Mapping Realistic Data Sets on Parallel Computers

R. Ponnusamy
A. Choudhary
G.Fox

CRPC-TR92265
September 1992

Center for Research on Parallel Computation
Rice University
P.O. Box 1892
Houston, TX 77251-1892

SCCS - 366

Mapping Realistic Data Sets on Parallel Computers

by
R. Ponnusamy, N. Mansour, A. Choudhary, and G. C. Fox

Technical Paper

submitted to
IPPS '93

September, 1992

Syracuse Center for Computational Science
Syracuse University
111 College Place
Syracuse, New York 13244-4100
<sccs@npac.syr.edu>
(315) 443-1723

Mapping Realistic Data Sets on Parallel Computers

R. Ponnusamy N. Mansour A. Choudhary
G. C. Fox
Northeast Parallel Architectures Center
111 College Place, Suite 3-201
Syracuse University
Syracuse, NY 13244-4100

Abstract

Mapping data to parallel computer aims at minimizing the execution time of the associated application. However, it can take unacceptable amount of time in comparison with the execution time of the application if the size of the problem is large. In this paper, we propose reducing the problem size by a mapping-oriented graph contraction technique. We present a parallel graph contraction (PGC) heuristic algorithm that yields smaller representation of the problems to which mapping is then applied. The mapping solution for the original problem is obtained by straight-forward interpolation. The experimental results show that the PGC algorithm still leads to good quality mapping solutions to the original problem, while producing remarkable reductions in mapping time. The PGC algorithm allows large-scale mapping to become efficient, especially when slow but high-quality mappers are used.

1 Introduction

Given an application based on an algorithm, α, 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 sub-optimal mapping solutions. Examples of heuristic algorithms are recursive bisection [Berger and Bokhari 1987; Dragon and Gustafson 1989; Fox 1988; Pothen et al. 1990; Simon 1991], mincut-based heuristics [Ercal 1988], clustering and geometry-based mapping [Chrisochoides et al. 1991; Farhat 1988; Houstis et al. 1990; Lee and Aggarwal 1987; Sadayappan and Ercal 1987], and scattered decomposition [Fox et al. 1988]. Examples of physical optimization algorithms are simulated annealing [Flower 1987; Fox et al. 1988; Mansour and Fox 1992b; Williams 1991], neural networks [Byun et al. 1992; Fox and Furmanski 1988; Mansour and Fox 1992b], and genetic algorithms [Mansour and Fox 1992a].

For large data sets, the high-quality physical optimization (PO) mapping algorithms are very slow [Mansour 1992]. 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) [Pothen et al.]. For example, mapping takes non-trivial amount of time relative to the actual solution time when the date set is reasonably large [Das et al. 1991]. 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 [Fox 1988; De Keyser and Roose 1991]. However, its implementation has not had much attention. We note that Nolting [Nolting 1991] 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 in [Nolting 1991], but it lacks flexibility and generalizability.

In this paper, 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 an 3D unstructured mesh [Mavriplis] 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 time. 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. Since 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: $(t_{pgc} + t_{map} + t_{app})$. Also, PGC is not restricted by assumptions about the problem structure and thus enjoys general applicability. The results show remarkable savings in mapping time, while preserving good mapping solutions.

This paper 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 to evaluate the performance of the PGC algorithm. 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 dependences among the data objects specified by the particular algorithm, α, 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 in [Ponnusamy et al. 1992] and [Balasundaram et al.]. 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], for $v = 0$ to $|V_C| - 1$, where MAP[v] is the processor number, from 0 to $|V_M| - 1$, to which v is mapped. For a given configuration MAP[v], the workload of a processor, p, is composed of computation and communication components. The computation load is determined by the sum of the degrees of the vertices mapped to p. The communication cost is determined by the sum of the numbers of boundary vertices, $B(p,q)$, with other processors q. 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 given in [Mansour 1992].

3 Mapping-oriented graph contraction

In this section, we explain pre-mapping 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 $\Theta(v_{ij}) = \Theta(v_i) + \Theta(v_j)$. 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 $\xi(v_{ij}, v_{nm}) = \sum_{v_x \in v_{ij}, v_y \in v_{nm}} \xi(v_x, v_y)$, where $\xi(v_x, v_y) = 1$ initially.

