A Comparison of Optimization Heuristics for the Data Mapping Problem

Nikos Chrisochoides*
Advanced Computing Research Institute
Cornell Theory Center, Cornell University
Ithaca, NY 14853-3801

Nashat Mansour
Computer Science
Lebanese American University
Lebanon

Geoffrey Fox
Northeast Parallel Architectures Center
Syracuse University
Syracuse, NY, 13244-4100

SUMMARY

In this paper we compare the performance of six heuristics with suboptimal solutions for the data mapping problem of two dimensional meshes that are used for the numerical solution of Partial Differential Equations (PDEs) on multicomputers. The data mapping heuristics are evaluated with respect to seven criteria covering load balancing, interprocessor communication, flexibility and ease of use for a class of single-phase iterative PDE solvers. Our evaluation suggests that the simple and fast block distribution heuristic can be as effective as the other five complex and computational expensive algorithms.

1. INTRODUCTION

The single-phase iterative 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 and independently process parts of the linear system that are assigned to them. That is,

*This work was supported by the Alex Nason Prize Award and in part by the Cornell Theory Center.

processor P_i computes the unknowns x^i of the subsystem $A^i x^i = b^i$ and communicates with other processors when nonlocal or global data are needed. Thus, the execution time of the data-parallel solver is given by:

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

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 [19], [22].

For parallel iterative PDE solvers, the data mapping problem can be formulated in two different levels: (i) at the geometric level, using the discrete geometrical data structures (element-meshes or tensor-grids) associated with the PDE domain and (ii) at the algebraic level using the linear system of algebraic equations associated with some discretization of the continuous PDE problem. In this paper we evaluate data mapping strategies based on geometrical data structures [11].

The minimization of the execution time, T_{solver}, of data-parallel, iterative solvers requires the equal distribution of processors' workload (calculation and communication) and minimization of overheads due to communication of nonlocal unknowns, global parameters, and convergence tests. The problem of finding data distributions that minimize T_{solver} is an intractable optimization problem. Thus, several heuristics have been proposed for finding good suboptimal solutions. Some heuristics are based on greedy schemes and divide-and-conquer method. Examples are nearest neighbor mapping, block partitioning, recursive coordinate bisection, recursive graph bisection, recursive spectral bisection, CM_Clustering, and scattered decomposition [1], [4], [11], [12], [13], [14], [15], [20], [21], [37] [41], [49], [50], [42], [52], [57]. Other algorithms are based on deterministic optimization, where local search techniques are used to minimize cost functions that approximate the execution time T_{solver}; examples are Kernighan-Lin algorithm [35] and geometry graph partitioning [4]. Yet, another class of mapping algorithms are based on physical optimization that employs techniques from natural sciences [18]; examples are neural networks, simulated annealing, and genetic algorithms [30], [16], [17] [38], [58].

Although a good deal of work has been published on data mapping, only few attempts have been made at comparing some algorithms using aggregate or a limited number of performance measures [4], [11], [52], [39], [58], [27]. In this paper, we use several measures to evaluate and compare the performances of six data mapping heuristics for irregular iterative PDE computations. The heuristics considered are: (1) the block (P$\times$Q) partitioning, (2) the recursive spectral bisection, (3) the geometry graph partitioning, (4) a neural network based algorithm, (5) a simulated annealing based algorithm, and (6) a genetic based algorithm. These heuristics have been chosen since they are among the most popular and frequently used data mapping algorithms in the literature. We report performance data that are machine-dependent and machine-independent; Our data have been obtained using the DecTool [6] and Parallel ELLPACK [29].

The findings from this evaluation are summarized in a table that compares these algorithms with respect to seven measures: (a) load balance, (b) submesh connectivity, (c) splitting of submeshes, (d) message size, (e) flexibility, (f) number of machine and solver-dependent parameters, and (g) execution time. Our evaluation suggests that the simple and very fast P×Q mapping algorithm is as effective as the other five complex and computationally expensive algorithms.

