1994

Mapping Algorithms and Software Environment for Data Parallel

Nikos Chrisochoides
Syracuse University, Northeast Parallel Architectures Center

Elias Houstis
Purdue University, Computer Science Department, enh@cd.purdue.edu

John Rice
Purdue University, Computer Science Department, jrr@cs.purdue.edu

Mapping Algorithms and Software Environment for Data Parallel PDE Iterative Solvers

Nikos Chrisochoides,*
Northeast Parallel Architectures Center, Syracuse University
111 College Place, Syracuse, NY, 13244-4100
nikos@npac.syr.edu

Elias Houstis† and John Rice†
Computer Science Department, Purdue University
West Lafayette, IN, 47906
{enh, jrr}@cs.purdue.edu

Abstract

We consider computations associated with data parallel iterative solvers used for the numerical solution of Partial Differential Equations (PDEs). The mapping of such computations into load balanced tasks requiring minimum synchronization and communication is a difficult combinatorial optimization problem. Its optimal solution is essential for the efficient parallel processing of PDE computations. Determining data mappings that optimize a number of criteria, like workload balance, synchronization and local communication, often involves the solution of an NP-Complete problem.

Although data mapping algorithms have been known for a few years there is lack of qualitative and quantitative comparisons based on the actual performance of the parallel computation. In this paper we present two new data mapping algorithms and evaluate them together with a large number of existing ones using the actual performance of data parallel iterative PDE solvers on the nCUBE II. Comparisons on the performance of data parallel iterative PDE solvers on medium and large scale problems demonstrate that some computationally inexpensive data block partitioning algorithms are as effective as the computationally expensive deterministic optimization algorithms. Also, these comparisons demonstrate that the existing approach in solving the data partitioning problem is inefficient for large scale problems. Finally, a software environment for the solution of the partitioning problem of data parallel iterative solvers is presented.

1 Introduction

Partial Differential Equations (PDEs) are the fundamental mathematical tool for describing the physical behavior of many applications in science and engineering. Most of the existing PDE software systems deal primarily with the solution of specific classes of PDE problems on sequential or vector machines. The techniques and software tools developed and analyzed in this paper have been applied to general second order elliptic PDEs defined on 1, 2 and 3 dimensional domains. They can easily be extended to computations associated with the numerical simulation of more complicated "steady-state" mathematical models. The structure of the PDE problem assumed throughout this paper is depicted in Figure 1.

*Most of this work was done while the author was at Purdue University. The research of this author was supported in part by NSF grant CCR86-10817, AFOSR grant F49620 and Alex G. Nason Foundation at Syracuse University.

†Work supported in part by NSF grants CCR 86-19817, CCR 92-02536, and AFOSR grant 91-F49620.

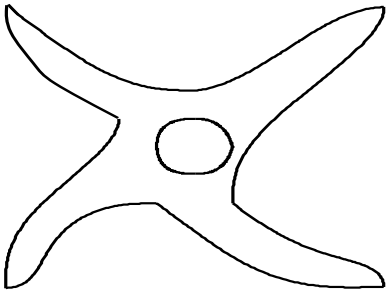


Figure 1: The components of a typical continuous PDE problem and example of continuous domain D in R^2. The PDE equation $Lu = f$ is satisfied within the interior D and the boundary conditions $Bu = g$ are satisfied on the boundary ∂D of D.

There are two general parallel methodologies for solving PDEs on distributed and shared memory MIMD machines. The first methodology is based on the decomposition of the continuous PDE domain D into non overlapping substructures or subdomains (see [GGMP88], [CSS86], [CR87] and [KG87]). The original PDE problem is reduced to a set of "smaller" PDE problems defined on each subdomain where auxiliary conditions have been "artificially" extended on the interior subdomain interfaces. The components of the decomposed PDE problem are depicted in Figure 2.

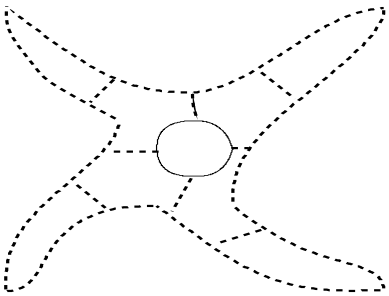


Figure 2: The components of the decomposed PDE problem based on the splitting of the domain D into a substructure of domains D_i.

Continuity or smoothness of u at the subdomain interfaces is usually required; these requirements are usually handled by an iterative technique over the subdomains. The proof of the equivalence of the decomposed PDE problem to the original one is not trivial. It depends very much on the artificial conditions employed and the operator L. The theoretical results in this area are limited

The second methodology is based on the decomposition of the mesh or grid D^h of the PDE domain which results into a splitting of the corresponding algebraic data structures consisting of the discrete equations corresponding to the node or grid points of the subdomain and their interfaces (boundary). Figure 3 describes the decomposition of the discrete PDE problem. Throughout this paper, we refer to the first approach as the *continuous domain decomposition* approach and the second one as the *discrete domain decomposition* approach for partitioning PDE problems.

The computation associated with data parallel iterative PDE solvers that preserve the ordering

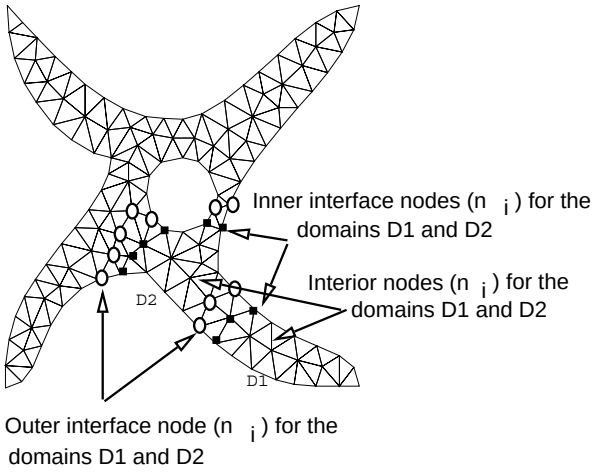


Figure 3: The components of the decomposed discrete PDE problem based on the splitting of the mesh or grid D^h used numerically. This discrete mesh is partitioned by interfaces nodes (shown as circles) into discrete subdomains D_i^h

of the corresponding sequential computation is *loosely synchronous* [Fox91]. The programming model for loosely synchronous computations is single-program-multiple-data, where parallelism is achieved by partitioning the underlying geometric data (continuous or discrete) of the PDE problem and allocating the disjoint subproblems or subcomputations to the processors. During each iteration the processors perform : (i) an exchange of local data (interface unknowns) with the processors that handle geometrically adjacent subdomains in order to enforce continuity requirements for the PDE solution *(local synchronization/communication)* (ii) an execution of matrix-vector operations *(local computation)* on the local subdomain data, and (iii) an evaluation of stopping criteria and acceleration of the convergence *(global synchronization)*. The high performance of these solvers on distributed memory MIMD machines depends on the minimization of the local and global communication time and synchronization delays, assuming that the local computations properly use the memory hierarchy (registers-cache-memory) of each processor. The global communication time depends on the efficient hardware/software implementation of reduction operations. During the last five years such operations have been identified and studied extensively (see in [JH89], [SS89], [SW90], and [FK89]) and as a result there are implementations for these operations on the commercially available parallel machines.

In this study we focus on the minimization of the local communication time per iteration. The local communication time depends both on data partitioning characteristic like interface length and degree of connectivity of the subdomains, and machine characteristics like the interconnection network and routing. The data partitioning problem is NP-Complete [GJ79] and many heuristic methods have been proposed for finding good suboptimal partitions of the data. These heuristics are divided into three classes, namely, data clustering, deterministic optimization and stochastic optimization. The rest of the paper is organized as follows. First, in Section 2, we describe the data mapping problem for the parallel PDE iterative solvers. In Section 3 we summarize the data partitioning algorithms. A brief high level description of the parallel iterative PDE solvers pertinent to the data partitioning problem is presented in Section 4. Sections 5 and 6 describe two new data partitioning algorithms. Section 7 evaluates most of the existing clustering and deterministic data partitioning algorithms. In Section 8 we present an interactive software environment for the manipulation and visualization

of data partitionings for 2-dimensional iterative PDE solvers. We conclude with a discussion of the evaluation of data partitioning algorithms. The evaluation of different partitions is based on the actual performance of the Jacobi Semi Iterative (Jacobi-SI) method [CHK+92].

2 Data Mapping Problem

The objective function for the mapping of a mesh M (or grid) onto a distributed memory MIMD machine so that the workload of the processors is balanced and the required communication and synchronization among the processors is minimum, can be formulated by :

$$\min_{m} \max_{1 \le i \le \mathbf{P}} \{ \; W(m(D_i)) \; + \sum_{D_j \in C_{D_i}} C(m(D_i), m(D_j)) \; \} \tag{2.0}$$

where D_i is the set of mesh points (subdomain) that are assigned to the same processor, C_{D_i} is the set of the subdomains that are adjacent to the subdomain D_i, $m : \{D_i\}_{i=1}^{P} \rightarrow \{P_i\}_{i=1}^{\mathbf{P}}$ is an assignment function that maps the subdomains to processors, $W(m(D_i))$ is the computational load of the processor $m(D_i)$ per iteration, which is related to the number of mesh points in D_i, and $C(m(D_i), m(D_j))$ is the communication required (per iteration) between the processors $m(D_i)$ *and* $m(D_j)$, and $\mathbf{P}$ the number of available processors of the target parallel machine. The synchronization of the processors is a nonlinear correlation of computational and communication work-load and overlapping. In the case of data parallel PDE iterative solvers without the overlapping between the computation and the communication phases the synchronization term in equation (2.0) is included in $W(m(D_i))$.

One approach to solve the optimization problem (2.0) is to approximate its of the objective function (2.0) by another function which is smoother, more robust and suitable for the existing optimization methods [Fox86a], [FOS88], [Wil90] and [Man92]. A second approach is to split the optimization problem into two distinct phases corresponding to the *partitioning* and *allocation* of the mesh [CHENHR89], [CHH90], [Chr92] and [Sim90]. In the *partitioning phase* we decompose the mesh (or grid) in a prespecified number (usually equal to the number of processors) of subdomains such that the following criteria are approximately satisfied:

(i) the maximum difference in the number of active mesh (or grid) points of the subdomains is minimum,

(ii) the ratio of the number of active interface points to the number of active interior points for each subdomain is minimum,

(iii) the number of subdomains that are adjacent to a given subdomain is minimum,

(iv) each subdomain is a connected domain.

