UCLA
COMPUTATIONAL AND APPLIED MATHEMATICS

On the Influence of the Partitioning Schemes on the Efficiency of Overlapping Domain Decomposition Methods

P. Ciarlet, Jr.
F. Lamour
B.F. Smith

July 1994
CAM Report 94-23

Department of Mathematics
University of California, Los Angeles
Los Angeles, CA. 90024-1555

On the influence of the partitioning schemes on the efficiency of overlapping domain decomposition methods

P. Ciarlet, Jr [*] F. Lamour [†] B.F. Smith [‡]

July 18, 1994

Abstract

One level overlapping Schwarz domain decomposition preconditioners can be viewed as a generalization of block Jacobi preconditioning. The effect of the number of blocks and the amount of overlapping between blocks on the convergence rate is well understood. This paper considers the related issue of the effect of the scheme used to partition the matrix into blocks on the convergence rate of the preconditioned iterative method. Numerical results for Laplace and linear elasticity problems in two and three dimensions are presented. The tentative conclusion is that using overlap tends to decrease the differences between the rates of convergence for different partitioning schemes.

1 Introduction

One level overlapping Schwarz domain decomposition methods are very simple, natural preconditioners for the parallel solution of the linear systems that arise from the discretization of elliptic PDEs. In order to apply these methods efficiently in parallel one must partition the unknowns among the processors to achieve both load balancing and reduce required interprocessor communication. However, for block Jacobi methods the particular partition used can have a large effect on the **numerical** convergence rate.

Farhat and Lesoinne [9] and Farhat and Simon [10] addressed the problem of load balancing and communications between processors on a parallel architecture, and have run some experiments on parallel machines in order to compare the execution times. In this paper we study the effects of several partitioning strategies (with and without overlap) on the load balancing and communications but we focus on the convergence rate of the method or, equivalently, on the number of iterations required to reach a prescribed tolerance.

We consider the numerical solution of elliptic PDEs in both two and three dimensions on unstructured grids using the finite element method. The resulting linear system is solved using a domain decomposition preconditioned Krylov space method designed for parallel computing. All of our numerical experiments are performed on sequential machines because we are mainly interested in the effect on the numerical convergence rate. Future studies will focus on the non-numerical effects introduced by parallel machines.

[*]Department of Mathematics, Univ. of Calif. at Los Angeles, CA 90024 and Commissariat à l'Energie Atomique, CEL-V, D.MA, MCN, 94195 Villeneuve-St-Georges Cedex, France. E-mail: ciarlet@math.ucla.edu, ciarlet@limeil.cea.fr. The work of this author was partially supported by the DGA/DRET under contract 93-1192.

[†]Department of Mathematics, Univ. of Calif. at Los Angeles, CA 90024. E-mail: lamour@math.ucla.edu. The work of this author was partially supported by the National Science Foundation under contract ASC 92-01266, the Army Research Office under contract DAAL03-91-G-0150, and ONR under contract ONR-N00014-92-J-1890.

[‡]Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439-4844. E-mail: bsmith@mcs.anl.gov.

The paper is organized as follows. In Section 2, we present the elliptic problems to be solved, that is the equations and the boundary conditions. We also include a brief overview of the discretization, the iterative and the domain decomposition methods used in the paper. In the next Section, we give a detailed description of the partitioning heuristics, the influence of which we compare in Section 4.

2 Solving the elliptic problems

2.1 The problems and their discretization

Let Ω denote the domain imbedded in R^2 or R^3 and $\Gamma = \Gamma_0 \bigcup \Gamma_1$ its boundary. The elliptic problems to be solved are either the Poisson problem with Dirichlet boundary conditions on Γ_0 and homogeneous Neumann boundary conditions on Γ_1 or the equations of linear elasticity in the domain Ω with Dirichlet boundary conditions on Γ. Let u denote the (scalar) solution of the Poisson problem and $\underline{u}$ the (vector) solution of the elasticity equations. Finally let E, Young's modulus, and ν, Poisson's ratio, be two positive constants. Then the problems can be written as:

$$-\Delta u = f \text{ in } \Omega, \; u = g \text{ on } \Gamma_0, \; \frac{\partial u}{\partial n} = 0 \text{ on } \Gamma_1,$$

$$-\frac{E}{1+\nu}\{\Delta \underline{u} + \frac{1-\nu}{1-2\nu}\nabla \; \nabla \cdot \underline{u}\} = \underline{f} \text{ in } \Omega, \; \underline{u} = \underline{g} \text{ on } \Gamma.$$

In order to derive a discretization of these problems, the equations are replaced by a suitable variational formulation and then the finite element method [2] with the P1 finite element is used on the triangulations. Even though the value of the solution is known on Γ_0 via the Dirichlet boundary condition, we have chosen to keep these (trivial) equations in the linear system. Because of this, the unknowns are the approximations of the value of the true solution at all vertices of the triangulation, and the matrix is nonsymmetric.

2.2 Iterative and DD methods

The resulting nonsymmetric linear system is solved using the preconditioned Bi-CGSTAB method, due to van der Vorst [21]. The preconditioner considered in this paper is designed with the help of domain decomposition principles.

In recent years, many papers have been devoted to the study of new domain decomposition methods. One of the main reasons for this renewed interest is the emergence of parallel and massively parallel computers. This is because many domain decomposition methods are naturally parallel. The additive Schwarz method we consider in this paper is very representative. This method was introduced by Dryja in [7] and Dryja and Widlund in [8]. For some results obtained on parallel architectures with this method, we refer the reader to [15].

Here, we consider the case of the one-level additive Schwarz, that is local problems are solved without a coarse problem. It is well known that the addition of a coarse solver greatly reduces the condition number and therefore the number of iterations, but a coarse solver is defined from a coarse triangulation of the subdomain which maybe costly to compute. In order to get a better condition number, it is possible, on the other hand, to consider overlapping subdomains. Here, the overlapping subdomains are simply obtained by extending the nonoverlapping partitions, at little additional cost. In the following, we thus allow subdomains to overlap.

Finally, let us briefly mention how the local problems are solved: the (complete) LU-factorization of each local problem is computed and then used as a solver at every iteration of the method. To minimize the fill (number of non-zero entries) during the factorization, the matrix columns and rows are reordered with nested dissection following George's Sparsepack [13].

3 The partitioning heuristics

In order to partition the domain, the triangulation can be considered as a graph by identifying the notions of vertices and edges. We call this graph a *finite element graph*. Let $G = (V, E)$ be a graph where V is the set of nodes and E is the set of edges. Let p be an integer, then we define the p edge-partition of G as a partition of V into p disjoint sets $V_1, V_2, ...V_p$ such that:

- $\bigcup_{i=1}^{p} V_i = V$,
- $| V_i | \approx | V_j |$ for $1 \leq i, j \leq p$ and $i \neq j$,
- $m = | \{(v_i, v_j) \in E$ and $v_i \in V_i,\ v_j \in V_j$ with $i \neq j\} |$ is as small as possible.

Besides the requirements of the p edge-partitioning problem it may be necessary to impose that each resulting subset has to be connected.

