In the Proceedings of the Scalable Parallel Libraries Conference, pp 36–44 Mississippi State University, MS, 1993.

An Alternative to Data Mapping for Parallel PDE Solvers : Parallel Grid Generation

Nikos P. Chrisochoides
Northeast Parallel Architectures Center and
Computer Science Department
Syracuse University
111 College Place, Syracuse, NY, 13244-4100

Abstract

*In this paper we identified and outlined the disadvantages of the traditional data mapping methods for the numerical solution of PDEs on distributed memory MIMD machines and we proposed a new approach that eliminates some of the disadvantages. Specifically, we presented a data-mapping approach based on parallel structured grid generation. The new approach is based on composite block structures to contract the size of the data-mapping problem. It is ten times faster than the fastest traditional data-mapping method, for relatively small problems, and approximately O(***P***) times faster, for very large problems (i.e., millions of grid points) that are processed on coarse-grain distributed memory MIMD machines with* **P** *processors.*

1 Introduction

The mapping of sequentially generated regular and irregular grids into distributed memory MIMD machines 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. Thus, several algorithms have been proposed for finding good suboptimal mapping solutions. These algorithms can be classified into three classes. The first class, clustering algorithms, includes greedy schemes, divide-and-conquer algorithms, and block partitioning methods [2]; The second class, deterministic optimization algorithms, includes local search techniques based on profit functions [22]. Finally, the third class, physical optimization, includes mapping algorithms that employ techniques from natural sciences [17]. Most of the above methods have a major drawback : they process sequentially unnecessary data.

An attempt to improve the performance of physical optimization algorithms was presented by Nashat et. al in [26]. This attempt is based on reducing the size of the grid or the mesh by applying a computationally cheap clustering algorithm on the original grid, and then on farther partitioning the resulted graph (or super-grid) by giving it as an input to a physical optimization algorithm. This approach reduces the problem size and thus the preprocessing time but at the same time it reduces the searching space (i.e., the freedom of the algorithm) and many times it returns solutions that are fragmented. This results to data mappings that require extensive communication [3]. Another attempt for irregular grids has been made by Lohner [25]. This attempt was based on the use of a coarse background mesh which was proved unsuccessful for problems with complex geometries because in those problems one had to start with very fine meshes in order to resolve the geometric and topologic inconsistencies.

In this paper we propose a method for the data mapping of structured grids for general 2 and 3-dimensional domains. The method is based on the idea of composite block structures that is used on numerical grid generation. For a given domain Ω we decompose the domain, Ω, into a small number of contiguous hexahedron subregions, Ω_i, that can be mapped into rectangular computational blocks B_i which form an initial composite block structure $C_o(\Omega) = \{B_i\}_{i=1}^{N}$. This decomposition can be done either interactively [38] or using a Medial Axis Transformation method [36]. Then, we generate sequentially an algebraic grid that provides an explicit control of the physical grid shape and requires a minimal number of grid points. This grid is used for the generation of a finer composite block structure,

$C_f(\Omega) = \{B_i^f\}^{N_{i=1}^f}$. $C_f(\Omega)$ decomposes the domain Ω into blocks of uniform sizes. Finally, we apply the boundary-conforming curvilinear P×Q partitioning method [2] for each block $B_i \in C_o(\Omega)$. Since the computational space of each block B_i is rectangular P×Q needs no sorting and thus it is even faster.

This approach is ten times faster than the fastest and one of the most effective traditional data-mapping methods (i.e., P×Q) [3], for relatively small problems, and can be approximately O(**P**) times faster, for very large problems (i.e., millions of grid points), processed on coarse grain distributed memory MIMD machines with **P** processors. Also, it is suitable for developing fast block Euler solvers, and preconditioned solvers for elliptic boundary value problems [30].

2 Data Parallel PDE Solvers

The parallelism of PDE computations can be explored on three levels, namely the continuous PDE operator, the discrete PDE operator and thirdly the PDE solver's arithmetic. In the first level, locality and thus parallelism can be explored on the continuous PDE operator. The PDE $Lu = f \ \ in \ \ \Omega \ \ \& \ \ Bu = g \in \vartheta\Omega$ is reduced to a set of "smaller" PDEs : $Lu^i \mid_{\Omega_i} = f \mid_{\Omega_i} \ \ \& \ \ Bu^i \mid_{\vartheta\Omega_i} = g \mid_{\vartheta\Omega_i}, \ \ i = 1, ..., \mathbf{P}$, where auxiliary conditions have been artificially extended at the interfaces of the non-overlapping subdomains $\{\Omega_i\}_{i=1}^{\mathbf{P}}$ (see [19], [35], [34] and [23]). The **P** coarse grain tasks solve **P** loosely synchronous PDE problems and exchange some information to force continuity at the interfaces of the subdomains (see Figure 1).