In the *allocation phase* the these subdomains are assigned to processors such that the following objective is satisfied:

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

For a given discrete domain D^h with N mesh points, the merit of a partition into $\mathbf{P}$ non-overlapping subdomains $\{D_i\}_{i=1}^{\mathbf{P}}$ is characterized in terms of the set of geometrical adjacent subdomains C_{D_i} to subdomain D_i and the number of the interface mesh points, $c(D_i, D_j)$, shared by the subdomains

D_i *and* D_j. Then, the optimal partitioning, as defined by criteria (i) to (iv), can be viewed as the one which simultaneously minimizes :

$$\max_{1\leq i,j\leq \mathbf{P}} \mid |D_i| - |D_j| \mid \tag{2.1}$$

$$\max_{1\leq i\leq \mathbf{P}} \{\frac{(\sum_{D_j\in C_{D_i}} c(D_i, D_j))}{|D_i|}\} \tag{2.2}$$

$$\max_{1\leq i\leq \mathbf{P}} |C_{D_i}| \tag{2.3}$$

where $|D_i|$ is the size of the subdomain D_i and it is defined as the cardinality of the set of mesh points that belong in D_i.

3 Overview of Data Partitioning Algorithms

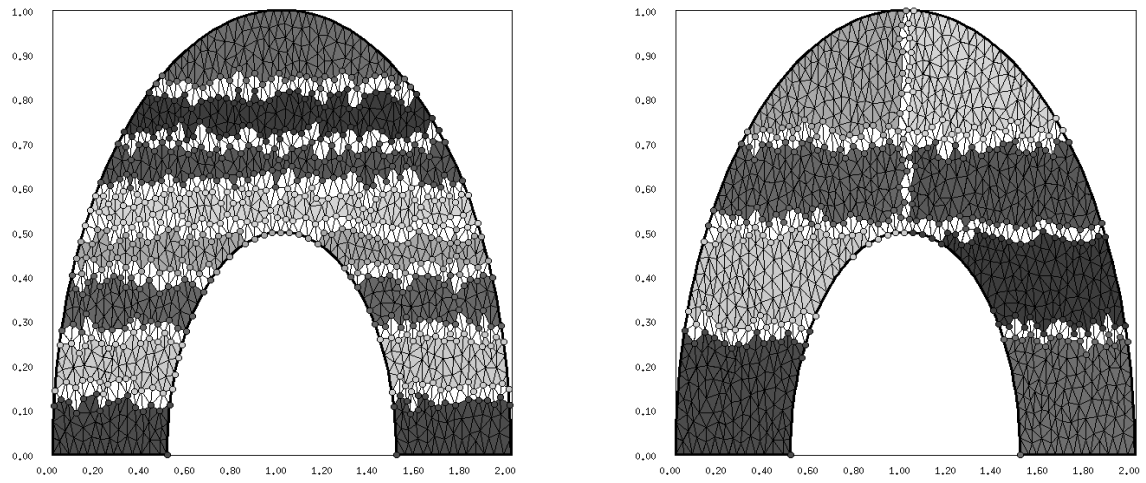
In this section we identify the three classes of data partitioning algorithms, namely, data clustering, deterministic optimization and stochastic (or physical) optimization and discuss some examples..

Data Clustering Algorithms

The objective of a data clustering algorithm is to group the mesh points into clusters such that the points within a cluster have a high degree of "natural association" among themselves while the clusters are "relatively distinct" form each other. In our case, the "natural association" is expressed in terms of the locality properties of the finite element and finite difference stencils that are used to approximate a continuous PDE operator, and the "relative distinction" is expressed in terms of the address space that is associated to the unknowns of the mesh or grid points that belong in the same cluster.

The simplest, oldest and one of the most effective data partitioning methods is to sort the geometric or topological mesh data in some direction and then partition the resulting ordered sequence of nodes into **P**-groups, where **P** is the number of available processors. The sorting of geometric data like the coordinates of node points, the coordinates of the sector origin of the elements, and the coordinates of the centroid of the elements of a mesh have been considered by many researchers (see [Bok81], [SE87], [FOS88], [LF90] and [PAF90]). This idea is referred in the literature 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 refer to this clustering algorithm as a *block partitioning algorithm.* In the case of 2-dimensional domains the block partitioning algorithm is called *P×Q partitioning algorithm*, where P is the number of subdomains (blocks or strips) along the x-axis, Q is the number of subdomains (blocks or strips) along the y-axis, and P × Q = **P**. Some of the advantages of the P×Q partitioning algorithm are that it satisfies criteria (i) and (ii), it is not sensitive to a predefined enumeration of the nodes (or elements), and it is suitable for the mapping of the subdomains onto a linear array and 2D-mesh architectures. Its disadvantage is that it usually partitions a non-convex domain into disconnected subdomains. In this paper we present a block clustering method that avoids this disadvantage for star-shape domains - a large class of non-convex domains. Figure 4 illustrates the partitioning of triangular meshes of a semi-annulus 2D non-convex domain using (a) 1×8 and (b) 4×4 algorithms.

A generalization of the block data partition method is *scattered decomposition* [MO87] which consists of the following two steps : (i) embed the machine's interconnection graph into a two-dimensional processor lattice and (ii) cover the mesh with several copies of this processor lattice. Its advantage is the ability to map a large class of irregular scientific computations without ever analyzing them (see



(a) (b)

Figure 4: (a) Partition of a discretized semi-annulus domain using the 1×8 partitioning algorithm. (b) Partition of a discretized semi-annulus domain using the 2×4 partitioning algorithm.

[CT88]) and it is inexpensive (by using lattices of fine granularity). Its main disadvantage is the higher communication cost due to fine granularity of the mapping (see [CT88] and Section 7).

Another class of data clustering heuristics is based on *reordering* methods developed to solve the *fill-in problem* [GL81]. These problems are NP-Complete [Gil80]. See [LS76], [Geo73], [GL78], and [GM78] for the following methods : *Cuthill McKee, reverse Cuthill McKee, automatic nested dissection,* and *minimum degree*. Generalizations of these algorithms appear in [Gil80] and [Liu89b] which can be used to partition a mesh into $\mathbf{P}$ connected submeshes ($\mathbf{P}$-way partition) . A generalization of these techniques for connected graphs has been made by Farhat [Far88] who presented a greedy algorithm based on the rooted level structure scheme. Throughout this paper we refer to this algorithm as the *CM-Cluster* algorithm. It produces load balanced partitionings with a minimum amount of interface points among the subdomains and handles domains with irregular geometry and arbitrary discretization, but it may generate disconnected partitions. In [ANN90], Al-Nasra et al. improve the CM-Cluster algorithm by using both the topology and the geometry of the mesh to avoid disconnected subdomains. The improvement introduces an additional weight for the nodes based on the calculation of the long and short directions of the two dimensional domain. The new weight ω_i of the ith node is

$$\omega_i := c_i + \mathbf{P}^2 * (\frac{\delta}{f}) * (\frac{f}{g} - 1)$$

where c_i is the node connectivity (i.e., the number of adjacent nodes to ith node) , δ is the step size of the mesh along the long direction of the smallest rectangular, say R of size $a \times b$, that encloses the domain, $f := \max\{a, b\}$, and $g = \min\{a, b\}$.

Finally, a divide-and-conquer class of algorithms, *recursive bisection* [Fox86b], [Sim90], [Wil90], [Man92], [S92] and [Chr92], have been used as data clustering algorithms. These algorithms bisect of a mesh by using either the coordinates of the mesh (or grid) points [Fox86b] or a rooted level structure or the *spectral properties of the Laplacian matrix* [Sim90]. The Laplacian matrix $L(M)$ of a mesh M is

defined as :

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

Deterministic Optimization Algorithms

A general approach for combinatorial optimization problems is *local (or neighborhood) search* [PS82]. Each feasible solution is associated with a cost which is to be optimized locally. Local search algorithms require the definition of a neighborhood structure for each feasible solution, i.e., a finite set of neighbors which are in some sense "close". For example, in partitioning the finite element mesh M, an obvious neighborhood of a given partition (M_1, M_2) of M is the finite set $\{(M_{1_i}, M_{2_i})$, where M_{1_i}, M_{2_i} are connected meshes and $M_{1_i} = (M_1 - \{x\}) \cup \{y\}$ and $M_{2_i} = (M_2 - \{y\}) \cup \{x\}$ with $y \in M_2$ and $x \in M_1\}$.

The only difference between various local search algorithms is in the definition of their neighborhood structures. Since the problem of partitioning the nodes of a mesh or grid is the same as the partitioning problem of a general graph, the neighborhood structures that have been defined for the graph partitioning problem can be used for the partitioning of PDE computations based on the discrete geometry of the physical domain.

See [KL70], [Got81], [PK89] and [TZTS92] [1]) for some of the neighborhood structures for the graph partitioning problem that appear in the literature. The simplest neighborhood structure for a given 2-way partitioning (A, B), is the set :

$$N_s(A, B) = \{\text{all partitionings } A^*, B^* \text{ that can be obtained from the partitioning A, B by a single swap operation }\},$$

where the *swap* operation of forming A^*, B^* is defined by :

$$A^* = (A \setminus \{a\}) \cup \{b\}, \text{ and } B^* = (B \setminus \{b\}) \cup \{a\}$$

with $a \in A$ and $b \in B$. Kernighan and Lin (KL) in [KL70] generalized the above idea by replacing a single swap operation with a sequence of swaps. At each step of the algorithm, the swap involving a pair of unswapped vertices is chosen that yields the best cost. As Figure 5 illustrates the first few swaps might worsen the initial partitioning but they will help the local search to climb out of some local minima. The algorithm stops at any point where no improvement can be made by further swapping.

A more complicated generalization by Satoshi Goto [Got81] replaces the pairwise swapping with an interchange of more than two vertices at the same time. The same extension can be used to define neighborhood structures for the P-way graph partitioning problem. Finally, Lee et al. [PK89] and Tao et al. [TZTS92] present a transformation of the bisection (and P-way) graph partitioning problem into the max-cut problem.

Stochastic (or Physical) Optimization Algorithms

Finally, the third class of heuristics are the *stochastic optimization* techniques which are not evaluated in this paper. These include the *physical optimization* mapping algorithms [Mans 92] and simulated annealing (SA) technique [KGV83]. Several authors [Fox86b], [FOS88], [Wil90], and [Man92] have applied this technique to the data partitioning problem. These techniques tend to be computationally very intensive [Wil90]. Another alternative is to use Hopfield neural networks [Hop82] whose objective is to minimize an energy function associated with the combinatorial problem. In [Fox86b], [HKB90] and [Man92] various artificial Hopfield neural networks have been developed for the solution of the data partitioning problem. This approach also tends to be computationally very intensive.

