In the Proceedings of the Scalable High-Performance Computing Conference pp 764–772, Knoxville, TN, 1994,

Performance evaluation of load balancing algorithms for parallel single-phase iterative PDE solvers

Nikos Chrisochoides,[*] Nashat Mansour[†] and Geoffrey Fox[*]

[*]Northeast Parallel Architectures Center
Syracuse University
111 College Place, Syracuse, NY, 13244-4100

[†] Computer Science
Lebanese American University
Lebanon

Abstract

We review and evaluate the performances of six data mapping algorithms used for parallel single-phase iterative PDE solvers with irregular 2-dimensional meshes on multicomputers. We provide a table that compares the six algorithms for eight measures covering load balance, interprocessor communication, flexibility, ease of use and speed. Based on the comparison results, we recommend the use of the simplest and fastest (P×Q) of the six algorithms considered for sequential compile-time mapping of 2-dimensional meshes.

1 Introduction

The data-parallel single-phase iterative Partial Differential Equation (PDE) solvers considered in this paper are based on mapping the discrete PDE operator (i.e., a linear system of algebraic equations, Ax=b) and the associated computations onto the **P** processors of a multicomputer. With the (most commonly used) single program multiple data programming model, processors execute the same program independently on parts of the linear system mapped on to them. That is, processor P_i computes the unknowns x^i of the sub-system $A^i x^i = b^i$ and communicates with other processors when nonlocal or global data are needed. Thus, the execution time of the data-parallel solver is given by

$$T_{solver} = \max_{1 \le i \le \mathbf{P}} \{T^i_{compute} + T^i_{communicate} + T^i_{synchronize}\} \quad (1)$$

assuming that computation and communication do not overlap. Equation (1) is particularly relevant for the loosely synchronous class of iterative solvers considered in this work.

For parallel iterative PDE solvers, the data mapping problem can be formulated at two levels : (i) the discrete geometrical data structures (element-meshes or tensor-grids) associated with the PDE domain and (ii) the linear system of algebraic equations associated with some discretization of the PDE equations. In this paper we evaluate data mapping strategies based on geometrical data structures [7]. These strategies are based on partitioning the mesh D^h representing the PDE domain and allocating the resultant submeshes to the multicomputer processors. The partitioning results in splitting the discrete equations associated with the mesh nodes and their interfaces. Figure 1 describes such a partitioning.

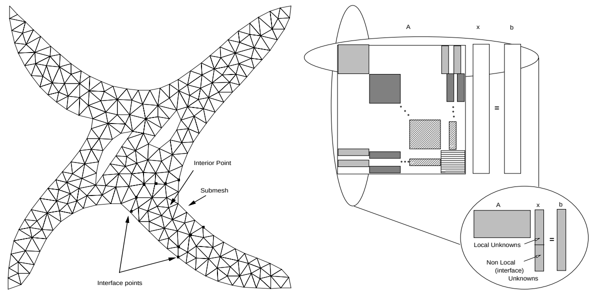

Figure 1: The components of the partitioned discrete PDE problem based on the splitting of the mesh D^h used numerically.

The minimization of the execution time, T_{solver}, of data-parallel iterative solvers requires equal distribution of the computation workload and minimization of overheads due to communication of nonlocal unknowns, update of global parameters, and test of convergence (synchronization). The problem of mapping data for minimizing T_{solver} is an intractable optimization problem. Thus, several algorithms have been proposed for finding good suboptimal mapping solutions. Some algorithms are based on greedy schemes, divide-and-conquer, or block partitioning. Examples are nearest neighbor mapping, P×Q partitioning, recursive coordinate bisection, recursive graph bisection, recursive spectral bisection, CM_Clustering, and scattered decomposition [1], [7], [3], [20], [9], [26], [13], [24], [27].

Other algorithms are based on deterministic optimization, where local search techniques are used to minimize cost functions related to T_{solver}; examples are Kernighan-Lin algorithm and geometry graph partitioning [18], [7]. Yet, another class of mapping algorithms are based on physical optimization that employs techniques from natural sciences; examples are neural networks, simulated annealing, and genetic algorithms [16], [10], [11] [21], [28].

