A Comparison of Data Mapping Algorithms for Parallel 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
corresponding author: Nikos Chrisochoides (nikos@npac.syr.edu)

SUMMARY

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

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

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. In the loosely synchronous model, computations are carried out in phases. Each phase consists of computations on the local subproblem followed by interprocessor communication for nonlocal data [FJL88], [Fox91].

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 [Chr93]. 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.

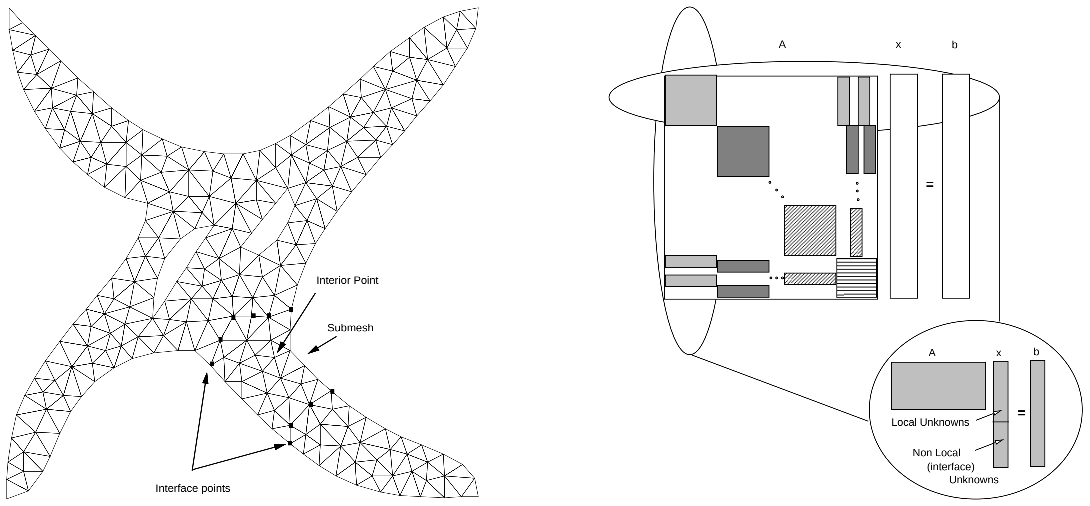


Figure 1: The components of the partitioned discrete PDE problem based on the splitting of the mesh D^h used numerically.

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$\times$Q partitioning, recursive coordinate bisection, recursive graph bisection, recursive spectral bisection, CM_Clustering, and scattered decomposition [BB87], [CHENHR89], [Chr93], [KR91], [DG89], [Erc88], [1] [SE87], [SE87], [Far88], [Fox86a], [Fox86b], [MO87], [Sim90], [Wal90].

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 [KL70] and [CHENHR89]. Yet, another class of mapping algorithms are based on physical optimization that employs techniques from natural sciences [Fox91b]; examples are neural networks, simulated annealing, and genetic algorithms [HKB90], [FOS88], [FF88] [MF91], [Wil91].

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 [CHENHR89], [Chr93], [Sim90], [MF92], [Wil91], [S92]. 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$\times$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 [CHENH+91] and Parallel ELLPACK, [HRC+90], 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$\times$Q algorithm for sequential mapping of 2-dimensional meshes for iterative PDE solvers.

This paper is organized as follows. Section 2 briefly describes the communication requirements of a class of parallel iterative PDE solver employed in this work. Section 3 presents objective functions based on the and communication requirements outlined in Section 2 and identifies two approaches for the data mapping problem. Section 4 gives a review of the six mapping algorithms. Section 5 presents and discusses the experimental results. Section 6 presents a summary of the findings. Section 7 concludes the paper.

2. COMMUNICATION REQUIREMENTS FOR PARALLEL ITERATIVE PDE SOLVERS

The iterative PDE solvers for the solution of a discrete linear system of algebraic equations can be reduced to matrix-vector multiplication operations (see [HY81] and [KRYG82]). The parallel processing and implementation of matrix-vector multiplication operations consists of two steps : (a) the *local communication* and (b) the *local computation* (see [FJL88], [CAHH92]). A high level description of the steps of an iterative solver (that preserves the ordering of the corresponding sequential computation) pertinent to the data mapping is the following : (i) *Local Communication*, (ii) *Local Computation*, and (iii) *Global Synchronization.* The local communication consists of an exchange of messages between the processors of the parallel machine; the messages transfer some of the local data (i.e., interface unknowns) required by the neighbor subdomains. The local computation mainly consists of matrix-vector and vector-vector operations. Finally, the global synchronization consist of reduction operations that are required for the acceleration of convergence and for the checking of stopping criteria [CHK+92].

3. DATA MAPPING

In this section, we present objective functions and criteria for 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:

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

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:

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

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. Further, OF_{appr} enjoys smoothness and computational locality (i.e., a change

ΔOF due to remapping node v from P_i to P_j is determined by information about v, P_i and P_j only). Also, we note that OF_{appr} shares with OF_{typ} the ability to allow a tradeoff between the computation workload and the communication cost for the purpose of minimizing their total sum.

The cost of interprocessor communication, $C(m(D_i^h), m(D_j^h))$, is difficult to express accurately. It depends on several hardware and software components of a multicomputer, some of which might be impossible to quantify. In this work , we use two expressions for $C(m(D_i^h), m(D_j^h))$. One expression has been proposed for modern multicomputers [Bok90], [Hey90]:

$$C_p(m(D_i^h), m(D_j^h)) \ = \sigma \ + \rho \ I(D_i^h, D_j^h) \ + \tau \ H(m(D_i^h), m(D_j^h)) \tag{4}$$

where σ is the message start-up time (latency); ρ is the machine time for communicating one byte; τ is the communication time per unit distance; $I(D_i^h, D_j^h)$ is the number of interface nodes of the submeshes D_i^h and D_j^h that determines the message size; $H(D_i^h, D_j^h)$ is the physical (e.g. Hamming) distance between $m(D_i^h)$ and $m(D_j^h)$. Note that the inclusion of σ in equation (4) accounts for the cost of the connectivity of the submeshes.