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

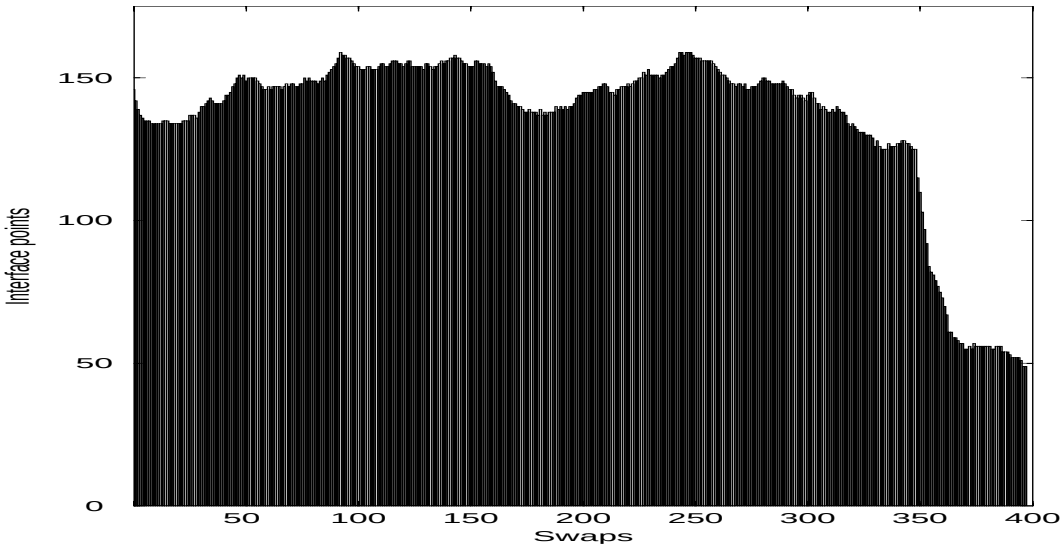


Figure 5: Performance of local search to reduce the size of the separator (number of interface points).

4 Data Parallel Iterative Solvers

We want to minimize the synchronization and communication costs of the data parallel PDE iterative solvers [CHK+92] based on discrete domain decomposition methods by finding an optimal solution for the partitioning and the allocation of the computation to the processors of a distributed memory MIMD machine. The structure of these computations is inherently parallel and suitable for MIMD machines. Figure 6 suggests a formulation of our approach implemented in the parallel ELLPACK system [HRC+90] using the nCUBE II machine.

Assumptions

First, we assume that the targeted parallel machine consists of a network of processors connected by communication links. Each processor exchanges information in *packets* whose lenghts vary from a few tens of bytes [nCU91] to several thousands of bytes [iPS90]. The bytes of a packet are consecutively transmitted without interruption. Sending or receiving a message stored in a buffer is the transmission of a number of packets. The local memory of each processor is used for storing some problem data and intermediate results (local data structures).

Communication requirements of the data parallel PDE iterative solvers

The iterative PDE solvers of a linear system of algebraic equations can be reduced to matrix-vector multiplication operations (see [HY81] and [KRYG82]). The operations consist of two steps : (a) the *local communication* of data between subdomains and (b) the *local computation* (see [FJL88], [CR92]). Throughout this paper, we also refer to it as *local synchronization*. A high level view of the steps of an iterative solver (that preserves the ordering of the corresponding sequential computation) for the *discrete domain decomposition* methods pertinent to the data mapping issue is : (i) *Local Synchronization*, (ii) *Local Computation*, and (iii) *Global Synchronization.* In this work we address only the local synchronization issue and not global synchronization. The local synchronization 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, see Figure 3. 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].

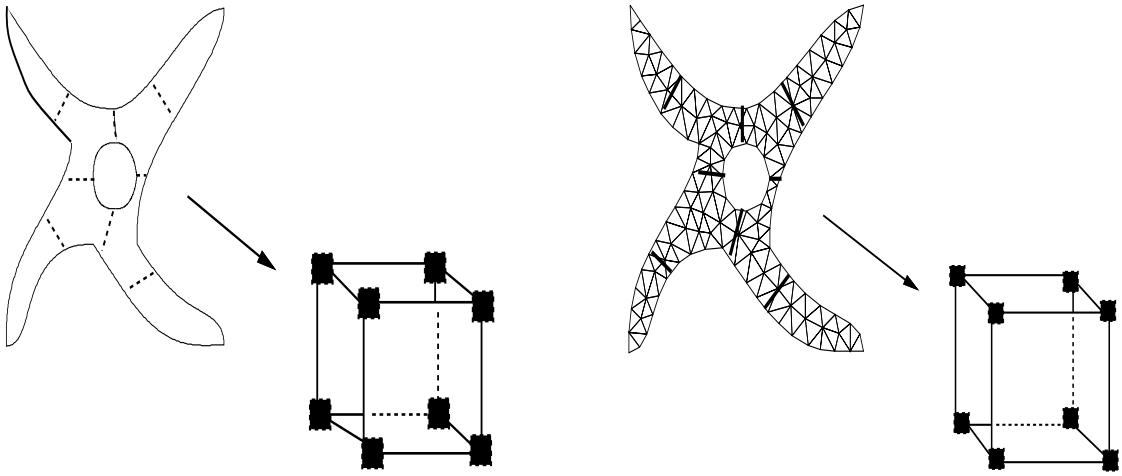


Figure 6: A problem partitioning and allocation methodology for PDE solvers on MIMD machines based on the domain decomposition approach.

The local synchronization mechanism used is as follows :

1. T_{copy} : Copy inner interface unknowns from local data structures to a buffer.
2. T_{send} : For each $D_j \in C_{D_i}$ send $S_{buffer(j)}$ to the processor $m(D_j)$
3. T_{recv} : For each $D_j \in C_{D_i}$ receive $R_{buffer(j)}$ from the processor $m(D_j)$
4. T_{copy} : Copy the outer interfaces $R_{buffer(j)}$ $\forall D_j \in C_{D_i}$ to local data structures.

The execution time of the local synchronization scheme is analyzed as follows. The execution time for the processor $m(D_i)$ is decomposed into three components, namely the time to send (T_{send}) a set of messages to processors $\Pi(m(D_i)) = \{m(D_j) \ \ processor, \ \ where \ D_j \in C_{D_i}\}$, the time to copy ($T_{copy}$) the local data structures into and from a buffer, and the time to receive (T_{recv}) the messages from the set of processors $\Pi(m(D_i))$. Thus, the total local synchronization time is modeled by :

$$T_{LS}^{m(D_i)} = 2T_{copy} + T_{send} + T_{recv} \tag{4.1}$$

In this relation the T_{send} is the time required by the processor to assemble the message and move it to the appropriate buffer. This includes tasks like appending and addressing information as well as selecting a link on which to transmit the message. T_{send} depends on architectural parameters like packet or circuit switching mechanisms, the size of the message buffer, and resource management , as well as on problem parameters like the number of neighbor subdomains (i.e., $\mid C_{D_i} \mid$) and the number of interface points (i.e., $\sum_{D_j \in C_{D_i}} c(D_i, D_j)$). The time T_{copy} mainly depends on the interface length (i.e., $\sum_{D_j \in C_{D_i}} c(D_i, D_j)$). Finally, the time T_{recv} depends on all the factors of T_{send} plus any delays due to the messages not being ready to receive; note that for almost all commercially available distributed memory MIMD parallel machines the receive operation is a blocking operation.

There are two ways to minimize the processors' idle time due to the blocking of the receive operation and due to the difference in actual and expected order of message arrivals. The first obvious way is to order the messages of each processor so that the actual and expected orders of message arrivals are identical. The computation of such an ordering scheme is yet another difficult optimization problem. We feel that the preprocessing overhead is too high for this approach.

A simpler and less expensive way to implement T_{recv} is to use primitives like *ntest* [nCU91] and busy-wait mechanisms. The following algorithm demonstrates a run time ordering scheme which minimizes the message passing idle time in the processors $m(D_i)$ by slightly increasing its computation by few cycles (*ntest* execution time).

Run Time Ordering Scheme :

1. isrc = -1
2. for i = 1 to C_{D_i}
3. while ((isize = ntest(isrc, isize) > 0)
4. get message form source isrc
5. set isrc = -1;
6. endfor

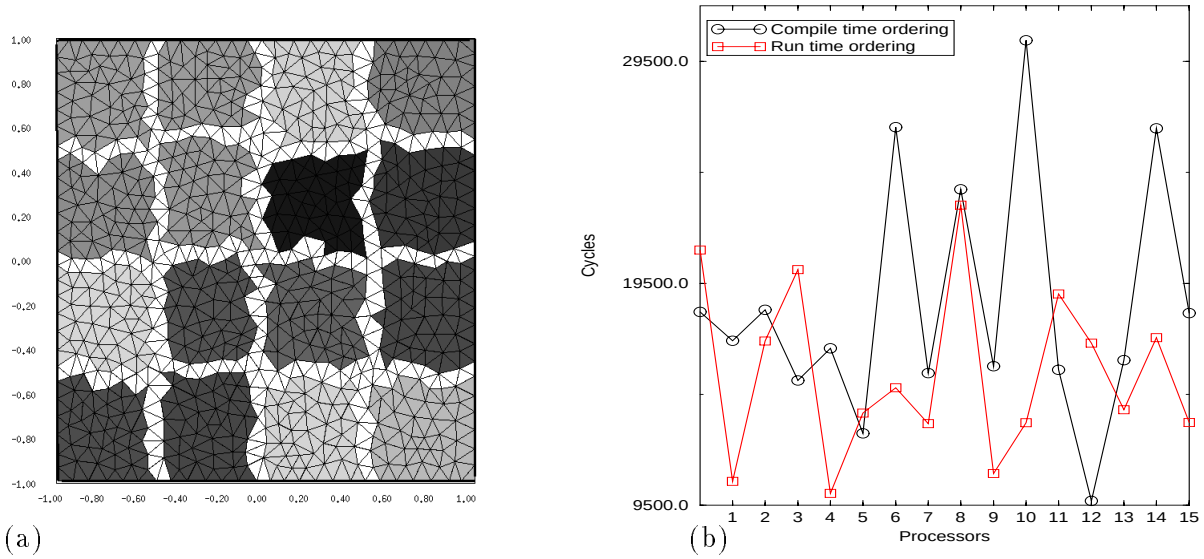


Figure 7: (a) A 4×4 partitioning of a rectangular domain. (b) Performance comparison of run time (squares) and compile time (circles) ordering of message arrivals for this problem on a 16 node nCUBE II.

Figure 7b illustrates the performance of the receive operation for two different implementations and the 4×4 partition of a rectangular domain (7a). The partitions of a moderate sized problem (27,000 equations) are mapped using 2-dimensional gray code (optimum mapping). The first implementation (shown with circles) is a blocking compile time ordering of the local messages and the second one (shown with squares) is a non-blocking run time ordering using busy-wait and primitives like *ntest* and *nread* of the nCUBE II or *irecvc* on the iPCS/860 and DELTA machines. The non-blocking run time FIFO mechanism is cheaper than the compile time ordering of the local messages by 10,000 cycles.