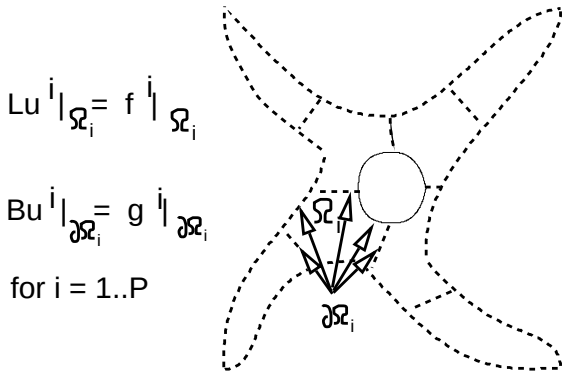

Figure 1: The components of the decomposed PDE problem based on the splitting of the domain Ω into a substructure of domains Ω_i.

At the second level, the locality can be explored on the discrete PDE operator (i.e., the system of algebraic linear equations generated by a finite difference or finite element module). The linear system $[L]\underline{x} = \underline{f} \ \ and \ \ [B]\underline{x} = \underline{g}$ is decomposed into subsystems : $[A_i]\underline{x}_{\Omega_i} = \underline{f}_{\Omega_i} \ \ \& \ \ [B_i]\underline{x}_{\vartheta\Omega_i} = \underline{g}_{\vartheta\Omega_i}, \ \ i = 1, ..., \mathbf{P}$ where x_{Ω_i} is the vector of the unknowns that correspond in the interior of the subdomain Ω_i and $x_{\vartheta\Omega_i}$ is the vector of the unknowns that correspond on the interface $\vartheta\Omega_i$ of the subdomain Ω_i (see Figure 2). In this case the computational model is loosely synchronous with medium size subcomputations that are based on a global distributed data structure (see Figure 3). Each processor independently solves part of the distributed linear system and synchronizes with other processors due to coupling of the linear equations. Parallel compilers in the near future should be able to find optimal uncouplings of the equations in order to reduce the communication and synchronization overheads.

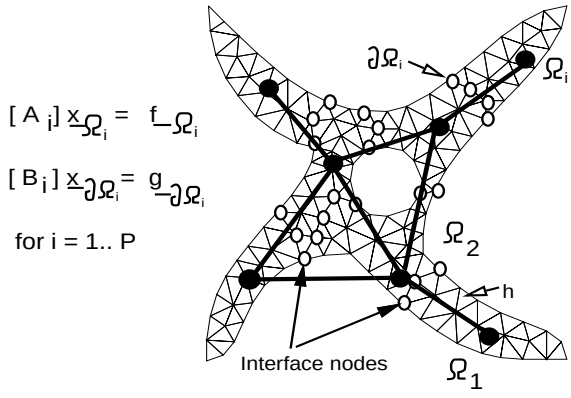

Figure 2: The components of the decomposed discrete PDE problem based on the splitting of the mesh D^h used numerically. This discrete mesh is partitioned by interfaces nodes (shown as circles) into discrete subgrids D_i^h

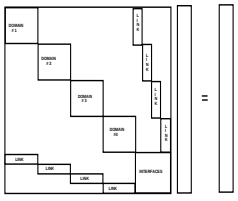

Figure 3: Distributed data structure (coefficient matrix of the linear system of equations) assembled in parallel on the processors of the parallel machine.

Finally, in the third level, locality can be explored in the arithmetic performed by a stream of instruc-

tions resulted from the compilation of a PDE solver. Parallel compilers based on Trace Scheduling [11] are capable of exploiting fine-grain parallelism in instructions even hundreds of blocks apart - basic block is defined as a set of instructions having no jumps into them except at the beginning and no jumbs out except at the end of the block. Such compilers can achieve speedups of about 90 [9].

