# Min-cut methods for mapping dataflow graphs

Volker Elling Karsten Schwan

Topic number: 2, Topic title: Scheduling and Load Balancing

Volker Elling (RWTH Aachen)
Bärenstraße 5 (Apt. 12)
52064 Aachen, Germany
volker.elling@post.rwth-aachen.de
Phone: (+49)241-8794337

Karsten Schwan (Georgia Tech)
801 Atlantic Drive
Atlanta, GA 30332-0280, USA
schwan@cc.gatech.edu
Phone: (+1)404-894-2589

**Abstract.** High performance applications and the underlying hardware platforms are becoming increasingly dynamic; runtime changes in the behavior of both are likely to result in inappropriate mappings of tasks to parallel machines during application execution. This fact is prompting new research on mapping and scheduling the dataflow graphs that represent parallel applications. In contrast to recent research which focuses on critical paths in dataflow graphs, this paper presents new mapping methods that compute near-min-cut partitions of the dataflow graph. Our methods deliver mappings that are an order of magnitude more efficient than those of DSC, a state-of-the-art critical-path algorithm, for sample high performance applications.

## 1 Introduction

Difficult steps in parallel programming include decomposing a computation into various pieces ('tasks'), distributing these tasks to the processors ('mapping'), determining an order of execution on each processor ('scheduling'), and providing means for communicating data between tasks. In the past, parallel programs were typically run on dedicated multiprocessors with processor speeds, network topologies, and bandwidths that were known in advance. For such cases, programmers could use fixed task-to-processor mappings or even develop good mappings by trial-and-error, using performance analysis tools to identify bottlenecks.

The drawbacks to such adhoc mapping methods are well known. First, some problems like 'sparse triangular solves' (see Section 4.3) are too irregular for partitioning by a human. Second, when programs are decomposed by compilers, at a fine grain of parallelism, the potential number of parallel tasks is too large to admit the use of manual methods, prompting most compiler writers to employ techniques like Dominant Sequence Clustering (DSC) to construct the parallel tasks and their schedule to be executed on the underlying parallel machine. Third, when using LAN-connected clusters of workstations or entire computational grids (as in Globus, see [FK97]) for parallel program execution, manual methods are difficult to employ due to irregular network topologies and changes in network or machine performance due to changes in load or platform failures. Concerning program structure, for grid applications, for example, a typical parallel application is comprised of a collection of parallel and sequential tasks, with certain task sets implementing single parallel applications linked to each other in order to solve a larger problem, other task sets implementing the I/O required by these applications using disk and visualization devices scattered across the grid, and finally, end users from a variety of 'access stations' interacting with these programs to perform tasks like output inspection, collaboration via the 'computational instruments' (see [PEE$^+$]) implemented by task sets, and 'program steering' (see [GESV98], [ES98]). As a result, task sets often change at runtime. Finally, in load balancing a busy processor occasionally shares tasks with a processor that has become idle. It is useful to choose the share of the other processor so that the communication between both is minimized; this requires min-cut partitioning methods for dataflow graphs (see below). In all these situations, automatically generated mappings are attractive or even required. Therefore, the ultimate goal of our research is to develop black-box mapping and scheduling packages that require limited degrees of human intervention.

In Section 2, we define the formal problem that is to be solved and give some examples for its usefulness. Section 3 discusses critical-path vs. min-cut mapping methods. Our algorithmic contributions, 'spectral mapping' and 'greedy mapping', are presented in Sections 3.3 and 3.4. Section 3.5 explains the need for periodic run-time remapping and discusses ways to adapt our algorithms for this purpose.

Section 4.1 introduces an abstract model for parallel hardware, suited for multiprocessor machines as well as for workstation clusters. This model is used for extending 2-processor mapping to an arbitrary number of processors with different speeds and an irregular interconnect. In Section 4.3, we present two real-world sample problems, 'Sparse Triangular Solves' and a climate simulation, that have rather complementary properties and reflect the wide applicability of our methods. Simulation results for these two sample problems are shown, for our two problems as well as DSC, a critical-path algorithm, and 'DSC-spectral', a variant of DSC.

## 2 Formal Problem Definition

*Sample Applications* Parallel applications are often modelled using dataflow graphs (also known as 'macro dataflow models', 'task graphs', or 'program dependence graphs'). A dataflow graph is a directed acyclic graph $(V, E)$ consisting of vertices $v \in V$ representing computation steps, connected by directed edges $(v, w) \in E$ which represent data generated by $v$ and used by $w$. The vertices are weighted with the computational costs $t(v)$, whereas edges are weighted with the amounts of communication $c(v, w)$.