Although a good deal of work has been published on data mapping, only a few attempts have been made at comparing some algorithms using aggregate or a limited number of performance measures [7], [27], [22], [28]. In this paper, we use several measures to evaluate and compare the performances of six data mapping algorithms for irregular iterative PDE computations. The algorithms considered are: (1) the P×Q partitioning, (2) the recursive spectral bisection, (3) the geometry graph partitioning, (4) a neural network algorithm, (5) a simulated annealing, and (6) a genetic algorithm. These algorithms have been chosen since they are among the best known data mapping algorithms in the literature. We report experimental machine-dependent and machine-independent results, obtained using DecTool [4] and Parallel ELLPACK, [15], for the evaluation of their performance. The experimental results and the operation of the algorithms are employed to produce a table that compares the algorithms on eight measures: (a) load balance, (b) submesh connectivity, (c) splitting of submeshes, (d) message size, (e) interprocessor distance traveled by messages, (f) flexibility, (g) dependence on parameters, and (h) execution time. The comparison leads us to recommend the use of the P×Q algorithm for sequential mapping of 2-dimensional meshes for iterative PDE solvers. A more detailed analysis of the algorithms and discussion on the above measures appears in [8].

This paper is organized as follows. Section 2 presents objective functions and identifies two approaches for the data mapping problem. Section 3 gives a review of the six mapping algorithms. Section 4 presents and discusses the experimental results. Section 5 presents a summary of the findings. Section 6 concludes the paper.

2 Data Mapping

In this section, we present objective functions and a number of criteria for the data mapping and describe two approaches in addressing the mapping problem. An objective function that reflects the cost of mapping a mesh D^h (with $|D^h| = N$) onto a multicomputer can typically be formulated as:

$$OF_{typ} = \max_{1 \le i \le \mathbf{P}} \{ \; W(m(D_i^h)) + \sum_{D_j^h \in \kappa_{D_i^h}} C(m(D_i^h), m(D_j^h)) \; \} \tag{2}$$

where $m : \{D_i^h\}_{i=1}^{\mathbf{P}} \rightarrow \{P_i\}_{i=1}^{\mathbf{P}}$ is a function that maps the nodes of submesh D_i^h to the processors; $W(m(D_i^h))$ is the computational load of the processor $m(D_i^h)$ per iteration, which is proportional to the number of nodes in D_i^h; $C(m(D_i^h), m(D_j^h))$ is the cost of the communication required (per iteration) between the processors $m(D_i^h)$ *and* $m(D_j^h)$; finally, $\kappa_{D_i^h}$ is the set of submeshes that are adjacent to D_i^h and its cardinality $|\kappa_{D_i^h}|$ is henceforth referred to as the submesh connectivity. The formulation of OF_{typ} assumes that computation and communication do not overlap. OF_{typ} approaches its minimum if the computation load $W(P_i)$ is near-evenly distributed among the processors and the communication cost of the processors is minimum. Clearly, such conditions are also necessary for minimizing T_{solver} (equation 1). However, the synchronization term in T_{solver} is not explicitly reflected in OF_{typ} because synchronization cost is a nonlinear function of communication, computation, and communication-computation overlapping. Thus, it is difficult to express quantitatively. Nevertheless, OF_{typ} is considered a reasonable measure for the quality of data mapping solutions and two approaches can be identified in the mapping literature for its minimization.

The first mapping approach is based on the expansion of the components of OF_{typ} and the use of explicit machine-dependent and algorithm-dependent parameters. This approach is adopted in the physical optimization methods which are guided by an objective function. However, OF_{typ} is not a smooth function and its minimization gives rise to a minimax criterion which is computationally expensive. To avoid these two shortcomings, the following approximate objective function is used:

$$OF_{appr} = \lambda^2 \sum_{i=1}^{\mathbf{P}} |D_i^h|^2 + \mu \sum_{i=1}^{\mathbf{P}} \sum_{D_j^h \in \kappa_{D_i^h}} C(m(D_i^h), m(D_j^h)) \tag{3}$$

where μ is a scaling factor expressing the relative importance of the communication term with respect to the computation term, and λ is dependent on the solver and is equal to the number of computation operations per mesh node per iteration.

Although OF_{appr} is not equivalent to OF_{typ}, it still represents a good approximation to the cost of a mapping configuration. Its first term is quadratic in the deviation of computation loads from the average computation load and is minimal when all deviations are zero. A minimum of the second term occurs when the sum of all interprocessor communication costs is minimized.

The second mapping approach uses criteria that are qualitatively derived from the mapping requirements and addresses them in stages. It is based on splitting the optimization problem into two distinct phases that accomplish the *partitioning* and the *allocation* of the mesh [6] and [27]. In the *partitioning phase* we decompose the mesh into P submeshes such that the following criteria are approximately satisfied:

(i) the maximum difference in the number of nodes of the submeshes is minimum,

(ii) the ratio of the number of interface nodes to the number of interior nodes for each submesh is minimum,

(iii) the number of submeshes that are adjacent to a given submesh is minimum,

(iv) each submesh is a connected mesh.

In the *allocation phase* these submeshes are allocated to the processors such that the following criterion is satisfied:

(v) the communication requirements of the underlying computation between the processors of a given architecture are minimum.