In the rest of the paper we will focus on PDE solvers that explore locality only on the continuous and discrete PDE operators. The overall efficiency of such data parallel iterative PDE solvers depends upon the rate of convergence and the speedup per iteration of the method. The convergence rate depends on the condition number of the preconditioned system while the speed per iteration depends on the number of non-local unknowns to be exchanged among the processors and on the number of coupled subsystems. Specifically, for elliptic boundary value problems the condition number of a preconditioned system is of the order $O(1 + ln^2(\frac{d}{h}))$ where d is roughly the diameter of the subdomain and h is the grid size. Furthermore, the number of non-local unknowns depends on the number of interface grid points, I, and the coupling of the subsystems depends on the degree, C, of the dual graph of the subgrids. (see Figure 2). The search for optimum values for d, I, C and load balance of the computation is a difficult combinatorial optimization problem since many times the optimum values for d, I, C and load balance are conflicting with each other.

3 Data Mapping

The efficiency of data parallel PDE solvers on distributed memory systems depends highly upon the partition of the distributed data structures (i.e., grids, coefficient matrix, and unknown vector) and their placement onto the processors of the parallel machine. The data partition and placement (or *data mapping*) problem, so that the processors' workload is balanced and inter-communication and synchronization are minimum, is a difficult combinatorial optimization problem. The mathematical formulation of the data mapping problem is given by equation (1).

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

where D_i is the set of grid points (subgrid) that are placed (assigned) to the same processor, C_{D_i} is the set of the subgrids that are adjacent to the subgrid D_i, $m : \{D_i\}_{i=1}^{P} \rightarrow \{P_i\}_{i=1}^{\mathbf{P}}$ is an assignment function that places the subgrids 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 grid points in D_i, $C(m(D_i), m(D_j))$ is the communication required (per iteration) between the processors $m(D_i)$ *and* $m(D_j)$, and $\mathbf{P}$ is the number of available processors of the target parallel machine. In the case of data parallel PDE iterative solvers, without the overlapping between the computation and the communication phases, the synchronization term in equation (1) is included in $W(m(D_i))$.

This formulation of the data mapping problem assumes that computation and communication do not overlap. Clearly, such conditions are also necessary for the efficient execution of the data parallel PDE solvers on distributed memory MIMD machines. However, there is no explicit term for the synchronization in equation (1) because the synchronization cost is a nonlinear correlation of computational and communication work-load, computational and communication overlapping and network contention. Thus, it is difficult to be quantified. Nevertheless, equation (1) is considered a reasonable measure for the quality of data mapping solutions and two approaches can be identified in the mapping literature for its solution. The first approach is based on a smoother approximation of the equation (1) and the second approach is based on the splitting of the mapping problem into two simpler but still NP-Complete problems.

The evaluation of equation (1) is computational expensive. An approximation can be used instead :

$$\min_{m} \lambda^2 \sum_{i=1}^{\mathbf{P}} |D_i^h|^2 \; + \mu \sum_{i=1}^{\mathbf{P}} \sum_{D_j^h \in C_{D_i^h}} C(m(D_i^h), m(D_j^h)) \tag{2}$$

where μ is a scaling factor expressing the relative importance of the communication term with respect to the computation term, and λ depends on the solver and is equal to the number of computation operations per grid node per iteration. The first term is quadratic in the deviation of computation loads from the average computation load and is minimal when all deviations are zero. A minimum of the second term occurs when the sum of all interprocessor communication costs is minimized. The reevaluation of equation (2) due to replacement of a gird point g_i is determined by information about g_i, P_i and P_j only. The minimization of equation (2) allows a tradeoff between the computation workload and the communication cost for the purpose of minimizing their total sum.

Unfortunately, the cost of interprocessor communi-

cation, $C_{cst} = C(m(D_i^h), m(D_j^h))$, is difficult to quantify at compile time since it depends on phenomena that occur at run-time like node and link contention. Throughout the literature there are two expressions that have been proposed for the compile time approximation of the communication cost

$$C_{cst} = \begin{cases} \sigma + \rho I(D_i^h, D_j^h) \ + \tau H(m(D_i^h), m(D_j^h)) & (3a) \\ \rho' I(D_i^h, D_j^h) H(m(D_i^h), m(D_j^h)) & (3b) \end{cases}$$

where σ is the message start-up time (latency); ρ is the machine time for communicating one byte; τ is the communication time per link; $I(D_i^h, D_j^h)$ is the number of interface nodes of the subgrids D_i^h and D_j^h that determine 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 (3a) accounts for the cost of the connectivity of the subgrids.

