# Graph Contraction for Mapping Data on Parallel Computers: A Quality–Cost Tradeoff

**R. PONNUSAMY[1], N. MANSOUR[2], A. CHOUDHARY[3], AND G. C. FOX[1]**

[1]*Northeast Parallel Architectures Center, Syracuse University, Syracuse, NY 13244*
[2]*Beirut University College, Lebanon*
[3]*ECE Department, Syracuse University, Syracuse, NY 13244*

## ABSTRACT

Mapping data to parallel computers aims at minimizing the execution time of the associated application. However, it can take an unacceptable amount of time in comparison with the execution time of the application if the size of the problem is large. In this article, first we motivate the case for graph contraction as a means for reducing the problem size. We restrict our discussion to applications where the problem domain can be described using a graph (e.g., computational fluid dynamics applications). Then we present a mapping-oriented parallel graph contraction (PGC) heuristic algorithm that yields a smaller representation of the problem to which mapping is then applied. The mapping solution for the original problem is obtained by a straightforward interpolation. We then present experimental results on using contracted graphs as inputs to two physical optimization methods; namely, genetic algorithm and simulated annealing. The experimental results show that the PGC algorithm still leads to a reasonably good quality mapping solutions to the original problem, while producing a substantial reduction in mapping time. Finally, we discuss the cost–quality tradeoffs in performing graph contraction. © 1994 John Wiley & Sons, Inc.

## 1 INTRODUCTION

Given an application based on an algorithm and a data set, $D$, the data mapping problem refers to mapping disjoint subsets of $D$ to the processors of a distributed-memory multiprocessor such that the execution time of the application, $t_{app}$, on the multiprocessor is minimized. Data mapping is an NP-hard optimization problem, and several heuristic and physical optimization algorithms have been proposed for finding good suboptimal mapping solutions. Examples of heuristic algorithms are recursive bisection [1–5], mincut-based heuristics [6], clustering and geometry-based mapping [7–9], and scattered decomposition [10]. Examples of physical optimization algorithms are simulated annealing [10–13], neural networks [12, 14, 15], and genetic algorithms [16].

For large data sets, the high-quality physical optimization (PO) mapping algorithms are very slow [17]. Their execution time is unacceptable when compared with typical execution times of applications using the data sets. In fact, the same assessment holds even for faster good-quality heuristic mapping algorithms, such as recursive spectral bisection (RSB) [4]. For example, map-

ping takes a nontrivial amount of time relative to the actual solution time when the data set is reasonably large [18]. Therefore, for realistic applications, we need to minimize the sum of $t_{app}$ and $t_{map}$, where $t_{map}$ is the mapping time. That is, the goal is to reduce the mapping time significantly while preserving a favorable mapping quality.

An approach to reducing $t_{map}$ is to shrink the problem first, and then map the reduced-size problem to the multiprocessor. The mapping solution of the coarse problem can be simply interpolated to yield the mapping solution of the original problem. The need for such an approach has been recognized in previous works [3, 19]. However, its implementation has not had much attention. We note that Nolting [20] has proposed the formation of blocks of data objects during the process of generating the data set itself. This technique may be useful for the data and application dealt with by Nolting [20], but it lacks flexibility and generalizability. Recently a graph contraction method is also proposed in the context of RSB algorithm [21].