For a given mesh D^h with N nodes, the merit of a partition into $\mathbf{P}$ non-overlapping submeshes $\{D_i^h\}_{i=1}^{\mathbf{P}}$ is characterized in terms of the set of geometrically adjacent submeshes $\kappa_{D_i^h}$ to submesh D_i^h and the number of interface mesh nodes, $I(D_i^h, D_j^h)$, shared by the submeshes D_i^h *and* D_j^h. Then, the optimal partitioning, as defined by criteria (i) to (iv), can be viewed as the one which simultaneously minimizes :

$$\max_{1 \le i,j \le \mathbf{P}} \left|\, |D_i^h| - |D_j^h| \,\right| \qquad (6)$$

$$\max_{1 \le i \le \mathbf{P}} \left\{ \frac{\left(\sum_{D_j^h \in \kappa_{D_i^h}} I(D_i^h, D_j^h)\right)}{|D_i^h|} \right\} \qquad (7)$$

$$\max_{1 \le i \le \mathbf{P}} |\kappa_{D_i^h}| \qquad (8)$$

3 Data Mapping Algorithms

In this section we briefly review six algorithms for the solution of the data mapping problem, namely : (1) the P×Q algorithm, (2) the recursive spectral bisection algorithm, (3) the geometry graph partitioning algorithm, (4) a neural network algorithm, (5) a simulated annealing algorithm, and (6) a genetic algorithm. The last three algorithms, which are physical optimization algorithms, are based on the first mapping approach described in Section 2 while the first three algorithms adopt the second approach.

3.1 P×Q Partitioning

A simple and attractive mapping method considered by many researchers (see [2], [10], and [6]) is the so-called data strip or block partitioning heuristic. This heuristic is referred under different names, some of them are : one-dimensional (1D) strip partitioning, two-dimensional (2D) strip partitioning, multilevel load balanced method, median splitting, and sector splitting. Throughout this paper, we are referring to this clustering algorithm as P×Q [7], where P is the number of sub-meshes (blocks or strips) along the x-axis, Q is the number of sub-meshes (blocks or strips) along the y-axis, and P×Q = $\mathbf{P}$ (for 2D domains). The algorithm based on two sorts of the node points : (1) sort the node points along the x-coordinate axis (2) group the node points into P subgroups, (3) sort the points of each subgroup along the y-coordinate axis, and (4) group the node points of each of the P subgroups into Q subgroups. In [7] the algorithm is generalized by using boundary-conforming curvilinear coordinate systems.

3.2 Spectral Bisection

Recursive spectral bisection (RSB) utilizes the spectral properties of the Laplacian matrix associated with the mesh for bisecting it [27]. It recursively applies the bisection step $log_2\mathbf{P}$ times and allocates the generated 2-dimensional submeshes to the corresponding subcubes in the multicomputer.

The Laplacian matrix L(M) is defined as:

$$L_{i,j}(M) = \begin{cases} +1 & \text{if e(i,j) exists} \\ -degree(\ of\ vertex\ i\ ) & \text{if i = j} \\ 0 & \text{otherwise} \end{cases}$$

In each bisection step the eigenvector corresponding to the second largest eigenvalue of the Laplacian matrix is computed; this vector is called Fiedler vector and can be computed using the Lanczos algorithm. The components of this vector provide distance information about the nodes of the mesh. Then, the nodes are sorted according to the values of the eigenvector's components. Using the sorted list, the nodes are split to form two equal-size submeshes.

3.3 Geometry Graph Partitioning

The *geometry graph partitioning* (GGP) heuristic is a local optimization algorithm. A local optimization algorithm for given initial solution t and neighborhood structure $N(t)$ performs local search of the the neighborhood $N(t)$ and replaces the current solution t with a neighbor solution u of t that optimizes (minimizes or maximizes) the cost function f, [25]. This process is repeated until no such better solution exists. At this point a "locally optimal" solution has been identified. The GGP heuristic uses the geometrical properties of the mesh graph (Euclidean graph) in order to deliver quasi-uniform partitionings with the minimal diameter.

The GGP algorithm's profit function is a weighted combination of the KL algorithm's profit function and of a function which is used in selecting pairs of node points whose swapping reduces the diameter of the subdomains. The GGP algorithm climbs out of local minima of the cost function by swapping points that might increase temporarily the value of the cost function but will decrease the diameter of the subdomains by bringing their mass centers far apart.

3.4 Genetic Algorithms

In genetic algorithms a population of candidate solutions, called individuals, evolve over successive generations, starting with random solutions. In every generation, individuals are selected for reproduction according to their fitness, then genetic operators are applied to the selected mates, and offspring replace their parents. In this process, fitness is gradually increased and optimal solutions evolve by the propagation and the combination of high-performance fit building blocks [14]. The genetic algorithm for data mapping encodes an individual as a string of N integers, where an integer refers to a processor and its position in the string represents the mapped mesh node. The fitness of an individual is the reciprocal of the value of the objective function, so that maximizing the fitness would correspond to minimizing the objective function.

