribution expectation of all its direct successors over the in-degree. Here, $S(i)$ represents the set of direct successors of u_i. Formula (5.2) also means that the expectation loads of each vertex are evenly spawned to its upstream vertices. The more successors each vertex has, the more expectation loads it has. So, the expectation loads can be used to spawn vertices among processors.

The optimal executable time and the distribution expectation can be calculated by inversely tracing the digraph from those vertices with zero out-degree to those vertices with zero in-degree.

The heuristic partitioning method consists of four steps. The first step organizes all vertices into multiple layers, the second step decomposes vertices in each layer to virtual processors, the third step combines virtual processors into P processors, and the fourth step optimizes the load balancing efficiency. In the second step, arc cuts are optimized.

Algorithm 5.1. *A partitioning method toward algorithm speedup for acyclic digraphs using P processors.*

(1) $P_0 = P, P_{max} = \alpha P_0 (\alpha > 1)$; P_c is the number of virtual processors.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

(2) Generate multiple layers. The bottom layer consists of vertices with zero out-degree. The second layer consists of vertices for each of its downstream vertices located at the bottom layer. Continue this process until the top layer is formed.

Denotes:

$l(u)$	the rank of layer to which vertex u belongs;
$V(l)$	the set of vertices in the lth layer;
u_l^i	the ith vertex in $V(l)$;
$V(p_k)$	the set of vertices distributed to processor p_k;
$p(u)$	the processor to which vertex u is distributed;
$\wp(u, p_i)$	the action of spawning vertex u to processor p_i;
L_{max}	the number of layers.

(3) Distribute vertices to virtual processors.

```
(3.1) IF ( |V(0)| ≤ P_0) {
          • ℘(u_0^i, p_i);  P_c = |V(0)|;
      } ELSE {
          • evenly decompose V(0) to P_0 processors; P_c = P_0;
      }
(3.2) FOR (l = 1, 2, ..., L_max) DO {
    (3.2.1) FOR (i = 1, 2, ..., |V(l)|) DO {
                IF (u_l^i has unique dominator processor p_j) {
                    • ℘(u_l^i, p_j);     /* geometry connectivity */
                } ELSE {
                    IF (a dominator does not own vertices of V(l)) {
                        /* balancing the communication overheads. */
                        • ℘(u_l^i, p_j): p_j has minimal number of cuts;
                    } ELSE {
                        /* balancing the calculation loads. */
                        • ℘(u_l^i, p_j): p_j has minimal loads;
                    }
                }
            }  ENDFOR (3.2.1)
    (3.2.2) FOR (for each processor p_j to which V(l) is distributed) DO{
                /* expectation loads of processors.*/
                • solve δ(p_j) = Σ_{i∈V(p_j)} δ(u_i)
                /* control loads of each virtual processor. */
                FOR (each vertex u_l^i ∈ V(p_j)) DO  {
                    IF ((δ(p_j) − δ(u_l^i)) > ξ_min && P_c < P_max) {
                        • P_c = P_c + 1;  ℘(u_l^i, P_c + 1);
                        • δ(p_j) = δ(p_j)-δ(u_l^i)  }
                } ENDFOR (each vertex u_l^i)
            } ENDFOR (3.2.2)
    (3.2.3) FOR (i = 1, 2, ..., |V(l)|) {  /* load balancing*/
                IF (|V(p(u_l^i))| > ξ_max && d^−(u_l^i) > 1) {
                  • ℘(u_l^i, p_j): p_j is the dominator processor and has
                                minimal distribution expectation;
                }
    }  ENDFOR (3.2)
(3.3) Combine virtual processors.
    (3.3.1) N_it = K_max; /* maximal number of iterations. */
    (3.3.2) WHILE (P_c > P_0 and N_it > 0) DO {
```