In this article, we propose graph contraction for reducing the problem size prior to mapping. For example, to study air flow over an aircraft, the structure of the aircraft can be represented as a three-dimensional unstructured mesh [22] and the flow variables are computed only at the vertices of the mesh. In a typical mesh representation, for a good quality solution, there will be thousands of vertices and millions of edges connecting the vertices in the mesh. Efficiently mapping such a realistic mesh, as it is, onto a multiprocessor system might take more time than the solution. We propose to merge (cluster) the vertices of the original mesh to form a contracted mesh maintaining the edges between the vertices. The contracted mesh is given as input to the data mapping algorithms. Because the problem size is reduced, the mapping can be done in an acceptable amount of time. The result of contracted mesh mapping can be used to map the original mesh. We present a parallel graph contraction (PGC) heuristic algorithm oriented to satisfying the requirements of the mapping step. One of these requirements is that its execution time, $t_{pgc}$, is significantly smaller than $t_{map}$. That is, the ultimate goal can be recast as the minimization of the total sum: $(t_{pgc} + t_{map}^c + t_{app})$, where $t_{map}^c$ is the mapping for the contracted graph and graph expansion time is not included as it is negligible. Also, PGC is not restricted by assumptions about the problem structure and thus enjoys general applicability. The results show
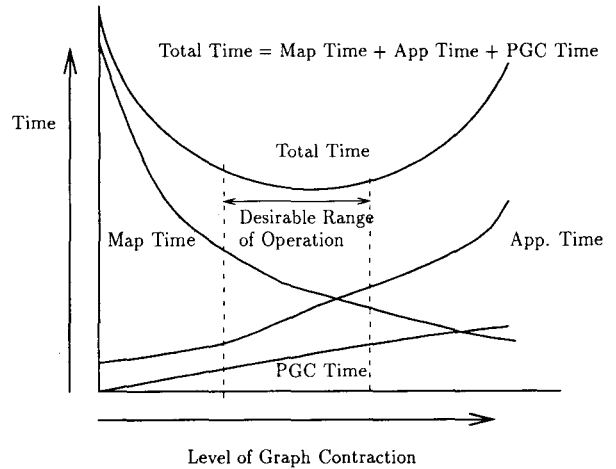


**FIGURE 1** Performance tradeoffs in mapping data using graph contraction.

remarkable savings in mapping time, while preserving good mapping solutions.

The tradeoffs involved in contraction level versus the overall application execution time is illustrated in Figure 1. Note that the figure does not correspond to any specific values and is not drawn to scale. The goal of the figure is to show the trend. As graph contraction is performed, the mapping time reduces rapidly first, where as the contraction time as well as the application execution time increases (a point on the Y-axis signifies no contraction). Increase in the application execution time is expected because the quality of mapping in terms of communication and load balance will degrade due to contraction. But initially, this degradation will be more than compensated by the reduction in the mapping time. However, for very large contraction levels, the quality may degrade so much that overall execution time will rise again. Therefore, normally we may expect a range of contraction levels (as shown in the figure) that will provide reasonably good performance, and at the same time provide a reduction in the overall execution time.

This article is organized as follows. Section 2 describes the data mapping problem. Section 3 explains graph contraction and discusses requirements for guiding the development of the graph contraction heuristics. Section 4 presents a sequential graph contraction algorithm. Section 5 presents a parallel algorithm based on the sequential one. Section 6 describes how graph contraction can be employed by PO and other mapping algorithms. Section 7 presents and discusses

the experimental results. We use a parallel genetic algorithm (PGA) mapper and a parallel simulated annealing (PSA) mapper to study the tradeoffs involved in graph contraction. Section 8 presents conclusions and future work.

## 2 DATA MAPPING

To characterize processor workloads for a data mapping instance, we define a computation graph, $G_C = (V_C, E_C)$, where its vertices, $V_C$, represent the data set and its edges, $E_C$, represent the computation dependencies among the data objects specified by the particular algorithm used by the application. Hence, the degree, $\theta(v)$, of a vertex $v$ determines its computation weight. The two terms, data objects and computation graph vertices, will henceforth be used interchangeably. We note that in this representation the weights of edges, $\xi(v, u)$, are unity for all vertices $v$ and $u$. Automatic methods for determining computation graphs are discussed [23]. The multiprocessor to which $G_c$ is mapped, is also represented by a graph $G_M = (V_M, E_M)$. The vertices, $V_M$, refer to the processors, and the edges, $E_M$, refer to their interconnections. Data mapping becomes a function from $V_C$ to $V_M$ such that $t_{app}$ is minimized. A data mapping configuration can be represented by an array $MAP[v_i]$, for $i = 0$ to $|V_C| - 1$, where $MAP[v_i]$ is the processor number, from 0 to $|V_M| - 1$, to which $v_i$ is mapped. For a given configuration $MAP[v_i]$, the workload of a processor, $p$, is composed of computation and communication components. The computation component is dependent on the sum of the degrees of the vertices mapped to $p$. The communication cost is dependent on the number of vertices that are boundary with other processors. A vertex is a boundary vertex if it has an incident edge whose other end is a vertex mapped to a different processor; we refer to such an edge as crossedge. Thus, a high-quality mapping solution is that which balances computation loads among the processors and minimizes interprocessor communication. A more formal formulation of the mapping problem is reported by Mansour [17].