The reproduction scheme determines which individuals survive and selects pairs of surviving individuals for reproduction. The genetic operators employed in GA are two-point crossover and mutation. Crossover is accomplished by randomly selecting equal-length substrings in the two parents and swapping them. Mutation refers to randomly remapping a randomly chosen mesh node. Crossover is applied to 70% of the individuals in the population and the rate of mutation used is 0.3%.

The last step in creating a new generation is a greedy hill-climbing procedure applied to all offspring solutions for improving their structure. The procedure considers all interface mesh nodes in a candidate solution and allows remapping of interface nodes only from overloaded to underloaded processors. That is, remapping is invoked only if ΔOF is negative.

3.5 Simulated Annealing

The SA starts with an initial random mapping solution which corresponds to a system in a high energy/temperature state, where the energy is given by the objective function OF_{appr}. The SA algorithm then reduces the temperature of the system gradually to a freezing point according to a cooling schedule. At each temperature, regions in the solution space are searched by the Metropolis algorithm [19]. An iteration of the Metropolis algorithm starts with proposing a random perturbation and evaluating the resultant change in OF_{appr}. A perturbation, or a move, is accomplished by a random remapping of a randomly chosen mesh node. A remapping that leads to a lower objective function value corresponds to a downhill move in the energy landscape and is always accepted. An increase in objective function (uphill move) may be accepted only with a temperature-dependent probability, $e^{-\Delta OF/\theta}$.

Perturbations are repeated at each temperature until thermal equilibrium. Equilibrium is reached when the number of attempted or accepted perturbations is equal to predetermined maximum numbers. Perturbations followed by the computation of ΔOF occur in every inner iteration of the SA algorithm. Hence, it is important to compute ΔOF as efficiently as possible.

3.6 Neural Network Algorithm

A Hopfield-type Neural Network for data mapping, described in [11] and [22], aims at quickly finding low minima for the objective function. The network is represented by a matrix of neurons. Each row corresponds to a mesh node v. The number of neurons per row is equal to $\log_2 \mathbf{P}$. Each neuron is associated with a neural variable $n(v, i)$, where i refers to column i in the network.

The NN starts with initial random neural values and converges to a fixed point, after a number of sweeps. The fixed point of the network is associated with a minimum of the energy function, OF_{appr}. The NN repeats this procedure $\log_2 \mathbf{P}$ times, each time determining the bits in column i in the network and, hence, the subcubes to which the mesh nodes are mapped. After the last iteration, the mesh will be partitioned into submeshes mapped to the P processors.

4 Performance Evaluation

In this section, we present and discus the machine-independent and machine-dependent performance analysis for the two data mapping approaches and the following algorithms :

P×Q	:	Partitioning along the x and y direction.
RSB	:	Recursive Spectral Bisection.
GGP	:	Geometry Graph Partitioning.
GA	:	Genetic Algorithm.
SA	:	Simulated Annealing.
NN	:	Neural Network.

The evaluation of these data mapping algorithms is performed on a Model Problem with a Poisson PDE operator and Dirichlet boundary conditions (the data mapping is independent of the PDE operator). The domain of the Model Problem is an irregular non-convex domain with two holes as shown in Figure 2. The mesh , M_{13K}, consists of 24,202 elements and 12,724 nodes and the PDE is approximated by a linear system with 11,676 number of equations. The PDE is discretized by a bilinear finite element method and the linear system is solved using a Jacobi Semi Iterative (Jacobi-SI) method [5].

4.1 Machine-Independent Analysis

The machine-independent measures considered are : (i) the submesh connectivity, (ii) the number of interface nodes , (iii) the splitting of the submeshes, (iv) the Hamming distance among communicating processors, and (v) the load balance. The analysis is based

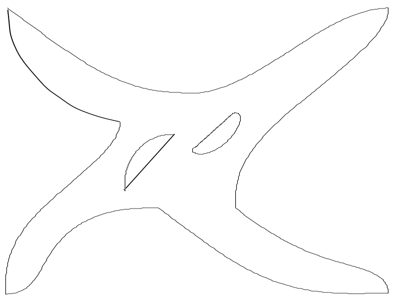

Figure 2: Model Problem.

on the solutions for mapping M_{13K} to nCUBE II with 8 to 128 processors. From these solutions the maximum values for the different measures are computed and plotted.