The second mapping approach uses criteria that are qualitatively derived from the mapping requirements and addresses them in stages. It is based on splitting the optimization problem into two distinct phases that accomplish the *partitioning* and the *placement* of the grid [4] and [33]. In the *partitioning phase* we decompose the grid into P subgrids such that the following criteria are approximately satisfied:

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

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

(iii) the number of subgrids that are adjacent to a given subgrid is minimum,

(iv) each subgrid is a connected grid.

In the *placement phase* these subgrids are placed to the processors such that the following criterion is satisfied:

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

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

$$\max_{1 \le i,j \le \mathbf{P}} \ | \ |D_i^h| - |D_j^h| \ | \tag{4}$$

$$\max_{1 \le i \le \mathbf{P}} \{ \frac{(\sum_{D_j^h \in C_{D_i^h}} I(D_i^h, D_j^h))}{|D_i^h|} \} \tag{5}$$

$$\max_{1 \le i \le \mathbf{P}} |C_{D_i^h}| \tag{6}$$

The approximation of the data mapping problem by the above approaches is still an intractable problem. Thus, several algorithms have been proposed for finding good suboptimal mapping solutions. Some algorithms are based on greedy schemes, divide-and-conquer , or block partitioning methods. Examples are the nearest neighbor mapping, the P×Q partitioning, the recursive coordinate bisection, the recursive graph bisection, the recursive spectral bisection, the CM_Clustering, and the scattered decomposition techniques [1], [2], [8], [10], [24] [31], [32], [12], [15], [16], [29], [33], [39].

Other algorithms are based on deterministic optimization, where local search techniques are used to minimize cost functions related to execution time of the PDE solver; examples are the Kernighan-Lin algorithm and the geometry graph partitioning (see in [22] and [7]). Finally, another class of mapping algorithms are based on physical optimization that employs techniques from natural sciences [17]; examples are neural networks, simulated annealing, and genetic algorithms [13], [27], [40]. Figure 4 presents a possible classification of these methods.

The two approaches and the methods (traditional methods) for the solution of the data mapping problem we described in this section have a number of weak points. For example, the data distribution is based on the actual grid or mesh data that are required by the PDE solver to achieve certain accuracy. For real problems the number of grids can be of the order of $O(10^6)$. Table 1 presents some timings (in seconds) for the sequential preprocessing required for two relatively small problems. The mesh for the first problem, Problem A, consists of 57,756 elements, 29,223 nodes and generates 28,535 equations, while the mesh for the second problem, Problem B, consists of 18,890 elements and 9,880 nodes and generates 8,981 equations.

Even if we use parallel algorithms to speedup the algorithms for the data mapping problem we should load all the data structures required for the parallel processing of these algorithms. In this case the I/O required is very large relative to the time needed for the

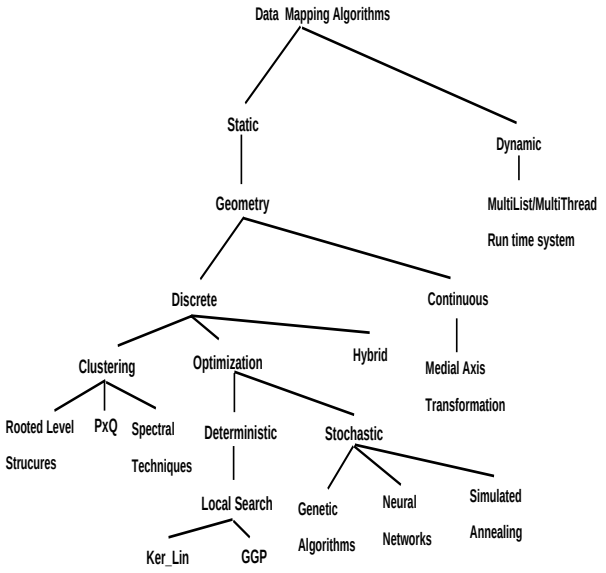

Figure 4: Data Mapping Algorithms

Table 1: The execution time of the different phases required for the sequential preprocessing for the numerical solution of Problems A and B using the P×Q partition algorithm. The time is in seconds on the SPARC workstation.