## 3 MAPPING-ORIENTED GRAPH CONTRACTION

In this section, we explain premapping graph contraction and its parameters. We also discuss the requirements of data mapping that should guide the development of graph contraction heuristics.

The basic graph contraction operation involves merging two adjacent vertices, $v_i$ and $v_j$, to form a supervertex $v_{ij}$ whose computational weight is $\Theta(v_{ij}) = \Theta(v_i) + \Theta(v_j)$; initially $\Theta(v) = \theta(v)$. $v_i$, and $v_j$ are henceforth referred to as partner vertices. Merging two vertices, $v_i$ and $v_j$, is equivalent to the contraction of the edge connecting them. Also, a superedge connecting supervertices $v_{ij}$ and $v_{nm}$ is assigned a weight $\xi(v_{ij}, v_{nm}) = \Sigma_{v_x \in v_{ij}, v_y \in v_{nm}} \xi(v_x, v_y)$.

The contract-and-merge operations are applied to all vertices in the graph in an iteration k. The number of such iterations is equal to a user-defined level of contraction determined by the parameter

$$\chi = log_2 \left\langle \frac{|V_c|}{|V_c|_\chi} \right\rangle \tag{1}$$

where $|V_c|_\chi$ is the size of the contracted graph and $\langle X \rangle$ is the nearest higher power of 2 integer to X. Equivalently, the level of contraction is determined by the parameter

$$\kappa = \frac{|V_c|_\chi}{|V_M|} \tag{2}$$

the ratio of the sizes of the contracted graph and the multiprocessor. Graph contraction, with parameter $\kappa$, leads to big reduction in the search space of data mapping from $|V_M|^{|V_c|}$ to $|V_M|^{\kappa|V_M|}$, where $\kappa|V_M|$ is the size of the contracted graph and can be considerably smaller than the original size, $|V_c|$. This makes the mapping of contracted graphs a much faster step.

When mapping a contracted graph, the weights of supervertices determine the computational workload of processors, and the edge weights affect the interprocessor communication cost. Hence, for mapping purposes, an optimally contracted graph would be a fairly homogeneous weighted graph that involves relatively small edge weights. That is, optimal graph contraction is identical to finding an optimal solution to the mapping problem, which is intractable. Therefore, we can only hope for reasonable heterogeneous contracted graphs. The heterogeneity of contraction contributes to placing an upper bound on the contraction parameter, $\chi$, as shown in Section 7. On the other hand, PO mapping algorithms have degrees of flexibility and adaptability, which allows them to utilize graph contraction despite nonoptimality.

Based on these considerations, the requirements guiding the development of graph contraction heuristics can be stated as follows. The first requirement is making edges with large weights intrasupervertices edges, ensuring that most of the intersupervertices edges have relatively small weights. This requirement helps in reducing the communication cost in a mapping configuration. The second requirement is having a small average supervertex degree in the contracted graph. Small supervertex degrees are useful for decreasing the number of communicating processors, and hence the communication cost, in a mapping configuration. The third requirement is keeping the $\Theta_{max}$ to $\Theta_{min}$ ratio as small as possible; smaller variations in the weights of the vertices of a contracted graph reduces heterogeneity and yields smaller size graphs. This requirement is also necessary to support the second requirement. The fourth requirement is that a graph contraction heuristic algorithm must be efficient; its execution time must be smaller than the mapping time.

## 4 SEQUENTIAL GRAPH CONTRACTION ALGORITHM

A sequential graph contraction (SGC) heuristic algorithm that aims for satisfying the above mentioned requirements is presented in this section.