Figure 3 shows the maximum number of the total submesh interface nodes; the length of the interfaces is proportional to the the message size term of the communication cost (equation 3). Figure 3 shows that GGP and RSB yield the smallest number of interface nodes. RSB minimizes node separators in the mesh while GGP at the same time maximizes the inter-center distance of submeshes, and thus reduces the size of node separators even more. GA, SA and P×Q also yield good number of interfaces, whereas NN yields the largest number of interfaces. However, the graph contraction pre-mapping step [23] used for speeding-up the three physical optimization algorithms does increase the length of the submesh interfaces due to the ill-shaped (contracted) super-nodes it produces.

Figure 4 shows the maximum submesh connectivity; the total message latency is proportional to the submesh connectivity. Figure 4 indicates that GA, GGP and RSB yield very good connectivities. This is expected for GA since its objective function explicitly includes a significant message latency cost The minimum node separator requirement sought by GGP and RSB seems to help in minimizing submesh connectivity for 2-D meshes.

Figure 5 shows the maximum distances for messages among communicating processors; longer distances for messages in circuit-switching machines increase the probability of link-contention and thus increase the communication time. SA nad NN show very good distances since interprocessor distances are included in their objective functions with a large weight [22] P×Q, RSB, and GGP also show good distances, whereas GA's distances are acceptable.

Figure 6 gives the standard deviation of the number of nodes per submesh of M_{13K}; M_{13K} is partitioned into 6 submeshes. The deviation values il-

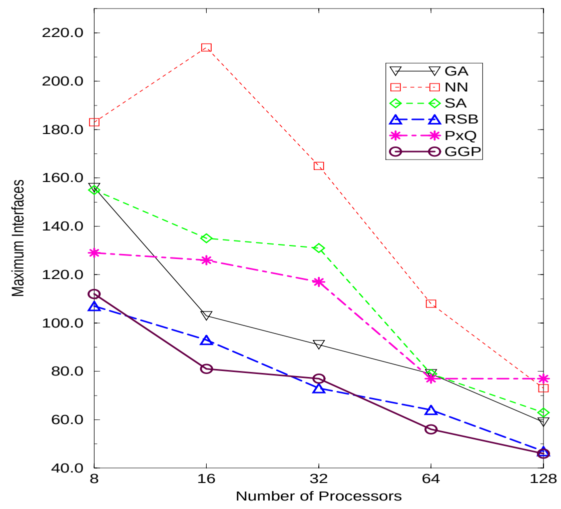

Figure 3: Maximum number of interface nodes for the mapping solution of M_{13K}.

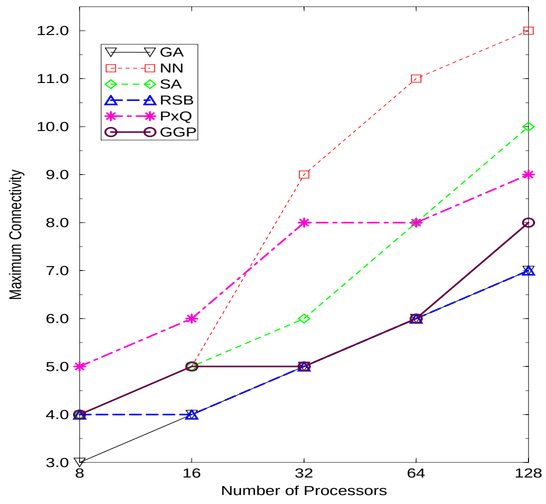

Figure 4: Maximum connectivity of the submeshes for the mapping solution of M_{13K}.

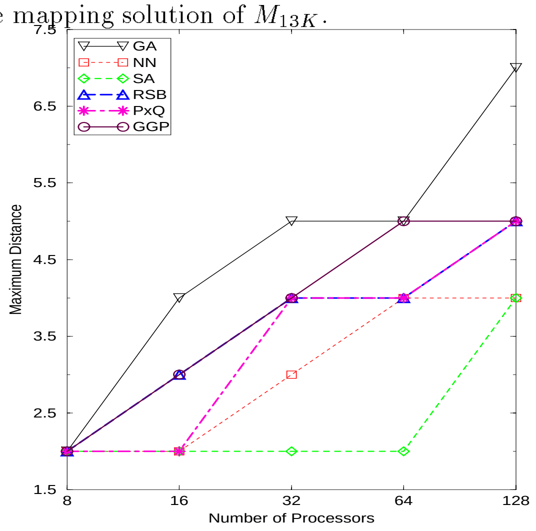

Figure 5: Maximum distances among the communicating processors for the mapping solution of M_{13K}.

lustrate how well-balanced is the computational load. Clearly, P×Q, RSB and GGP produce mapping that are perfectly load balanced since these algorithms first optimize this criterion. The three physical algorithms do not insist on perfect load balance. Instead, their aim is to minimize the total sum of both the computational load and communication cost. Although they do not produce mapping with large imbalances they offer a tradeoff between the computation load and the communication cost of the individual processors for the aim of minimizing the total workload of the slower processors.