There are various choices for the unit of $t$ and $c$. Many mapping and scheduling algorithms, especially critical-path methods (see Section 3.1), require $t$ and $c$ to have the unit 'computation time in seconds' and 'point-to-point delay in seconds'. This is undesirable when dealing with different processor or network speeds since computation and communication times depend on the task-to-processor assignment. In the sequel, we assume that $t$ and $c$ are specified in units that are invariant under change of processor or network, for example 'cycles', 'FLOPs' or 'computation time on a reference processor' (for $t$) resp. 'bytes' or 'delay on a reference network' (for $c$).

An example is the graph shown in Figure 1. This graph represents a two-hour iteration step in the Georgia Tech Climate Model (GTCM) which simulates atmospheric ozone depletion. On our UltraSparc-II-cluster, the tasks in the 'Lorenz iteration' execute for about 2 seconds while the chemistry tasks run for about 10 seconds. 'Debugging' simulations have to run for at least 1 month of simulated time; simulations in which certain

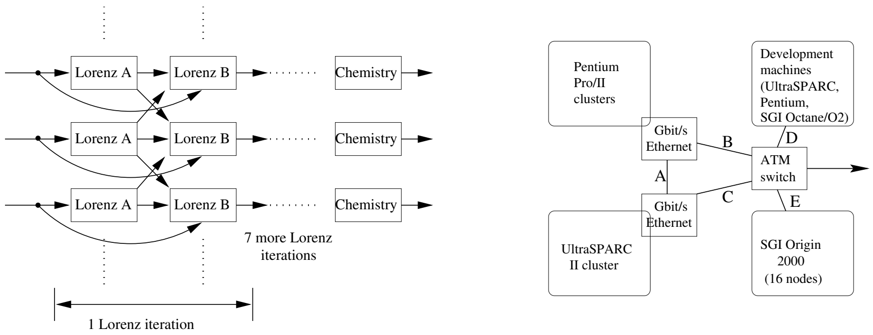

**Fig. 1.** Left: task graph in the Georgia Tech Climate Model; right: high-performance hardware in the Georgia Tech IHPCL lab

atmospheric phenomena are investigated run from at least 6 months to 6 years of simulated time. In this graph, each edge corresponds to roughly 100-400 KB of data carried across tasks. This application constitutes one of the examples addressed by our mapping (and remapping) algorithm. Another example exhibiting finer grain parallelism is presented in Section 4.3 below.

*Hardware Infrastructure* Figure 1 shows a part of the high-performance hardware in the Georgia Tech IHPCL lab. In addition to Gigabit Ethernet, the Pentium clusters are partially connected by SCI resp. Myrinet interfaces. All of the machines are shared by several research groups and usually run several applications at a time in addition to interactive jobs on the 'development' stations, each on a subset of the clusters. The UltraSPARC cluster consists of of 16 single-processor machines, the Pentium Pro cluster is composed of 64 quad-processor nodes, and the Pentium II cluster contains 32 dual-processor boxes. The development machines span a wide range of types and speeds. The operating systems include Solaris 2 (SPARC and Intel), Irix 6.4/6.5, and Windows NT.

*Problem Definition* 'Mapping' is the task of assigning each of the vertices $v$ to $p(v)$, one of $P$ available processors. Often, the mapping stage is subdivided into forming 'clusters' of tasks and assigning clusters instead of single tasks. 'Scheduling' (more precisely, 'local scheduling') is the subsequent stage of determining an order of execution for the vertices on each processor. This order can be strict or advisory. Usually, the objective is to minimize the 'makespan', the time between start of the first and completion of the last task. We define

$$\text{efficiency} = \frac{\sum_{v \in V} \text{load}_v}{\text{makespan} \cdot \sum_{p=1}^{P} \text{speed}_p}$$

where $\text{load}_v$ refers to the number of $t$-units task $v$ takes, and $\text{speed}_p$ to the speed of processor $p$ in $t$-units per second. Obviously, a higher efficiency is equivalent to a lower makespan, and the maximum efficiency is 100%.
Existing mapping and scheduling algorithms are subject to various restrictions and are therefore, not easily used to address the problem definition shown above. First, many algorithms are restricted to dealing with homogeneous processors connected by a uniform network. For the cluster and grid machines used in our work, however, we have to deal with processors of different speeds ('weakly heterogeneous multiprocessors') or even different architectures ('strongly heterogeneous multiprocessors'). Some processors

are well-suited for floating-point computations while others were designed for integer tasks, as is the case for the UltraSPARC-II vs. Pentium II clusters used in our work; the size of on-chip caches is important as well. Some processors, like the SGI Origin machine, utilize high performance interconnects, whereas the workstation clusters employ commodity networks like switched Ethernet for interprocessor communication. Finally, processor and network speeds can vary over time due to their shared use by other applications. The methods we describe deal with weak heterogeneity, and they can address changes in program behavior or resource availability by considering the remapping problem.