This paper is organized as follows. In Section 2 we describe the communication requirements, for a class of parallel iterative PDE solver, and we present two approaches for the mathematical formulation of the data mapping problem. In Section 3 we present a brief description of the six data mapping algorithms we evaluate in this paper. In Section 4 we present our comparisons based on seven criteria and actual performance data from a parallel PDE solver; our findings are summarized in a table. Finally, in Section 5 we present our conclusions.

2. DATA MAPPING FOR PARALLEL ITERATIVE PDE SOLVERS

A number of iterative PDE solvers for the solution of a discrete linear system of algebraic equations can be reduced to matrix-vector multiplication (see [32] and [36]). The matrix-vector multiplication on distributed memory MIMD machines is implemented in two steps : (a) the *local communication* and (b) the *local computation* (see [8] and [19]). Thus a high level description, for a class parallel iterative solvers, pertinent to the data mapping is given by the following three steps: (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., inner and outer interface unknowns, see Figure 1) 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 [7].

An objective function that reflects the computation and communication cost for a single-phase parallel iterative PDE solver and a given mapping of data (i.e., mesh D^h, with $|D^h| = N$) onto the processors of a distributed memory multicomputer can typically be written 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 $m : \{D_i^h | 1 \le i \le P\} \to \{P_i | 1 \le i \le P\}$ is a function that maps the submeshes D_i^h to the processors P_i; $W(m(D_i^h))$ is the computational load of the processor $m(D_i^h)$ per iteration, $W(m(D_i^h))$ 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,

*Notice that we are interested for single-phase iterative PDE solvers, and thus we do not consider solvers with multiple-phases (i.e., different communication requirements for each phase) like multigrid and *hp*-refinement methods.

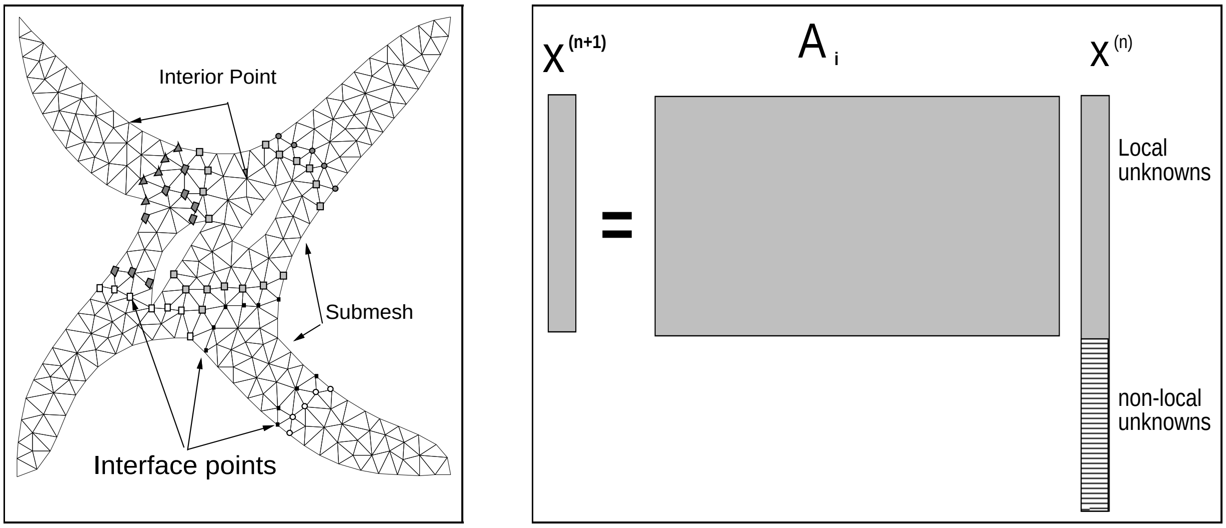

Figure 1: Interior points and inner interface points correspond to local unknowns (stored in local memory) while outer interface points correspond to non-local unknowns (stored in remote memory). For the local matrix vector multiply (right) we need the values of the non-local unknowns form other processors that hadle geometrically adjacent subdomains.