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

FOR ($i = 1, 2, \ldots, P_c$) DO {
/* *small loads should be combined.* */
IF ($|V(p_i)| < \xi_{min}$ and $\exists j \neq i$ such that {
($|V(p_j)| < \xi_{min}$ or $|V(p_i)| + |V(p_j)| < \xi_{max}$)
and (p_i is the neighbor of p_j) }) {
• combine p_i to p_j and delete p_i;
• $P_c = P_c - 1$;
• reorder virtual processors and break; }
} ENDFOR ($i = 1, 2, \ldots, P_c$)
• $N_{it} = N_{it} - 1$;
} ENDFOR (3.3.2)
(3.3.3) IF ($P_c > P_0$) { modify ξ_{max} and ξ_{min}, goto (3.3.1).}
(4) Load balancing optimization.
(4.1) FOR ($l = L_{max}, L_{max} - 1, \ldots, 2, 1$) DO {
(4.1.1) FOR ($i = 1, 2, \ldots, |V(l)|$) DO {
IF ($|V(p(u_l^i)| > \kappa$ and $\exists j : i \in S(j)$) {
• redistribute u_l^i to one of its dominator processors having the smallest loads.
}
ENDFOR (4.1.1)
ENDFOR (4.1)

The parameter ξ_{min} (ξ_{max}) is the lower (upper) load bound per processor. The parameter κ is a similar parameter but is less than ξ_{max}. In general, we take the parameters

$$P_0 = P,\ K_{max} = 4,\ \alpha = 4,\ \xi_{min} = 0.6S/P,\ \xi_{max} = 1.3S/P,\ \kappa = 1.1S/P. \tag{5.3}$$

Here, $S = |V(D)|$.

Algorithm 5.1 preserves the geometry connectivity because it always distributes vertices to its dominator processor or a new processor. Loads are especially balanced in step (3.2), followed by step (3.3), and are finally optimized in step (4). Lower communication complexity is expected in step (3.2.1). Parallelism is guaranteed in step (3.1).

Apply Algorithm 5.1 to the digraph in Figure 1 for two processors using the parameters given in formula (5.3). Two subdigraphs are generated. One subdigraph owns the set of vertices {1,2,3,4,5,9,10,14} and another subdigraph owns the set of vertices {6,7,8,11,12,13,15,16}. The algorithm speedup, $S_A = 1.6$, is equal to the optimal speedup S_∞.

In order to test Algorithm 5.1, we individually construct 16 acyclic digraphs from 16 sweeping angles for the neutron transport applications discussed in section 4.1. We compare the algorithm speedup of **GP1** and **GP3** with Algorithm 5.1 for these digraphs. The results are listed in the third row of Table 6. By the way, the load balancing efficiency η_P and the communication surface volume γ_P with respect to that of **GP1** are also listed in the last two rows. These results show that Algorithm 5.1 is the most superior. In fact, its algorithm speedup is very close to the optimal speedup of 21.4 for $P = 64$. Moreover, the loads are also well balanced and the communication overheads are greatly optimized.

Unfortunately, Algorithm 5.1 cannot be easily applied to the digraphs arising from the grid-based scientific applications where multiple downstream sweepings are simultaneously executed. This problem will be investigated in our further works.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Table 6
*Average algorithm speedup using three partitioning methods. P is the number of processors, "**new**" represents Algorithm* 5.1*, and "speedup" means algorithm speedup.*

P	4			16			64		
Methods	GP1	GP3	**new**	GP1	GP3	**new**	GP1	GP3	**new**
Speedup	3.22	2.54	3.46	8.32	5.31	9.13	16.8	11.2	18.3
η_P	0.98	1.00	0.84	0.96	1.00	0.75	0.93	1.00	0.86
γ_P	3.42	1.00	3.02	2.98	1.00	2.67	2.51	1.00	1.76

6. Conclusion. The acyclic digraph models are presented in this paper and they are universal for the accurate description of data flow for various grid-based scientific computations from numerical algebra to S_n transport, from structured grids to unstructured grids, and from conforming grids to nonconforming grids.

A framework of heuristic parallel algorithms is presented in this paper. It consist of three components: the digraph partitioning methods, the parallel sweeping algorithm given by Algorithm 3.1, and the heuristic strategies. Algorithm 3.2 gives the locally shortest path priority strategy which is more efficient than other works, and Algorithm 3.3 gives the heuristic packing strategy for the tolerance of large MPI latency. As shown in Table 3, this framework can scale up to 2048 processors for S_n transport.