The performance of parallel applications is affected by various factors, including available CPU performance, the amounts of main memory or disk space, network latencies, or network bandwidths. Given these factors, some parallel programs are bandwidth-limited in that the amount of necessary communication among tasks is sufficiently large to cause slowdown due to network congestion. Other programs are latency-limited, which means that delays in frequent, fine-grain communications result in program slowdown. This paper considers both latency- and bandwidth-limited programs.

## 3 Mapping Algorithms

This section first describes a popular mapping algorithm called 'Dominant Sequence Clustering' (DSC). Next, we present alternative mapping algorithms based on min-cut methods, called 'spectral bisection' and 'greedy spectral bisection'. In Section 4, these algorithms are shown to be superior to DSC in terms of mapping quality.

### 3.1 Critical-path Methods

Critical-path methods are based on $t(v)$ resp. $c(v, w)$ being computation resp. communication time. A path in the dataflow graph is a 'critical path' if the sum of $t(v)$ and $c(v, w)$ on this path equals the makespan. In order to decrease the makespan, one has to decrease $t(v)$ or $c(v, w)$ on the critical paths. Most mapping and scheduling algorithms for dataflow graphs proceed by looking for critical paths in the graph and assigning adjacent tasks on the path to the same cluster. Later, tasks $v, w$ in the same cluster are assigned to the same processor which corresponds to 'zeroing' the communication time $c(v, w)$. The most advanced critical-path algorithm known to us is 'Dominant Sequence Clustering' ('DSC'; see [YG94]). It is faster than older algorithms since it avoids recomputing the critical paths after every zeroing step, by careful selection of edges to be zeroed. On the other hand, it is as efficient as the other algorithms in minimizing makespan (see [GY92]). We will use this algorithm to assess the performance of critical-path methods as compared to the min-cut methods we describe below.

Usually, critical-path methods do not attempt to minimize cut size. The clusters formed by DSC are assigned to processors in blocks or cyclically. In [YG94], the use of Bokhari's algorithm (see [Bok81]) for overcoming this limitation is proposed: An undirected graph is formed, with the clusters as vertices and edges $(c, d)$ weighted by the amount of communication $C(c, d)$ between clusters $c$ and $d$:

$$C(c, d) := \sum_{v \in c, w \in d} (c(v, w) + c(w, v)).$$

The cut size is reduced by computing small-cut partitions of the cluster graph and assigning them to processors. Instead of Bokhari's algorithm, we partition the graph by spectral bisection resp. greedy bisection (see below). We refer to this variant as 'DSC-spectral' resp. 'DSC-greedy'.

### 3.2 Min-cut Methods

Rather than shortening critical paths, min-cut methods try to find a mapping $\pi : V \to \{1, \dots, P\}$ of the task graph with small cut size, defined as

$$\mathrm{cutsize}(\pi) := \sum_{v,w \in E,\ \pi(v) \neq \pi(w)} c(v, w),$$

and loads

$$\mathrm{load}_p(\pi) := \sum_{\pi(v)=p} t(v)$$

which are roughly proportional to the respective speeds of the processors $p = 1, \dots, P$.

*Previous Work* To the best of our knowledge, there has been no previous attempt to define explicit min-cut mapping algorithms for *dataflow* graphs. However, dataflow graphs often arise from undirected graphs like finite-element grids. There has been progress on min-cut partitioning algorithms for *undirected* graphs ([KL70], [FM82], [PSL90], [KK]; a good overview is [Els97]). Unfortunately, these methods cannot be trivially applied to the problem we are solving, since our complex parallel applications typically consist of several coupled subcomponents (e.g., a finite-element elasticity code, a finite-volume gas dynamics code, chemistry and and a visualisation) that cannot be scheduled easily by partitioning a single physical grid.
Communicating long-running processes form an undirected doubly-weighted graph. Bollinger and Midkiff ([BM91]) have developed a method to map processes onto processors which is based on Simulated Annealing. Chaudhary and Aggarwal ([CJ93]) propose a method for a problem similar to our dataflow graphs which proceeds by greedy pairwise exchange. However, they recompute the makespan (or a similar objective function) after every exchange. This is very expensive for large graphs but seems to be inevitable in absence of the 'time intervals' we use for 'greedy mapping' as described in Section 3.4. Min-cut algorithms for directed graphs cannot easily be adapted to dataflow graphs since the partition with smallest cut size might be the *worst* rather than the best mapping (see Figure 2). As evident from the example in the figure, it is important to take the directedness of the graph into account. Toward this end, we define the earliest-start