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 it is difficult to be expressed quantitatively. The synchronization cost is a nonlinear function of communication, computation, and communication-computation overlapping. Another term that does not explicitly reflected is the network or bus contention, this term appears implicitly in the communication term, $\sum_{D_j^h \in \kappa_{D_i^h}} C(m(D_i^h), m(D_j^h))$; the network contention among other factors depends on $|\kappa_{D_i^h}|$ which is minimized in the effort to minimize the overall communication of the processor $m(D_i^h)$, for $1 \leq i \leq P$. Nevertheless, OF_{typ} is considered in literature a reasonable approximation for the execution time T_{solver}.

Two approaches can be identified in the literature for the minimization of OF_{typ}. The first approach is based on the expansion of the components of OF_{typ} and the use of explicit machine-dependent as well as 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 floating point and integer operations per mesh node per iteration.

Although OF_{appr} is not equivalent to OF_{typ}, it still represents a good approximation for OF_{typ}. Its first term is quadratic in the deviation of computation loads from the average computation load and is minimal when all deviations are zero. The second term of OF_{appr} is minimized 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 be expressed 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 multicomputers with circuit-switching based interconnection network [3], [28]:

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

where σ is the message start-up time (latency); ρ is the machine time for communicating one word; τ is the communication time per unit distance; $I(D_i^h, D_j^h)$ is the number of interface nodes between 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 submesh connectivity.

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 [28]. 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 qualitative criteria and is derived from the mapping requirements we outline in this paragraph. This approach address the data mapping problem into two distinct phases, namely the *partitioning* phase and the *allocation* phase [4], [5], [10] and [52]. In the *partitioning phase* the mesh is decomposed 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, In the *allocation phase* these submeshes are distributed to the processors such that the following criterion is satisfied:

(iv) 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 $\{D_i^h | 1 \leq i \leq P\}$ is characterized (i) in terms of the set of geometrically adjacent submeshes $\kappa_{D_i^h}$ to submesh D_i^h and (ii) in terms of 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 (iii), can be viewed as the one that 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. 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 use the first formulation for the data mapping problem while the first three algorithms adopt the second approach.

3.1 P×Q Partitioning Algorithm

A simple and attractive mapping method considered by many researchers (see [2], [49], [16], [45] and [10]) 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, sector splitting, and block partitioning algorithm. Throughout this paper, we are referring to this block partitioning algorithm as P×Q algorithm [11], where P is the number of submeshes (blocks or strips) along the x-axis, Q is the number of sub-meshes (blocks or strips) along the y-axis, and P×Q = $\mathbf{P}$ is the total number of submeshes for 2D domains.

The allocation phase, for the P×Q partitioning algorithm and for hypercube and mesh architectures, utilizes the identity mapping in order to distribute the submeshes to processors. The indexing of the submeshes that results form gray code guarantees that geometrically neighbor submeshes are allocated to physically neighbor processors.

The P×Q algorithm, when is used for the partitioning of 2D or 3D meshes with non-convex geometry, often produces submeshes with more than one connected component. To avoid the splitting of the submeshes into more than one components, for star-shaped, non-convex 2D domains Chrisochoides et. al. presented in [9] the *boundary conforming P×Q* algorithm. This extension uses boundary-conforming curvilinear coordinate systems instead of Cartesian coordinate systems. Finally an other generalization of the P×Q algorithm that handles non-star-shaped domains appears in [41]. A description of the P×Q algorithm for a 2D mesh (with N 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) submesh
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**(=P×Q) points of the l = gray_code_1d(i) submesh to the k = gray_code_2d(i,j) submesh
endfor

3.2 Recursive Spectral Bisection

Recursive spectral bisection (RSB) utilizes the spectral properties of the Laplacian matrix, L(M), associated with the mesh, M [46],[52]. RSB recursively bisects the mesh M $log_2\mathbf{P}$ times. The submeshes are distributed to the processors using allocation methods presented in [5], however for the results we report in this paper we use the identity distribution function (i.e., the submesh with ID i is assigned to the processor i), since the RSB implementation we are using [52] enumerates the submeshes so that adjacent submeshes are assigned (whenever possible) to physically neighbor processors.