An outline of SGC is given in Figure 2. In each contraction iteration, k, pairs of adjacent vertices, i.e., partners, are selected from $G_C^{k-1}$, to be merged. The first vertex, $v_i$, is that which has the minimum $\Theta(v_i)$. Its partner, $v_j$, is an unpaired vertex adjacent to $v_i$ with maximum $\xi(v_i, v_j)$. If $v_j$ does not exist, $v_i$ becomes a vertex of $G_C^k$. The way $v_i$ is selected ensures that vertices with smaller weights are merged before those with larger weights, which limits the differences in the weights of supervertices in $G_C^k$. It has been observed that this yields a $\Theta_{max}$ to $\Theta_{min}$ ratio in $G_C^k$ that is smaller than or same to that in $G_C^{k-1}$, which is a reasonable result satisfying the third design requirement mentioned in the previous section. A partner vertex, $v_j$, is selected with maximum $\xi(v_i, v_j)$ to satisfy the first design requirement. Also, both techniques for selecting partner vertices support the second design requirement.

SGC is an efficient heuristic algorithm. A counting sort algorithm, with complexity of order $(|V_c|_{k-1} + \Theta_{max(k-1)})$, can be used for sorting vertex weights because the maximum weight is known and is relatively small in every contraction iteration [24]. It can be easily shown that the complexity of SGC is of the order of $(\theta_{max}|V_c|)$. It is also clear that SGC's complexity is considerably less than that of any of the PO mapping algorithms [17].

---

Input: $G_C^0(V_C, E_C)$; $\chi$;

$\Theta_0(v) = \theta(v); \xi_0(v_i, v_j) = 1; |V_c|_0 = |V_c|$;

for k = 1 to $\chi$ do

    Counting-Sort();

    repeat (of order of $|V_c|_{k-1}$)

        $v_i$ = unpaired vertex with minimum $\Theta_{k-1}(v_j)$;

        /* find $v_i$'s partner, if exists */

        if k = 1 then

            $v_j$ = randomly chosen unpaired vertex adjacent to $v_i$, if exists;

        else

            $v_j$ = unpaired vertex adjacent to $v_i$ with max $\xi_k(v_i, v_j)$, if exists;

        end-if-else

        Form supervertex $v_{ij} = v_i, v_j$;

    until all vertices are paired or considered

    Determine $|V_c|_k$;

    Construct_contracted_graph($G_C^k(V_C, \Theta_{k-1}(v_{ij}), \xi_k(v_{ij}, v_{nm}))$);

endfor

Output: $G_C^\chi(V_C^\chi, E_C^\chi)$ with size $|V_c|_\chi$;

---

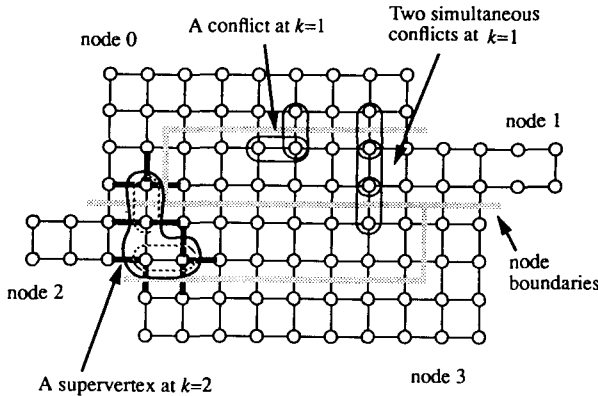**FIGURE 2** Sequential graph contraction algorithm.

**FIGURE 3** Possible conflicts and supervertex produced by PGC.

## 5 PGC ALGORITHM

A PGC algorithm is presented in this section. The PGC algorithm is based on distributing the vertices among the $|V_M|$ processing nodes and executing SGC concurrently on the distributed subgraphs. This strategy involves conflicts in different nodes over nonlocal partner vertices. Resolving conflicts in accordance with SGC requires sequential processing of boundary vertices over all nodes, which leads to deterioration in PGC's efficiency. Because our goal is to efficiently produce contracted graphs that satisfy the design requirements mentioned in Section 3 to a reasonable extent, deviating from SGC is both acceptable and necessary. Another issue that PGC has to address is the expansion in the amount of nonlocal information needed in successive contraction iterations. Figure 3 illustrates how a supervertex formed across node boundaries leads to an increase in nonlocal and nonboundary information; it also shows examples of conflicts. The design of PGC presented next addresses the two issues of conflicts and expanding nonlocal information. The guiding concerns are: making the decisions in PGC as close as possible to those in SGC, and keeping the PGC's time significantly smaller than the mapping time.