For all the formulations and variants of graph partitioning, the general problem is a NP-hard problem. Many heuristics have been developed while few optimal results have been proved.

The heuristic methods proposed in the literature for finite element graphs can be roughly arranged in two categories: the greedy methods and the recursive methods. A detailed study of the heuristics presented here as well as a fair comparison of the corresponding partitions are given in [6].

Let first introduce some definitions and notations about graphs that will be used in the following.

Let $N = | V |$ and $M = | E |$ be the number of nodes and edges respectively. The *degree* of a node v, denoted $d(v)$, is the number of edges that have one endpoint in v, i.e. the number of neighbors of v. Moreover we use d to denote the average degree of a node in G.

The Laplacian matrix of a graph G, denoted $L(G) = (l_{ij})_{i,j=1...N}$, is defined by:

$$l_{ij} = \begin{cases} -1 & \text{if } (v_i, v_j) \in E \\ d(v_i) & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

As we study finite element graphs we shall talk about the *boundary* of the graph which is clearly defined with respect to the meshes.

3.1 Recursive methods

3.1.1 Spectral bisection

Spectral partitioning methods are based on a particular eigenvalue, and on its associated eigenvector, of the Laplacian matrix of the graph to be partitioned.

In order to understand the intuitive justifications of these methods we must summarize some known properties of the eigenvalues of the Laplacian matrix of a graph and display their links with the graph bisection, or 2 edge-partitioning, problem.

Let $L(G)$ be the Laplacian matrix of G and $N \geq 2$. Let $\lambda_1 \leq \lambda_2 \leq \lambda_3 \leq ... \leq \lambda_N$ be the eigenvalues of $L(G)$.

To begin with, λ_1 is always equal to zero. The second smallest eigenvalue, λ_2, is also known as the algebraic connectivity of G. Let us recall that $\lambda_2 = 0$ if and only if G is a disconnected graph. In other words, the number of eigenvalues equal to zero gives exactly the number of connected components of the graph. Let us assume, without loss of generality, that λ_2 is the first non zero eigenvalue.

The 2 edge-partitioning problem can be reformulated in the following terms. Let us consider a vector $q \in Q = \{(q_i) \in R^N,\ q_i = \pm 1,\ \sum q_i = 0\}$ and the induced balanced partition for which a node i is assigned to subset V_1 if $q_i = +1$ or to subset V_2 if $q_i = -1$.

The number of intersubset edges (or edge cuts) is equal to

$$m = \frac{1}{4}(q, L(G)q).$$

So the 2 edge-partitioning problem amounts to finding a vector $q \in Q$ which minimizes this quantity.

By relaxing this discrete minimization problem to a continuous problem, that is to the search of a vector x such that $\| x \|_2^2 = N$ and $\sum x_i = 0$, one finds that the minimum of $(x, L(G)x)$ is obtained for $x = x_2$, where x_2 is the eigenvector associated with λ_2, named the *Fiedler* vector, [12].

Knowing this, the idea is to compute a vector $q \in Q$ by using x_2 in the following way. Let x_l be the median value of the components of x_2. Then q_i is defined as:

$$q_i = \begin{cases} +1 & \text{if } (x_2)_i > x_l \\ -1 & \text{if } (x_2)_i < x_l \\ \pm 1 & \text{if } (x_2)_i = x_l \text{ (for the balancing)} \end{cases}$$

The partition induced by such a vector q is known as the *median cut partition.* Of course this is not an optimal partition. Although we cannot say how close to the optimal the median cut is, there are still some interesting properties about it.

First, there is an important result about the connectivity of a median cut partition which has been stated by Fiedler [12]. Briefly, if there are exactly $\frac{N}{2}$ strictly positive components and $\frac{N}{2}$ strictly negative components then both balanced subsets of the partition are connected. Regardless, the connectivity of one of the subsets is always guaranteed.

Secondly, it have been proved by Ciarlet, Chan and Szeto in [3] that for all $p \in Q$, $\| x_2 - p \| \geq \| x_2 - q \|$. This result, which does not allow us to conclude about the optimality of q, can nevertheless reassures the promoters of the median cut method by assuming that it is a close enough choice.

3.1.2 Recursive spectral bisection

In order to partition a given graph into any number of subsets that is a power of two, the median cut partitioning method has been used as a step in a divide and conquer process, as in the *Recursive Spectral Bisection* algorithm (*RSB*) due to Simon [20]. There the median cut algorithm is recursively applied to each subgraph induced by the bisection previously computed until the required number of subsets is obtained.

Moreover, as the computation of the Fiedler vector is time consuming, many studies have been devoted to speeding up the calculation of this particular vector. Barnard and Simon, [1], have accelerated the RSB algorithm by approximating the Fiedler vector. For this, they contract some edges in the graph in order to obtain a smaller graph and repeat this operation a certain number of times until the contracted graph is small enough. Then, they compute the Fiedler vector of the smallest graph. From the smallest graph, an approximation of the Fiedler vector of the previous graph is deduced by an interpolation technique. This approximated vector is then used as a starting point in an iterative method that computes the Fiedler vector. Once the refined Fiedler vector is obtained the process goes back to the larger graph and recomputes a Fiedler vector for this graph using the same strategy (interpolation and refinement), and so on. The algorithm is given below and it will be referred to as "RP 1".

Algorithm RP 1

1. Compute the Fiedler vector for the graph by:
   (a) Constructing a series of smaller graphs $(G^l)_{l=1,\cdots,L}$ obtained by some contraction operations applied to the original graph, (contraction step).
   (b) Computing the Fielder vector for the smallest graph G^L.
   (c) Constructing a series of Fiedler vectors corresponding to the series of graphs by:
      i. interpolating the previously found Fiedler vector to the next larger graph in a way that provides a good approximation to next Fielder vector (interpolation step),
      ii. computing from the given approximated Fielder vector, a more accurate vector (refinement step).
2. Sort vertices according to size of entries in Fielder vector.

3. Assign half of the vertices to each subdomain.
4. Repeat recursively (divide and conquer).

The complexity of RP 1 is estimated to $O(M \log_2 p)$.

3.1.3 Recursive multilevel algorithm

In a similar manner, Hendrickson and Leland in [18] used a multilevel technique to partition a graph. They first reduce the size of the graph, and derive a series of smaller graphs by contracting the edges of the original graph until the size of the last graph is small enough. They partition the smallest graph using the median cut technique. Then, they reflect the partition when uncontracting the series of graphs. Moreover, to improve the quality of the partition (in term of balancing and of number of intersubset edges) they perform a local optimization method which exchanges nodes between the subsets of the partition. Finally they repeat the previous sequence of steps on each subset until the total number of subsets is obtained. The algorithm below is called "RP 2".

Algorithm RP 2

1. Construct a series of smaller graphs obtained by contraction operations applied to the original graph.
2. Partition the smallest graph using the median cut method.
3. Propagate the partition by:
   (a) Uncontracting the smallest graph.
   (b) Reflecting back the partition to the uncontracted graph.
   (c) Refining locally the partition using a local optimization method.
   (d) Repeating steps (a), (b), (c), until the original graph.
4. Repeat recursively (divide and conquer).