SPARC Workstation phases	A	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

Table 2: The execution time of the different phases required for the data loading, synchronization and parallel numerical solution of Problems A and B. The time is in seconds on the nCUBE 2.

nCUBE 2 phases	A	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

parallel solution of the PDE. Table 2 presents performance timings (in seconds) from nCUBE 2 for the four phases required to solve numerically a PDE problem using the traditional data mapping methods. From the data of the Table 2 we can see that the time to discretize a PDE operator and perform 100 iterations on nCUBE 2 using Jacobi semi-iterative together with Chebyshev polynomials for the acceleration of the convergence (see [6]) is 2.9 % and 3.4 % of the total execution time for problems A and B respectively.

Another major weakness of the traditional data mapping methods is the difficulty to quantify, at compile time, phenomena that occur at run time. Thus they are not capable to maintain efficiently the data distribution for applications that require the use of different PDE operators in different subregions (eg. Navier-Stokes in a boundary layer around and near an airfoil and Euler in the far field) that change at run time. We have already mentioned the weakness to quantify phenomena that occur at run time due to network contention. Equally important is the weakness to efficiently distribute the data for distributed systems with time sharing heterogeneous work stations and high speed networks. Finally, these methods assume that the data structures are static throughout the computation and thus they can not be used for PDE problems that require solution methods with mesh movement or some form of refinement (i.e., h-refinement, p-refinement or hp-refinement).

4 Parallel Grid Generation

In the previous section we presented data indicating that even the least expensive data mapping methods introduce a lot of overhead in the process of solving numerically PDEs on parallel machines. In this section we present a new efficient approach for the solution of the data mapping problem. This approach is based on parallel grid generation. Grids or meshes can be classified into two basic types, namely, the *structured grids*, formed by intersecting grid lines, and the *unstructured meshes*, formed by first creating node points and then connecting the nodes to form the "best" possible triangles. In the rest of the paper we describe a method for the parallel generation of structured grids that is used to reduce the pre-processing overhead due to data mapping. As a result the new approach is ten times faster than the fastest traditional data mapping method for small problems (i.e., tens of thousands of grid points) and O(**P**) times faster for larger problems (i.e., millions of points).

The parallel generation of structured grids for general 2 and 3-dimensional domains is based on composite block structures (see [37] for a comprehensive survey of the method for sequential machines). The basic idea of the composite block structure is based on the decomposition of the physical domain Ω into contiguous four-sided (or hexahedrons for 3-dimensional domains) subregions Ω_i (Figure 5a) which are mapped to rectangular computational blocks (Figure 5b), B_i. In each of the computational blocks an independent curvilinear coordinate system (ξ^1, ξ^2, ξ^3) is generated. The grid of the full domain is generated by composing "properly" the separate coordinate systems of the computational blocks (Figure 5c). This composition requires an interaction between adjacent rectangular blocks. Note that (i) the size (i.e., the number of grid points) of the computational blocks B_i may vary and (ii) the number of the subregions and thus of the computational blocks usually is smaller than the number of the available processors for large and massively scale parallel machines.

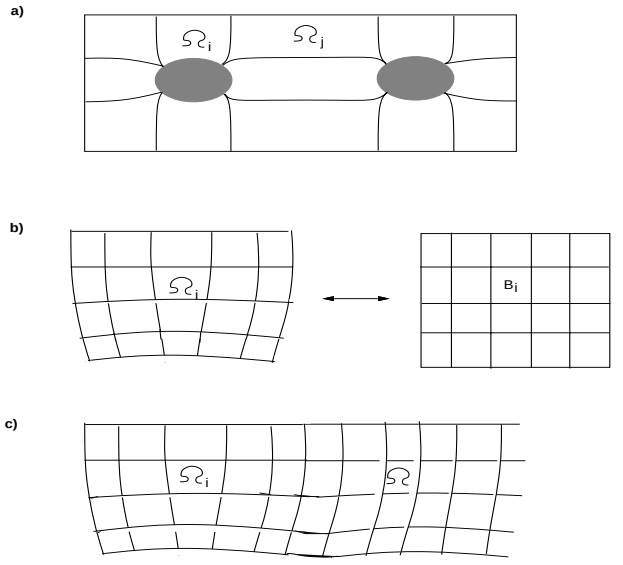

Figure 5: Three steps of grid generation process.