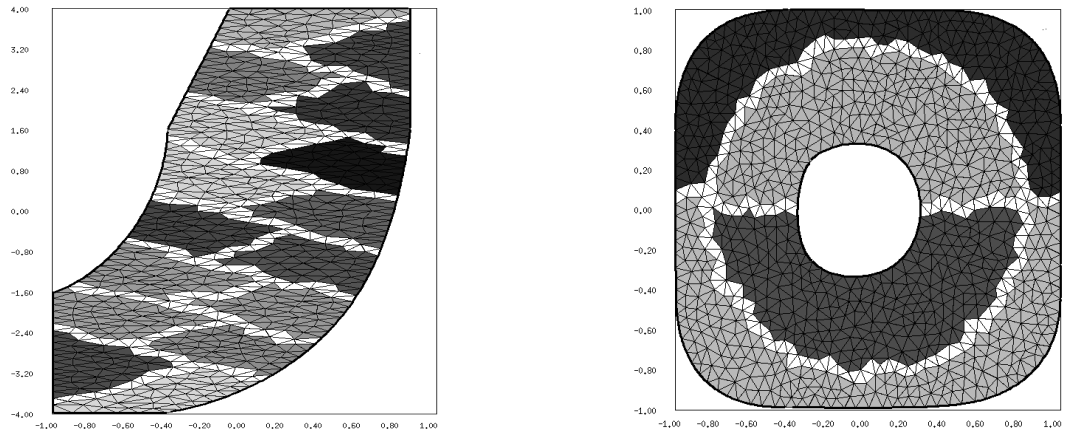
5 Data Block Partitioning Algorithm

The failure of the P×Q data partitioning algorithm to always produce partitions with connected subdomains is the result of the choice of attributes (Cartesian Coordinates) used for the clustering. We propose to replace the Cartesian coordinates of the nodes by attributes that characterize the boundary shape (geometry) of the physical domain, namely by attributes associated with the curvilinear coordinate system that is defined by a boundary-value problem on the physical domain. This idea is used in numerical mesh generation [TWM85] and provides the key to remove the problem of boundary shape from data partitioning algorithms. Examples are seen in Figure 8, where on the left, the clustering of the nodes is first along the x^*-axis and then along the y^*-axis of the coordinate system (x^*, y^*) defined by the following transformation :

$$\begin{bmatrix} x^* \\ y^* \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

On the right, cylindrical coordinates (r, θ) are used, where $r(x, y) = \sqrt{(x^2 + y^2)}$, $\theta(x, y) = tan^{-1}\frac{y}{x}$.

We can use *boundary-conforming curvilinear coordinate systems* to generalize this heuristic for more general 2D (or 3D) star-shaped [Prep 88] domains as follows. For $\mathbf{P}$ = P×Q processors : (1) sort the node points (or elements) along the coordinate lines conforming to the boundaries (analogous to the way in which lines of constant radial coordinate coincide with circles in cylindrical coordinate system), (2) group the node points (or elements) into P subgroups, and (3) sort the points of each subgroup along the other curvilinear coordinate (analogous to the angular coordinate in the cylindrical coordinate system). This coordinate varies monotonically along the boundary. Finally, group the node points (or elements) of each of the P subgroups into Q subgroups. Figure 9 illustrates the curvilinear lines of a 2D curvilinear coordinate system and shows a 16-way partitioning based on these curves. We call this the *boundary conforming P×Q algorithm.* This algorithm seems to appear expensive since we solve one PDE in the preprocessing step for solving another PDE. However the accuracy requirement in solving the preprocessing PDE that defines the curvilinear coordinate system can be quite low. This makes this approach feasible for practical use.



(a) (b)

Figure 8: (a) A 16-way partition based on attributes associated to (x^*, y^*) coordinate system and (b) A 4-way partition based on the cylindrical coordinates.

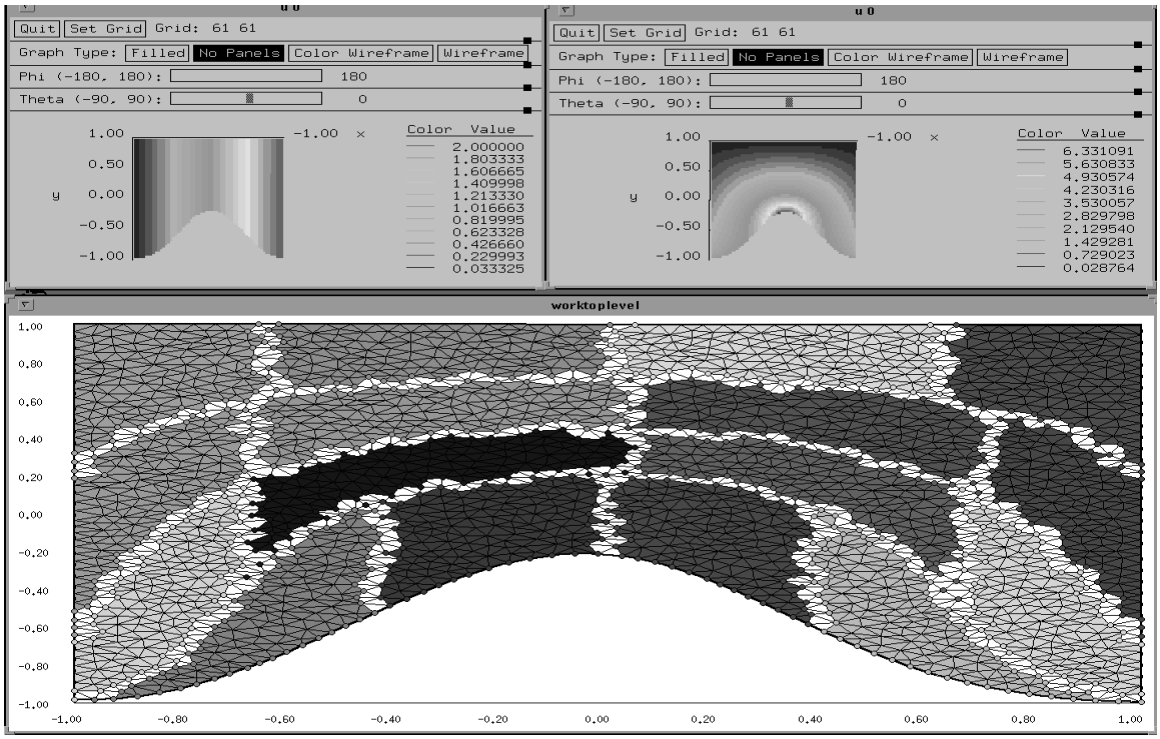


Figure 9: Block data partition based on curvilinear coordinate system defined by level curves of an elliptic problem on the domain.

6 Geometry Graph Partitioning (GGP) Heuristic

This section presents a partitioning heuristic based on local search algorithms for Euclidean graphs. The element mesh (or tensor-grid) of a 2D or 3D domain is an Euclidean graph, with vertices being the node points and links being the edges of the elements. The matrix and domain decomposition methods require quasi-uniform partitionings of the spatial domain with a minimum diameter. A partitioning heuristic for arbitrary graphs, like the KL heuristic, is unable to use the geometric properties of Euclidean graphs and produce partitionings appropriate for matrix and domain decomposition methods. In [CHENHR89] we present the *geometry graph partitioning* (GGP) heuristic which uses the geometrical properties of mesh graphs by using Euclidean metrics (see Figure 10) and minimizes the diameter of the subdomains, thus it can deliver quasi-uniform partitions with the minimal diameter. Next we give an improved (in terms of time and space complexity) version of the algorithm presented in [CHENHR89].

The partitioning problem of a discrete PDE domain is transformed into a *graph partitioning* problem of an Euclidean graph (mesh graph). Then this graph is decomposed by the GGP algorithm. The performance of the GGP algorithm is improved by representing the geometry and the topology of the graph with two *augmented open hash tables*. These data structures guarantee the *linear space* and *quasi-linear time* complexity of the KL and thus the GGP algorithm (see in [FM82] for more details on time complexity.) The cost function that GGP algorithm minimizes is given by :

$$\sum_{k,\ell=1}^{\mathbf{P}} \sum_{e_i \in D_k} \sum_{e_j \in D_\ell} \chi(e_i, e_j) \tag{6.1}$$

where $\chi(e_i, e_j) = 1$ if e_i and e_j are adjacent and in different subdomains and $\chi(e_i, e_j) = 0$ otherwise. The criteria (ii) and (iv) are imposed implicitly during the minimization of the objective function (6.1)

by seeking solutions that optimize certain function known as *profit* functions :

$$\sum_i (\omega_1 f(a_i, b_i) + \omega_2 g(a_i, b_i)) \tag{6.2}$$

where :

$$f(a_i, b_i) = 2 \sum_{e \in c_{a_i}} \chi(a_i, e) - |c_{a_i}| + 2 \sum_{u \in c_{b_i}} \chi(u, b_i) - |c_{b_i}| - 2\chi(a_i, b_i) \tag{6.3}$$

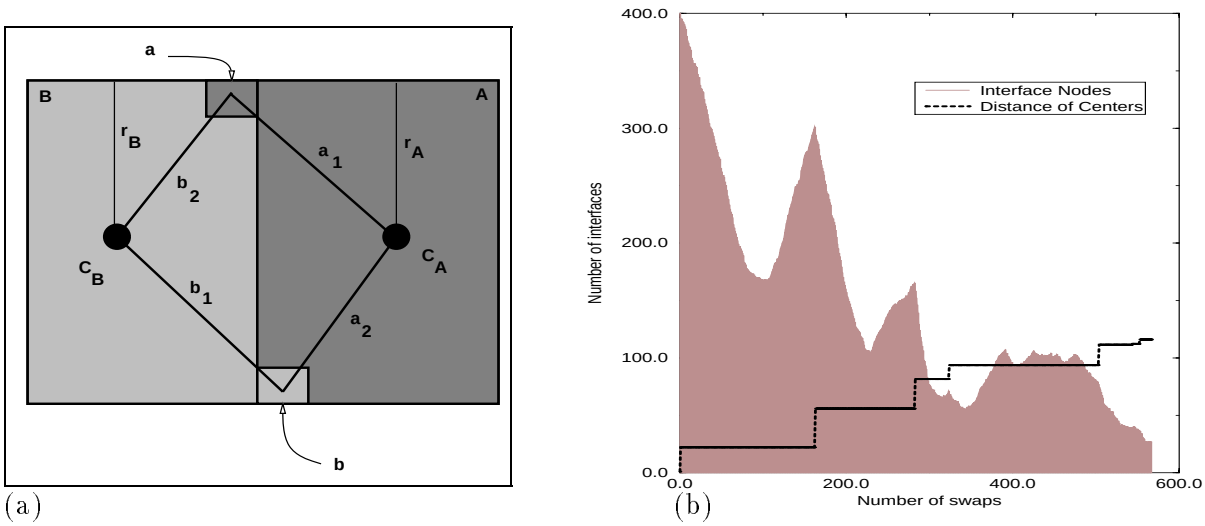
and

$$g(a_i, b_i) = (\frac{d_{a_i,c_A}}{r_A} - 1) - (\frac{d_{b_i,c_A}}{r_A} - 1) + (\frac{d_{a_i,c_B}}{r_B} - 1) - (\frac{d_{b_i,c_B}}{r_B} - 1) \tag{6.4}$$

These formulas use the following notation :

- $|c_{a_i}|$ and $|c_{b_i}|$ are the number of vertices adjacent to the vertices $a_i \in A$, $b_i \in B$ respectively,
- c_A, c_B are the mass center of the subdomains A, B (see Figure 10),
- d_{a_i,c_A} and d_{b_i,c_B} are the distances between the elements 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,
- 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 the function g which selects pairs of nodes whose swap reduces the diameter of the subdomains. The GGP algorithm climbs out of local minima of the objective function (6.1) by swapping points that might increase temporarily the value of the objective function but will decrease the diameter of the subdomains by bringing their mass centers far apart. The GGP algorithm is described in complete detail in Appendix A.