The complexity of RP 2 is estimated to $O(M \log_2 p)$.

3.2 Greedy algorithms

3.2.1 Principles

According to the p edge-partitioning problem a greedy algorithm can be described as an algorithm that computes each subset V_i by simply accumulating nodes when traveling through the graph. The problematical questions are only: how to start and how to stop?

The way of accumulating nodes in each subset is obvious from the graph structure of the problem. A starting node v_s is chosen and marked. The accretion process is done by selecting and marking the unmarked neighbors of v_s, then the unmarked neighbors of the neighbors of v_s and so on as long as the expected total number of nodes is not reached. This can be viewed as successively building *fronts.*

The way of choosing a starting node v_s will clearly affect the shape of the final partition. It will also influence the communication scheme, i.e. the number of existing edges between different subsets of the partition.

In the same way, the manner that one chooses the prescribed number of nodes among all the candidate nodes of the last front contributes to the quality of the final partition.

Thus a greedy heuristic for solving the p edge-partitioning problem can be defined roughly by iterating the following 3 steps:

1. Choose a "good" starting node v_s,
2. Build fronts,
3. Stop according to some tie-break strategy in case of multiple choices and mark all the chosen nodes.

At present, there are no theoretical results on the "goodness" of one starting node. Neither are there results on how good a tie-break strategy is. For those two points only intuitive guesses help to design p partitioning problem heuristics. However, an obvious justification of using greedy heuristics for solving the p partitioning problem is that they are inexpensive. We have shown in [5], that for the general case the overall complexity of such algorithm is $O(N \max(p, d, \log_2(\frac{N}{p})))$.

3.2.2 Front oriented algorithm

The next algorithm, presented in detail in [5], implements the principles of a greedy method as well as some other original features. First, this algorithm builds connected subsets. On the other hand, it does not always provide well balanced subsets. The subsets are constructed in a concentric way around the boundary of the graph. Finally, for each subset, in case of multiple choices between the nodes of the last front, the tie-break strategy chooses those which are linked to as few unmarked nodes as possible.

More precisely, in the partitioning process, the starting node of each iteration i is chosen in order to belong simultaneously to the *boundary* of G, to be an unmarked neighbor of a node of V_{i-1} and to have a minimal positive *current degree*. Here, the current degree of a node is the number of nodes connected to it which have not yet been selected, i.e. marked, during the accumulation step.

As for the tie-break strategy, it is achieved by keeping the nodes that have a minimal current degree. In fact, the algorithm does not simply build the subsets as mentioned but also check the connectivity of each of them. Whenever a subset is found to be multiconnected, the algorithm corrects the feature by reassigning small components to other subsets and by keeping the largest component. The algorithm "GP" is described next.

Algorithm GP

```
1. If $i < p$
   (a) Compute $n_i = \frac{N - \sum_{j=1}^{i-1} n_j}{p-(i-1)}$.
   (b) Choose an unmarked node $v_s$ such that:
       i. $v_s$ belongs to the current boundary,
       ii. if the current boundary is not new, $v_s$ is a neighbor of a node belonging to $V_{i-1}$ (if possible[1]),
       iii. $v_s$ has a minimal current degree.
       Mark $v_s$ and initialize $V_i$ with $v_s$.
   (c) If there are unmarked neighbors of nodes of $V_i$, let $k$ be their number.
       i. If $|V_i| + k < n_i$ then mark those nodes, add them to $V_i$ and update the current degree of their
          neighbors, and then return to 1.(c).
       ii. Mark $(n_i - |V_i|)$ minimal current degree nodes and add them to $V_i$.
           Update the current and virtual boundaries.
           Do $i = i + 1$ and return to 1.
   (d) If there are no more unmarked neighbors of nodes of $V_i$ and if $|V_i| < n_i$ then unmark the nodes in
       $V_i$ and assign them to neighboring subsets. Return to 1.
2. Mark all the remaining nodes and add them into $V_p$. If $V_p$ is multiconnected then keep the largest
   component and unmark the nodes of the other components and assign these nodes to neighboring subsets.
```

The complexity of GP is $O(M)$.

[1]It may not be possible to find a node neighboring the previously built subset if the boundary is multiconnected.

3.3 Local optimization methods

Whatever partitioning methods may be used, many experiments have shown that local optimization methods can improve greatly the load balancing or the intersubset structure. One of the most well known optimization methods for improving a given partition is due to Kernighan and Lin [19].

Fundamentally the method is based on a given 2 edge-partition (V_1, V_2) of the node set V of a graph and tries to improve it by exchanging a subset of V_1 with one of V_2. The selection criterion of the subsets is determined from a gain function which is defined as follows. First, for $v \in V_1$, $g_v = d_{V_2}(v) - d_{V_1}(v)$ ($d_A(v)$ is the number of neighbors of v which belong to A) and for $w \in V_2$, $g_w = d_{V_1}(w) - d_{V_2}(w)$. Then the gain obtained by exchanging a node $v \in V_1$ with a node $w \in V_2$ is equal to:

$$g_{v,w} = g_v + g_w - 2\delta(v,w)$$

where $\delta(v,w)$ is defined by:

$$\delta(v,w) = \begin{cases} 1 & \text{if } (v,w) \in E \\ 0 & \text{otherwise} \end{cases} .$$

The authors of RP 1 ([1]), have chosen to use the KL algorithm as a postprocessing step after each bisection has been performed. Nevertheless, as the complexity of the KL method is in the order of $O(N^2 \log N)$, the KL method is actually invoked only when the number of nodes in each subset is less than 300.

Because of the complexity of the KL algorithm, Fiduccia and Mattheyses [11] slightly modified the algorithm, and reduced the overall complexity to $O(M)$. Principally, the FM algorithm chooses to move one node after another instead of directly exchanging a pair of nodes. The authors of RP 2 generalized the FM method to an arbitrary number of subsets, [17] and [18]. In this case, the overall complexity is estimated to $O((p-1)M)$. This method is then invoked in step 3.(c) for refining the partition.

Finally, the authors of GP have designed, what they call a retrofitting method which has been suggested by the experiments conducted on their partitioning heuristic [4]. The first step tries to reshape the outlines of the subsets by deleting some excrescences. Generally these occur during the accumulation step of the partitioning process when some of the chosen nodes encounter prematurely another subset. Then some nodes are attached to the subset to which they belong by a single edge. By reassigning those nodes to neighboring subsets and by iterating the process until no more excrescences remain, the shape of the subsets is improved. Note that these boundary nodes are transferred to the subset which holds the highest number of its neighbors. It is not always possible to eliminate all the excrescences, so the reshaping step has to stop when the overall shape of the partition, i.e. the number of excrescences, does not seem to be improved over the iterations (more precisely five iterations). Here, one iteration consists of spanning the whole set of nodes V and reassigning the excrescences.