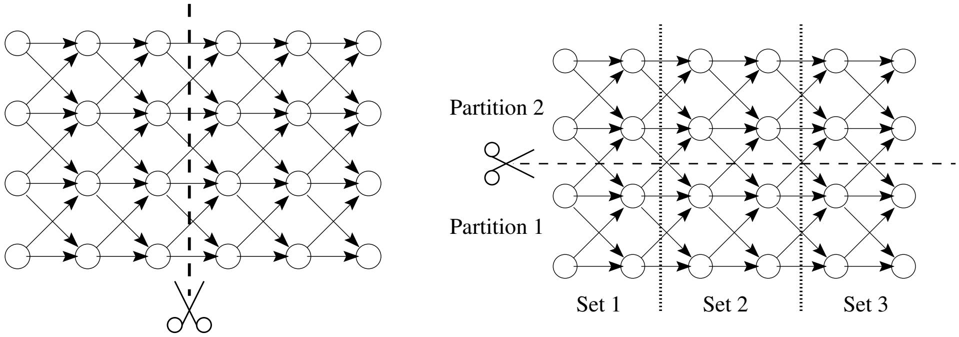

**Fig. 2.** Left: a bad min-cut mapping; right: improved mapping

resp. earliest-finish 'times' $\mathrm{est}(v)$ and $\mathrm{eft}(v)$ by

$$\mathrm{est}(v) := \max_{(w,v)\in E} \mathrm{eft}(w)$$

and

$$\mathrm{eft}(v) := \mathrm{est}(v) + t(v)$$

(note that this definition is valid because the graph is acyclic). The nodes are sorted by est and separated into $K$ sets $V_1, \ldots, V_K$ so that

$$i \leq j, v \in V_i, w \in V_j \quad \Rightarrow \quad \mathrm{est}(v) \leq \mathrm{est}(w).$$

Instead of requiring load proportional to speed on each processor, we require proportionality *in each set* $V_k$:

$$\mathrm{load}_{p,k}(\pi) := \sum_{v \in V_k,\ \pi(v)=p} t(v).$$

This means that we have load balance in $K$ 'time' intervals rather than overall. Of course, the exact start and finish times of the tasks are not known in advance; in addition, our definition of est and eft does *not* involve $c(v, w)$. Nevertheless, the est and eft are a practical and useful way of estimating the relative execution order of tasks in advance. In our experiments, we compute the longest path in the dataflow graph and set $K$ to half its length (node count). The mapping on the right side of Figure 2 shows the improvement.

### 3.3 Spectral Mapping

We have adapted spectral bisection (see [PSL90], [DH73]) to dataflow graphs since it delivers, along with multilevel partitioning, the best partitions for undirected graphs. Its disadvantages are low speed and difficult implementation. For simplicity, we assume that $V = \{1, \ldots, |V|\}$. The 'Laplacian matrix' (compare [Chu97]) $L = (L_{vw})$ is defined by

$$L_{vw} := \begin{cases} -c(v,w), & (v,w) \in E,\ v \neq w \\ -c(v,w), & (w,v) \in E,\ v \neq w \\ \sum_{(v,w)\in E} c(v,w), & v = w \\ 0, & (v,w),(w,v) \notin E \end{cases}.$$

It is a positive semidefinite matrix, with $(1, \ldots, 1)$ as eigenvector for the eigenvalue 0. The second smallest eigenvalue is always positive if the graph is connected. The corresponding eigenvector $x$ is called 'Fiedler vector'. It minimizes the function $\phi(x)$,

$$\phi(x) := \frac{\sum_{v,w\in E} c(v,w)(x_v - x_w)^2}{\sum_{v\in E} x_v^2}$$

Note that $\phi(x)$ is small if, for adjacent vertices $v, w$, the difference $x_v - x_w$ is small. The 'closer' vertices are in the graph, the closer are their $x$-values. For dataflow graphs, we minimize $\phi(x)$ with respect to the constraints

$$\sum_{v\in V_k} t(v) x_v = 0.$$

This corresponds to finding the smallest eigenvalue and corresponding eigenvector of the operator

$$P^{-1}LP$$

where $P$ is a linear mapping from $\mathbb{R}^{n-K}$ into the constraint subspace of $\mathbb{R}^n$.