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

1. Compute second (Fiedler) eigenvector of the Laplacian matrix
2. Sort nodes according to the values of Fiedler components;
3. Assign each half of the nodes to a submesh;
4. Repeat recursively steps 1 to 3 for each of the above submeshes;

3.3 Geometry Graph Partitioning Algorithm

Local optimization algorithms [48], search a set of finite perturbations for a given 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 [35], [26], and [47]†). 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 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) ≠ null do
        t := improve(t);
        return t;
    endwhile
end
```

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

The *geometry graph partitioning* (GGP) heuristic [4] is a local optimization algorithm. 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 cost function that GGP algorithm minimizes is given by :

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

†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.

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 (iii) (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. The profit function we use here us given by:

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

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.

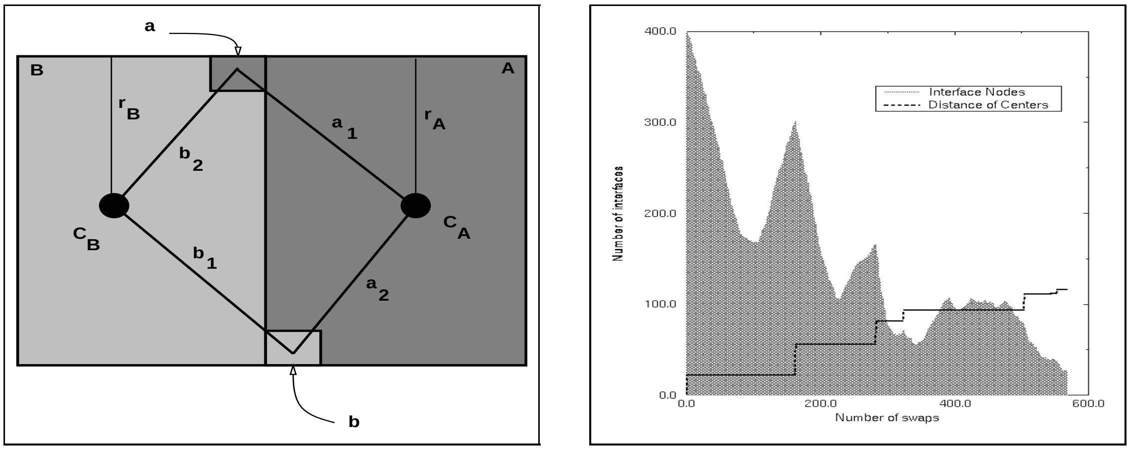

Figure 2: (left) Illustration of the points mass centers c_A and c_B, distances d_{a_i, c_A} and d_{b_i, c_B} and radii r_A and r_B of quadrilateral sub-meshes A and B. (right) The values of the cost function and of the distance between the mass centers c_A and c_B of the two subdomains for the 2-way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

The profit function, p, for GGP algorithm is a weighted combination of the profit function, f, used by KL algorithm and of the function g which is used in selecting pairs of mesh-nodes whose swapping reduces the diameter of the subdomains. The GGP algorithm climbs out of local minima 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. It is easy to see (Figure 2), for simple, convex geometries and 2-way partitioning, that the decrease in the diameter of subdomains implies the decrease of the interfaces, too.

3.4 Genetic Algorithms

In genetic algorithms (GAs) 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 [25]. For more details on data distribution and graph-partitioning heuristics based on GAs see [39], [43] and [55]
An outline of a genetic algorithm 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.

	Individ.	Fitness	Rank	#copies
0 0 0 1 1 1 1 1 1	0	0.019	0.93	1
0 0 1 0 0 1 1 1 1	1	0.022	1.2	2
0 0 0 0 0 0 0 0 0	2	0.012	0.8	0
1 1 1 1 1 1 1 0 0	3	0.017	1.07	1

Parents selected randomly: 0 and 1
3 and 1

Figure 3: Reproduction scheme for the Genetic Algorithm.

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 30%.

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 simulated annealing (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 [34]. 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, $e^{-\Delta OF/\theta}$.

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

Simulated Annealing Algorithm