Because of the reassignment of nodes to neighboring subsets in order to preserve the connectivity of the subsets, the resulting partition provided by GP is not balanced in most of the cases. Thus the second step of the retrofitting method looks for rebalancing the subsets by moving nodes from large subsets to small ones. One iteration of this consists of three parts. First, the largest and smallest subsets are determined. Then a set of nodes (or front) of the largest subset is reassigned to its smallest neighbor. Last, a set of nodes is reassigned to the smallest subset, coming from its largest neighbor. Moreover this process includes a connectivity test step which allows a move of a front only if the subsets of the new partition remain connected. The balancing process is iterated until the standard deviation approaches the optimal standard deviation or until it does not decrease anymore.

The complexity of one iteration of the reshaping step is $O(M)$ while the complexity of the balancing step is in the order of $O(nd)$, where n is the average number of node by subset. Note that when the retrofitting method is performed, neither the number of iterations for the reshaping steps nor for the balancing step are bounded *a priori*.

4 Numerical experiments

For our numerical experiments, we consider three domains. One is imbedded in R^2 and is the exterior of the section of a wing, made of four parts. The triangulation of this domain is due to T. Barth (from NASA

Ames). The other two are imbedded in R^3: they are a domain bounded by two concentric spheres and an axle (part of an automobile). The first triangulation comes from L. Crouzet (from CEA) and the second one from M. Vidrascu (from INRIA). The three domains have been scaled in order to be in $[0,1]^2$ or $[0,1]^3$. The wing grid has 33677 nodes and 99520 edges, the spheres grid has 9020 nodes and 59418 edges and finally the axle grid has 25058 nodes and 156842 edges.

The Poisson problem is solved on the wing and the spheres, f is set to $-9x^2 \sin 3y + 2 \sin 3y$ in 2D and 0 in 3D. $\Gamma_0 = \Gamma \bigcap \{(x,y), x < 0.2\}$ and $g = x^2 \sin 3y$ in 2D, $\Gamma_0 = \Gamma$ and $g = x$ in 3D. For the linear elasticity equations, solved on the axle, the constants are set to $E = 1$ and $\nu = 0.3$. $\underline{f}$ is set to $\frac{30E(1-\nu)}{(1+\nu)(1-2\nu)} \begin{pmatrix} x^4 \\ y^4 \\ z^4 \end{pmatrix}$. Finally, $\underline{g} = \begin{pmatrix} x^6 \\ y^6 \\ z^6 \end{pmatrix}$.

The numerical experiments including both the partitioning heuristics and the iterative solver are tested through a common tool: the *Portable Extensible Tools for Scientific Computation* (PETSc) developed by Gropp and Smith [14], [16]. PETSc is a software library for parallel and serial scientific computations. It provides a variety of packages and in particular the iterative solver. In addition to these existing packages, PETSc makes it easy to include any software written either in C or Fortran. The softwares corresponding to the partitioning heuristics have been given by their authors and have been used in PETSc as is. We have written interfaces for transferring inputs and parameters as well as for interpreting the results. The RP 1 code, written by Barnard and Simon, can be obtained by a request to their authors[2]. The RP 2 code is copyrighted but can be obtained via a request to Hendrickson or Leland[3]. The GP code has been written by Ciarlet and Lamour.

The experiments were carried out on a Sun Viking SuperSparc.

4.1 The partitioning characteristics

We summarize the characteristics of the three partitioning heuristics according to two parameters. The first one measures the balancing of the subsets. It gives a realistic idea of the difference of the size between large subsets and small ones according to the average size of the subsets: σ/n (%). The second one measures the percentage of intersubset edges (m/M).

We list below the notations that describe the different parameters attached to a given partition:

p is the number of subsets,

n is the average number of nodes by subset,

σ is the standard deviation of the number of nodes by subset and is equal to $\sqrt{\frac{1}{p}\sum_{i=1}^{p}(n_i - n)^2}$,

σ/n (%) is the average standard deviation of the number of nodes by subset,

m is the number of intersubset edges,

m/M (%) is the percentage of intersubset edges.

Although σ/n measures the size of the subsets it is not representative of the actual load balancing. We can see from Tables 1-3 that σ/n is very small in all cases (and optimal by definition for the recursive heuristics), with the exception of the first two values for GP in Table 3. But this is not enough to ensure the load balancing if one looks at Tables 7-9. Further comments are made in the next Section.

[2]simon@nas.nasa.gov or barnard@nas.nasa.gov.
[3]bahendr@cs.sandia.gov or rwlelan@cs.sandia.gov.

p	16	32	64	128	256
RP 1	0.02	0.05	0.08	0.11	0.38
RP 2	0.02	0.05	0.08	0.11	0.38
GP	1.04	1.45	0.56	2.29	2.54

Table 1: Wing: σ/n (%).

p	16	32	64	128	256
RP 1	0.08	0.12	0.17	0.71	1.20
RP 2	0.08	0.12	0.17	0.71	1.20
GP	0.08	0.29	0.39	0.87	1.64

Table 2: Spheres: σ/n (%).

p	16	32	64	128	256
RP 1	0.02	0.03	0.13	0.22	0.33
RP 2	0.02	0.03	0.13	0.22	0.33
GP	25.39	14.24	7.36	3.54	3.59

Table 3: Axle: σ/n (%).

The ratio m/M measures the communication volume between the subdomains, where M is given while m is the parameter to be minimized by the heuristics. In general we can see that this volume is smallest for RP 2, followed closely by RP 1. Nevertheless, GP gives reasonably close values, as can be seen in Tables 4-6. The impact of differences between communication volumes on iteration time is not measurable in our experiments as we have run them on a sequential machine.

p	16	32	64	128	256
RP 1	2.01	3.17	4.61	6.89	10.17
RP 2	1.44	2.47	4.11	6.47	9.74
GP	2.69	3.74	5.13	7.73	11.24

Table 4: Wing: m/M (%).

p	16	32	64	128	256
RP 1	15.83	20.59	26.53	34.09	43.08
RP 2	14.44	19.45	25.73	33.26	42.39
GP	18.29	22.56	29.83	37.77	47.54

Table 5: Spheres: m/M (%).

p	16	32	64	128	256
RP 1	5.26	8.35	12.86	18.85	26.09
RP 2	4.36	7.40	11.80	17.74	25.09
GP	6.68	11.99	17.44	24.04	31.74

Table 6: Axle: m/M (%).

4.2 The numerical characteristics

Basically, the numerical characteristics of the iterative method are the number of iterations needed to reach the prescribed tolerance of 10^{-5}, and the amount of work by iteration. Let us first address the latter for nonoverlapping subdomains.

Mainly, an iteration consists in solving p linear systems, one for each subdomain. Therefore the amount of work is proportional to the number of nonzero entries, or fill, of the LU-factorization of the local matrices, that is the restriction of the matrix to each subdomain. Average fills are listed Tables 7-9 in thousands for nonoverlapping subdomains. Interestingly, GP gives generally better results than the recursive heuristics. This can be explained in part by the average fill of the local matrices before factorization. It is equal to $\frac{(M+N)-m}{p}$ and therefore smaller for GP, as shown by Tables 4-6. The reordering and factorization process does not seem to reduce these original differences. The reordering is done using Sparsepack's nested dissection routines.