A 'bisection' (2-partition) is obtained by choosing thresholds $T_k$, $k = 1, \dots, K$, and setting, for $v \in V_k$,

$$p(v) := \begin{cases} 1, & x_v > T_k \\ 0, & \text{else} \end{cases} .$$

The thresholds are chosen so that

$$\frac{\text{load}_{0,k}(p)}{\text{load}_{0,k}(p) + \text{load}_{1,k}(p)} \approx \alpha$$

where $\alpha$ is the speed ratio of the processors executing partition 0 resp. 1. A $P$-partition is obtained by repeated bisection.

Spectral mapping is much slower than DSC or greedy mapping (see below), but it takes only about 1.7 seconds for a 1000 vertices/4000 edge graph (20 seconds for 10000/40000) on an SGI O2 (R10000 2.6 195 MHz). These times can be improved significantly, as our current implementation is not very sophisticated and sequential; [BS93] and [Bar95] propose multilevel parallelized variants for spectral bisection that achieve an order-of-magnitude performance improvement and could be adapted to spectral mapping.

### 3.4 Greedy Mapping

A faster but lower-quality method for min-cut bisection is 'greedy mapping. It corresponds to the greedy bisection methods developed for undirected graphs (see [KL70], [FM82]). At the beginning, an arbitrary mapping $p$ is chosen. The 'gain' of a vertex is defined as the decrease in cutsize when this vertex is moved to the other partition ($p(v)$ is changed from 0 to 1 or vice versa). The vertices are moved between partitions, in order of decreasing gain. In every iteration, a vertex may be moved only once. A vertex $v \in V_k$ may not be moved if

$$|\alpha - \frac{\text{load}_{0,k}(p)}{\text{load}_{0,k}(p) + \text{load}_{1,k}(p)}|$$

becomes larger than a threshold (for example, 0.07). An iteration finishes when no vertices with nonnegative gain are left. In our experience, there is no improvement after 10–15 iterations.

### 3.5 Remapping

As we have mentioned, it is desirable to compute a new mapping if processor or network performance change during execution. Our atmospheric simulation has running times of several minutes to several days. During this time, network links and computers can break down and become available again; other users start and stop their own high-performance applications on a part of the clusters. It is impossible to adapt to these changes manually because 24-hour operator supervision would be necessary. On the other hand, not adapting would degrade simulation speed severely. The only alternative is to develop automatic mapping methods like the ones we describe.

Greedy mapping is sufficiently fast for a remapping frequency on the order of 1/second and is easily adapted to take initial data location into account. For spectral bisection, the underlying problem to be solved is no longer an eigenvalue problem (see the discussion in [Ell98]), but replacing the eigensolver by a more general optimization routine should not be difficult. Unfortunately, for many parallel applications and execution platform, given the rate at which they change, the speed of spectral mapping is likely to make remapping difficult except for long-running applications with stable, long lasting phases like our atmospheric simulation.

# 4 Evaluation

## 4.1 Topologies and their Modeling

We model real-world network topologies by a simple but representative model. The cluster consists of processors with different speed. Each processor is connected to an arbitrary number of buses. Buses themselves can be connected by switches (which are 'zero-speed processors').

In order to apply our bisection algorithms to irregular network topologies with more than 2 processors, we compute a hierarchy of processors. A cluster is modeled as a mesh of processors, each of which is connected to an arbitrary number of buses. Processors are weighted with their speed, while buses are weighted with their bandwidth (alternatively, latency or another characteristic can be used). At each step, the bus with highest bandwidth is chosen and, together with the connected processors, contracted into a single parent 'processor'. When all buses have been contracted, only one 'processor' is left. The clustering is undone in reverse order. At each step, one processor is unfold into a bus and the adjacent processors. The dataflow graph partition corresponding to the parent processor is distributed to the children by repeated bisection. The ratio $\alpha$ is chosen according to the performance value of the children in each bisection step. Finally, all clustering steps have been undone, and every processor has been assigned a partition of the graph.