An outline of PGC is given in Figure 4. PGC is based on executing SGC concurrently in $V_M$ nodes. The initial graph, $G_C^0$, is partitioned among the nodes in a naive way: each node is allocated $|V_C^0|/V_M$ vertices; node $n_i$ is allocated vertices $n_i(|V_C^0|/V_M)$ to $(n_i + 1)V_C^0/V_M - 1$. Such subgraphs are denoted as $(G_C^0/V_M)$. A PGC iteration includes the same steps of SGC concerning the selection of

vertices and their partners for forming supervertices. Selection of nonlocal partners is allowed, which sometimes causes conflicts as illustrated in Figure 3. We note that only the vertices at the node boundaries may be involved in such conflicts. Although there are many ways in which the conflicts can be resolved, a simple rule would be to respect a nonlocal request for a partner vertex only if the requested vertex is still free or has also selected the requesting vertex as a partner. This simple rule prevents any ambiguities in forming supervertices.

After deciding about nonlocal partnership requests, the decisions are exchanged among neighboring nodes in order to update the local information about the nonlocal request in the most recent period. Those vertices that find that their requests have been turned down select a new partner, if possible, within the local set of vertices before proceeding to the next PGC period. This offers these smaller-computation-weight vertices earlier chances for merging than the other free local vertices, in accordance with SGC.

After partner selection process, the node boundaries are redrawn in order to place whole supervertices in one node. This avoids the problem of expanding nonlocal information. Boundary shifting is accomplished by some nodes transferring their part of the cross-supervertices to the other nodes that own the other part. Figure 5 shows an example of boundary shifting after the first iteration. While merging two vertices, the vertex with the lower number is merged with its higher numbered partner in even iteration steps. It is done the other way in odd iteration steps. Finally, the new contracted graph is constructed.

It is worth noting that the results of PGC depend on the mapping of the initial graph, the number of processors used, and the order in which conflicts are resolved.

## 6 MAPPING USING GRAPH CONTRACTION

Some remarks are given in this section about how the PO algorithms make use of premapping graph contraction. The algorithms include PSA and PGA.

All the algorithms map the contracted graph first; we refer to this step as coarse-structure mapping. Then, the mapped graph is decontracted by a simple interpolation in order to specify $MAP[v_i]$, for $i = 0$ to $|V_c|^{-1}$. That is, a vertex, $v_i$, in the

```
Read computation subgraph (G_c^0/N_H);
for k = 1 to χ do
    counting-sort(Θ_{k-1}(v));
    while (m=0 to m< |V_c|_{k-1}) do
        Select v_i and its partner v_j as in SGC;
        resolve_conflicts();
        for (all boundary vertices v_b requesting nonlocal partners) do
            if (request_of[v_b] = REJECT) then
                Select another local partner by an SGC step;
            end-if
        end-for
    end-while
    merge(); /* remap vertices */
    Build contracted subgraph (G_c^k/N_H); /* involves communication */
end-for
/* ————— */
resolve_conflicts()
    exchange_boundary(v_j, mate[v_j]); /* exchange decisions */
    for all local boundary vertices v_b
        match mate[v_b] with a received v_j;
        if (mate[v_b] ≠ v_j )
            request_of[v_b] = REJECT;
        end-if
    end-for
    exchange_result(request_of[v_b])
    for all local boundary vertices v_b
        if (request_of[v_b] == REJECT) then mate[v_b] = FREE;
    end-for
/* ————— */
merge()
    if (k is odd) then
        if (v_i < v_j ) then merge v_j with v_i;
        else merge v_i with v_j;
    else
        if (v_i < v_j ) then merge v_i with v_j;
        else merge v_j with v_i;
    end-if-else
```

**FIGURE 4**  PGC algorithm.

original graph is mapped to the same processor as the supervertex it belongs to: we refer to this step as fine-structure mapping.
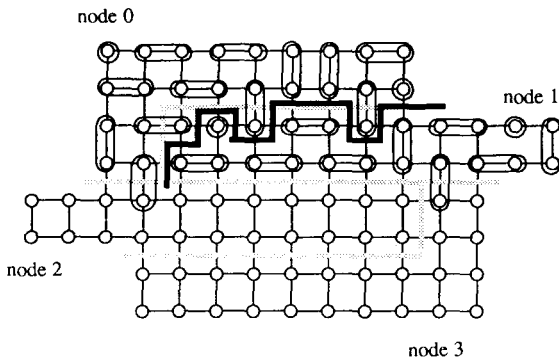


**FIGURE 5**  New node boundaries due to remapping in PGC (only 0-1 boundary shown).

In coarse-structure mapping, the PO algorithms lose some information in computing their objective functions. For example, it becomes impossible to compute the correct number of initial boundary vertices from supervertices. These are replaced by an approximation derived from the crossedges.