Moreover, as these methods are designed for parallel architectures, it is natural to study the load balancing, that is not the average fill but the fill for each subdomain. The most important values are the minimum and the maximum fills for a given partition. Tables 7-9 also indicate the minimum and maximum fill in square brackets. In most cases, there is a great imbalance.

The maximum value is representative of the largest amount of work on a processor, in the case of a parallel experiment. Generally, these are very close for the three heuristics. If we look at these values as a function of the number of partitions p, then RP 1 gives the best results for small p whereas GP performs best for large p. Note that here we only commented (and gave) results concerning nonoverlapping subdomains. This is because the conclusions remain identical for overlapping subdomains.

p	16	32	64	128	256
RP 1	84.3 [75.2,94.9]	35.9 [30.6,41]	15.0 [12.1, 16.8]	6.1 [4.7,7.0]	2.4 [1.5,2.9]
RP 2	90 [81,99.1]	37.8 [31.9,43.7]	15.5 [12.1,18.9]	6.2 [4.8,7.1]	2.4 [1.2,2.9]
GP	82.5 [70.8,94.9]	35.3 [24.4,41.2]	15.0 [11.4,18.1]	6.0 [4.7,7.22]	2.3 [1.5,2.9]

Table 7: Wing: Average fills [min,max] without overlap.

p	16	32	64	128	256
RP 1	34.8 [25.7,40.9]	11.8 [8.6,14.5]	3.9 [2.2,5.3]	1.3 [0.5,1.9]	0.4 [0.2,0.7]
RP 2	35.6 [31.1,43.1]	12.3 [9,16.3]	4 [2.1,5.9]	1.3 [0.7,2.1]	0.4 [0.2,0.7]
GP	32.9 [13.1,45.2]	11.3 [7.7,15.3]	3.6 [2.5,5.3]	1.2 [0.6,1.9]	0.4 [0.1,0.7]

Table 8: Spheres: Average fills [min,max] without overlap.

p	32	64	128	256
RP 1	387.7 [238.1,612]	132.4 [56.4,206.9]	43.8 [14,78.6]	14.3 [2.5,28.0]
RP 2	398.9 [228.4,604.2]	136.1 [60.7,222.6]	45 [9.4,72.8]	14.7 [1,26.8]
GP	356.7 [145.4,704]	115.8 [45.6,242.0]	37.9 [9.2,70.4]	12.5 [2.8,25.9]

Table 9: Axle: Average fills [min,max] without overlap.

Let us now investigate the possible origins of the imbalance. It can be caused by some original imbalance, i.e. the discrepancy in fill of the local matrices before they are factored. It can also be caused by the structure of these matrices which affects the reordering and factorization process. The second cause is not clearly related to the partitioning heuristic, but the first one is.

Indeed, the constraint, when partitioning, is on the number of vertices. Unfortunately, the nonzero entries of the local matrices correspond exactly to the edges of the graph. Thus, in order to have balanced local matrices, one has to replace the constraint on vertices by a constraint on edges. But it can not be easily done, since the average number of edges by subset $\frac{(M+N)-m}{p}$ is not known *a priori*: a value for m is obtained only after the partition is computed.

Nevertheless, for this particular problem, it is possible design special postprocessing strategies. Here, once m is determined, some "retrofitting" technique could be applied very simply, targeting the number of edges instead of the number of vertices. Note that, in this case, the original constraint on the number of vertices is not crucial any more. Briefly, let us mention another way of avoiding this problem in 2D. As a matter of fact, for a given region R of the 2D-triangulation, if T_R denotes the number of triangles and M_R the number of edges in the region, then

$$3T_R \approx 2M_R.$$

Based on this observation, it can be interesting to partition the triangulation in terms of triangles, as well balanced subsets of triangles correspond roughly to well balanced subsets of edges. In 3D, however, there is no such relationship between the number of tetrahedra and edges in a region.

Now, we consider the number of iterations as a function of both the number of subdomains p and the overlap.

First, we consider the nonoverlapping case. It is the only case we calculate for the axle because of memory limitations. We can see from Tables 10-12 that no heuristic performs really better than the others in terms of the number of iterations. Moreover, these numbers are relatively close to one another for a given p, especially when they are small (see the 3D examples). Finally, for the spheres and the axle we note that the number of iterations increases only slowly with the number of subdomains. This is particularly true in Table 12.

p	16	32	64	128	256
RP 1	146	158	134	136	152
RP 2	83	94	107	142	79
GP	105	83	96	119	144

Table 10: Wing: Number of iterations without overlap.

p	16	32	64	128	256
RP 1	21	25	22	28	27
RP 2	19	21	21	29	22
GP	19	21	27	24	29

Table 11: Spheres: Number of iterations without overlap.

p	32	64	128	256
RP 1	11	13	14	17
RP 2	10	13	16	16
GP	15	16	17	20

Table 12: Axle: Number of iterations without overlap.

Figures 1-10 illustrate the behavior of the number of iterations according to the overlap, the number of subdomains being fixed. For Figures 8 to 10, the missing parts of the curves are due to memory limitations. In the case of overlapping subdomains we can also see that no heuristic performs dramatically better. The different values of the number of iterations tend to be more clustered as the overlap increases. Once again, the behavior of the curves is not strictly monotonic. Yet, it tends to decrease as the overlap increases.

We note that the reason for the larger number of iterations for the problem in two dimensions is due to the Neumann boundary conditions which usually decrease the convergence rate of block Jacobi type preconditioners.

5 Conclusion

We solved some elliptic PDEs in two and three dimensions on unstructured grids using the one level overlapping Schwarz method. We partitioned the triangulations with three different heuristics and studied the impact of the partitioning on the numerical results. Based on these experiments, it appears that the three heuristics behave very similarly in terms of load balancing. For the convergence rate the increasing amount of overlap tends to smooth out the differences between the iterative solvers. For nonoverlapping subdomains it is difficult to point out any advantage of one heuristic over the others, since different heuristics perform best on different problems.

Acknowledgements

We thank Horst Simon for providing the code of RP 1 and Bruce Hendrickson and Robert Leland for allowing us to use the RP 2 code.

References

[1] S.T. Barnard and H.D. Simon. A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems. Technical report, NASA Ames Research Center, RNR-092-033, 1992.

[2] P. Ciarlet. *The finite element method for elliptic problems.* North Holland, 1978.

[3] P. Ciarlet, Jr, T.F. Chan, and W.K. Szeto. On the optimality of the median cut spectral bisection graph partitioning method. Technical report, UCLA, CAM 93-14, 1993.

[4] P. Ciarlet, Jr and F. Lamour. In preparation. Technical report, UCLA.

[5] P. Ciarlet, Jr and F. Lamour. An efficient low cost greedy graph partitioning heuristic. Technical report, UCLA, CAM 94-1, 1994.

[6] P. Ciarlet, Jr and F. Lamour. Recursive partitioning methods and greedy partitioning methods: a comparison on finite element graphs. Technical report, UCLA, CAM 94-9, 1994.