The second expression for the communication cost between the processors $m(D_i)$ and $m(D_j)$ is based only on the physical distance between processors and the message size:

$$C_d(m(D_i^h), m(D_j^h)) \ = \rho \ I(D_i^h, D_j^h) \ H(m(D_i^h), m(D_j^h)) \tag{5}$$

$C_d(m(D_i^h), m(D_j^h))$ appeared in the literature in mid 80's and is relevant only for early multicomputer machines [Hey90]. Nevertheless, its advantage is that computing its incremental change, ΔC_d, is faster than computing ΔC_p. Since the physical optimization algorithms considered in this work employ incremental changes as a basic step, efficient computation of such a change becomes important for the efficiency of the physical algorithms.

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 [CHENHR89], [CHH90], [Chr92] and [Sim90]. 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 :

$$\max_{1 \leq i,j \leq \mathbf{P}} |\, |D_i^h| - |D_j^h|\, | \tag{6}$$

$$\max_{1 \leq i \leq \mathbf{P}} \{ \frac{(\sum_{D_j^h \in \kappa_{D_i^h}} I(D_i^h, D_j^h))}{|D_i^h|} \} \tag{7}$$

$$\max_{1 \leq i \leq \mathbf{P}} |\kappa_{D_i^h}| \tag{8}$$

3. COMMUNICATION REQUIREMENTS FOR PARALLEL ITERATIVE PDE SOLVERS

The iterative PDE solvers for the solution of a discrete linear system of algebraic equations can be reduced to matrix-vector multiplication operations (see [HY81] and [KRYG82]). The parallel processing and implementation of matrix-vector multiplication operations consists of two steps : (a) the *local communication* and (b) the *local computation* (see [FJL88], [CAHH92]). A high level description of the steps of an iterative solver (that preserves the ordering of the corresponding sequential computation) pertinent to the data mapping is the following : (i) *Local Communication*, (ii) *Local Computation*, and (iii) *Global Synchronization.* The local communication consists of an exchange of messages between the processors of the parallel machine; the messages transfer some of the local data (i.e., interface unknowns) required by the neighbor subdomains. The local computation mainly consists of matrix-vector and vector-vector operations. Finally, the global synchronization consist of reduction operations that are required for the acceleration of convergence and for the checking of stopping criteria [CHK+92].

4. 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. But, the first three algorithms adopt the second approach.

4.1 P×Q Partitioning Algorithm

A simple and attractive mapping method considered by many researchers (see [Bok81], [SE87], [FOS88], [PAF90] and [Chr92]) 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 [Chr93], 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 = **P** (for 2D domains). A description of the P×Q algorithm for a 2D mesh (with N = k **P**) points is given below.

P×Q Algorithm

```
Sort the mesh points along the x-coordinate axis;
for i = 1 to P do
    Assign the ith set of N/P points to the l = gray_code_1d(i) sub-mesh
endfor
Sort the mesh points along the y-coordinate axis;
for i = 1 to P do
    for j = 1 to Q do
        Assign the jth set of N/P points of the l sub-mesh to the
        k = gray_code_2d(i,j) sub-mesh
endfor
```

The P×Q algorithm often produces submeshes with more than one connected component when partitioning non-convex 2D meshes. To avoid the splitting of the sub-meshes for star-shaped non-convex 2D domains Chrisochoides et. al. in [CR92] present the *boundary conforming P×Q algorithm* which uses *boundary-conforming curvilinear coordinate systems.* instead of Cartesian coordinate systems.

4.2 Spectral Bisection

Recursive spectral bisection (RSB) utilizes the spectral properties of the Laplacian matrix, L(M), associated with the mesh M [PSL90],[Sim90]. It recursively bisects $log_2\mathbf{P}$ times and allocates the generated 2-dimensional submeshes to the processors of the multicomputer. The submeshes are allocated to the processors using allocation methods presented in [CHH90].

The Laplacian matrix L(M) is defined as:

$$L_{i,j}(M) = \begin{cases} +1 & \text{if vertex i and j are joined by an edge (i,j)} \\ -degree(\ of\ vertex\ i\ ) & \text{if i = j} \\ 0 & \text{otherwise} \end{cases}$$

In each bisection step the eigenvector corresponding to the second largest eigenvalue of the Laplacian matrix is computed -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. An outline of RSB is given below.

Recursive Spectral Bisection

```
Compute second (Fiedler) eigenvector of the Laplacian matrix
Sort nodes according to the values of Fiedler components;
Assign each half of the nodes to a subdomain;     Repeat recursively for each submeshes;
```

4.3 Geometry Graph Partitioning Algorithm

Local optimization algorithms [PS82], for a given initial solution search a set of finite perturbation of the initial solution until a perturbation with lower cost function is found. Examples of such perturbations, for the graph partitioning problem, appear in the literature (see [KL70], [Got81], and [PK89]*). Two feasible solutions t and t' are called neighbors iff t' is the result of a finite number of consecutive perturbations on t. The set of all neighbors of t is called neighborhood structure. The simplest neighborhood structure, for the partitioning of the graph G and an initial 2-way partitioning (A, B), is given by the following set : $N_s(A, B) = \{$ all partitionings A^*, B^* that can be obtained from the partitioning A, B by a single swap operation $\}$ where the *swap* operation (i.e., perturbation) of forming A^*, B^* is defined as : $A^* = (A \setminus \{a\}) \cup \{b\}$, and $B^* = (B \setminus \{b\}) \cup \{a\}$ with $a \in A$ and $b \in B$.

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. This process is repeated until no such better solution exists. At this point a "locally optimal" solution has been identified. An outline of a local optimization algorithm is described below.