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

$$\chi = \ln\langle \frac{|V_c|}{|V_c|_X} \rangle \tag{1}$$

where $|V_c|_X$ 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

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

the ratio of the sizes of the contracted graph and the multiprocessor. Graph contraction, with parameter κ , leads to big reduction in the search space of data mapping from $|V_M|^{|V_c|}$ to $|V_M|^{\kappa|V_c|}$, where $\kappa|V_c|$ 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, χ , 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 non-optimality.

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 intra-supervertices edges, ensuring that most of the inter-supervertices 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 Θ_{max} to Θ_{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 which aims for satisfying the above mentioned requirements is presented in this section.

An outline of SGC is given in Figure 1 . In each contraction iteration, k, pairs of 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

```
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 with maximum $\xi_k(v_i, v_j)$, 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 1: Sequential graph contraction algorithm

Θ_{max} to Θ_{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 the order of $(|V_c|_{k-1} + \Theta_{max(k-1)})$, can be used for sorting vertex weights since the maximum weight is known and is relatively small in every contraction iteration [Cormen et al. 90]. 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 [Mansour 1992].

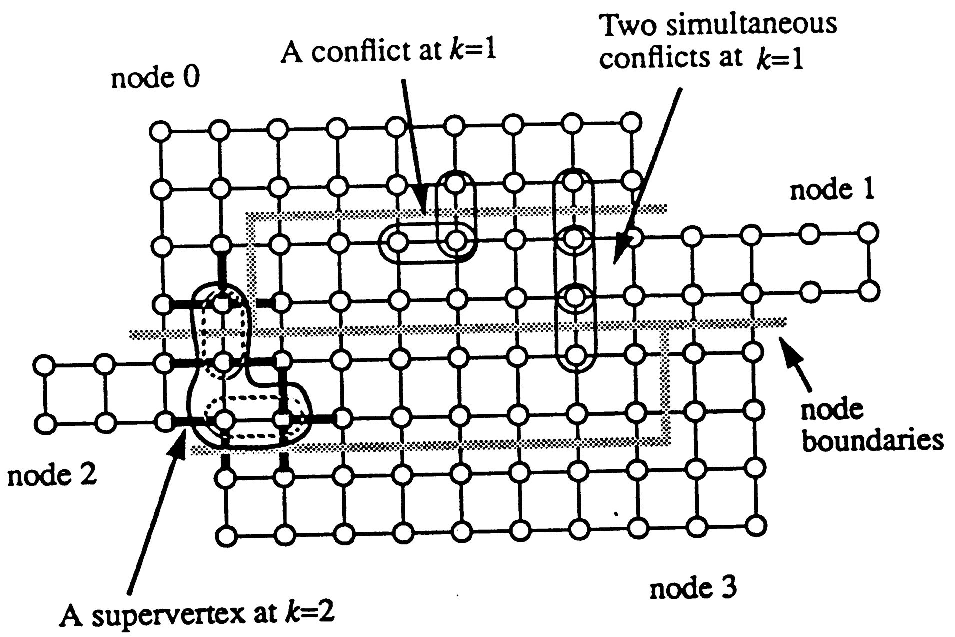

Figure 2: Possible conflicts and supervertex produced by PGC

5 Parallel graph contraction algorithm

A parallel graph contraction (PGC) algorithm is presented in this section. The PGC algorithm is based on distributing the vertices among the N_H processing nodes, PEs, 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. Since 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 2 illustrates how a supervertex formed across node boundaries leads to an increase in nonlocal and non-boundary 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 3. PGC is based on executing SGC concurrently in N_H nodes. The initial graph, G_C^0, is partitioned among the nodes in a naive way: each node is allocated $|V_C^0|/N_H$ vertices; node n_i is allocated vertices $n_i(|V_C^0|/N_H)$ to $(n_i+1)V_C^0/N_H - 1$. Such subgraphs are denoted as (G_C^0/N_H). 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 2. 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 requests 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 4 shows an example of boundary shifting after the first iteration. While merging two vertices, the vertex with 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.

6 Mapping using graph contraction

Some remarks are given in this section about how three PO algorithms and a recursive bisection algorithm make use of pre-mapping graph contraction. The algorithms include parallel simulated annealing (PSA), parallel genetic algorithm (PGA), parallel neural network (PNN) and a recursive spectral bisection (RSB).

All four 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], for v = 0 to $|V_c|$. That is, a vertex, v, in the original graph is mapped to the same processor as the supervertex it belongs to in ; we refer to this step as fine-structure mapping.

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