[7] M. Dryja. An additive Schwarz algorithm for two- and three-dimensional finite element elliptic problems. In T. Chan, R. Glowinsky, J. Périaux, and O. Widlund, editors, *Domain Decomposition Methods.* SIAM, Philadelphia, PA, 1989.

[8] M. Dryja and O. B. Widlund. An additive variant of the Schwarz alternating method for the case of many subregions. Technical report, 339, Dept of Computer Science, Courant Institute, 1987.

[9] C. Farhat and M. Lesoinne. Automatic partitioning of unstructured meshes for the parallel solution of problems in computational mechanics. *International Journal for Numerical Methods in Engineering,* vol 36, n^o 5, pp 745-764, 1993.

[10] C. Farhat and H.D. Simon. TOP/DOMDEC- a software tool for mesh partitioning and parallel processing. Technical report, NASA Ames Research Center, RNR-93-011, 1993.

[11] C.M. Fiduccia and R.M. Mattheyses. A linear-time heuristic for improving network partitions. *Proceedings of the 19th IEEE Design Automation conference,* IEEE, pp 175-181, 1982.

[12] M. Fiedler. Algebraic connectivity of graphs. *Czechoslovak Mathematical Journal,* 23, (98), pp 298-305, 1973.

[13] A. George and J. Liu. *Computer solution of large sparce positive definite systems.* Prentice-Hall, Englewood Cliffs, NJ, 1981.

[14] W.D. Gropp and B.F. Smith. Portable, Extensible Toolkit for Scientific Computation (PETSc). *Available at* `info.mcs.anl.gov` *in the directory* `pub/pdetools` *via anonymous ftp.*

[15] W.D. Gropp and B.F. Smith. Experiences with domain decomposition in three dimensions: overlapping Schwarz methods. In A. Quarteroni, Y.A. Kuznetsov, J. Périaux, and O. Widlund, editors, *Domain decomposition methods in science and engineering.* AMS contemporary mathematics, vol 157, 1993.

[16] W.D. Gropp and B.F. Smith. Scalable, extensible, and portable numerical libraries. *Proceedings of the Scalable Parallel Libraries Conference, IEEE,* pp 87-93, 1993.

[17] B. Hendrickson and R. Leland. Multidimensional spectral load balancing. Technical report, SAND 93-0074, 1993.

[18] B. Hendrickson and R. Leland. A multilevel algorithm for partitioning graphs. Technical report, SAND 93-1301, 1993.

[19] B.W. Kernighan and S. Lin. An efficient heuristic for partitioning graphs. *The Bell System Technical Journal*, vol 49, n° 2, pp 291-307, 1970.

[20] H.D. Simon. Partitioning of unstructured problems for parallel processing. *Computing Systems in Engineering*, vol 2, n° 2/3, pp 135-148, 1991.

[21] H. A. van der Vorst. Bi-cgstab: a fast and smoothly converging variant of bi-cg for the solution of nonsymmetric linear systems. *SIAM J. Sci. Stat. Comp.*, vol 13, n° 2, pp 631-644, 1992.

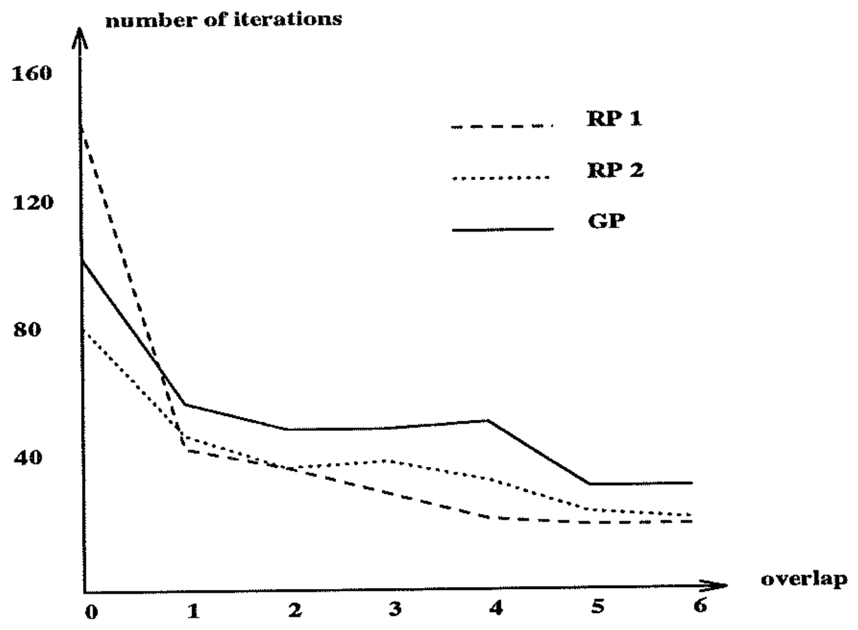

Figure 1: Wing in 16 subdomains.

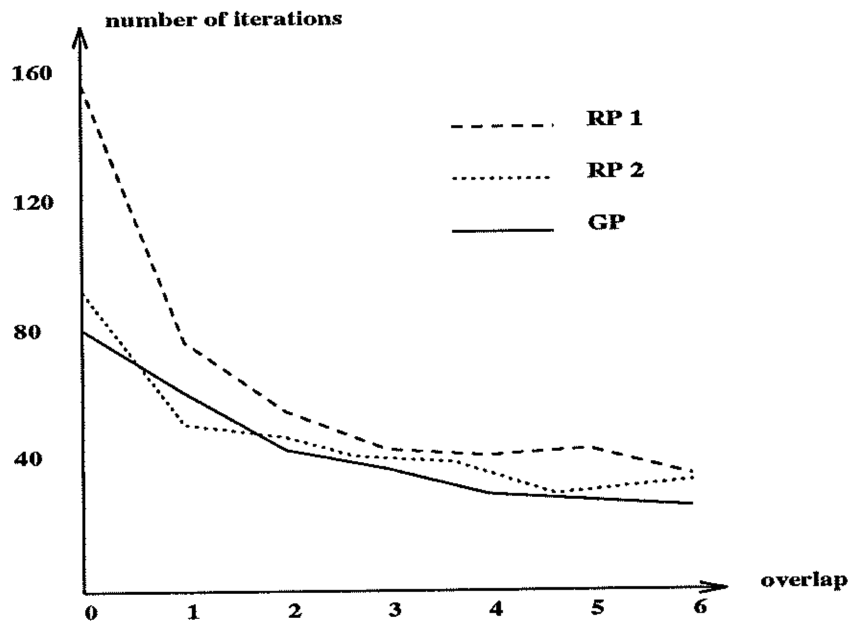

Figure 2: Wing in 32 subdomains.