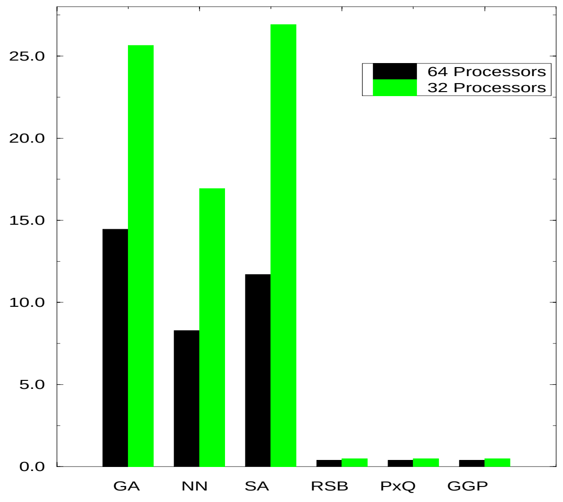

Figure 6: Standard deviation of the number of nodes per submesh for $\mathbf{P}$ = 32 and 64.

Figures 7 shows the submeshes produced by the six data mapping algorithms. These solutions show disconnected subdomains for NN, SA, GA and P×Q, but not for GGP and RSB. GGP uses profit functions that try to prevent disconnectedness.

4.2 Machine-Dependent Analysis

The machine-dependent measures are : the total elapsed execution time of the PDE solver (T_{solver}) and the interprocessor communication time ($T_{communicate}$). We have run the solver for M_{13K} using the mapping solutions on 32 and 64 processors.

Tables 1 and 2 present the maximum, mean, standart deviation values for T_{solver} and $T_{communicate}$. From these tables we observe the following : (1) The difference in maximum T_{solver} between the best and worst values is 15%, except for the NN value on 64 processors (25%). (2) The processor with maximum $T_{communicate}$ is not always the processor with maximum T_{solver} even for the algorithms with perfect load balance. According to the model (see equation 2) that

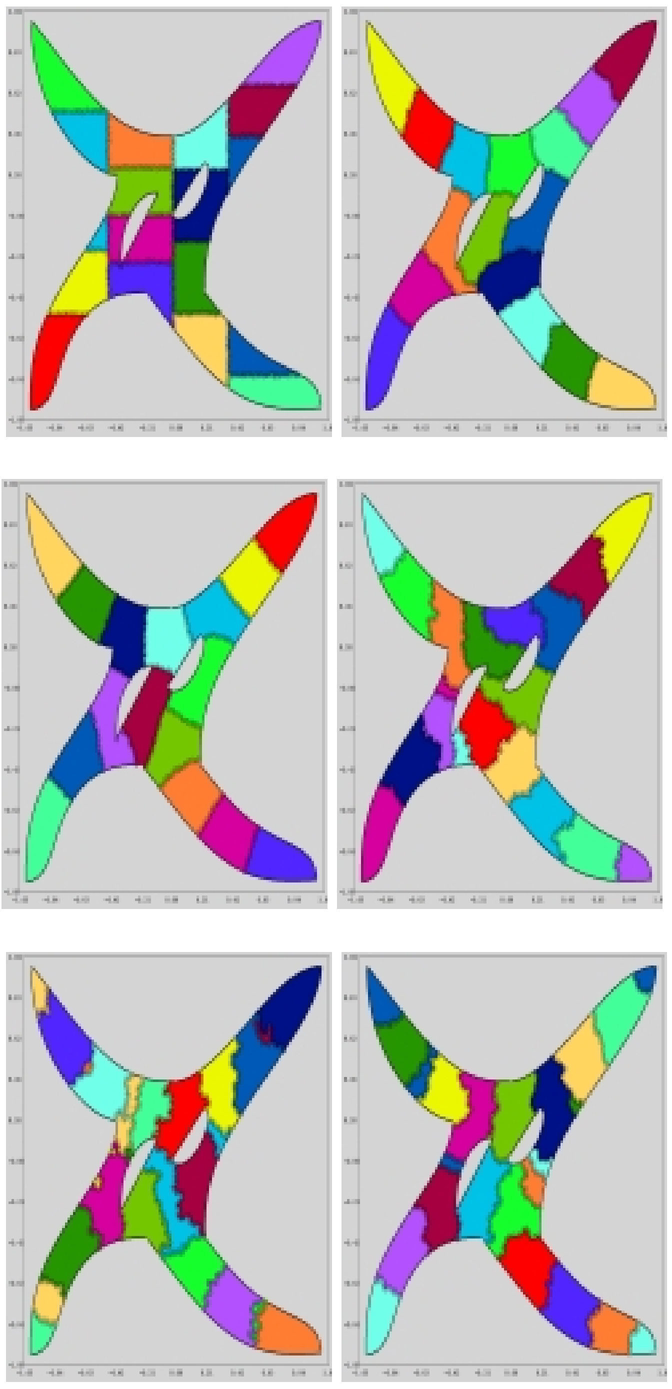

Figure 7: 16 submeshes produced by the P×Q (top left) RSB (top right), GGP algorithm (midle left) the GA (midle right), the NN (bottom left) and SA (bottom right) for the mesh M_{13K}.