Local Optimization Algorithm

```
begin
    $t$ := some initial solution;
    while $improve(t) \neq$ null do
        $t := improve(t)$;
        return $t$;
    endwhile
end
```

*These algorithms have a longer history, some were discussed in the elementary text Introduction to Computer Science, John R. Rice, 1969 and were analyzed mathematically in the early 1960's by Stanley Reiter.

$$improve(t) = \begin{cases} u & \text{where u} \in N(t) \text{ with } cost(\text{u}) \leq cost(t) \\ null & \text{otherwise.} \end{cases}$$

The *geometry graph partitioning* (GGP) heuristic [CHENHR89] is a local optimization algorithm. The GGP heuristic uses the geometrical properties of the mesh graph (Euclidean graph) in order to deliver deliver quasi-uniform partitionings with the minimal diameter. The cost function that GGP algorithm minimizes is given by :

$$\sum_{k,\ell=1}^{\mathbf{P}} \sum_{e_i \in D_k} \sum_{e_j \in D_\ell} \chi(e_i, e_j) \tag{9}$$

where

$$\chi(e_i, e_j) = \begin{cases} 1 & \text{if } e_i \text{ and } e_j \text{ are adjacent and in different subdomains} \\ 0 & \text{otherwise.} \end{cases}$$

The criteria (ii) and (iv) (see Section 2) are imposed implicitly during the minimization of the cost function (9) by seeking solutions that optimize certain function known as *profit* function :

$$\sum_i (\omega_1 f(a_i, b_i) + \omega_2 g(a_i, b_i)) \tag{10}$$

where

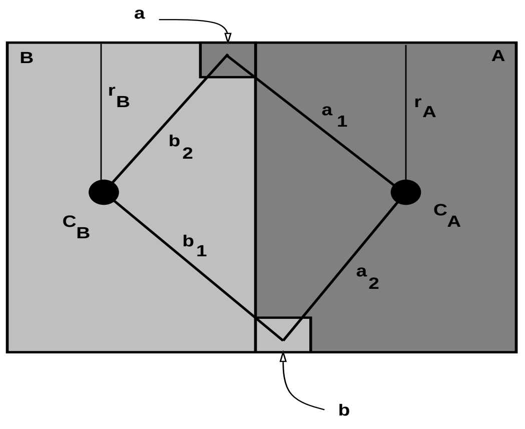
$$f(a_i, b_i) = 2 \sum_{e \in c_{a_i}} \chi(a_i, e) - |c_{a_i}| + 2 \sum_{u \in c_{b_i}} \chi(u, b_i) - |c_{b_i}| - 2\chi(a_i, b_i) \tag{11}$$

and

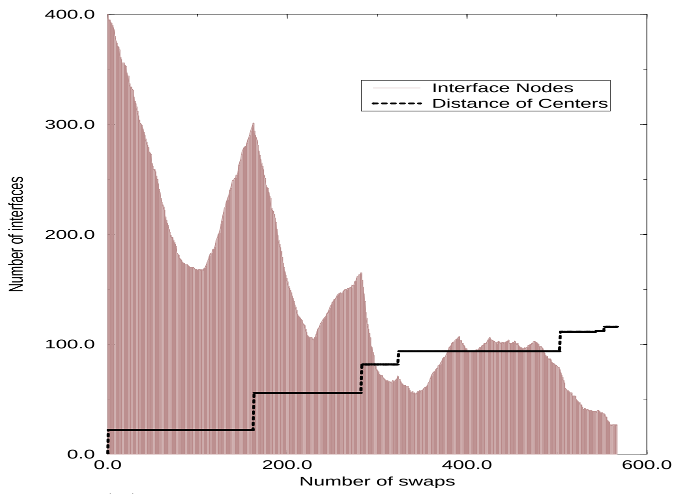
$$g(a_i, b_i) = (\frac{d_{a_i, c_A}}{r_A} - 1) - (\frac{d_{b_i, c_A}}{r_A} - 1) + (\frac{d_{a_i, c_B}}{r_B} - 1) - (\frac{d_{b_i, c_B}}{r_B} - 1) \tag{12}$$

c_e denotes the set of adjacent nodes to the node e whereas $|c_e|$ is the number of nodes adjacent to the node e, with $e = a_i \in A$ or $b_i \in B$. c_A, c_B are the mass center of the subdomains A, B (see Figure 2). d_{a_i, c_A} and d_{b_i, c_B} are the distances between the nodes a_i, b_i and the mass centers c_A, c_B of the subdomains A, B respectively. r_A, r_B are the "ideal" radius of the subdomains A, B, and ω_1 and ω_2 are positive weights.

The GGP algorithm's profit function is a weighted combination of the KL algorithm's profit function f and of the function g 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



(a)



(b)

Figure 2: (a) Illustration of the points mass centers c_A and c_B, distances d_{a_i,c_A} and d_{b_i,c_B} and radii r_A and r_B of quadrilateral sub-meshes A and B. (b) The values of the cost function and of the distance between the mass centers c_A and c_B of the two subdomains for the 2-way partition using the GGP algorithm.

of the cost function (9) 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.

4.4 Allocation Phase

In the allocation phase the submeshes generated by RSB and GGP can be assigned to processors using heuristics described in [CHH90]. In this work the allocation of the submeshes generated by RSB is based on the identity mapping (i.e., the submesh with ID i is assigned to the processor i. In the case of the GGP algorithm we use a Geometry Based Allocation (GBA) algorithm presented in [CHH90].

The GBA algorithm represents the partitioning graph (i.e., graph whose vertices are the submeshes and edges are defined by the connectivity of the submeshes) and the processors interconnection graph into a simpler space, such as the 2D Euclidean space (eg. [0, 1]x[0, 1]). Thus, the allocation problem is simplified into a planar assignment problem. The planar assignment problem can be solved either using spectral methods (see [FYK87]) or local optimization algorithms. GBA uses a local optimization algorithm that optimizes objective functions like the Rectilinear (or Manhattan) distance between the communicating processors, the L_2 norm of distance of the centers of the subdomains and representation points of the processors 2D Euclidean space. The minimization of these objective functions results in the allocation of neighbor subdomains onto neighbor processors.

4.5 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 [Gol89].

An outline of a genetic algorithm (GA) is given below.

Genetic Algorithm

```
Random generation of initial population;
repeat
    Evaluate the fitness of individuals;
    Allocate reproduction trials;
    for i = 1 to population_size step 2 do
        Select 2 parents from the list of trials;
        Apply crossover and mutation;
        Hill climbing by offspring;
    endfor
until convergence
Solution = Fittest.
```

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 objective function used is OF_{appr} involving the communication cost function C_p (equation 4). The reproduction scheme determines which individuals survive and selects pairs of surviving individuals for reproduction. This scheme involves sorting individuals in ascending order. These individuals are assigned a survival probabilities according to a uniform scale of values between 0.8 and 1.2. Then, the number of reproduction trials (copies) for each individual is determined according to these probabilities. Obviously, zero trials means death and two trials allow polygamy. Match-making between individuals is done by random choice from the list of reproduction trials. This reproduction scheme is illustrated by a simple example shown in Figure 3.