(a) (b)

Figure 10: (a) Illustration of the Euclidean graph metric. The points c_A and c_B are the mass centers of the subdomains A and B, d_{a_i,c_A} and d_{b_i,c_B} are the distances between the elements a_i and b_i and the mass centers c_A and c_B of the subdomains A and B, and r_A and r_B are the"ideal" radii of the subdomains A and B, for a 2-way partitioning of a quadrilateral mesh. (b) The values of the cost function (i.e number of interface points) 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.

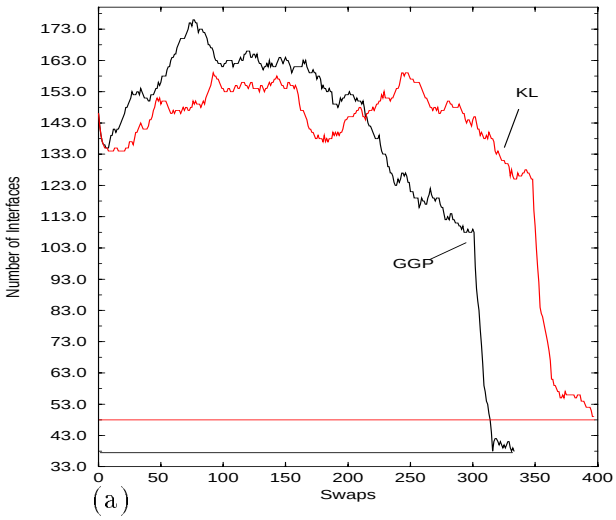
7 Performance Evaluation of Data Mapping Algorithms

In this section we present the performance evaluation of the following six data partitioning algorithms plus a comparison of the GGP and KL algorithms. The latter justifies not including a KL based recursive bisection method among the six algorithms evaluated carefully. The P×Q algorithm used in the evaluation is the one originally described in Section 3 and not the boundary conforming P×Q algorithm presented in Section 5. The latter will always be as good as the P×Q and in some cases be substantial better. Preliminary evaluation supports this belief but a complete set of performance data has not been collected similar to that given below for the six algorithms.

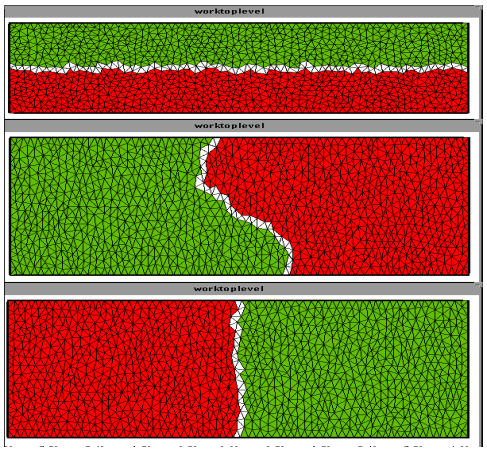
P×Q	:	Block partitioning along the x and y direction (Section 3).
1×Q	:	Strip partitioning along x or y direction (Section 3).
ScatDec	:	Scattered decomposition (Section 3).
CM-Clust	:	Clustering techniques based on an ordering of node points (Section 3).
RB	:	Recursive bisection based on 2-way rooted level structure (Section 3).
Hybrid	:	Recursive bisection using the GGP heuristic whose initial 2-way partitioning is determined by CM-Clust (Section 6).

7.1 Comparison of the GGP and KL Partitioning Heuristics

An experimental comparison of the GGP algorithm with the KL partitioning heuristic shows that GGP consistently returns, with less computation, partitionings whose separators are smaller. Figure 11a shows the evaluation of KL and GGP algorithms based on the quality of the partition (i.e., number of interface points) and the effectiveness of the swap operations of the algorithms. Both algorithm use as an initial partition the result of 1×2 algorithm (Figure 11b-top). Figure 11b also shows the final partition produced by KL algorithm (center), and the final partition produced by GGP algorithm (bottom).



(a)



(b)

Figure 11: Comparison of the GGP and KL algorithms. The initial partition is at the top, the result by KL algorithm is in the middle and by GGP algorithm is at the bottom. The efficiency of these algorithms is shown where the number of interface points is plotted as a function of the number of swaps made.

Figure 12 shows the partitions produced by KL algorithm (top middle) and GGP algorithm for different values of the weights ω_1 and ω_2. Both algorithms use as an initial partition the result of the CM-Clust algorithm (top right).

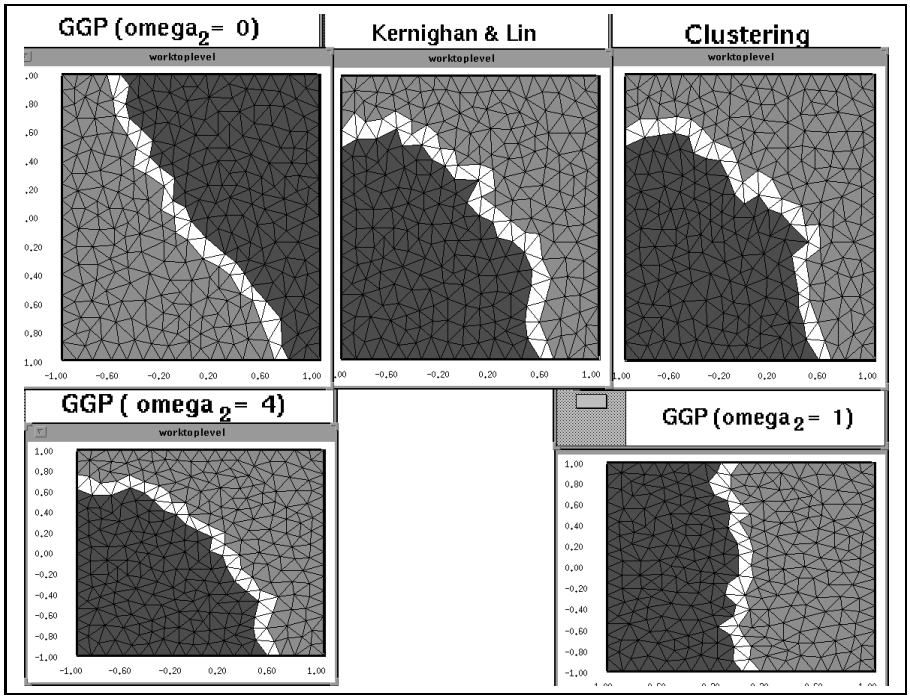


Figure 12: A comparison of 2-way partitioning algorithms. The initial partition is shown at the upper left corner. Three results (right top corner and bottom) for the GGP algorithm are shown with weights $\omega_1 = 1$ and $\omega_2 = 0$, 1, and 4. The KL algorithm result (top-middle) and GGP without using the distance criterion (i.e., $\omega_2 = 0$) are not so good.

7.2 Machine-independent Evaluation

The evaluation of the six algorithms is divided into two phases, a machine-independent phase and a machine-dependent phase. In the machine-independent phase we measure the relative satisfiability of the criteria (i) to (iv) presented in Section 2, since the optimum solution of the data partitioning problem is not known. In the machine-dependent phase, we first measure the impact of the different degrees of satisfiability of the load balance, the degree of subdomains connectivity and the number of interface points on the local communication time of the data parallel iterative PDE solvers. Then we measure the message passing overhead of the Jacobi-SI method for different data partitions. All the reported timing data reflect the performance per iteration, since the convergence rate of data parallel, point, semi-iterative methods does not depend on the data partition. The evaluation of the data partition algorithms is performed on two model problems with a Poisson PDE operator and Dirichlet boundary conditions (the data partition is independent of the PDE operator). The difference between the two problems is in the complexity of the domain. The domain of the Model Problem A (Figure 13) is very simple 2-dimensional, almost convex domain, while the domain of the Model Problem B (Figure 14) is an irregular non-convex domain with a hole. Another difference is in the size of the triangular mesh and thus, the size of the computation. The mesh for the Model Problem A consists of 57,756 elements, 29,223 nodes and generates 28,535 equations, while the mesh for the Model Problem B consists of 18,890 elements and 9,880 nodes and generates 8,981 equations.

For Model Problem A the P×Q and boundary conforming P×Q are essentially identical so we expect very little difference between their performance for this problem. Model problem B is just at the limit of geometric complexity that can be handled easily by the boundary conforming heuristic. The technique shown in Figure 8b does not quite work because the domain for this problem is not star-shaped. One could make an ad hoc modification for this domain to make the appropriate curvilinear coordinate system. An examination of Figure 17a supports that the P×Q partition might be improved substantially for criterion (iv), appearance of the splitting. The performance improvements in the other criteria might be less substantial.

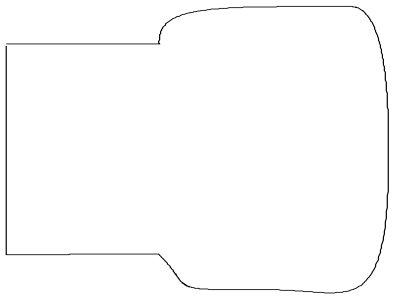


Figure 13: Model Problem A.

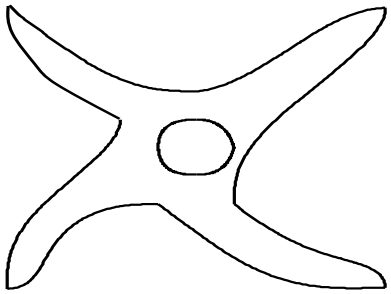


Figure 14: Model Problem B.

The evaluation of the data partitioning algorithms is based on the following indicators : a) the minimum, average, maximum, and difference between the maximum and minimum number of equations per processor. Note that the number of equations (or active node points) and *not the number of total mesh points* is the indicator of the load balance. b) the ratio of the total number of interface points to the total number of the points of the mesh, c) the ratio of the number of interface points to the total number of points per subdomain, d) the average connectivity of the subdomains, e) the maximum connectivity of the subdomains, and f) the splitting of the subdomains. The last criterion is evaluated by inspection.

Most data partitioning algorithms in the literature balance the work-load of the processors by minimizing the difference between the minimum and maximum number of active and non-active mesh points (Ω^h). These algorithms ignore the boundary conditions (Dirichlet, Newmann, Mixed) of the boundary points ($\partial\Omega^h$). In this paper the load balance of the data parallel iterative solvers is measured by the difference between the minimum and maximum number of *active* mesh points only (i.e., equa-

tions) per processor. This is possible by symbolically analyzing the boundary conditions of the PDE problem. Table 1 indicates the load imbalance produced by the data partitioning algorithms using only the active mesh points, while Table 2 indicates the load imbalances produced by the same data partitioning algorithms using both the active and non-active mesh points. Table 1 shows that only the data clustering algorithms produce partitions with perfect load balance. The optimization algorithms using local search techniques are capable of preserving the load balance of total (active and non-active) nodes of an initial partition (see swap operation, Section 3); but are not equipped with constraints to keep the number of active mesh points in balance. A version of the GGP that uses the information related to the boundary conditions of the node points and a swapping operation with additional constraints that enforce the balance of active node points is under development. Finally, Table 2 indicates that **none** of the above data partitioning algorithms applied on all node points (active and non-active) leads to the perfect load balance of the computation.