As an example, consider Figure 1. Depending on the type of switches, each cluster would be modeled as a bus to which machines and switch are connected. The $p$-processor machines can be treated as single-processor machines with $p$-fold speed, or as a separate bus with $p$ nodes connected to it. The latter case is important for large multiprocessor machines. Starting with the Gigabit Ethernet links, each link, together with machine and switch would be collapsed into a 'processor'. After all of them are gone, the ATM links are collapsed. Obviously, our bisection algorithm will try to assign coherent pieces of the dataflow graph to the Gigabit clusters and minimize communication via the slow ATM switch. However, the example also demonstrates the limitations of our simplistic topology treatment: the link 'A' interconnecting the two Gigabit switches might be collapsed first (before any of the Gigabit-switch-to-processor links is collapsed). Since it might represent a bottleneck, collapsing it last (i.e. assigning coherent pieces of the dataflow graph to the processors on each side) would be more appropriate. A more sophisticated algorithm could consider the cluster topology as an undirected graph and apply spectral bisection to it.

## 4.2 Local Scheduling

There are many good heuristics for computing local schedules, for fixed mapping, on a multiprocessor, even in the presence of communication delays. The survey paper of

Gerasoulis and Yang (see [YG93]) discusses various schemes and compares them for randomly generated graphs. An interesting aspect is that taking communication delays into account (as in the RCP* scheme) and neglecting them (in the RCP scheme) does not change the schedule quality. In our experiments we have used the RCP scheme (in its non-strict form, i.e. out-of-order execution is allowed if the highest-priority task is not ready).

### 4.3 Sample Problems

We have chosen two sample problems that are rather complementary and reflect the variety of parallel applications. The first, 'sparse triangular solves', is very fine-grain and latency-limited. The tasks take $\leq 1\mu$s; the edges correspond to 8 bytes. It is very hard to achieve good speedup for runtime-generated mappings because the administration overhead (for distributing data and scheduling tasks) might be larger than the actual computation time. The cost of runtime mapping can be amortized only if the mapping is reused many times (which is realistic).

The second problem, the Georgia Tech climate model (see [KSS$^+$96]) which has already been introduced in Section 2, is a very coarse-grain and usually bandwidth-limited problem. The tasks execute for several seconds; the edges correspond to data in the order of 100 kilobytes. The model runs for a very long time; this is typical for many of the so-called grand-challenge applications.

For our simulations below, we use the following simplifications: data is sent in packets that have equal size on each bus. The bandwidth of a bus is determined by the number of packets per second. Network interfaces have unlimited memory and perfect knowledge about the other interfaces on the bus. When a bus becomes available, one of the waiting interfaces is chosen randomly with uniform probability. These simplifications are not vital. By varying the packet size it is possible to simulate networks with different latencies.

*Sparse Triangular Solves* Our first test problem are sparse triangular solves (STS): solve for $x$ in the linear system

$$Ax = b$$

where $A$ is a lower triangular matrix with few nonzero entries. Task $i$ corresponds to solving for $x_i$; this requires all $x_j$ with $a_{ij} \neq 0$. The dataflow graph is determined by the sparsity structure of $A$ (see Figure 3).

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ -1 & 2 & 0 & 0 & 0 \\ 2 & 0 & 1 & 0 & 0 \\ 0 & 1 & -1 & 1 & 0 \\ 1 & 3 & 0 & -1 & 1 \end{pmatrix}$$

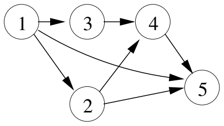

**Fig. 3.** Lower triangular matrix and its dataflow graph. The vertices are labeled with the index of the corresponding matrix row

The performance of DSC for STS in comparison to simple mapping heuristics was examined in [CSBS95] for a real-world implementation. One result is that DSC is too slow because the mapping time exceeds the actual execution time in practice. A workstation cluster is appropriate only if the matrix is very large; otherwise a multiprocessor machine with a good interconnect is mandatory. Since spectral mapping is slower than DSC, we consider sparse triangular solves as a source for real-world dataflow graphs rather than a practical application. The results shown in figure 4 were obtained by simulation using the hardware model described in section 4.1. For this experiment, we use 16 equal-speed processors connected by a bus with packet size 16 byte. Each task is assumed to cost 1 microsecond on these processors. $g$ is the latency (in time units) for sending one number over an idle network. The matrix is `bcspwr10` from the Harwell-Boeing collection ($5300 \times 5300$, 8271 below-diagonal nonzeros; available in 'netlib'); other matrices from the `bcspwr` collection yield similar results, as do randomly generated matrices.