```
Read computation subgraph ($G_c^0/N_H$);
for k = 1 to $\chi$ do
    counting-sort($\Theta_{k-1}(v)$);
    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[$v_b$] $\neq v_j$ )
            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 3: The Parallel graph contraction algorithm

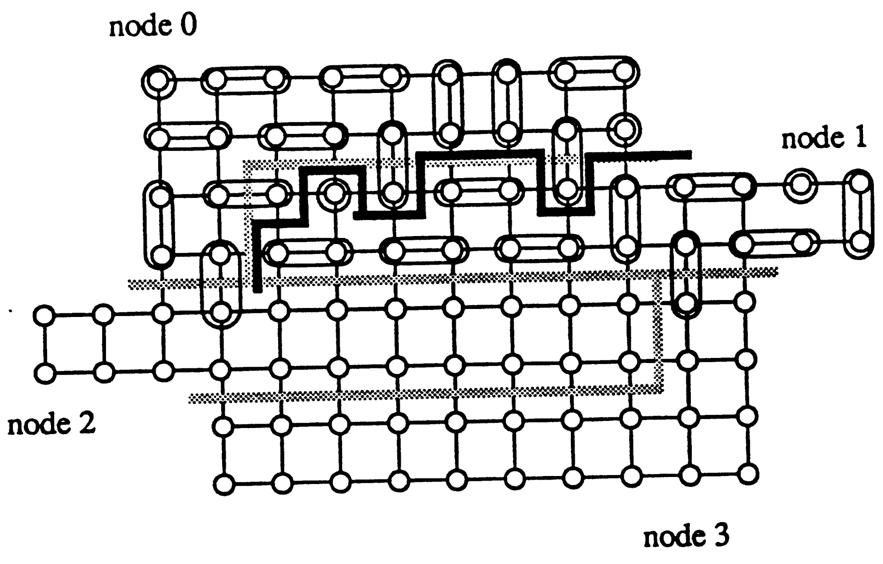

Figure 4: New node boundaries due to remapping in PGC (only 0-1 boundary shown)

Table 1: Graph contraction Time for USM(10K) (Time in Sec.)

No.	Contraction Level									
	1		2		4		6		8	
Procs	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

7 Experimental results and discussion

This section presents experimental results for graph contraction algorithm and use of its output graph on the Parallel Genetic Algorithm. The experiments employ data sets with different sizes. These data sets constitute coarse and fine discretizations of an aircraft wing [20] (unstructured mesh representations) and are hence forth referred to as USM(x), where x is the number of data points. These data are to be mapped to hypercube multiprocessors. We studied the effect of graph contraction on these data sets on one of the PO methods, genetic algorithm based data partitioner. The PGC and the PGA have been implemented on iPSC/860. The scheduling of irregular communications that occur in graph contraction algorithm are handled using PARTI software [5].

The performance of the PGC for USM(10K) on various processor sizes is shown in Table 1.

1

Table 2: Cost of Data Partitioning after graph contractions for PGA(Time in Sec.)

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

The table shows the time taken for executing the PGC algorithm and the corresponding size of the contracted graph. 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 three-fold 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.

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 PGA mapping time for meshes USM(2k), USM(3k) and USM(10K) is shown in Table 2. Note that there is a five fold 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 GC, 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 *cross edges* (Section 2). Cross edges determine the communication cost of mapping. The average cross edges for USM(2k) and USM(10k) for different levels(χ) of contraction is shown in Table 3. For example, for contraction level 6 for USM(10K), the number of crossedges increases by approximately 10%. That is, for a reduction in mapping time of five-fold, the mapping degrades by 10% in terms of average cross edges. 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 χ the number of crossedges stays close to the number of cross edges obtained without contraction. However, beyond a threshold, the degradation in terms of cross edges increases rapidly.

Table 3: Average Cross edges after contractions on 16 processors (for PGA)

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

8 Conclusions

A parallel graph contraction algorithm with a user-defined contraction parameter, χ, has been 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 good sub-optimal mapping qualities. The time reduction is larger for larger problems, because with graph contraction, time is determined by χ 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 future, we expect to obtain access to machines with larger memory to experiment with larger problems.

As an extension of this work, we will try the PGC on other PO methods such as Neural networks and Simulated Annealing as well as on other algorithms such as spectral bisection.

Acknowledgements

We would like to thank Dimitri Mavriplis for providing us with his unstructured meshes. We also would like to thank R. Das and J. Saltz for providing us PARTI software. This work was sponsored in part by DARPA under contract no. DABT63-91-C-0028 and in part by NSF grant MIP-9110810.