## 7 EXPERIMENTAL RESULTS AND DISCUSSION

This section presents experimental results for PGC and the use of its output graph for mapping. The experiments employ data sets with different sizes. These data sets constitute coarse and fine discretizations of an aircraft wing [22] (unstructured mesh representations) and are henceforth

Table 1.   Graph contraction time (sec) and Contracted Graph Size for USM(10 K)

| No. Procs | Contraction Level | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | | 2 | | 4 | | 6 | | 8 | |
| | $t_{pgc}$ | $V_{pgc}$ | $t_{pgc}$ | $V_{pgc}$ | $t_{pgc}$ | $V_{pgc}$ | $t_{pgc}$ | $V_{pgc}$ | $t_{pgc}$ | $V_{pgc}$ |
| 8 | 9.53 | 5003 | 12.29 | 2668 | 14.4 | 815 | 15.97 | 380 | 16.4 | 212 |
| 16 | 2.84 | 5153 | 4.05 | 2879 | 5.02 | 1021 | 5.72 | 470 | 6.00 | 300 |
| 32 | 1.00 | 5391 | 1.57 | 3429 | 2.22 | 1461 | 2.70 | 970 | 3.10 | 750 |

referred to as USM(x), where x is the number of data points. These data are mapped to hypercube multiprocessors. We study the effect of graph contraction on these data sets for two PO methods implemented on iPSC/860. The scheduling of irregular communication that occurs in PGC is handled using PARTI software [18].

## 7.1 PGC

The performance of the PGC for USM(10 K) on various processor sizes is shown in Table 1. The table shows the time taken for executing the PGC algorithm, $t_{pgc}$, and the corresponding size of the contracted graph, $V_{pgc}$. There are two important observations to be made from the table. First, the total time for contraction increases sublinearly as the contraction level is increased. For example, time to go from contraction level 1 to contraction level 8 results in only a threefold increment in the time. Second, the effect of approximating the sequential algorithm by a parallel one is illustrated when the number of processors is varied. As the number of processors is increased, the contracted graph's size also increases for the same contraction level. This is because the number of conflicts increases with the number of processors.

## 7.2 PGA

The most important performance metrics for the PGC algorithm, however, are the reduction in the

mapping time and the quality of the mapping based on the contracted graph. The effect of PGC on the PGA mapping time for meshes USM(2 K), USM(3 K), and USM(10 K) is shown in Table 2. Note that there is a fivefold improvement (reduction) in the mapping time for the PGA on a graph contracted from level 3 to level 6. Therefore, it can be seen that by paying a small penalty for graph contraction, mapping time can be considerably reduced. However, the reduction in mapping time should be coupled with the quality of the mapping solution to judge the overall performance. One of the ways to measure the solution quality of a mapper is using crossedges (Section 2). Crossedges determine the communication cost of mapping. The average number of crossedges (per processor) for USM(2 K) and USM(10 K) for different levels, $\chi$, of contraction is shown in Table 3. For example, for contraction level 6 for USM(10 K), the number of crossedges increases by approximately 10%. That is, for a reduction in mapping time of fivefold, the mapping degrades by 10% in terms of average crossedges. However, note that the corresponding increase in the communication time is expected to be less than 10% because a major factor of communication cost is the startup cost, which does not increase. As expected, for smaller values of $\chi$ the number of crossedges stays close to the numbe of crossedges obtained without contraction. However, beyond a threshold, the degradation in terms of crossedges increases rapidly.