For each block B_i an independent curvilinear coordinate system can be generated in parallel. The degree of continuity of grid lines across the interfaces of adjacent curvilinear systems requires either the specification of grid points at the same fixed locations on both of the adjacent coordinate systems (case of discontinuous grid line slope) or the treatment of grid lines as a branch cut on which the generation system is solved just as it is in the interior of the blocks (case of continuous grid line slope). In this case the interface locations are determined by the grid generation system. The continuity of grid line slope is handled by providing an extra layer of points (outer-layer) surrounding each block. The interface and outer-layer grid points of a block are forced to coincide with the interface and interior grid points of an other adjacent block. This coincidence of the points is maintained during the course of an iterative solution of an elliptic system over all blocks. This suggest that a local synchronization among the processors that process adjacent blocks is required at the end of each iteration.

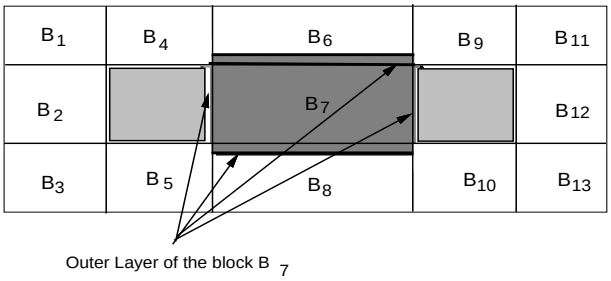

Figure 6: Computational domain.

The initial grid for the iterative elliptic system is generated using methods based on transfinite interpolation. The grids that are generated using interpolation are referred as *algebraic* grids and the grids that require the solution of elliptic partial differential equations (PDEs) are referred as *elliptic* grids. The efficient parallel generation of elliptic grids requires that the processor work load is balanced and communication and synchronization is minimum. The computational domains of elliptic grids are rectilinear (see Figure 6) and thus the mapping of the computations associated to parallel grid generation is not as difficult as the mapping for general PDE problems.

In order to apply the fastests and one of the most effective data mapping algorithms [2], [3] on the block structures $C_o(\Omega)$ we first generate sequentially an algebraic grid that provides an explicit control of the physical grid shape and requires a minimal number of grid points. This algebraic grid is the basis for the $C_f(\Omega) = \{B_i^f\}_{i=1}^{N^f}$ which satisfies the following three properties :

1. $\mid C_o(\Omega) \mid < \mid C_f(\Omega) \mid$
2. $\forall B_i \in C_o(\Omega)\ \exists I_i \subset \aleph\ \ni: B_i = \cup_{j \in I_i} B_j^f,\ B_j^f \in C_f(\Omega)$
3. $|B_i^f| = |B_j^f|\ \ \forall B_i^f, B_j^f \in C_f(\Omega)$.

Then we apply boundary-conforming curvilinear P×Q partitioning method [2] on each block (see Figure 7)

$B_i \in C_o(\Omega)$ and place all blocks B_i^f with the same color to the same processor. This approach is similar to the scattered decomposition for irregular domains presented in [13]. The difference is that the templates defined by the $C_o(\Omega)$ are not fixed, the subdomains reflect the boundary shape and thus there are no disconnected subdomains. Another advantage of the method is that eliminates link contention in the network [5] and thus minimizes communication time.

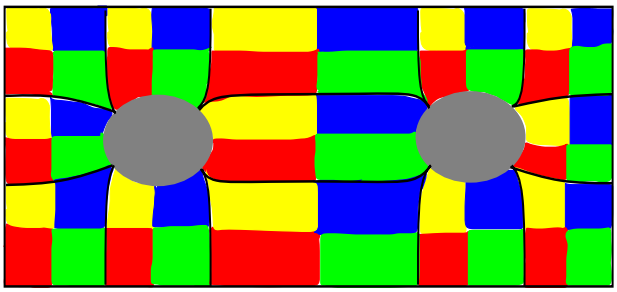

Figure 7: Block-by-Block P×Q partitioning of the $C_o(\Omega)$.