Offsprings are generated by applying genetic operators to the selected parents. 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

Figure 3: Reproduction scheme for the Genetic Algorithm.

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.

4.6 Simulated Annealing

An outline of a simulated annealing algorithm (SA) for data mapping is given below.

Simulated Annealing Algorithm

```
Initial configuration = random data mapping;
Determine initial temperature $\theta_0$;
Determine freezing temperature $\theta_f$;
while ($\theta_i > \theta_f$ and Not converged) do
  repeat
    Perturb(mapping solution);
    if ($\Delta OF \leq 0$) then
      Update mapping soln; /* accept perturbation */
    else
      if ($random(0,1) < e^{-\Delta OF/\theta_i}$) then
        Update mapping soln;
      else
        Reject perturbation;
  until equilibrium
  $\theta_{i+1} = 0.95\ \theta_i$; /* cooling schedule */
endwhile
```

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 [KGV83]. 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. The maximum number of attempts allowed is $\mathbf{P}$ per mesh node, whereas the maximum number of accepted moves is $0.75\mathbf{P}$. The initial temperature is determined such that the probability off accepting uphill moves is initially 0.85. The freezing point is the temperature at which this probability is very small (2^{-30}). The cooling schedule determines the next temperature as a fraction, 0.95, of the present one.

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. We have chosen to use $C_d(P_i, P_j)$ (equation 5) for the communication component of OF_{appr} since computing ΔC_d is faster than computing ΔC_p. This choice improves SA's execution time, but also affects the quality of its mapping solutions as will be discussed below.

4.7 Neural Network Algorithm

A Hopfield-type Neural Network for data mapping, described in [FF88] and [MF92], 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. An outline of the neural network algorithm (NN) is given below.

Neural network algorithm

```
for i = 0 to ($\log_2 \mathbf{P}$-1) do
    Generate random spins $s(v,i,0)$;
    repeat
        for all spins do
            Pick a spin randomly;
            Compute $s(v,i,t+1)$; /* equation (13) */
        endfor
    until convergence
    Determine bit $i$ in the neurons;
endfor
```

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.

To derive the network equation, the neural variables are replaced by magnetic spin variables, $s(v,i,t)$ = -1 or +1, in the energy expression. That is, for a given i we associate a spin s with every mesh node v. Then, a mean field approximation technique, from physics, is used to derive the spin update equation:

$$s(v,i,t+1) \; = \tanh\{\; -\alpha s(v,i,t) \; + \beta \sum_{s'} G(s,s') \; - \frac{\gamma}{|\Phi_{i-1}|} \sum_{s \in \Phi_{i-1}, s' \neq s} s'(v,i,t)\} \quad (13)$$

where α, β and γ are appropriate scaling factors [MF92]; G is the spin coupling matrix given by the mesh graph; Φ_{i-1} refers to the current submesh (to be further bisected) to which v belongs. The second term can be interpreted as the ferromagnetic interaction that aligns neighboring spins. The third term can be interpreted as the long-range paramagnetic force responsible for the global up/down spin balance. The first term in the NN equation is the noise term that tries to flip the current spin and, thus, helps the system avoid local minima. Note that the message latency information is missing from equation (13); this equation has been derived assuming that the communication cost function is given by C_d [FF88].

4.8 Graph Contraction

Previous work has shown that physical optimization algorithms are slow in mapping large problems [MF92]. Their execution time is unacceptable when compared with typical time for solving the problems being mapped. To make these physical optimization algorithms practical we have suggested the use of graph (or mesh) contraction for reducing the size of the problem with parameter χ, where the size of the contracted mesh is approximately $N/2^{\chi}$. Then the contracted mesh can be mapped and the mapping solution can next be interpolated. A simple and efficient graph contraction heuristic algorithm has been developed and its description can be found in [MPCF93].

5. 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	:	Block partitioning along the x and y direction (Section 4.1).
RSB	:	Recursive Spectral Bisection (Section 4.2).
GGP	:	Geometry Graph Partitioning (Section 4.3).
GA	:	Genetic Algorithm (Section 4.4).
SA	:	Simulated Annealing (Section 4.5).
NN	:	Neural Network (Section 4.6).

The performance of these algorithms strongly depends on the test cases (i.e, geometry of the domain and mesh). For this reason we compare the above algorithms using a Model Problem defined on a general non-convex domain, D, with two holes. This domain includes many geometric characteristics that appear in real applications. Also, it provides a fair test-case for the comparison of the above algorithms because it does not possesss properties like convexity and simply-connectivity that allow algorithms like P×Q and RSB to perform much better than the most general local search optimization algorithms??????????. For the Model Problem we use a Poisson PDE operator and Dirichlet boundary conditions (the data mapping is independent of the PDE operator). The mesh of the domain D, 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 [CHK+92].

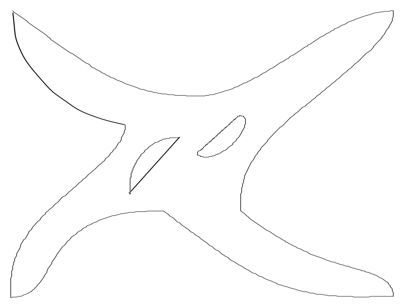


Figure 4: Model Problem.

In the experimental results described below, the physical optimization algorithms use the following problem-dependent parameter values: $\lambda = 5$, $\sigma = 325$, $\rho = 15$, and $\tau = 100$, which have been normalized with respect to the machine time for a floating point operation. The last three communication parameters are relevant to the target machine, nCUBE II [nCU91]. Aslo, suitable values for the contraction parameter χ are chosen for every mapping instance.

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

Figure 5 shows the average and 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 4). Figure 5 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$\times$Q also yield good number of interfaces, whereas NN yields the largest number of interfaces. The number of interfaces appears as a weighted term in the communication cost component of the objective function of GA and SA and is implicitly incorporated in the NN update equation (13). However, the graph contraction pre-mapping step 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.