Table 1: The number of equations per processor for data partition algorithms applied only on the active (non-boundary) points of the mesh for Model Problem A.

	P×Q	1×Q	ScatDec	CM-Clust	RB	Hybrid
MINIMUM	444	445	428	445	402	391
AVERAGE	446	446	446	446	446	446
MAXIMUM	446	446	448	446	458	476
MAX − MIN.	2	1	20	1	56	85

Table 2: Indicates the number of equations per processor for data partition algorithms are applied on active and non-active node points of the mesh for Model Problem B.

	P×Q	1×Q	ScatDec	CM-Clust	RB	Hybrid
MINIMUM	107	101	109	86	76	117
AVERAGE	140	140	140	140	140	140
MAXIMUM	155	152	144	155	155	155
MAX − MIN.	48	49	35	69	79	38

Figure 15 shows the percentage of interface node points per subdomain as the number of processors increases. This measure is the ratio of the number of interface points to the total number of points which is closely related to the ratio of communication time to computation time. Figure 16 shows the maximum and average degree of connectivity of the subdomains. Later we will see that subdomains (processors) with high degree of connectivity have higher local communication due to startup latency, edge/node contention and large differences in actual and expected message arrival times. Figure 17 shows two partitions for the Model Problem B and criterion f) is evaluated subjectively inspecting the splitting. In this case, as usual, the Hybrid algorithm produces the partition most pleasing to the eye.

7.3 Machine-dependent Evaluation

The objective of data partioning algorithms is to distribute the mesh over the processors so that the solvers spend minimum time in interchanging the data required for their local synchronization so they must be evaluated by the actual performance of the data parallel solvers. In [Chr92] we found

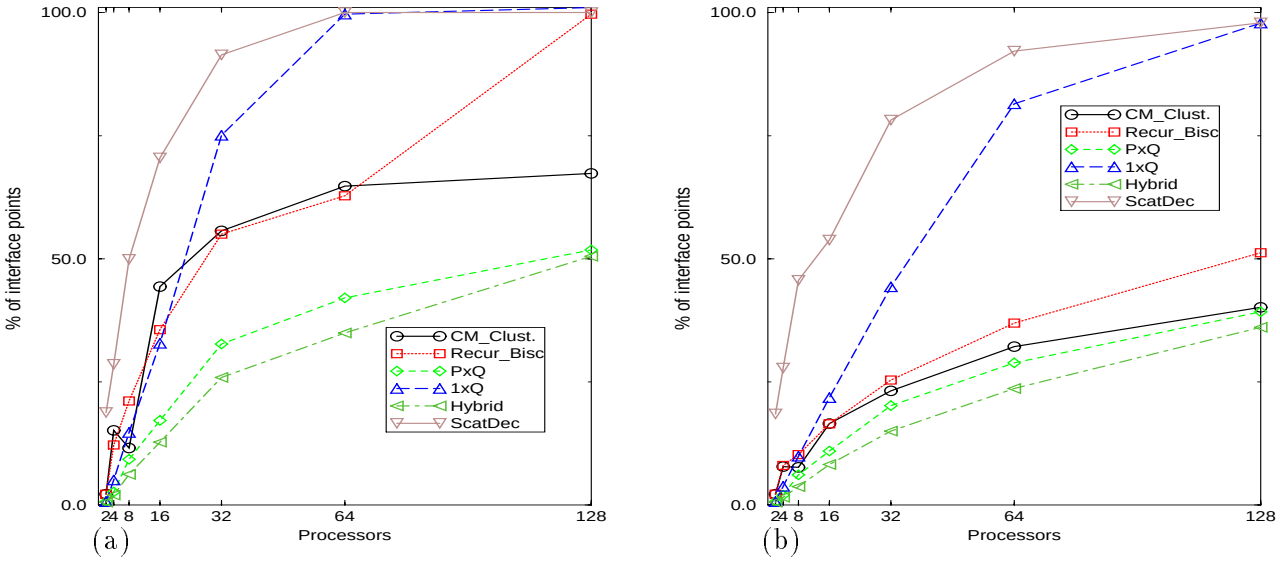


Figure 15: (a) The percentage of interface node points per subdomain and (b) the percentage of interface node points in the total points for Model Problem B and the six data partitioning algorithms listed above.

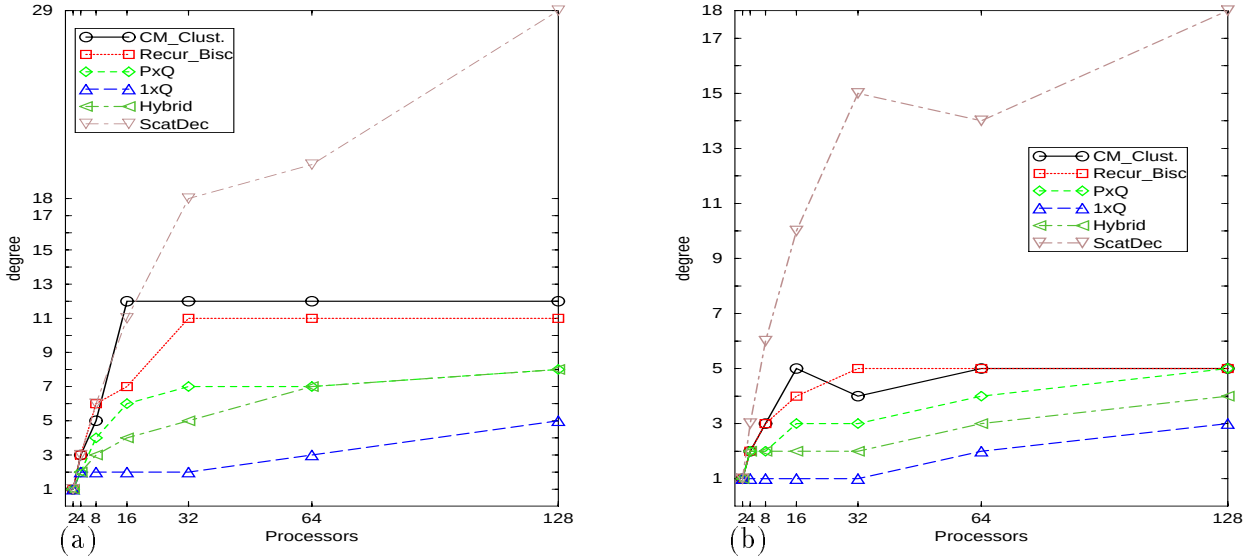
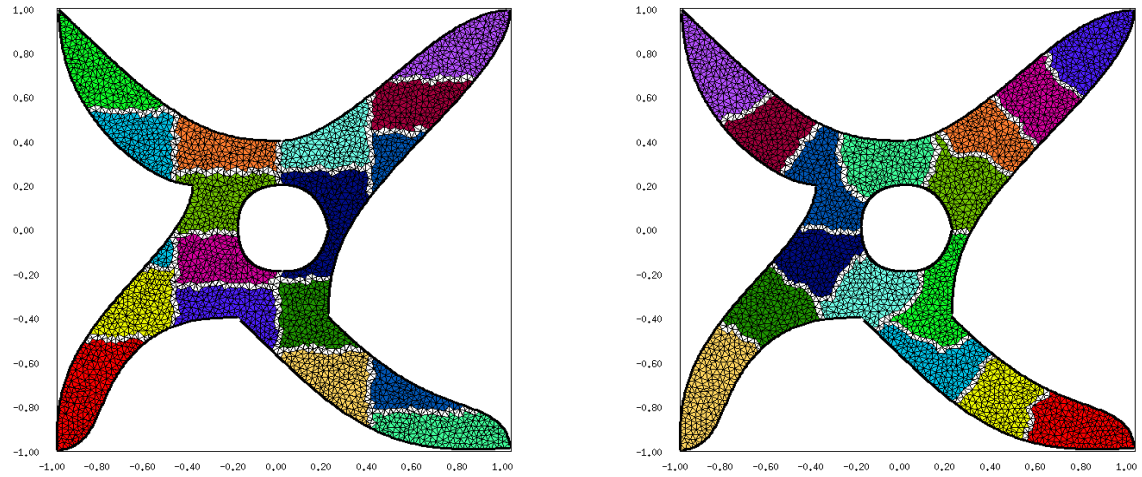


Figure 16: The maximum degree (a) and average degree (b) of the decomposition graph for Model Problem B and the six data partitioning algorithms listed above.



(a) (b)

Figure 17: The subdomains produced by the 4×4 (a) and Hybrid (b) data partitioning algorithms for Model Problem B.

that the performance of the solvers is independent of the allocation of the subdomains to processors for small configurations ($\mathbf{P} \leq 64$).

In this section we first examine the impact of the optimization of different criteria on the message passing overhead of the data parallel iterative solvers and then we evaluate the partitioning algorithms with respect to the time spent by the parallel solver for message passing and delays due to load imbalance.

Three measures for the performance of data parallel iterative solvers

We use three measures : (a) the impact of an uneven distribution of mesh points on the processors work load for the nCUBE II machine, (b) the relationship between the connectivity of the subdomains and the message passing overhead ($2T_{copy} + T_{send} + T_{recv}$) of the PDE solvers (see Section 4) , and (c) the relationship between the number of interface points (size of the graph separator) and the message passing overhead.

To measure the impact on the uneven distribution of the computation we use the Model Problem A (see Figure 13) and the P×Q partition algorithm in two different ways. In Case (I), we consider only the mesh points that are active (i.e., the points which correspond to an equation), these points are all the interior mesh points and the boundary points with boundary conditions other than Dirichlet. In Case (II), we use the same algorithm but we consider all the mesh points (active and nonactive). Table 3 gives the performance measures for the two cases.

To measure the relationship between the connectivity of the subdomains and the message passing overhead of the data parallel iterative solvers we use the P×Q partitioning algorithm and another algorithm, Ext_P×Q [Chr92], which is a simple modification of the P×Q algorithm. The Ext_P×Q algorithm reduces the connectivity of the subdomains by uncoupling subdomain (i,j) from the subdomains (i+1, j+1) and (i−1, j−1). The uncoupling takes place by properly extending the interfaces of the other surrounding subdomains. Figure 18 shows the relationship between the message passing overhead of the Jacobi SI method and the subdomain connectivity for P×Q and Ext_P×Q algorithms using the nCUBE II with 64 processors. There is very hight correlation between processors that handle

Table 3: The impact of the distribution of the points is measured in terms of the number of actual equations being solved per processor, and the resulting total elapsed time of the Jacobi-SI method used on Model Problem A.