```
Initial configuration = random data mapping;
Determine initial temperature θ_0;
Determine freezing temperature θ_f;
while (θ_i > θ_f and Not converged) do
    repeat
        Perturb(mapping solution);
        if (ΔOF ≤ 0) then
            Update mapping soln; /* accept perturbation */
        else
            if (random(0,1) < e^(-ΔOF/θ_i)) then
                Update mapping soln;
            else
                Reject perturbation;
    until equilibrium
    θ_{i+1} = 0.95 θ_i; /* cooling schedule */
endwhile
```

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 **P** per mesh node, whereas the maximum number of accepted moves is 0.75**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. For more details on data distribution and graph-partitioning heuristics based on SA see [39], [33], [34].

3.6 Neural Network Algorithm

A Hopfield-type Neural Network for data mapping, described in [17] and [39], 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 [39]; 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 [17].

3.7 Graph Contraction

Previous work has shown that physical optimization algorithms are slow in mapping large problems [39]. 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 [40].

3.8 Allocation Phase

In the allocation phase the submeshes generated by the above algorithm are assigned to processors using heuristics described in [5]; for this evaluation we are using Geometry Based Allocation (GBA) algorithm. The GBA algorithm first projects 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 2D (or 3D) Euclidean space for 2D (or 3D) problems. Thus, the allocation problem is reduced into easier but still NP-Complete problem, namely the planar assignment problem. Optimal solutions for the planar assignment problem can be obtained using either global