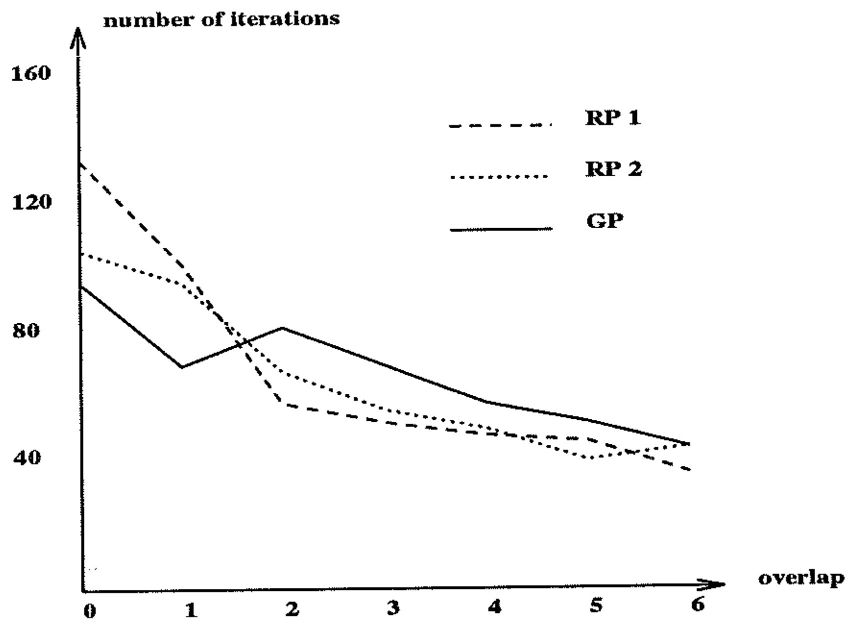

Figure 3: Wing in 64 subdomains.

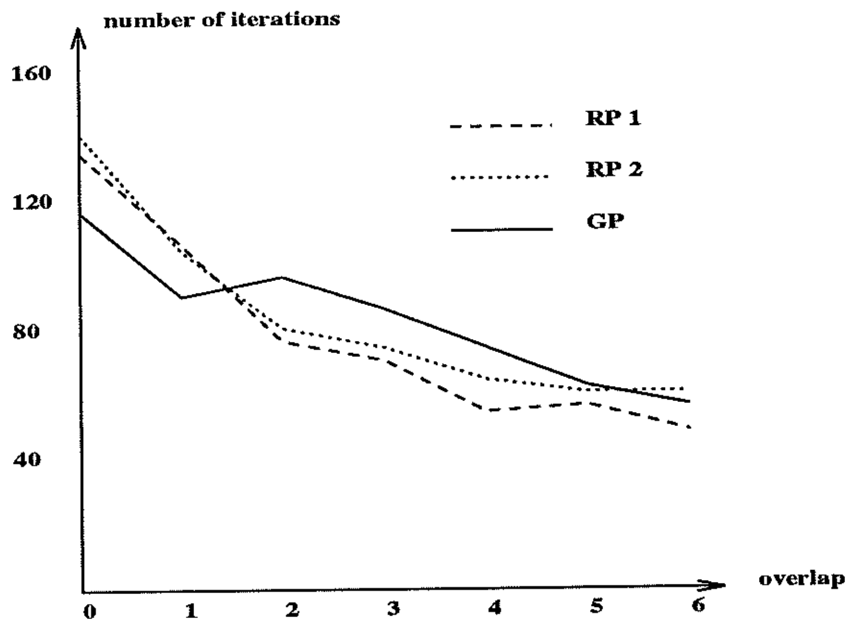

Figure 4: Wing in 128 subdomains.

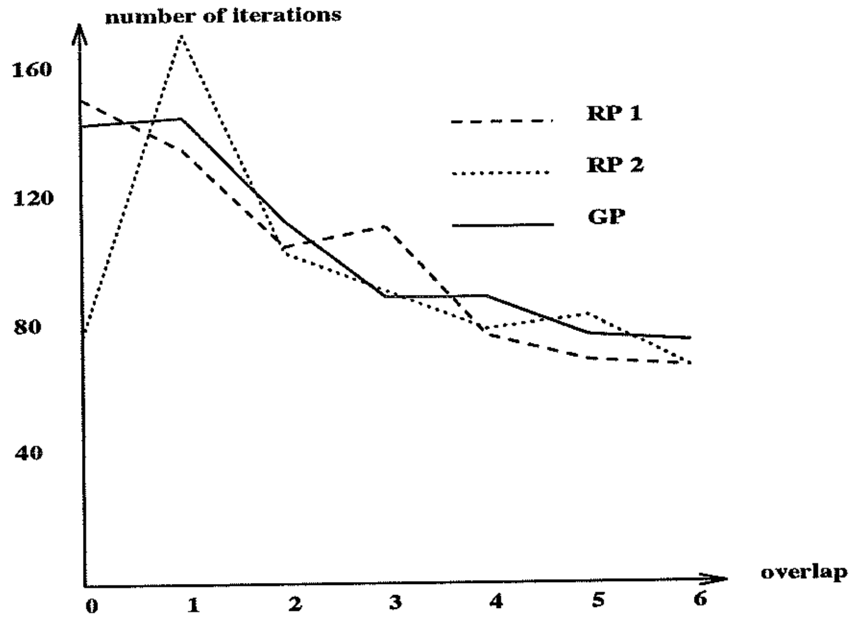

Figure 5: Wing in 256 subdomains.

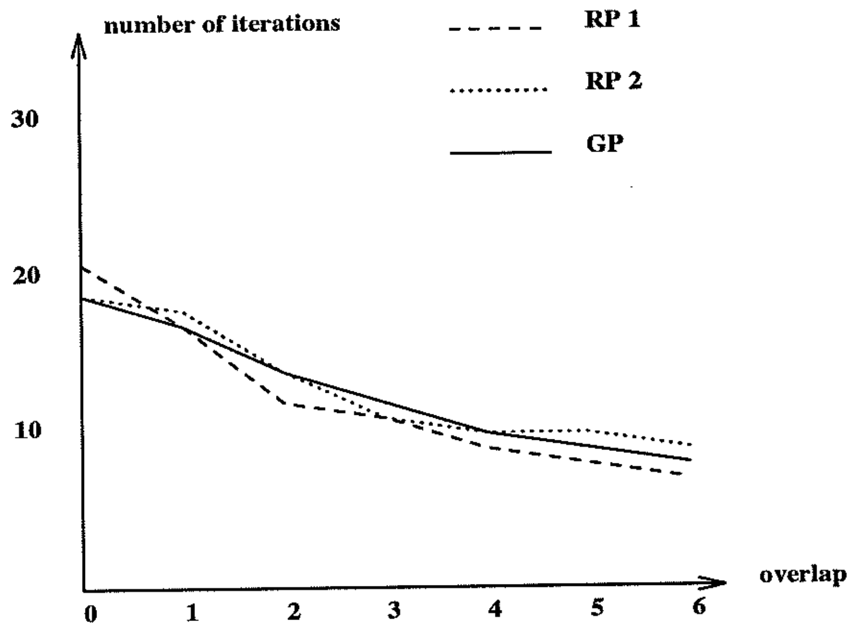

Figure 6: Spheres in 16 subdomains.