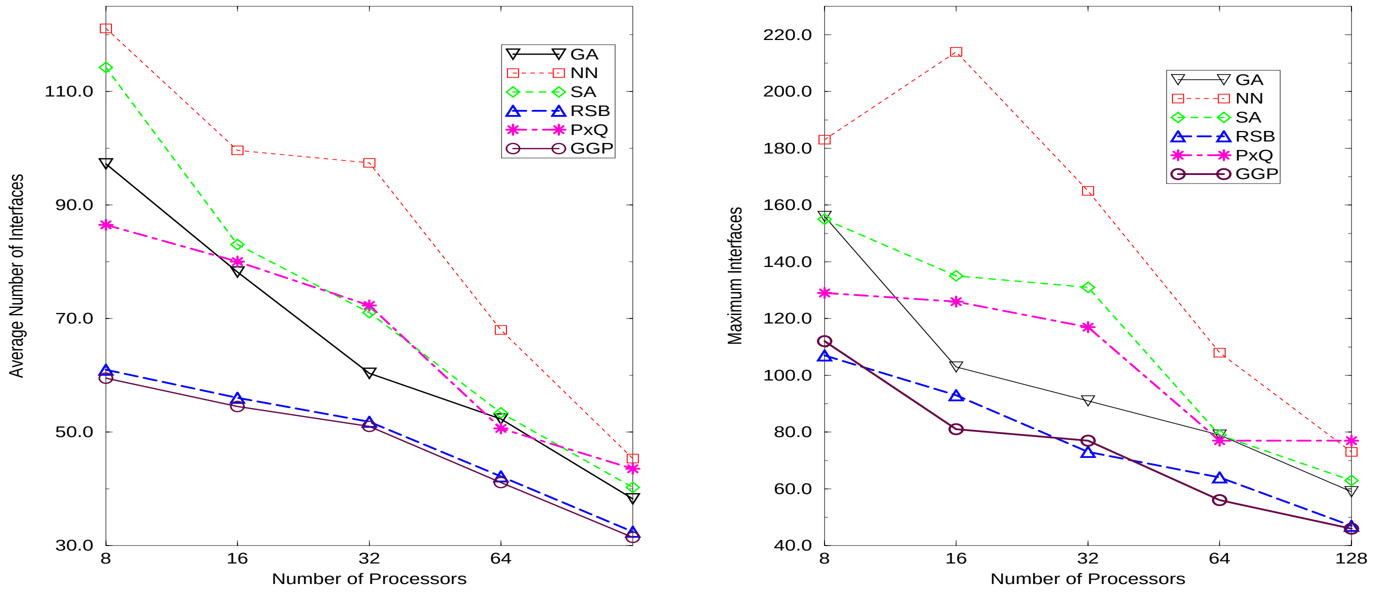
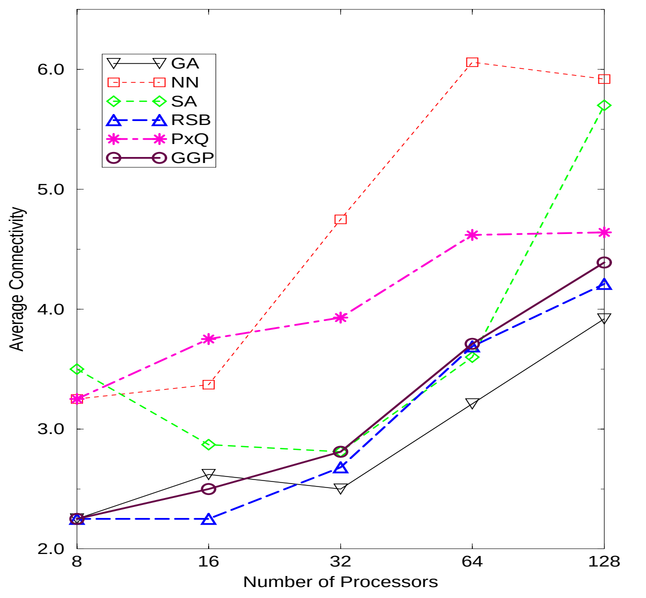


Figure 5: Average and maximum number of interface nodes for the mapping solution of M_{13K}.

Figure 6 shows the average and maximum submesh connectivity; the total message latency is proportional to the submesh connectivity. Figure 6 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 (equation 4) The minimum node separator requirement sought by GGP and RSB seems to help in minimizing submesh connectivity for 2-D meshes. Figure 6 also shows that the connectivity values of NN are worse since NN does not account for connectivity in its update equation. P$\times$Q and SA yield good connectivity values.

Figure 7 shows the average and 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 [Bok90]. SA nad NN show very good distances since interprocessor distances are included in their objective functions with a large weight (see equation 5). P$\times$Q, RSB, and GGP also show good distances,



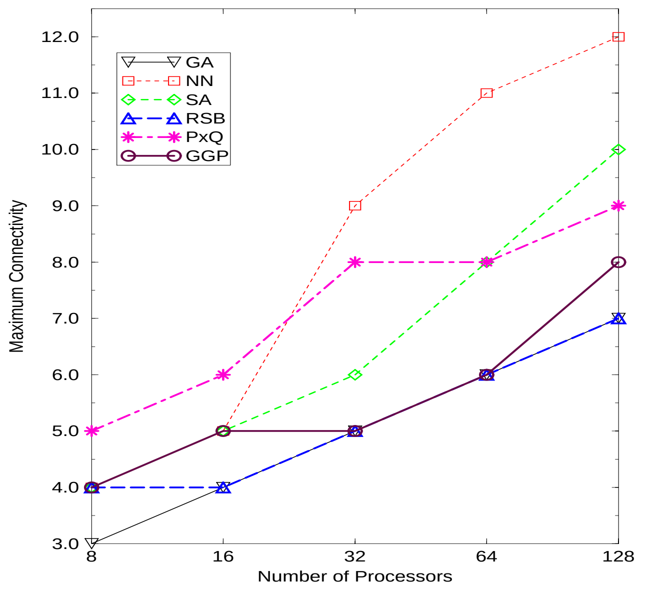


Figure 6: Average and maximum connectivity of the submeshes for the mapping solution of M_{13K}.

whereas GA's distances are acceptable. Note that interprocessor distances are included in the communication cost (equation 4) for GA with a reasonable weight.