techniques like spectral methods (see [23]) or local search techniques. GBA uses a local search algorithm that can optimize objective functions like: (i) the Rectilinear (or Manhattan) distance between the communicating processors, and (ii) the distance (L_2) of the centers of the subdomains from the projection points that represent the processors on the 2D Euclidean space. The minimization of these objective functions results in the allocation of neighbor subdomains onto neighbor 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$\times$Q	:	Block partitioning along the x and y direction (Section 3.1)
RSB	:	Recursive Spectral Bisection (Section 3.2)
GGP	:	Geometry Graph Partitioning (Section 3.3)
GA	:	Genetic Algorithm (Section 3.4)
SA	:	Simulated Annealing (Section 3.5)
NN	:	Neural Network (Section 3.6)

The performance of these algorithms strongly depends on the test cases (i.e, geometry of the 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 possesses properties like convexity and simply-connectivity that allow algorithms like P$\times$Q and RSB to perform much better than the most general local search optimization algorithms. The choice of data mapping methods is independent of the PDE operator, in this work we choose Poisson PDE operator on D and Dirichlet boundary conditions on the external boundary of D. The mesh, M_{13K}, of the domain D consists of 24,202 elements and 12,724 nodes. The PDE operator is discretized by a bilinear finite element method, the resulting linear system of equations (11,676 equations) is solved by a Jacobi Semi Iterative (Jacobi-SI) method [7].

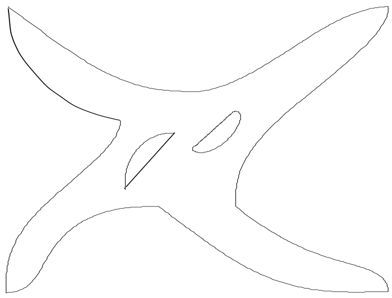

Figure 4: Model Problem.

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 [44]. Also, suitable values for the contraction parameter χ are chosen for every mapping instance.