	Case I	Case II
MINIMUM	444	409
AVERAGE	446	446
MAXIMUM	446	457
MAX − MIN.	2	48
SOLVER TIME	11.53	11.67

subdomains with high connectivity and those that have higher message passing overhead and that the pattern of the message passing overhead is dominated by the connectivity pattern of the subdomains.

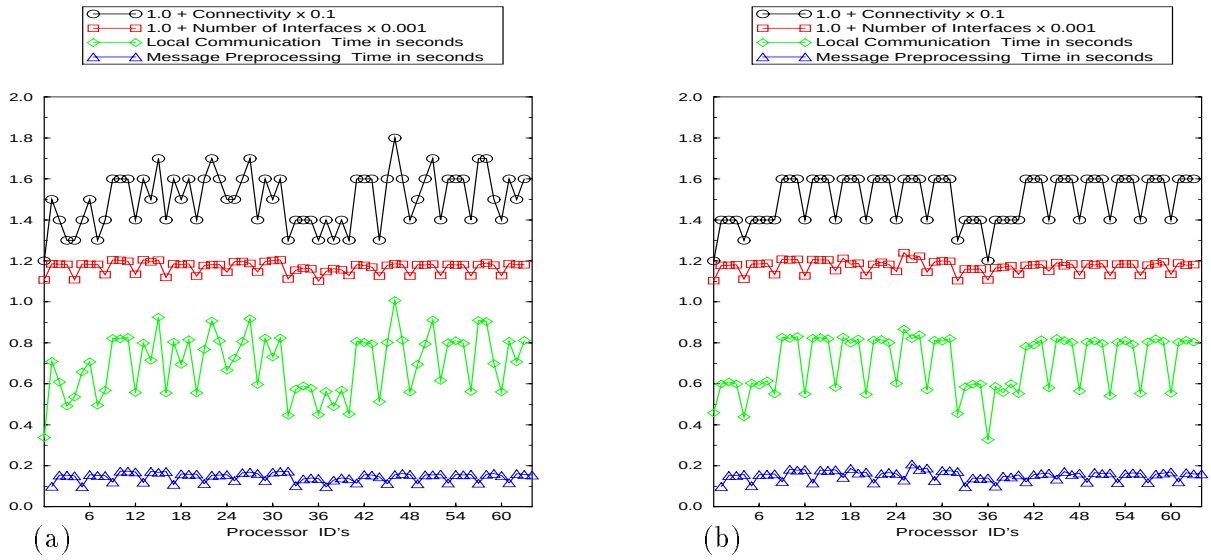


Figure 18: Relationship between the connectivity and message passing overhead of the Jacobi SI method on nCUBE II with 64 processors and Model Problem A for (a) P×Q block partition and (b) Ext_P×Q block partition.

To measure the relationship between the number of interface points and the message passing overhead for the Jacobi-SI method on the nCUBE II ($\mathbf{P} = 64$) we use the 1×Q partition algorithm. We force the 1×Q partitioning algorithm to generate subdomains with small connectivity but a very large number of interface points. Figure 19a shows the relationship between the number of interfaces and the message passing overhead for communication with physically neighboring processors. The 1-dimensional gray code mapping of the subdomains to processors is optimum and preserves the nearest-neighbor property. Figure 19b shows that the imbalance of the computation due to the difference in the number of interface points of the subdomains. Note that the global communication time includes the idle time of a processor due to waiting for other processors to finish their local communication since in this case the computational work-load of the processors is perfectly balanced (see Table 1).

Finally, we evaluate the partitioning algorithms with respect to the total performance of the iterative solver. Table 4 shows the difference in the load balance of the computation of the Jacobi-SI method for the six partitioning algorithms, and also shows the additional computation due to load imbalance.

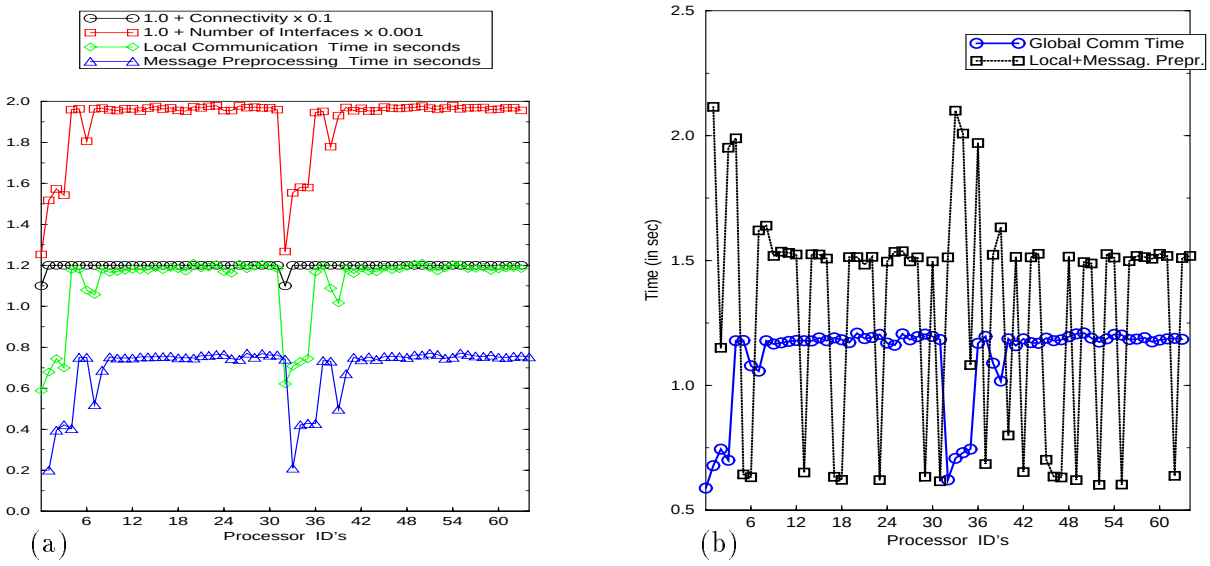


Figure 19: Relationship between the number of interfaces and message passing overhead of the Jacobi-SI method on the nCUBE II with 64 processors and Model Problem A, using the (a) 1×Q block partition algorithm, and (b) the global and local communication.

Clustering algorithms result in partitions with perfect load balance. Table 5 shows the degree of connectivity along with the ratio of the time spent in sending and receiving messages over the total elapsed time spent by slowest processor. It also shows the number of interface points along with the percentage of time spent in filling the communication buffer over the the total elapsed time required by the slowest processor for the Model Problem A. We use the Jacobi SI method and the nCUBE II with 64 processors. Table 6 shows the same data for Model Problem B. These data strongly suggest that the simple and cheap P×Q algorithm performs very well compared to the others.

Table 4: Load balance obtained using the data partitioning algorithms for the Model Problem A and 64 processors. The percentage of the additional time due to load imbalance is given at the bottom.

	P×Q	1×Q	ScatDec	CM-Clust	RB	Hybrid
MINIMUM	444	445	428	445	402	391
AVERAGE	446	446	446	446	446	446
MAXIMUM	446	446	448	446	458	476
MAX – MIN	2	1	20	1	56	85
ADDT WK	0.45	0.22	4.46	0.22	12.23	17.86

8 DecTool

In this section we describe a software system which assists the user to visualize and manipulate domain decompositions in the environment of the Parallel ELLPACK (// ELLPACK) system [HRC+90]. The // ELLPACK system is a prototype intelligent parallel programming environment for solving PDEs defined on two and three dimensional domains. It is implemented on a hardware facility consisting of a graphics workstation supporting the X11 window system and connected to the nCUBE II

Table 5: The column N gives the degree of connectivity and number of interfaces of the data partitioning algorithms for the Model Problems A and B. The column Time gives the send/receive time divided by the total elapsed time and percentage of the elapsed time spent in buffer filling.

Problem A	P×Q		1×Q		ScatDec		CM-Clust		RB		Hybrid	
	N	Time	N	Time	N	Time	N	Time	N	Time	N	Time
Connect	8	7.31	2	5.03	13	11.34	21	15.35	11	9.52	8	7.15
Interf	206	1.52	979	6.58	783	5.01	677	6.19	346	2.34	224	1.69

Problem B	P×Q		1×Q		ScatDec		CM-Clust		RB		Hybrid	
	N	Time	N	Time	N	Time	N	Time	N	Time	N	Time
Connect	7	14.40	3	7.81	20	32.08	22	38.36	11	21.74	7	14.96
Interf	135	2.68	388	7.06	431	6.65	215	3.21	211	4.05	112	2.85

machine through a local area network. The software infrastructure includes a) a PDE problem oriented language processor, b) a geometry processing tool which is capable of generating meshes and their decompositions either automatically or interactively, and c) an algorithm mapper facility for partitioning and mapping the underlying PDE computation onto the nCUBE.

The interactive environment called *DecTool* (short for Domain Decomposer Tool) [CHENH+91] provides facilities for both automatic (using predefined algorithms) and manual decomposition of a given 2-D or 3-D discrete domain. An example display is shown in Figure 20.

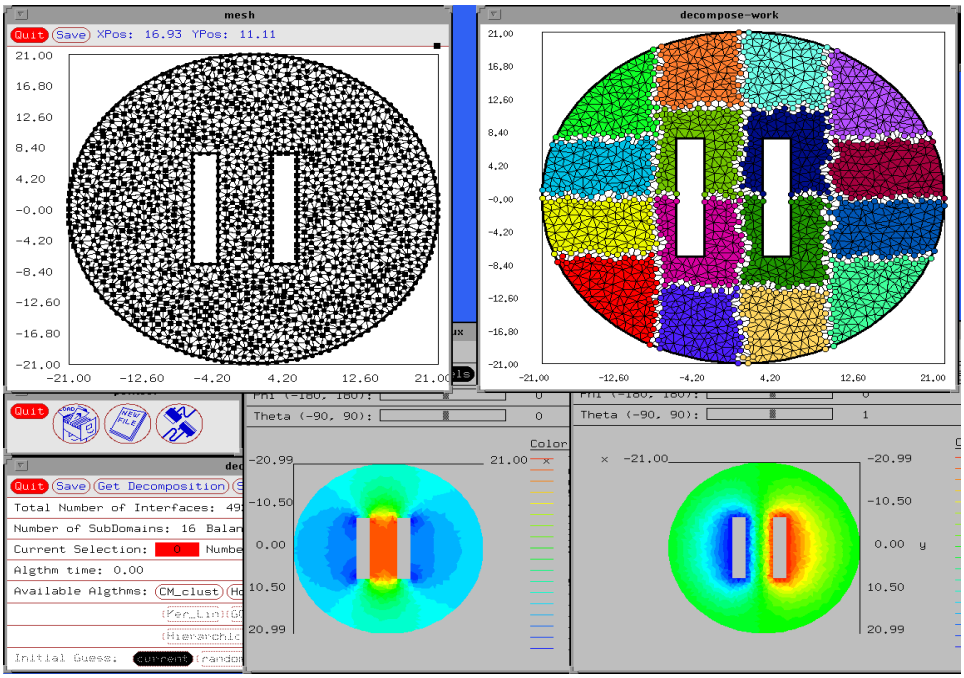


Figure 20: An instance of the the DecTool and Parallel ELLPACK environment.