Figure 8 gives the standard deviation of the number of nodes per submesh of M_{13K}; M_{13K} is partitioned into 8 submeshes. The deviation values illustrate how well-balanced is the computational load. Clearly, P$\times$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 9 shows two bar charts (for GA and RSB) for the four components of the total workload in each of the 8 processors to which the M_{13K} mesh is mapped. Figure 9 clearly shows that RSB produces perfect load balance regardless of the communication cost of the individual processors. However, GA often reduces the computational load when the

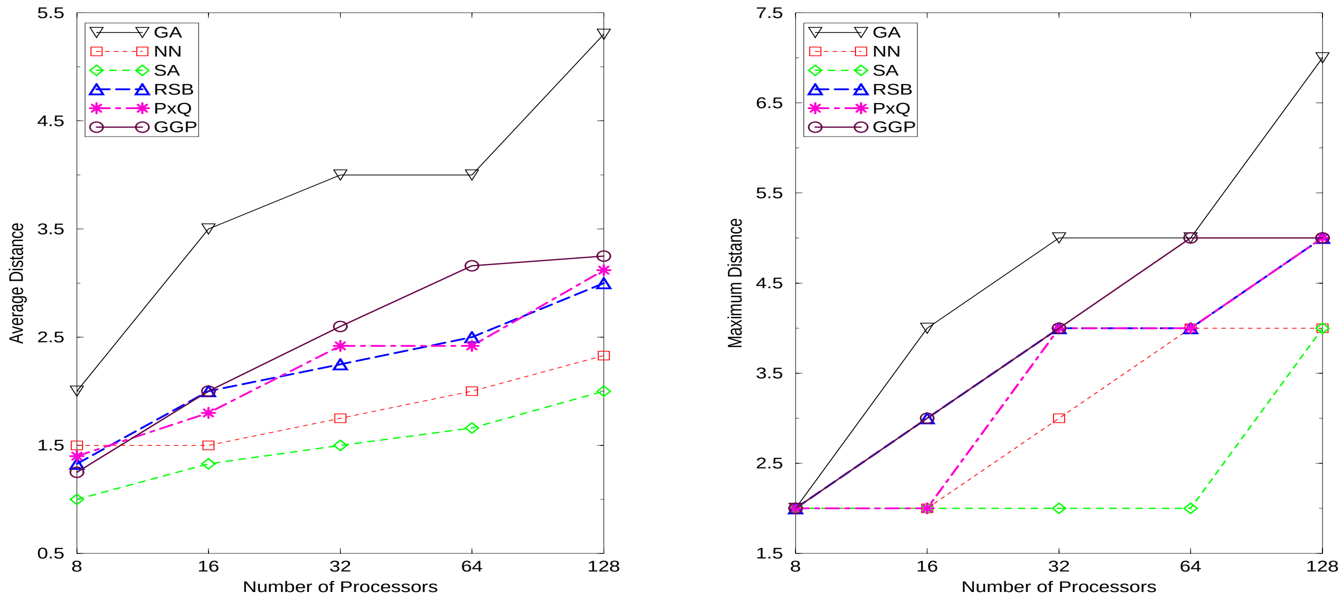


Figure 7: Average and maximum distances among the communicating processors for the mapping solution of M_{13K}.

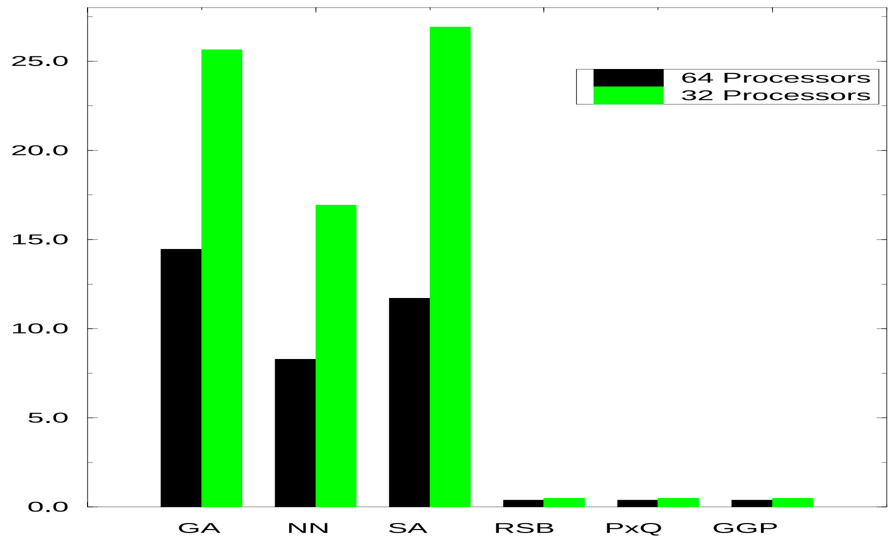
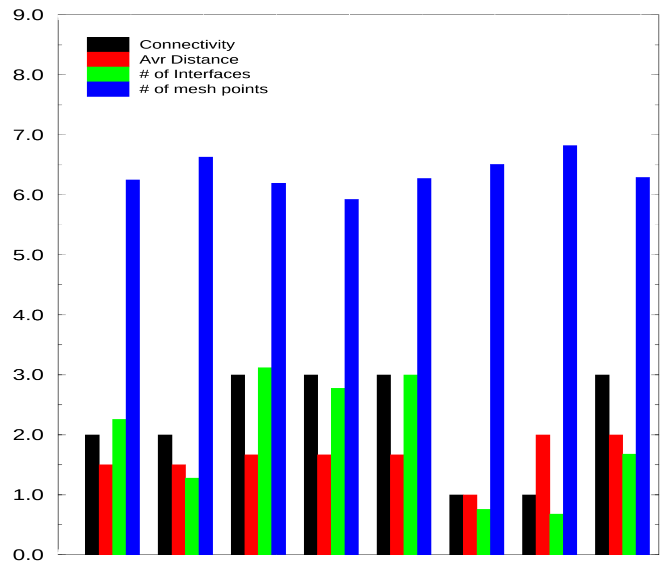


Figure 8: Standard deviation of the number of nodes per submesh for **P** = 32 and 64.