Spectral mapping achieves the best results, followed by greedy mapping which offers a fast alternative. Mapping the task clusters generated by DSC with spectral bisection ('DSC spectral') improves DSC performance but cannot compete with the genuine min-cut methods. It is worth noting that for the small `bcspwr05` matrix ($443 \times 443$, 590 below-diagonal nonzeros), 16 CPUs and an 'infinitely fast' network (bandwidth $10^{30}$ MB/s), spectral mapping achieves 95 % efficiency while DSC achieves about 51 %. Even in this case where communication delays can be neglected, the min-cut algorithm produces better mappings.

*Atmospheric Ozone Simulation* In our second application, speedup is bandwidth-limited due to large data items. Our topology consists of two $B$ MByte/s buses with 8 equal-speed processors at each, connected by a $B$ MB/s link. This topology represents typical bottleneck situations – in Figure 1, these could occur if, for example, a computation is distributed between the nodes in the UltraSPARC cluster and the 16-node SGI Origin.

Figure 4 gives results for this problem (with 32 atmospheric levels). In this simulation we used a network packet size of 256 byte; this size is representative for commodity workstation interconnects which are well-suited for GTCM.

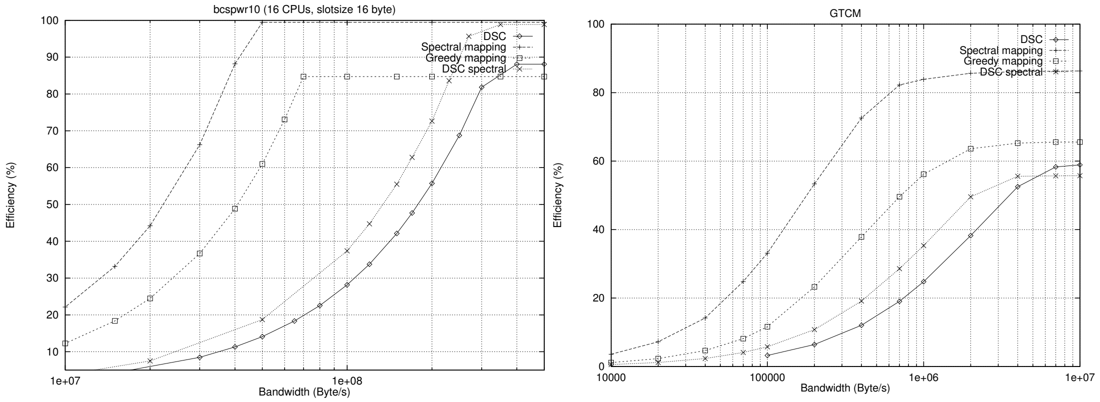

**Fig. 4.** Simulation results for Sparse Triangular Solves and GTCM

Obviously, the small cut size is essential: spectral mapping achieves the same efficiency as DSC for $B$ a factor ten smaller. Greedy mapping again qualifies as a fast alternative with fair quality. Although 'DSC spectral' distributes clusters of tasks with respect to min-cut, it is not effective in minimizing communication over the bottleneck link joining the two buses. This is most likely caused by the fact that DSC does not form task clusters based on cut size; DSC spectral can distribute whole task clusters only.

### 4.4 Summary of experimentation

It is apparent that the two min-cut methods we describe are comparable to DSC for infinitely fast networks and clearly superior for slow networks, since they achieve a small cut size. Greedy and spectral mapping are well-suited for black-box mapping software because they are applicable to a wide range of processor/network speed ratios, to diverse task graphs and regular as well as irregular network topologies. The quality of spectral mapping is higher, but greedy mapping is faster and therefore suited for runtime remapping.

## 5 Conclusions

The main contribution of our work is a method for applying undirected-graph min-cut methods to dataflow graph min-cut mapping, namely the 'time intervals' defined above. We have adapted spectral bisection and greedy bisection to dataflow graph mapping. These methods are applicable for a wide range of processor/network speed ratios. We have demonstrated that, with respect to quality, min-cut mapping methods are slightly better than critical-path methods for fast networks where small cut size does not seem to matter, and clearly superior for slow networks. We expect that min-cut-mapping will replace critical-path methods.

Future work to be done includes the acceleration of spectral mapping in order to make it practical for a wide range of applications. Furthermore, additional work on greedy spectral mapping is necessary in order to assess whether this method performs well for large dataflow graphs, because greedy algorithms 'look' at the graph in a 'local' way. Toward this end, multilevel bisection strategies as discussed in [KK] for undirected graphs are promising.

We have considered only weakly-heterogeneous clusters; for an application that consists, for example, of integer as well as floating-point tasks and runs on a mixed Intel and MIPS CPU cluster, this would lead to serious performance penalties. Also, it is not clear whether our simplistic topology clustering method is appropriate for all network topologies appearing in practice.