4.1 Machine-Independent Analysis

The machine-independent measures we consider here are : (i) the submesh connectivity, (ii) the number of interface nodes , (iii) the splitting of the submeshes, and (iv) the equidistribution of both interior and interface nodes of the submeshes. Our analysis is based on distributions of the mesh M_{13K} onto nCUBE II with configurations of 2^3, 2^4, 2^5, 2^6 and 2^7 processors. From these distributions 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 message size term of the communication cost (equation 4) is proportional to the number of submesh interface nodes. Figure 5 shows that GGP and RSB yield the smallest number of interface nodes. RSB tries to minimizes the node separator of the mesh while GGP algorithm tries to minimizes the node separator and submesh diameter as well as tries to maximize the inter-center distance of submeshes. 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 6 shows the average and maximum submesh connectivity; the total message latency is proportional to the submesh connectivity, since each processor send its messages to other processors in sequential manner. 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.

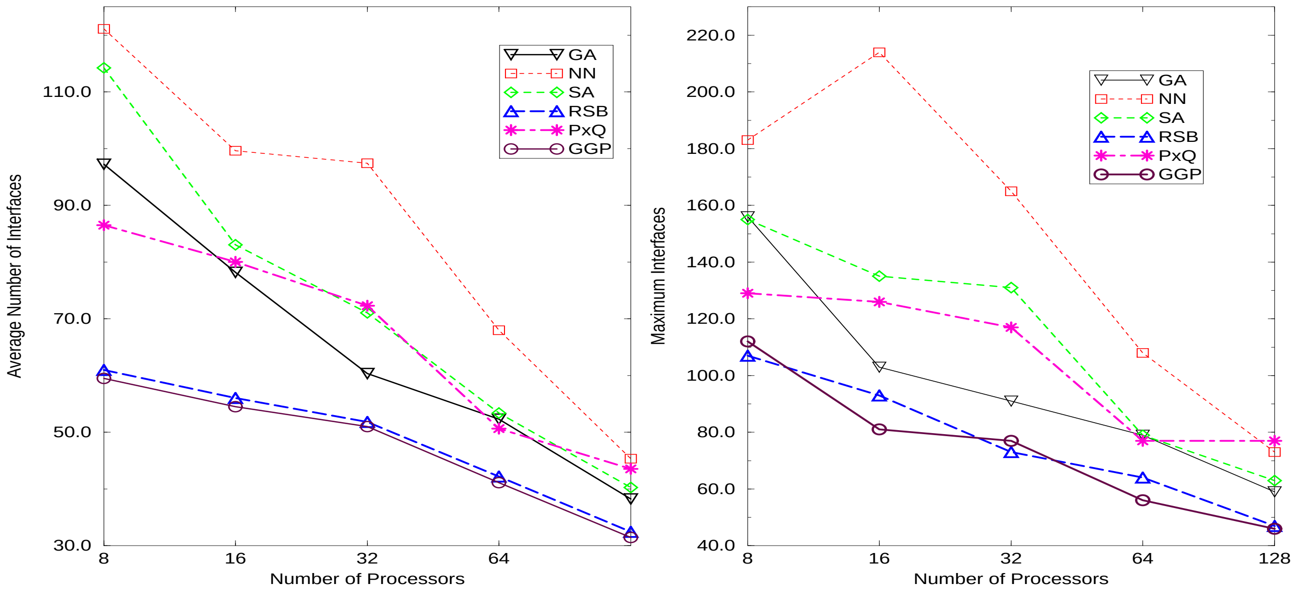

Figure 5: Average and maximum number of interface nodes for the mapping solution of M_{13K}.

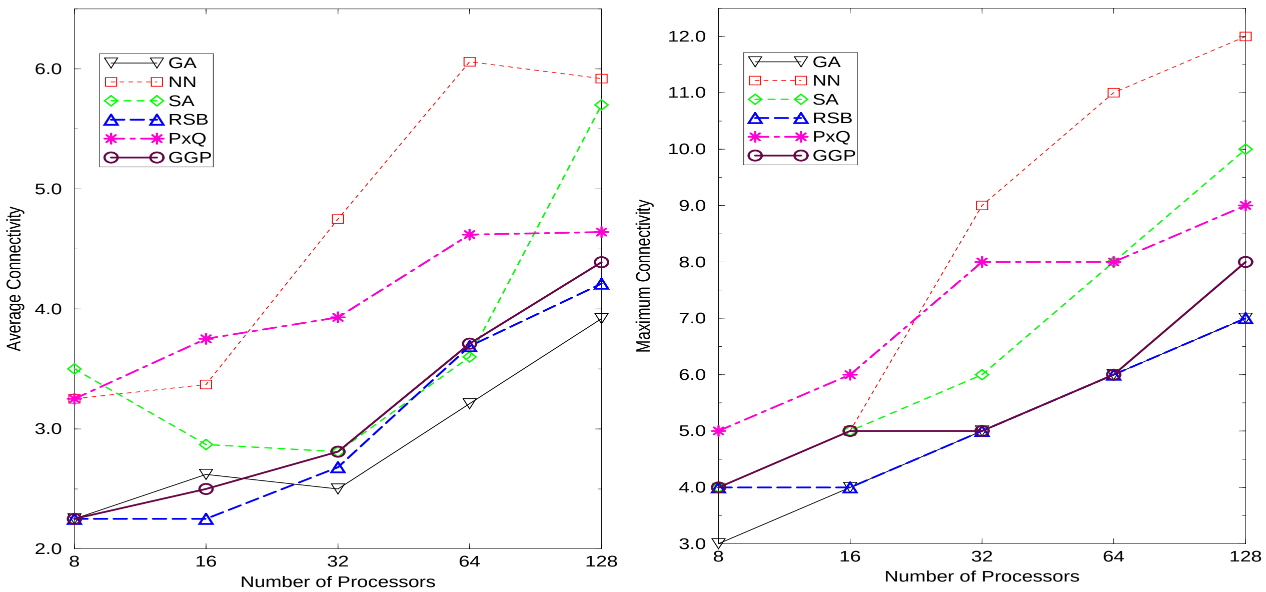

Figure 6: Average and maximum connectivity of the submeshes for the mapping solution of M_{13K}.

Figure 7 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×Q, RSB and GGP produce near equal distributions of points among the processors 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.

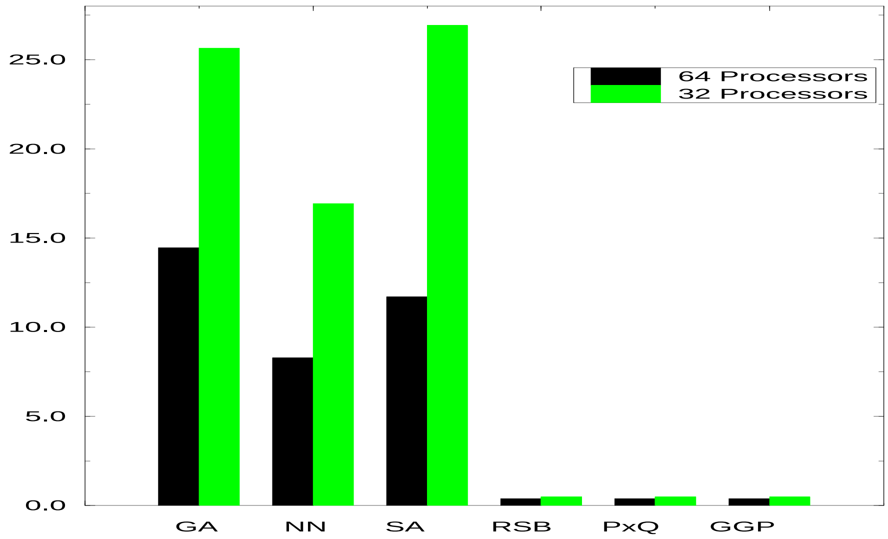

Figure 7: Standard deviation of the number of nodes per submesh for $\mathbf{P}$ = 32 and 64 .