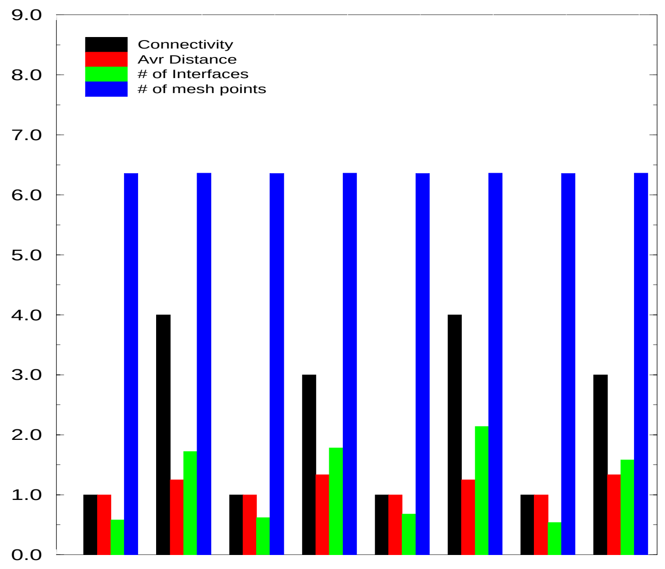


Figure 9: Bar charts for the connectivity, average distance, number of interfaces (divided by 25), and number of node (divided by 250) for GA (left) and RSB (right).

communication cost is large (eg. processors 2 and 3) and increases it when the communication cost is small (eg. processors 1 and 6). The SA and NN algorithms also involve such a tradeoff. On the other hand, P$\times$Q and GGP have similar behavior to RSB.

Figures 10, and 11 show the submeshes produced by the six data mapping algorithms. These solutions show disconnected subdomains for NN, SA GA and P$\times$Q, but not for GGP and RSB. GGP uses profit functions that try to prevent disconnectedness.

5.2 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, average, standart deviation values for T_{solver} and $T_{communicate}$. The tables also show the IDs of the processors (in parenthesis) that have maximum 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

Figure 10: 16 submeshes produced by the P×Q (top left) RSB (top right), GGP algorithm (bottom left) and the GA (bottom right) for the mesh M_{13K}.

Figure 11: 16 submeshes produced by the NN (left) and SA (right) for the mesh M_{13K}.

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 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$\times$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$\times$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.

Elapsed Time	P×Q	RSB	GGP	NN	SA	GA
MAXIMUM	3.433(29)	3.421(28)	3.403(14)	3.839(0)	3.697(24)	3.610(0)
MEAN-VAL	3.428	3.415	3.398	3.760	3.687	3.550
STRD-DEV.	0.003	0.004	0.004	0.014	0.006	0.021

Comm. Time	P×Q	RSB	GGP	NN	SA	GA
MAXIMUM	0.606(20)	0.844(28)	0.826(3)	1.289(29)	1.347(7)	1.169(22)
MEAN-VAL	0.511	0.517	0.499	0.819	0.778	0.657
STRD-DEV.	0.105	0.160	0.162	0.230	0.294	0.279

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

Elapsed Time	P×Q	RSB	GGP	NN	SA	GA
MAXIMUM	2.13(42)	1.94(56)	1.90(15)	2.37(0)	1.95(47)	2.18(60)
MEAN-VAL	1.95	1.93	1.89	2.17	1.94	2.05
STRD-DEV.	0.044	0.002	0.002	0.034	0.002	0.027

Comm. Time	P×Q	RSB	GGP	NN	SA	GA
MAXIMUM	0.777(42)	0.734(48)	0.669(44)	0.958(19)	0.755(56)	0.993(33)
AVERAGE	0.470	0.465	0.423	0.657	0.473	0.582
STRD-DEV.	0.080	0.101	0.108	0.109	0.104	0.164

6. SUMMARY OF RESULTS

Based on the results described in Section 5, 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.

7. 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 s 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 involve intricate parameter-dependence (eg. GA, SA). These findings, for sequential ab initio mapping of 2-dimensional meshes, together with comparisons of the P$\times$Q with greedy algorithms based on Cuthill-McKee ordering schemes [Chr93], [Far88] let us recomment that the very fast and simple P$\times$Q mapping algorithm.

7. ACKNOWLEDGMENTS

The authors thank Horst Simon for providing the RSB code. Also, they thank Pelayia Varodoglu and Elaine ..... for her comments and proofreading of the manuscript. The first author is grateful to John Rice and Elias Housis for their helpful discussions on the data mapping problem. The 64 processor nCUBE II at Purdue University was used for the performance evaluation.

Table 3: Summary of Results

References

[BB87] M. Berger, S. Bokhari. A partitioning strategy for nonuniform problems on multiprocessors. *IEEE Trans. Computers*, C-36, 5 (May), pp. 570–580, 1987.

[Bok81] Shahid H. Bokhari. On the mapping problem. *IEEE Transactions on Computers*, (3):207 – 213, 1981.

[Bok90] Shahid Bokhari. Communication overhead on the Intel iPSC-860 hypercube. Technical Report NAS1-18605, NASA, 1990.

[CHENHR89] N. P. Chrisochoides, C. E. Houstis, S. K. Kortesis E. N. Houstis, and J. R. Rice. Automatic load balanced partitioning strategies for PDE computations. In E. N. Houstis and D. Gannon, editors, *Proceedings of International Conference on Supercomputing*, pages 99–107. ACM Press, 1989.

[CHH90] 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, 1961.

[CHENH$^+$91] 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.

[CHK$^+$92] 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.

[CAHH92] N. P. Chrisochoides, M. Aboelaze, E. N. Houstis, and C. E. Houstis. The parallelization of some level 2 and 3 BLAS operations on distributed memory machines. In *Advances in Computer Methods for Partial Differential Equations VII*, (R. Vichnevetsky. D. Knight and G. Richter, eds) IMACS, New Brunswick, NJ, pages 119-126, 1992.