## References

[Bar95] Stephen T. Barnard, *PMRSB: Parallel multilevel recursive spectral bisection*, Proceedings of Supercomputing, 1995.

[BM91] S. Wayne Bollinger and Scott F. Midkiff, *Heuristic technique for processor and link assignment in multicomputers*, IEEE Transactions on Computers **40** (1991), no. 3, 325–333.

[Bok81] Shahid H. Bokhari, *On the mapping problem*, IEEE Transactions on Computers **C-30** (1981), no. 3, 207–214.

[BS93] Stephen T. Barnard and Horst D. Simon, *A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems*, Proceedings of the 6th SIAM Conference on Parallel Processing for Scientific Computing, 1993, pp. 711–718.

[Chu97] Fan R. K. Chung, *Spectral graph theory*, Americal Mathematical Society, 1997.

[CJ93] Vipin Chaudhary and J.K.Aggarwal, *A generalized scheme for mapping parallel algorithms*, IEEE Transactions on Parallel and Distributed Systems **4** (1993), no. 3, 328–346.

[CSBS95] Frederic T. Chong, Shamik D. Sharma, Eric A. Brewer, and Joel Saltz, *Multiprocessor runtime support for fine-grained, irregular dags*, Parallel Processing Letters **5** (1995), no. 4, 671–683.

[DH73] W.E. Donath and A.J. Hoffman, *Lower bounds for the partitioning of graphs*, IBM Journal of Research and Development (1973), 420–425.

[Ell98] Volker W. Elling, *A spectral method for mapping dataflow graphs*, Master's thesis, Georgia Institute of Technology, 1998.

[Els97] Ulrich Elsner, *Graph partitioning — a survey*, Tech. Report Preprint SFB393/97-27, Technische Universität Chemnitz, Sonderforschungsbereich "Numerische Simulation auf massiv parallelen Rechnern", December 1997.

[ES98] Greg Eisenhauer and Karsten Schwan, *An object-based infrastructure for program monitoring and steering*, Proceedings of the 2nd SIGMETRICS Symposium on Parallel and Distributed Tools (SPDT'98), August 1998, pp. 10–20.

[FK97] I. Foster and C. Kesselman, *Globus: A metacomputing infrastructure toolkit*, International Journal of Supercomputer Applications **11** (1997), no. 2, 115–128.

[FM82] C.M. Fiduccia and R.M. Mattheyses, *A linear-time heuristic for improving network partitions*, Proceedings of the 19th IEEE Design Automation Conference, 1982, pp. 175–181.

[GESV98] Weiming Gu, Greg Eisenhauer, Karsten Schwan, and Jeffrey Vetter, *Falcon: On-line monitoring for steering parallel programs*, Concurrency: Practice and Experience **10** (1998), no. 9, 699–736.

[GY92] Apostolos Gerasoulis and Tao Yang, *A comparison of clustering heuristics for scheduling DAGs on multiprocessors*, Journal of Parallel and Distributed Computing, Special Issue on scheduling and load balancing **16** (1992), no. 4, 276–291.

[KK] George Karypis and Vipin Kumar, *A fast and high quality multilevel scheme for partitioning irregular graphs*, To appear in SIAM Journal on Scientific Computing.

[KL70] B.W. Kernighan and S. Lin, *An efficient heuristic procedure for partitioning graphs*, Bell System Technical Journal **49** (1970), 291–307.

[KSS$^+$96] Thomas Kindler, Karsten Schwan, Dilma Silva, Mary Trauner, and Fred Alyea, *Parallelization of spectral models for atmospheric transport processes*, November 1996.

[PEE$^+$] Beth Plale, Volker Elling, Greg Eisenhauer, Karsten Schwan, Davis King, and Vernard Martin, *Realizing distributed computational laboratories.*

[PSL90] Alex Pothen, Horst D. Simon, and Kang-Pu Liou, *Partitioning sparse matrices with eigenvectors of graphs*, SIAM Journal on Matrix Analysis and Applications **11** (1990), no. 3, 430–452.

[YG93] Tao Yang and Apostolos Gerasoulis, *List scheduling with and without communication*, Parallel Computing Journal **19** (1993), 1321–1344.

[YG94] Tao Yang and Apostolos Gerasoulis, *DSC: Scheduling parallel tasks on an unbounded number of processors*, IEEE Transactions on Parallel and Distributed Systems **5** (1994), no. 9, 951–967.