The digraph partitioning methods should be improved. Algorithm 5.1 is well suited for a single sweeping. The parallel partitioning methods are often required for large scale applications. For the underlying supergraph, the parallel partitioning tools such as ParaMetis [24] can be used. However, for the digraph, new algorithms should be studied.

In this paper, it is assumed that digraphs must be constructed before a parallel sweeping begins. Sometimes such an assumption is impossible because data flow cannot be determined until vertices are computed. We will study such digraphs in the future.

Acknowledgments. The authors thank Mr. Guanhao Jin and Ms. Junxia Wei for their numerical experiments and Prof. W. Hackbusch for the discussions on cycle detections and breaking strategies for convection-dominant flows.

REFERENCES

[1] P. N. Brown, *Ardra: Scalable Parallel Code System to Perform Neutron and Radiation Transport Calculations*, http://www.llnl.gov/casc/Ardra.

[2] R. S. Baker and R. E. Alcouffe, *Parallel* 3*-d S_n performance for MPI on Cray-T*3*D*, in Proceedings of the Joint International Conference on Mathematics Methods and Supercomputing for Nuclear Applications, 1 (1997), pp. 377–393.

[3] R. S. Baker and K. R. Koch, *An S_n algorithm for the massively parallel CM-*200 *computer*, Nucl. Sci. Eng., 128 (1998), pp. 312–320.

[4] J. Bey and 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 (1997), pp. 177–192.

[5] R. H. Bisseling, *Parallel Scientific Computation: A Structured Approach using BSP and MPI*, Oxford University Press, Oxford, UK, 2004.

[6] R. L. Bowers and J. R. Wilson, *Numerical Modeling in Applied Physics and Astrophysics*, Jones and Bartlett, Sudbury, MA, 1991.

[7] P. N. Brown, R. D. Falgout, and J. E. Jones, *Semicoarsening multigrid on distributed memory machines*, SIAM J. Sci. Comput., 21 (2000), pp. 1823–1834.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