is usually adopted in the literature, the processor with the maximum $T_{communicate}$ is the slowest processor. In our experiments we see a deviation from this logic because of overheads due to imperfect work load (computation & communication) balance and synchronization.

The first observation, the machine-independent analysis, and the fact that the P×Q's execution time is only few seconds, while all the other algorithms' execution time is between several minutes to several hours indicate that the P×Q is the most suitable algorithm for the sequential compile-time data mapping of 2-dimensional irregular meshes for the solution of PDE problems on distributed memory MIMD machines. The second observation indicates that the model (see equation 2) that is usually adopted in the literature is not complete. This model ignores the effects of link-contention as well as blocking (idle) time due to synchronization.

Table 1: Elapsed & Communication Time for the the Model Problem on 32 processors.

Algs	Elapsed Time			Comm Time		
	max	mean	dev	max	mean	dev
P×Q	3.433	3.428	0.003	0.606	0.511	0.105
RSB	3.421	3.415	0.004	0.844	0.517	0.160
GGP	3.403	3.398	0.004	0.826	0.499	0.162
NN	3.839	3.760	0.014	1.289	0.819	0.230
SA	3.697	3.687	0.006	1.347	0.778	0.294
GA	3.610	3.550	0.021	1.169	0.657	0.279

Table 2: Elapsed & Communication Time for the the Model Problem on 64 processors.

Algs	Elapsed Time			Comm Time		
	max	mean	dev	max	mean	dev
P×Q	2.13	1.95	0.044	0.777	0.470	0.080
RSB	1.94	1.93	0.002	0.734	0.465	0.101
GGP	1.90	1.89	0.002	0.669	0.423	0.108
NN	2.37	2.17	0.034	0.958	0.657	0.109
SA	1.95	1.94	0.002	0.755	0.473	0.104
GA	2.18	2.05	0.027	0.993	0.582	0.164

5 Summary of Results

Based on the results described in Section 4, Table 3 summarizes the major properties of the six mapping algorithms. Note that the table reflects the quality and timings of the contracted graphs for NN, SA and GA.

Algs	P×Q	RSB	GGP	NN	SA	GA
load bal.	perf.	perf.	perf.	v. g.	v. g.	v. g.
connect.	good	v. g.	v. g.	accpt.	good	v. g.
interf.	good	v. g.	v. g.	accpt.	good	good
dist.	good	good	good	v. g.	v. g.	accpt
comp. -comm.	no	no	no	limit.	yes	yes
discon. subdom.	likl.	likl.	less likl.	likl.	likl.	likl.
depend. on pp.	no	no	no	yes	yes	yes
mapping speed	very fast	slow	very slow	fast	extr. slow	very slow

Table 3: Summary of results, with v. g. meaning very good, accpt. meaning acceptable, limit. meaning limited, extr. meaning extremely, perf. meaning perfect, likl. meaning likely, pp meaning problem parameters, discon. meaning disconnected, and comp-comm meaning computation to communication tradeoff. The the speed of the mapping depends on the algorithm implementation, data structures, optimizations, and even compilers used, thus we prefer to present a relative comparison.

6 Summary and Conclusions

We have presented performance evaluation results for six mapping algorithms used for PDE computations on irregular 2-dimensional meshes. The experiment results are concerned with the performance of the algorithms for eight measures. The properties of the algorithms are summarized in table 3 which can be used for selecting a mapping algorithm that suits different application requirements. For example, for applications where the same mesh is used many times, mapping algorithms with slower execution time and better solution quality can be chosen.

However, we have found that the machine-dependent performances of the algorithms do not differ by a great amount. Further, Table 3 shows that the algorithms that satisfy the mapping criteria to a better degree are slow (eg. GGP, RSB) and/or involve intricate parameter-dependence (eg. GA, SA). These findings lead us to recommend that for sequential compile time mapping of 2-dimensional meshes, the very fast and simple P×Q partitioning algorithm should be chosen.

Acknowledgements

The first author gratefully acknowledges the Alex G. Nason Foundation for the Nason Prize Award that supports him at NPAC. Also, the authors thank Horst Simon for providing the RSB code. The 64 processor nCUBE II at Purdue University was used for the performance evaluation.