References

[1] Berger M., and Bokhari S. 1987. A partitioning strategy for nonuniform problems on multiprocessors. IEEE Trans. Computers, C-36, 5 (May), 570-580.

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

[3] Chrisochoides N.P., Houstis C.E., Houstis E. N., Papachiou P.N., Kortesis S.K., and Rice J.R. 1991. Domain decomposer. In Domain Decomposition Methods for Partial Differential Equations, editors R. Glowinski et al. SIAM Publication.

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

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

[6] Das R., Mavripilis D., Saltz J., Gupta S., and Ponnusamy R. 1992. The design and implementation of a parallel unstructured Euler solver using software primitives. AIAA Aerospace Sci ences Meeting, January.

[7] De Keyser J., and Roose D. 1991. A software tool for load balanced adaptive multiple grids on dis tributed memory computers. Sixth Distributed Memory Computing Conference, April, 122-128.

[8] Dragon K., and Gustafson J. 1989. A low cost hypercube load-balance algorithm. 4th Conf. Hyper cube Concurrent Computers, and Applications, 583-590.

[9] Ercal F. 1988. Heuristic Approaches To Task Allocation For Parallel Computing. Ph.D. thesis, Ohio State University.

[10] Farhat C. 1988. A simple and efficient automatic FEM domain decomposer. Computers and Struc tures. Vol. 28, no. 5, 579-602.

[11] Flower J., Otto S., and Salama M. 1987. A preprocessor for finite element problems. Symp. Paral lel Computations and their Impact on Mechanics. ASME Winter Meeting (Dec.).

[12] Fox G.C. 1988. A graphical approach to load balancing and sparse matrix vector multiplication on the hypercube. In Numerical Algorithms for Modern Parallel Computers, ed. M. Schultz, Springer-Verlag.

[13] Fox G.C., and Furmanski W. 1988. Load balancing loosely synchronous problems with a neural network. 3rd Conf. Hypercube Concurrent Computers, and Applications, 241-278.

[14] Fox G.C., Johnson M., Lyzenga G., Otto S., Salmon J., and Walker D. 1988. Solving Problems on Concurrent Processors. Prentice Hall.

[15] Houstis E.N., Rice J.R., Chrisochoides N.P., Karathonases H.C., Papachiou P.N., Samartzis M.K., Vavalis E.A. , Wang K.Y., and Weerawarana S. 1990. //ELLPACK: A numerical sim ulation programming environment for parallel MIMD machines. Int. Conf. on Super computing, 3-23, ACM Press.

[16] Lee S-Y, and Aggarwal J.K. 1987. A mapping strategy for parallel processing. IEEE Trans. on Computers, Vol. C-36, No.4, April, 433-442.

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

[18] Mansour N., and Fox G.C. 1992a. Parallel genetic algorithms with application to load balancing parallel computations. Supercomputing Symposium, Montreal, June 8-10.

[19] Mansour N., and Fox G.C. 1992b. Parallel physical optimization algorithms for allocating data to multicomputer nodes. Syracuse Center for Computational Science, SCCS-305, sub mitted for publication.

[20] Mavriplis D., *Three dimensional unstructured multigrid for the Euler equations*, In AIAA 10th Computational Fluid Dynamics Conference, June 1991.

[21] Nolting S. 1991. Nonlinear adaptive finite element systems on distributed memory computers. Eu ropean Distributed Memory Computing Conference, April, 283-293.

[22] Ponnusamy R., Saltz J., Das R., Koelbel C., and Choudhary A. 1992. A runtime data mapping scheme for irregular problems. Scalable High Performance Computing Conference, May, 216-219.

[23] Pothen A., Simon H., and Liou K-P. 1990. Partitioning sparse matrices with eigenvectors of graphs. SIAM J. Matrix Anal. Appl., 11, 3 (July), 430-452.

[24] Sadayappan P., and Ercal F. 1987. Nearest-neighbor mapping of finite element graphs onto proces sor meshes. IEEE Trans. on Computers, vol. C-36, no. 12, Dec., 1408-1424.

[25] Simon H. 1991. Partitioning of unstructured mesh problems for parallel processing. Conf. Parallel Methods on Large Scale Structural Analysis and Physics Applications, Permagon Press.

[26] Williams R.D. 1991. Performance of dynamic load balancing algorithms for unstructured mesh calculations. Concurrency Practice and Experience, 3(5), 457-481.