By applying the Block-by-Block P×Q method on $C_o(\Omega)$ we use a natural way to contract the size of the problem and thus reduce the pre-processing time. Table 3 depicts the time required to generate (Grid-Gen.) structured grids on a SPARC workstation, the time to map (P×Q) and load the subgrids onto the memory of the 64 processors of the nCUBE 2 and the sum (Total) of the grid generation, mapping and loading times. Table 4, for the same grids, depicts the time to generate $C_f(\Omega)$, map and load the data on the memory of the 64 processors of the nCUBE 2. For the mapping the Block-by-Block P×Q method was used on the same SPARC workstation. This table also depicts the time to generate on a 64 node nCUBE 2 an algebraic grid. As it is indicated from the third columns of the Tables 3 and 4 the new approach is ten times faster than the fastest traditional data-mapping method. For applications with millions of grid points the method can be approximately O(**P**) times faster. Also, it is easy to see that this approach returns a data mapping with the same number of interface grid points and degree of connectivity of the subgrids as the traditional P×Q data mapping method.

5 Summary and Conclusions

A number of weak points for the traditional data mapping methods have been identified. We addressed the pre-processing overhead due to processing of large amounts of unnecessary data. A new approach that reduces the size of the data to be processed to a minimum has been proposed. The new approach is based on parallel grid generation and for small problems is ten times faster than the fastest traditional method (i.e., P×Q see in [3]), and for large problems is O(**P**) times faster without compromising the quality of the solution.

Table 3: Preprocessing time (in sec) required for generating and mapping sequentially an algebraic grid on the 64 nodes of the nCUBE 2.

Grid Points	Grid-Gen.	Pre-Proc.	Total
2.5×10^3	0.38	17.81	18.19
10×10^3	1.61	46.45	48.06
22.5×10^3	3.61	100.15	103.76
40×10^3	6.45	195.13	201.58

Table 4: Preprocessing time (in sec) required for preprocessing and parallel generation of an algebraic grid on the 64 nodes of the nCUBE 2.

Grid Points	Pre-Proc.	(//) Grid-Gen.	Total
2.5×10^3	3.44	0.08	3.52
10×10^3	4.38	0.35	4.73
22.5×10^3	8.48	0.71	9.19
40×10^3	16.06	1.25	17.31

Acknowledgements

The author gratefully acknowledges the Alex G. Nason Foundation for the Nason Prize Award that supports him at NPAC.

References

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

[2] N. P. Chrisochoides, Elias Houstis and John Rice. *Mapping Algorithms and Software Environment for Data Parallel PDE Iterative Solvers* To appear in the special issue of the Journal of Parallel and Distributed Computing on Data-Parallel Algorithms and Programming.

[3] N. P. Chrisochoides, Nashat Mansour and Geoffrey Fox. *Performance evaluation of data mapping algorithms for parallel single-phase iterative PDE solvers* Submitted in the special issue of the Journal of Concurrency Practice and Experience on Load Balancing and Graph partitioning for parallel machines.

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

[5] 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.

[6] 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.

[7] 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.

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

[9] DeCegama, L. A. Parallel Processing Architectures and VLSI Hardware, Volume 1. Englewood Cliffs, NJ: Prentice Hall, 1989.

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

[11] Ellis, R. J. BULLDOG : A compiler for VLIW Architectures. Cambridge, MA: The MIT Press, 1986.

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

[13] Flower, J., 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.

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

[15] 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.

[16] 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.

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

[18] 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.

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

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

[21] 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.

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

[23] 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.

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

[25] Lohner, R., J. Camberos and M. Merriam Parallel Unstructured Grid Generation Unstructured Scientific Computation on Scalable Multiprocessors, (eds. P. Mehrotra, J. Saltz and R. Voight), The MIT Press, 31–64, 1992

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

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

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

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

[30] Pasciak, J. Domain Decomposition preconditioners for Elliptic in Two and Three Dimensions : First Approach *First International Symposium on Domain Decomposition Methods for Partial Differential Equations.* SIAM, Philadelphia, 62–72, 1988.

[31] 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.

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

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

[34] 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.

[35] 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.

[36] Tan, T.K. H., M. A. Price, C. G. Amstrong and R. M. McKeag Computing the critical points on the medial acis of a planar object using a Delaunay point triangulation algorithm Submitted to IEEE PAMI.

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

[38] Thompson, J., The National Grid Project NSF Engineering Research Center for Computational Field Simulation, 1991.

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

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