The DecTool consists of three different windows. In the main window, there are three additional widgets for invoking the library decomposition techniques and specifying the appropriate initializations. The decompositions are displayed and manipulated in another window. Each subdomain is colored differently and the interface nodes are displayed as colored circles or squares. The colors indicate the

assignment of subdomains (processes) to processors with a color map (color palette) displayed in a different window.

DecTool was used in the performance evaluation to create the domain partitions using various algorithms. It provided the visual displays and data for measuring the criteria used. A few of the measurements (e.g. number of interface points) are provided automatically by DecTool but most required specific programs to be run within the DecTool environment.

9 Discussion - Conclusions

The qualitative comparison of the data clustering and deterministic optimization algorithm is based on the inspection of the partitionings resulting from the six representative algorithms we evaluated. The P×Q, 1×Q and Scattered Decomposition algorithms for general domains result into partitions with *disconnected subdomains*. Moreover, the 1×Q algorithm partitions the domains into narrow subdomains with *large diameter*, while the CM-Clust and Recursive Bisection algorithms partition the domains into disconnected subdomains with "fuzzy" interfaces. *None* of the above algorithms utilizes information about the shape of the boundary of the domain. The Block partitioning algorithm based on curvilinear-coordinates has been designed to reflect the boundary shape of the star-shaped domains, a large class of non-convex domains, and gives partitions with connected subdomains. Finally, the Hybrid algorithm, a recursive bisection algorithm based on data clustering and deterministic optimization, gives more compact subdomains with relatively small diameter. The Hybrid algorithm is designed to minimize the diameter of the initial partition and the size of the interfaces. The Hybrid algorithm creates partitions that are visually pleasing.

The CM-Clust algorithm has been implemented with open hash tables and thus is of O(N) time complexity. The P×Q, 1×Q and Scattered Decomposition algorithms have O(NlogN) time complexity. The divide-and-conquer algorithms, Recursive Bisection and Hybrid are more computationally intensive algorithms. The Hybrid is the most expensive algorithm for three reasons : (1) it requires an initial partition which is at least of O(N) time, (2) it performs a number of iterations on the initial partition, each iteration is of O(N) time but the number of iterations depends on the initial partition, and (3) it requires integer and floating point operations while the rest of the algorithms require only integer operations with an exception of spectral bisection that uses the Lanczos algorithm to compute the Fiedler eigenvector.

Table 7 summarizes the qualitative and quantitative evaluation of the data partition algorithms. The overall evaluation of the algorithms is based on their performance on general non-convex domains. Tables 8 and 9 show the impact of different partition algorithms on the total elapsed time and message passing overhead.

Table 6: Performance evaluation of all six partitioning methods considered with respect to the satisfiability of criteria (i) to (iv) and minimization of local communication time.

Algorithm	(i) Load Balance	(ii) Interfaces	(iii) Connectivity	(iv) Splitting of Subdomains
P×Q	Perfect	Very Good	Small	Most of the Time
1×Q	Perfect	Poor	Small	Always
ScatDec	Very Good	Poor	Large	Always
CM-Clust	Perfect	Poor	Large	Always
RB	Good	Good	Large	Always
Hybrid	Good	Very Good	Small	Some Times

Table 7: Elapsed time per iteration (in microseconds) of the Jacobi-SI method for Model Problems A and B.

	P×Q	1×Q	ScatDec	CM-Clust	RB	Hybrid
Problem A	2.89e−2	2.94e−2	3.16e−2	3.20e−2	3.06e−2	3.00e−2
Problem B	1.39e−2	1.26e−2	1.63e−2	1.48e−2	1.38e−2	1.24e−2

Table 8: The message passing overhead of the Jacobi-SI for Model Problems A and B.

	P×Q	1×Q	ScatDec	CM-Clust	RB	Hybrid
Problem A	2.51e−3	2.94e−3	5.07e−3	5.46e−2	2.99e−3	1.99e−3
Problem B	2.17e−3	1.83e−3	6.01e−3	6.15e−3	3.36e−3	2.09e−3

Tables 4, 5, 6, 7, 8, 9 and the above discussion indicate that the simplest, oldest and one of the least expensive algorithms, **P×Q**, is the most suitable data partition algorithm for the data parallel iterative solvers based on the *discrete domain decomposition* approach for solving PDEs, while the **Hybrid** algorithm is the most suitable for the data parallel iterative solvers based on the *continuous domain decomposition* approach. These six tables provide 14 quantitative measures of performance for these six algorithms. Table 9 gives the average and worst rank (rank 1 = best, rank 6 = worst) of each of the algorithms for these measures. In computing the ranks similar measurements (ones within 5 or 10 %) are counted as ties. The overall superiority of the P×Q and Hybrid algorithms is quite apparent from this summary data. Table 10 indicates that **none** of the above algorithms is suitable for the solution of the data partition problem for very large meshes since most of the time is spent *not on the data partition* but on initializing, meshing and loading sequentially the huge data structures onto the nodes of the processors. These data imply that problem preprocessing work such as mesh generation and problem partitioning must be moved from the workstation to the MIMD machine. This, in turn, requires that new parallel approaches to mesh generation, data partition, and problem initialization be developed. Thus a basic change in the paradigm for solving the PDE problem is needed rather than just finding efficient parallel implementation for the data partitioning algorithms. Of course the parallel I/O of the most recent parallel machines will improve the loading time but still the I/O will remain a bottleneck.

Table 9: Average and worst ranks of the algorithms for the 14 quantitative measures of performance (rank = 1 is best).

	P×Q	Hybrid	1×Q	RB	ScatDec	CM-Clust
Average rank	1.6	2.1	2.5	3.6	4.4	4.8
Worst rank	3	5	6	5	6	6

Table 10: The execution time of the different phases required for the sequential preprocessing for the numerical solution of Model Problems A and B using the P×Q partition algorithm. The time is in seconds on the SPARC workstation and nCUBE II.

SPARC Workstation phases	Problem A.	Problem B
Mesh generation	24.01	10.56
Initialize decomposer	7.73	2.56
Partition of domain.	8.35	2.70
Save data	51.88	20.78

nCUBE II phases	Problem A.	Problem B
Load data	57.45	11.37
Synchronize/initialize processors	11.93	17.91
Discretize PDE	2.06	0.75
100 iterations of solver	2.93	1.40

Acknowledgements We thank all members of the parallel ELLPACK group from 1988 to 1992 for their dedication in the parallel ELLPACK project.

10 Appendix A

References

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

[ANN90] M. Al-Narsa and D. T. Nguyen. An algorithm for domain decomposition in finite element analysis. *Computers and Structures*, 39(3/4):277 – 289, 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 (PhD thesis), Computer Science Department, Purdue University, W. Lafayette IN, 1992.

[CR87] Tony F. Chan and Diana C. Resasco. A domain-decomposed fast Poisson solver on a rectangle. In *Selected Papers from the Second Conference on Parallel Processing for Scientific Computing* (C. W. Gear and R. G. Voigt, editors), pages 14–26. SIAM, Philadelphia, 1987.

[CSS86] Tony F. Chan, Youcef Saad, and Martin H. Schultz. Solving elliptic partial differential equations on hypercubes. In *Hypercube Multiprocessors 1986* (Michael T. Heath, editor), pages 196–210. SIAM, Philadelphia, PA, 1986.

[CT88] T.F. Chan and R. S. Tuminaro. A survey of parallel multigrid algorithms. *Parallel Computers and Their Impact on Mechanics*, 86:155–170, 1988.

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

[FM82] Charles M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *Proceedings of the 19th IEEE Design Automation Conference*, pages 175–181, IEEE Press, 1982.

[FK89] Horance Flatt and Ken Kennendy. Performance of parallel processors. *Parallel Computing*, 12:1–20, 1989.

[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.

[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.

[Geo73] Alan George. Nested dissection of a regular finite element mesh. *SIAM Journal Numerical Analysis*, 10(2):345 – 363, 1973.

[GGMP88] R. Glowinski, G. Golub, G. Meurant, and J. Periaux. *First International Symposium on Domain Decomposition Methods for Partial Differential Equations*. SIAM, Philadelphia, 1988.

[Gil80] John Russel Gilbert. *Graph Separation Theorems and Sparse Gaussian Elimination*. PhD thesis, Stanford University, Department of Computer Science, 1980. STAN-CS-80-833.

[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.

[GL78] Alan George and Joseph W. Liu. An automatic nested dissection algorithm for irregular finite element problems. *SIAM Journal Numerical Analysis*, 15(5):1053–1069, 1978.

[GL81] Alan George and J. W. Liu. *Computer solution of large sparse positive definite systems*. Prentice-Hall, Englewood Cliffs, NJ, 1981.

[GM78] Alan George and David R. McIntyre. On the application of the minimum degree algorithm to finite element systems. *SIAM Journal Numerical Analysis*, 15(1):1053–1069, 1978.

[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.

[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*, 79 : 2554–2558, USA, 1982.

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

[iPS90] Int$_e$l Corporation, 3065 Bowers Avenue, Santa Clara, California, 95051,. *iPSC/2 and iPSC/860 User's Guide*, 1990.

[JH89] Lennart Johnsson and Ching-Tien Ho. Optimum broadcasting and personalized communication in hypercubes, *IEEE Trans. on Computers*, 38(9) :1248-1268, 1989.

[KG87] David E. Keyes and William D. Gropp. A comparison of domain decomposition techniques for elliptic partial differential equations and their parallel implementation. In *Selected Papers from the Second Conference on Parallel Processing for Scientific Computing* (C. W. Gear and R. G. Voigt, editors), pages s166–s202, Philadelphia, 1987. SIAM.

[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.

[LF90] M. Loriot and L Fezoui. Mesh-splitting preprocessor. unpublished notes, 1990.

[Liu89a] Josehp W. H. Liu. A graph partitioning algorithm by node separators. *ACM Transactions on Mathematical Software*, 15(3):198 – 219, 1989.

[Liu89b] Josehp W. H. Liu. The minimum degree ordering with constraints. *SIAM J. Sci. Stat. Comput.*, 10(6):1136 – 1145, 1989.

[LS76] Wai-Hung Liu and Andrew H. Sherman. Comparative analysis for the Cuthill-McKee and the reverse Cuthill-McKee ordering algorithms for sparse matrices. *SIAM Journal Numerical Analysis*, 13(2):199 – 213, 1976.

[Man92] Nashat Mansour. *Physical Optimization Algorithms for Mapping Data to Distributed-Memory Multiprocessors*. PhD thesis, Computer Science Department, Syracuse University, 1992.

[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.

[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.

[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.

[TZTS92] L. Tao, Y.C. Zhao, K. Thulasiraman and M.N.S. Swany. Simulated annealing and tabu search algorithms for multi-way graph partitioning. Unpublished manuscript, 1992.

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

[Wil90] R. D. Williams. Performance of dynamic load balancing algorithms for unstructured mesh calculations. Technical Report, Concurrent Supercomputing Consortium, Cal. Tech., 1990.