## 7.3 PSA

Tables 4 and 5 show the mapping time and the number of crossedges for PSA. For example, for USM(10 K) the cost of mapping reduces by a factor of 10 when contracted from level 2 to level 6. But the number of crossedges increases by almost 50%. Comparing PGA with PSA indicates that the quality of the mapping is better for PGA but the cost of mapping is also higher for PGA. For exam-

Table 2.   Mapping time (sec) After Graph Contractions for PGA on 16 Processors

| Mesh | Contraction Level | | | | | |
|---|---|---|---|---|---|---|
| | 3 | 4 | 5 | 6 | 7 | 8 |
| USM(2 K) | 102 | 64 | 69 | 30 | | |
| USM(3 K) | 113 | 78 | 42 | 25 | | |
| USM(10 K) | 365 | 162 | 141 | 76 | 60 | 43 |

**Table 3.   Average Crossedges After Contractions for PGA on 16 Processors**

| Mesh | Contraction Level ($\chi$) | | | | | |
|---|---|---|---|---|---|---|
| | 3 | 4 | 5 | 6 | 7 | 8 |
| USM(2 K) | 536 | 596 | 796 | 924 | | |
| USM(3 K) | 660 | 740 | 785 | 941 | | |
| USM(10 K) | 1554 | 1537 | 1618 | 1713 | 2100 | 2691 |

**Table 4.   Mapping Time (sec) After Graph Contractions for PSA on Eight Processors**

| Mesh | Contraction Level ($\chi$) | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| USM(2 K) | 199 | 123 | 67 | 40 | 31 | 18 | | |
| USM(3 K) | 300 | 162 | 98 | 35 | 19 | 9 | | |
| USM(10 K) | — | — | 244 | 182 | 69 | 43 | 25 | 18 |

**Table 5.   Average Crossedges After Contraction for PSA on Eight Processors**

| Mesh | Contraction Level ($\chi$) | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| USM(2 K) | 669 | 701 | 716 | 776 | 910 | 1067 | | |
| USM(3 K) | 837 | 815 | 859 | 909 | 74 | 1190 | | |
| USM(10 K) | — | — | 1854 | 2002 | 2311 | 2530 | 2804 | 3014 |

ple, for contraction level 4, the mapping cost for PGA is a factor of 3 greater than that for PSA, but the number of crossedges is more than 50% larger for PSA. Hence, the selection of the contraction level also depends on the choice of the PO method along with the cost of mapping. After a certain contraction level, the quality of partitioning degrades due to increase in number of conflicts to be resolved in choosing partner nodes.

In Table 6, we present average crossedges for the HPF BLOCK distribution. From Tables 6, 3, and 5, it can be seen that it is important to have a good mapping of data as number of crossedges, which reflect the volume of communication, for BLOCK distribution is enormously high.

## 8 CONCLUSIONS

In this article we motivated the case for using graph contraction when using PO methods for data mapping because for large-size problems the cost of mapping on the original data size can be exorbitant. A parallel graph contraction algorithm with a user-defined contraction parameter, $\chi$. was presented for reducing the problem size prior to mapping. The experimental results show that PGC leads to considerable reductions in the execution time of the mapping algorithms, while maintaining reasonable suboptimal mapping qualities. The time reduction is larger for larger problems, because with graph contraction, time is determined by $\chi$ and $|V_M|$. not by $|V_C|$. These findings make the application of physical optimization algorithms to large problems feasible and allows the mapping step itself to be an efficient and scalable operation. Therefore, the use of graph contraction is imperative for large problems.

It was shown that with a small degradation in the quality of the mapping solution. considerable savings in the mapping time can be obtained. In our experiments we were limited by the memory size of the available parallel computer to apply our algorithms to even larger problems. We expect that the performance gains are expected to be even better for much larger problems. In the near

**Table 6.    Average Crossedges for BLOCK Partition**

| Data | Processors | |
|------|------|------|
| | 8 | 16 |
| USM(2 K) | 2159 | 1700 |
| USM(3 K) | 2225 | 1858 |
| USM(10 K) | 7476 | 6509 |

future, we expect to obtain access to machines with larger memory to experiment with larger problems.

The graph contraction method can be used with other physical optimization methods such as neural networks. It can also be used for other mapping methods that can use a weighted graph as an input. Further work needs to be done to improve the PGC algorithm itself. Also, techniques to perform good quality but low cost initial graph mapping based on domain information (e.g., geometric information) will be useful to reduce the cost of PGC as well as to improve the quality of the PGC. Finally, graph contraction techniques that are parallel yet deterministic need to be explored.