[CR92] N. P. Chrisochoides, J. R. Rice. Partitioning heuristics for PDE computations based on parallel hardware and geometry characteristics. In *Advances in Computer Methods for Partial Differential Equations VII*, (R. Vichnevetsky. D. Knight and G. Richter, eds) IMACS, New Brunswick, NJ, pages 127-133, 1992.

[Chr92] 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.

[Chr93] N. P. Chrisochoides, Elias Houstis and John Rice. *Mapping Algorithms and Software Environment for Data Parallel PDE Iterative Solvers Special Issue of the Journal of Parallel and Distributed Computing on Data-Parallel Algorithms and Programming, Vol 21, No 1, pp 75–95, April, 1994.*

[KR91] J. De Keyser, D. Roose. A software tool for load balanced adaptive multiple grids on distributed memory computers. Sixth Distributed Memory Computing Conference, April 1991, pp. 22–128.

[DG89] K. Dragon, J. Gustafson. A low cost hypercube load-balance algorithm. 4th Conf. Hypercube Concurrent Computers, and Applications, 583–590, 1989.

[Erc88] F. Ercal. Heuristic Approaches To Task Allocation For Parallel Computing. Ph.D. thesis, Ohio State University, 1988.

[Far88] C. Farhat. A simple and efficient automatic fem domain decomposer. *Computers and Structures*, 28:579–602, 1988.

[FOS88] 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.

[FF88] G. C. Fox, W. Furmanski. Load balancing loosely synchronous problems with a neural network. 3rd Conf. Hypercube Concurrent Computers, and Applications, 241–278, 1988.

[Fox91b] G. C. Fox. Physical computation. *Concurrency Practice and Experience*, Dec., 627–654. 1991b.

[FJL88] G. C. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon and D. Walker *Solving problems on concurrent processors.* Prentice Hall, New Jersey, 1988.

[Fox86a] G. C. Fox. A graphical approach to load balancing and sparse matrix vector multiplication on the hypercube. In *Proceedings of IMA Institute* (M. Schultz, editor), pages 37–51. Springer–Verlag, 1986.

[Fox86b] 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.

[Fox91] G. C. Fox. The architecture of problems and portable parallel software systems. Technical Report SCCS-134, NPAC, Syracuse University, 1991.

[FYK87] Kunio Fukunaga, Shoichiro Yamada, and Tamotsu Kasai. Assignment of job modules onto array processors. *IEEE Transactions on Circuits and Systems*, C–36(7):888–891, 1987.

[GJ79] Michael R. Gary and David S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, San Francisco, 1979.

[Gol89] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley. 1989.

[Got81] Satoshi Goto. An efficient algorithm for the two-dimensional placement problem in electrical circuit layout. *IEEE Transactions on Circuits and Systems*, CAS–28:12–18, 1981.

[S92] W. S. Hammond *Mapping Unstructured Grid Computations to Massively Parallel Computers.* PhD thesis, Computer Science Department, Rensselaer Polytechnic Institute, Troy, NY, 1992.

[Hey90] A. J. G. Hey. Concurrent supercomputing in Europe. 5th Distributed Memory Computing Conf., 639-646. 1990.

[HRC$^+$90] 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.

[HKB90] 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.

[Hop82] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*,

[HY81] L.A. Hageman and D.M. Young. *Applied Iterative Methods.* New York, 1981.

[KGV83] S. Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

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

[KRYG82] D.R. Kincaid, J.R. Respess, D.M. Young, and R.G. Grimes. ITPACK 2C: a Fortran package for solving large sparse linear systems by adaptive accelerated iterative methods. *ACM Transactions on Mathematical Software*, 6:302–322, 1982.

[1] 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.

[MF91] 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.

[MF92] 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.

[MPCF93] 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.

[MO87] R. Morrison and S. Otto. The scattered decomposition for finite elements. *Journal of Scientific Computing*, 2:59–76, 1987.

[nCU91] nCUBE Corporation, *nCUBE 2 Supercomputers*, 1991.

[PAF90] C. Pommerell, M. Annaratone, and W. Fichtner. A set of new mapping and coloring heuristics for distributed-memory parallel processors. In , *Proceedings of Copper Mountain Conference on Iterative Methods* (T. M. Manteuffel editor), volume 4, pages 1–27, 1990.

[PSL90] A. Pothen, H. Simon, K-P Liou. Partitioning sparse matrices with eigenvectors of graphs. *SIAM J. Matrix Anal. Appl.*, 11, 3 (July), 430–452. 1990.

[PK89] C.-H. Lee, C.-I. Park and M. Kim. Efficient algorithm for graph-partitioning problem using a problem transformation method. *Computer-Aided Design*, 21(10):611 – 618, 1989.

[PS82] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization Algorithms and Complexity.* Prentice-Hall, Englewood Cliffs, NJ 07632, 1982.

[SE87] P. Sadayappan and F. Ercal. Cluster-partitioning approaches to mapping parallel programs onto a hypercube. In *Proceedings of Supercomputing '87* (E. N. Houstis, T. S. Papatheodorou, and C. Polychronopoulos, editors), pages 476–497. Springer–Verlag, 1987.

[SE87] 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.

[SBW91] J. Saltz, H. Berryman, J. Wu. Multiprocessors and run-time compilation. *Concurrency Practice and Experience*, 3(6), 573-592. 1991.

[Sim90] D. Horst Simon. Partitioning of unstructured problems for parallel processing. Technical Report RNR-91-008, NASA Ames Research Center, Moffet Field, CA, 94035, 1990.

[SS89] Y. Saad and M. H. Schultz. Data communication in parallel architectures. *Parallel Computing*, 11:131–150, 1989.

[SW90] Quintin Stout and Bruce Wagar. Intensive hypercube communication. *Journal of Parallel and Distributed Computing*, 10:167–181, 1990.

[TWM85] Thompson, F. Joe, Z. U. A. Warsi and C. Wayne Mastin. *Numerical Grid generation.* North-Holland, New York, 1985.

[Wal90] D. Walker. Characterizing the parallel performance of a large-scale, particle-in-cell plasma simulation code. *Concurrency Practice and Experience*, Dec., 257-288. 1990.

[Wil91] R. D. Williams. Performance of dynamic load balancing algorithms for unstructured mesh calculations. *Concurrency Practice and Experience*, 3(5), 457-481. 1991.