[8] Committee on High Energy Density Plasma Physics, Plasma Science Commitee, and National Research Council, *Frontiers in High Energy Density Physics: The X-Games of Contemporary Science*, National Academies Press, Washington, DC, 2003.
[9] J. Dongarra, I. Foster, G. Fox, W. Gropp, K. Kennedy, L. Torczon, and A. White, *Sourcebook of Parallel Computing*, Morgan Kaufmann, San Francisco, CA, 2003.
[10] E. W. Larsen, *Diffusion-synthetic acceleration methods for the discrete-ordinates equations*, in Proceedings of the ANS Topical Meeting on Advances in Reactor Computations (Salt Lake City, UT), American Nuclear Society, LaGrange Park, IL, 1983, pp. 57–62.
[11] G. C. Fox, R. D. Williams, and P. C. Messina, *Parallel Computing Works*, Morgan Kaufmann, San Francisco, CA, 1994.
[12] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message-Passing Interface*, 2nd ed., MIT Press, Cambridge, MA, 1999.
[13] J. L. Gross and J. Yellen, eds., *Handbook of Graph Theory*, Discrete Mathematics and Its Applications 25, CRC Press, Boca Raton, FL, 2003.
[14] W. Hackbush and G. Wittum, eds., *Incomplete Decompositions (ILU): Algorithms, Theory and Applications*, Notes Numer. Fluid Mech. 41, Vieweg, Braunschweig, 1993.
[15] W. Hackbush and T. Probst, *Downwind Gauss–Seidel smoothing for convection dominated problems*, Numer. Linear Algebra Appl., 4 (1997), pp. 85–102.
[16] W. Hackbusch, *On the feedback vertex set problem for a planar graph*, Computing, 58 (1997), pp. 129–155.
[17] B. Hendrickson and R. Leland, *The Chaco User's Guide: Version* 2.0, Technical report SAND94-2692, Sandia National Laboratories, Albuquerque, NM, 1994.
[18] B. J. Jørgen and G. Gregory, *Digraphs: Theory, Algorithms and Applications*, Springer-Verlag, London, 2001.
[19] C. Koutsougeras and C. A. Papachristou, *Data flow graph partitioning to reduce communication cost*, in Proceedings of 19th Annual Workshop on Microprogramming, 1986, pp. 82–91.
[20] B. Lee, A. R. Hurson, and T. Y. Feng, *A vertically layered allocation scheme for data flow systems*, J. Parallel Distributed Comput., 11 (1991), pp. 175–187.
[21] Y. F. Lee and B. Ryder, *A comprehensive approach to parallel data flow analysis*, in Proceedings of the 1992 International Conference on Supercomputing, 1992, pp. 236–247.
[22] E. E. Lewis and W. F.Miller, *Computational Methods of Neutron Transport*, John Wiley & Sons, New York, 1984.
[23] D. Li and G. Xu, *Numerical Methods for Two-Dimensional Nonsteady Hydrodynamics*, Chinese Academic Publisher, Beijing, 1987.
[24] G. Karypis, K. Schloegel, and V. Kumar, *ParMetis: Parallel Graph Partitioning and Sparse Matrix Ordering Library, Version* 3.1, Minneapolis, MN, 2003.
[25] Z. Mo and X. Li, *Parallel multigrid computation for anisotropic diffusion problems*, Chinese J. Numer. Math. Appl., 20 (1998), pp. 31–43.
[26] Z. Mo and X. Li, *On the Jacobi components at pseudo-boundaries for parallel multigrid computations*, Math. Numer. Sin., 21(1999), pp. 9–18.
[27] Z. Mo and L. Fu, *Parallel flux sweeping algorithm for neutron transport on unstructured grid*, J. Supercomputing, 30 (2004), pp. 5–17.
[28] Z. Mo, *Concatenation algorithm for parallel numerical simulation of hydrodynamics coupled with neutron transport*, Internat. J. Parallel Programming, 33 (2005), pp. 57–71.
[29] S. Plimpton, B. Hendrickson, S. Burns, and W. McLendon, *Parallel algorithms for radiation transport on unstructured grids*, in Proceeding of SuperComputing '2000, Dallas, 2000, pp. 1–17.
[30] Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2nd ed., SIAM, Philadelphia, 2003.
[31] K. Schloegel, G. Karypis, and V. Kumar, *Graph partitioning techniques for high performance scientific simulations*, in Sourcebook of Parallel Computing, J. Dongarra et al., eds., Morgan Kaufmann, San Francisco, CA, 2003, Chapter 18.
[32] J. A. Sharp, *Data Flow Computing*, Hardback Ellis Horwood, New York, 1985.
[33] H. Wasserman, *SWEEP3D Discrete Ordinates Neutron Transport Benchmark Codes*, http://www.llnl.gov/asci_benchmarks/asci/limited/sweep3d/sweep3d_readme.html.
[34] L. G. Valiant, *A bridging model for parallel computation*, Communications of the ACM, 33 (1990), pp. 108–111.
[35] F. Wang and J. Xu, *A crosswind block iterative method for convection-dominated problems*, SIAM J. Sci. Comput., 21 (1999), pp. 620–645.
[36] T. A. Wareing, J. M. McGhee, J. E. Morel, and S. D. Pautz, *Discontinuous finite element S_n methods on 3-D unstructured grids*, in Proceeding of the International Conference on Mathematics and Computation, Reactor Physics and Environment Analysis in Nuclear

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Applications, Madrid, Spain, 1999, pp. 126–130.

[37] B. Wilkinson and M. Allen, *Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers*, Prentice–Hall, Englewood Cliffs, NJ, 2002.

[38] L. Zhang, *Pipelining parallelization of a cluster of recursive formulae*, Chinese J. Numer. Math. & Comput. Appl., 20 (1999), pp. 184–191.

[39] F. Zuo and Z. Mo, *Alternating plane parallel algorithms for numerical simulation of interface instability driven by laser ablation*, Chinese J. Numer. Math. Appl., 16 (2005), pp. 1–12.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.