References

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

[2] Shahid H. Bokhari. On the mapping problem. *IEEE Transactions on Computers*, (3):207 – 213, 1981.

[3] N. P. Chrisochoides, C. E. Houstis, and E. N. Houstis. Geometry based mapping strategies for PDE computation. In E. N. Houstis and D. Gannon, editors, *Proceedings of International Conference on Supercomputing*, pages 115-127. ACM Press, 1991.

[4] N. P. Chrisochoides, C. E. Houstis, P. N. Papachiou E. N. Houstis, S. K. Kortesis, and J. R. Rice. Domain decomposer: A software tool for mapping PDE computations to parallel architectures. In R. Glowinski et al., editors, *Domain Decomposition Methods for Partial Differential Equations IV*, pages 341–357, SIAM Publications, 1991.

[5] N. P. Chrisochoides, E.N. Houstis, S.B. Kim, M.K. Samartzis, and J.R. Rice. Parallel iterative methods. In *Advances in Computer Methods for Partial Differential Equations VII*, (R. Vichnevetsky. D. Knight and G. Richter, eds) IMACS, New Brunswick, NJ, pages 134-141, 1992.

[6] N. P. Chrisochoides. *On the Mapping of PDE Computations to Distributed Memory MIMD Machines.* CSD-TR-92-101, Computer Science Department, Purdue University, W. Lafayette IN, 1992.

[7] N. P. Chrisochoides, Elias Houstis and John Rice. *Mapping Algorithms and Software Environment for Data Parallel PDE Iterative Solvers* To appear in the Special Issue of the Journal of Parallel and Distributed Computing on Data-Parallel Algorithms and Programming, Vol 21, No 1, April 1994.

[8] N. P. Chrisochoides, Nashat Mansour and Geoffrey Fox. *Performance evaluation of load balancing algorithms for parallel single-phase iterative PDE solvers* Submitted to the Journal of Concurrency Practice and Experience, SCCS-551, 1993.

[9] C. Farhat. A simple and efficient automatic fem domain decomposer. *Computers and Structures*, 28:579–602, 1988.

[10] J. Flower, S. Otto, and M. Salana. Optimal mapping of irregular finite element domains to parallel processors. *Parallel Computers and Their Impact on Mechanics*, 86:239–250, 1988.

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

[12] G. C. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon and D. Walker *Solving problems on concurrent processors.* Prentice Hall, New Jersey, 1988.

[13] G. C. Fox. A review of automatic load balancing and decomposition methods for the hypercube. In *Proceedings of the IMA Institute* (M. Schultz, editor), pages 63–76. Springer–Verlag, 1986.

[14] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley. 1989.

[15] E. N. Houstis, J. R. Rice, N. P. Chrisochoides, H. C. Karathanasis, P. N. Papachiou, M. K. Samartzis, E. A. Vavalis, Ko-Yang Wang, and S. Weerawarana. //ELLPACK: A numerical simulation programming environment for parallel MIMD machines. In *Proceedings of Supercomputing '90* (J. Sopka, editor), pages 97–107. ACM Press, 1990.

[16] E. N. Houstis, S. K. Kortesis, and H. Byun. A workload partitioning strategy for PDEs by a generalized neural network. Technical Report CSD–TR–934, Department of Computer Sciences, Purdue University, 1990.

[17] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*,

[18] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, Feb., 291 – 307, 1970.

[19] S. Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

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

[21] Nashat Mansour and Geoffrey Fox. A Hybrid Genetic Algorithm for Task Allocation in Multicomputers. *International Conference on Genetic Algorithms*, pp 466-473, July 1991, Morgan Kaufmann Publishers.

[22] Nashat Mansour and Geoffrey Fox. Allocating Data to Multicomputer Nodes by Physical Optimization Algorithms for Loosely Synchronous Computations. *Concurrency: Practice and Experience*, Vol. 4, Number 7, pp 557-574, October 1992.

[23] N. Mansour, R. Ponnusamy, A. Choudhary, and G. Fox. Graph Contraction for Physical Optimization Methods: A Quality-Cost Tradeoff for Mapping Data on Parallel Computers. *International Supercomputing Conference*, Japan, July 1993, ACM Press.

[24] R. Morrison and S. Otto. The scattered decomposition for finite elements. *Journal of Scientific Computing*, 2:59–76, 1987.

[25] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization Algorithms and Complexity.* Prentice-Hall, Englewood Cliffs, NJ 07632, 1982.

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

[27] D. Horst Simon. Partitioning of unstructured problems for parallel processing. Technical Report RNR-91-008, NASA Ames Research Center, Moffet Field, CA, 94035, 1990.

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