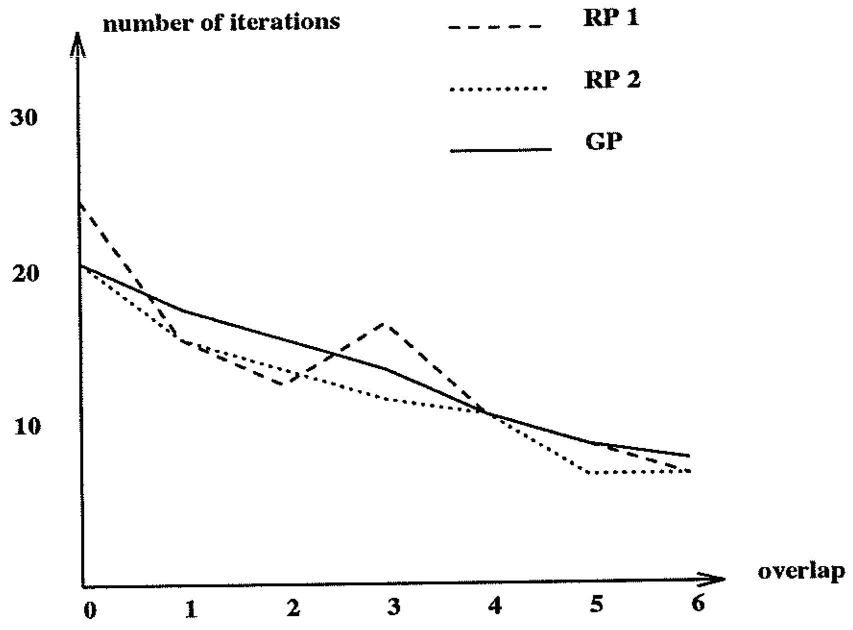

Figure 7: Spheres in 32 subdomains.

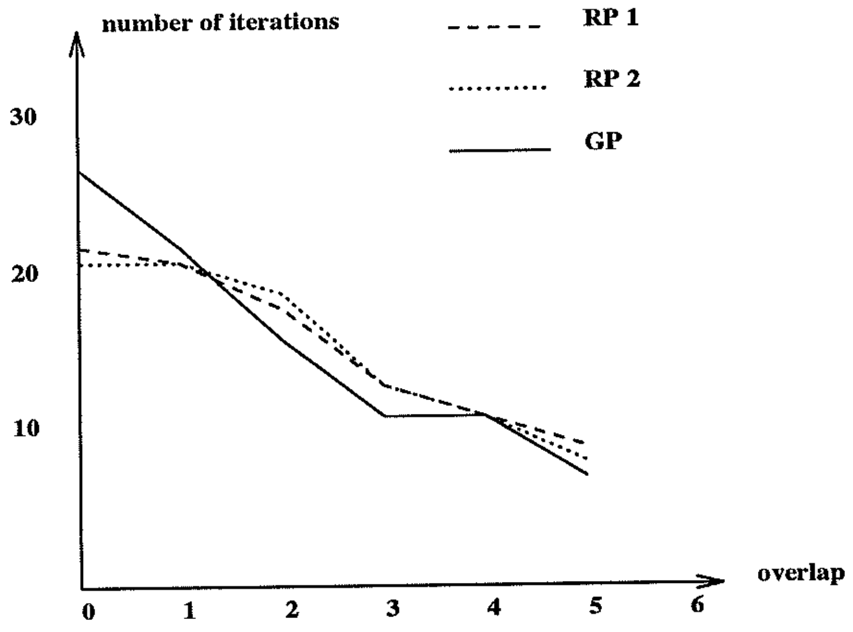

Figure 8: Spheres in 64 subdomains.

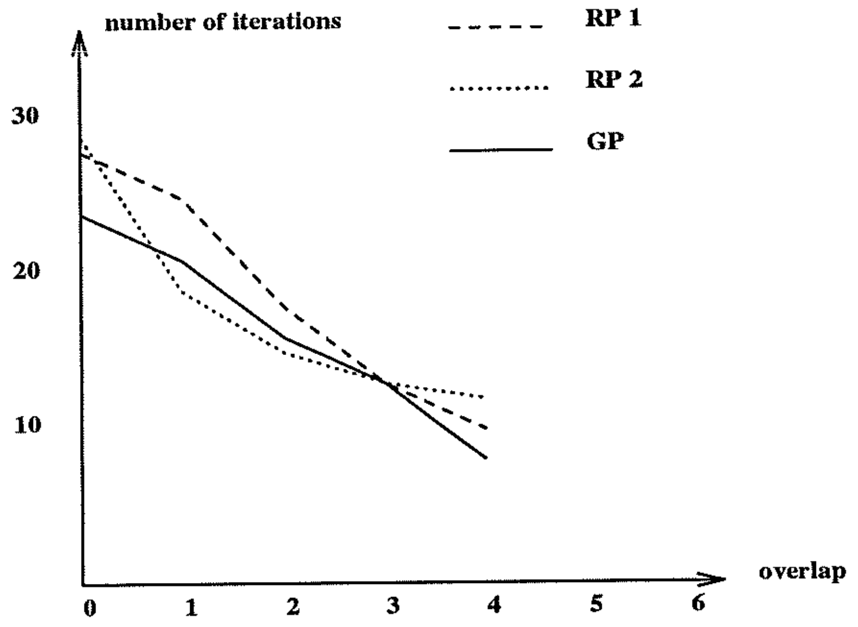

Figure 9: Spheres in 128 subdomains.

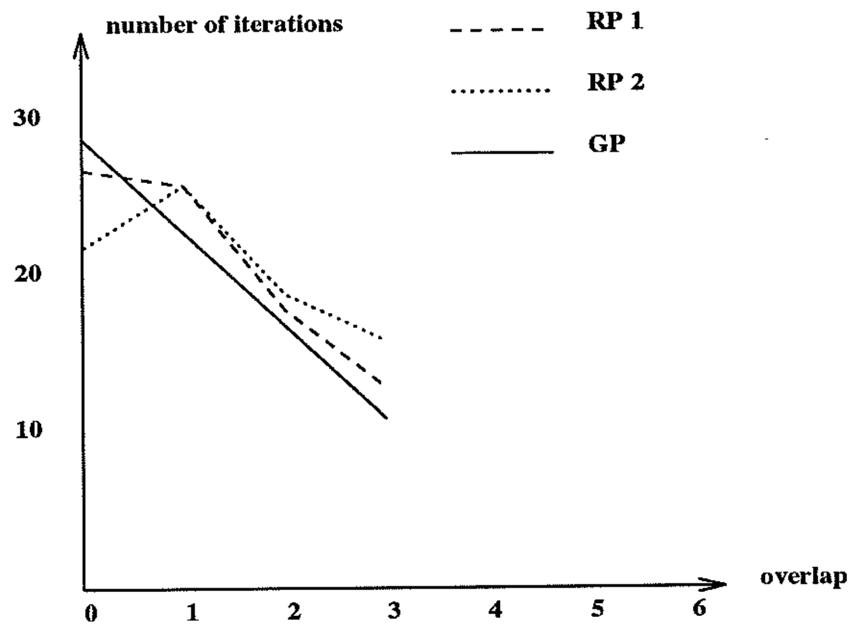

Figure 10: Spheres in 256 subdomains.