Figure 8 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 8 clearly shows that RSB, produces perfect load balance regardless of the communication cost of the individual processors, the same is true for P×Q and GGP. However, GA often reduces the computational load when the 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×Q and GGP have similar behavior to RSB.

Table 1 summarizes the comparison of the six data mapping algorithms with respect to seven criteria. Note that the table reflects the quality and timings of the contracted graphs for NN, SA and GA. We report some timing data for the M_{13K} problem. The execution times of these algorithms can be reduced farther with the propr optimizations in the data structures and implementation.

Figure 9 shows the data partitions that were generated by the six data mapping algorithms. These solutions show disconnected subdomains for NN, SA GA and P×Q, but not for GGP and RSB.

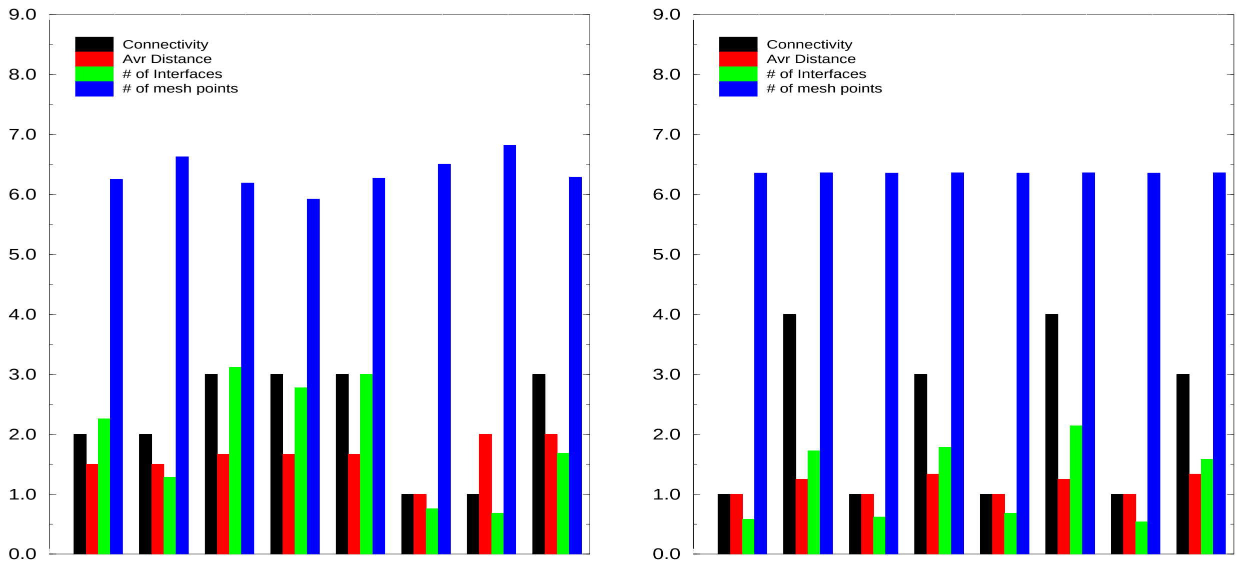

Figure 8: Bar charts that depict the tradeoff between load balance and communication for GA (left) and RSB (right). Each chart consists of eight groups (x-axis) of four bars that depict connectivity, average distance, number of interfaces divided by 25, and total number of nodes divided by 250 per submesh. Y-axis has no metric.

Table 1: Summary of Results

Algo.	load bal.	connect.	interf .	comp-comm tradeoff	discon. submeshes	params	map. code speed
PxQ	perfect	good	good	no	likely	no	very fast few secs
RSB	perfect	v. good	v. good	no	likely	no	slow 45 mins
GGP	perfect	v. good	v. good	no	less likely	no	very slow 133 mins
NN	v. good	acceptable	acceptable	limited	likely	yes	fast 6 mins
SA	v. good	good	good	yes	likely	yes	extr. slow 199 mins
GA	v. good	v. good	good	yes	likely	yes	very slow 86 mins