## ACKNOWLEDGMENTS

## REFERENCES

[1]  M. Berger and S. Bokhari, "A partitioning strategy for nonuniform problems on multiprocessors," *IEEE Trans. Comput.*, vol. C-36, pp. 570–580, 1987.

[2]  K. Dragon and J. Gustafson, *Fourth Conference on Hypercube Concurrent Computers, and Applications*. New York: ACM, 1989, pp. 583–590.

[3]  G.C. Fox, *Numerical Algorithms for Modern Parallel Computers*. New York: Springer-Verlag, 1988.

[4]  A. Pothen, H. Simon, and K.-P. Liou, "Partitioning sparse matrices with eigenvectors of graphs," *SIAM J. Matrix Anal. Appl.*, vol. 11, pp. 430–452, 1990.

[5]  H. Simon, *Conference on Parallel Methods on Large Scale Structural Analysis and Physics Applications*. Permagon Press, 1991.

[6]  F. Ercal, "Heuristic approaches to task allocation for parallel computing," Ph.D. thesis, Ohio State University, 1988.

[7]  N.P. Chrisochoides, C.E. Houstis, E.N. Houstis, P.N. Papachiou, S.K. Kortesis, and J.R. Rice, *Domain Decomposition Methods for Partial Differential Equations*. SIAM Publication, 1991.

[8]  S.-Y. Lee and J.K. Aggarwal, "A mapping strategy for parallel processing," *IEEE Trans. Computers*, vol. C-36, pp. 433–442, 1987.

[9]  P. Sadayappan and F. Ercal, "Nearest-neighbor mapping of finite element graphs onto processor meshes," *IEEE Trans. Computers*, vol. C-36, pp. 1408–1424, 1987.

[10]  G.C. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon, and D. Walker, *Solving Problems on Concurrent Processors*. Englewood Cliffs, NJ: Prentice Hall, 1988.

[11]  J. Flower, S. Otto, and M. Salama, *Symposium on Parallel Computations and their Impact on Mechanics*. ASME, 1987.

[12]  N. Mansour and G.C. Fox, "Parallel physical optimization algorithms for allocating data to multicomputer nodes," Syracuse Center for Computational Science, Technical Report SCCS-305, 1992.

[13]  R.D. Williams, "Performance of dynamic load balancing algorithms for unstructured mesh calculations," *Concurrency Practice Exp.*, vol. 3, pp. 457–481, 1991.

[14]  H. Byun, S.K. Kortesis, and E.N. Houstis, "A workload partitioning strategy for PDEs by a generalized neural network," Purdue University, Computer Science, Technical Report CSD-TR-92-015, 1992.

[15]  G.C. Fox and W. Furmanski, *Third Conference on Hypercube Concurrent Computers, and Applications*. New York: ACM, 1988, pp. 241–278.

[16]  N. Mansour and G.C. Fox. *Proceedings from the Supercomputing Symposium*, 1992.

[17]  N. Mansour, "Physical optimization algorithms for mapping data to distributed-memory multiprocessors," Ph.D. Dissertation, School of Computer Science, Syracuse University, 1992.

[18]  R. Das, R. Ponnusamy, J. Saltz, and D. Mavriplis, "Distributed memory compiler methods for irregular problems - data copy reuse and runtime partitioning," ICASE Report No. 91-73, 1991.

[19]  J. De Keyser and D. Roose, *Sixth Distributed Memory Computing Conference*. New York: ACM, 1991, pp. 122–128.

[20] S. Nolting, *European Distributed Memory Computing Conference*, 1991, pp. 283–293.

[21] S. Barnard and H. Simon, "A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems," Report RNR-92-033, NASA Ames Research Center, Moffett Field, CA 94035, 1992.

[22] D. Mavriplis, *AIAA 10th Computational Fluid Dynamics Conference*, 1991.

[23] R. Ponnusamy, J. Saltz, R. Das, C. Koelbel, and A. Choudhary, *Scalable High Performance Computing Conference*, 1992, pp. 216–219.

[24] T. Cormen, C. Leiserson, and R. Rivest, *Introduction to Algorithms*. McGraw Hill, 1990.