A retrofit based methodology for the fast generation and optimization of large-scale mesh partitions: beyond the minimum interface size criterion

D. Vanderstraeten\textsuperscript{a}, C. Farhat\textsuperscript{b,*}, P.S. Chen\textsuperscript{b}, R. Keunings\textsuperscript{a}, O. Ozone\textsuperscript{a}

\textsuperscript{a}Center for Systems Engineering and Applied Mechanics, Universit\'e Catholique de Louvain, B-1348 Louvain-la-Neuve, Belgium
\textsuperscript{b}Department of Aerospace Engineering Sciences, and Center for Aerospace Structures, University of Colorado at Boulder, Boulder, CO 80309-0429, USA

Received 17 September 1994

Abstract

Mesh partitioning is often the preferred approach for solving unstructured computational mechanics problems on massively parallel processors. Research in this area has focused so far on the automatic generation of subdomains with minimum interface points. In this paper, we address this issue and emphasize other aspects of the partitioning problem including the fast generation of large-scale mesh decompositions on conventional workstations, the optimization of initial decompositions for specific kernels such as parallel frontal solvers and domain decomposition based iterative methods, and parallel adaptive refinement. More specifically, we discuss a two-step partitioning paradigm for tailoring generated mesh partitions to specific applications, and propose a simple mesh contraction procedure for speeding up the optimization of initial mesh decompositions. We discuss what defines a good mesh partition for a given problem, and show that the methodology proposed herein can produce better mesh partitions than the well celebrated multilevel Recursive Spectral Bisection algorithm, and yet be an order of magnitude faster. We illustrate the combined two-step partitioning and contraction methodology with several examples from structural mechanics and fluid dynamics problems, and highlight its impact on the total solution time of realistic applications on current massively parallel processors. In particular, we show that the minimum interface size criterion does not have a significant impact on a reasonably well parallelized application, and highlight other criterion which can have a significant impact.

1. Nomenclature

\begin{itemize}
  \item $E$ set of edges of the dual graph of the mesh
  \item $P$ partitioning vector: $P_i = k$ means that the mesh entity $i$ belongs to subdomain $k$.
  \item $C$ cost function to be optimized.
  \item LBF load balance factor
  \item $L$ computational load of a given application
  \item $N_p$ number of processors
  \item $N_c$ number of subdomains
  \item $N_e$ number of elements in the mesh
  \item $N_n$ number of nodes in the mesh
  \item $N_d$ number of degrees of freedom in the model
  \item $N_k$ number of some specific mesh entities in subdomain $k$ (including its interface boundary)
\end{itemize}

* Corresponding author.

0045-7825/96/$15.00 \textcopyright$ 1996 Elsevier Science S.A. All rights reserved

PII S0045-7825(96)01024-9
2. Introduction

It is often argued and demonstrated that if unstructured computational mechanics problems are to be efficiently solved on distributed-memory parallel computers, their data structures must be partitioned and distributed across the processors in a way that maximizes load balance and minimizes interprocessor communication [1, 2]. However, research in mesh partitioning algorithms has mostly focused on the second issue—that is, on minimizing interprocessor communication costs only, and the number of interface points in a mesh partition, or the number of edge cuts in its corresponding graph, has rapidly become the main ‘acceptance test’ for a proposed mesh decomposer.

While several mesh partitioning algorithms have already been presented and/or discussed in the literature [1–8], two radically different schemes have particularly attracted the attention of the user and developer communities: the Greedy algorithm [3, 4, 9], and the Recursive Spectral Bisection algorithm [1, 7, 13].

The Greedy (GR) mesh partitioning algorithm was first proposed in [9] and applied to the parallel solution of finite element structural equations on the iPSC-1 system. This mesh decomposition scheme is referred to as the Greedy algorithm because it essentially ‘bites’ into the mesh in order to construct the subdomains. It exploits only the mesh connectivity information, which makes it the fastest partitioning algorithm we know about. In general, the GR algorithm tends to generate mesh partitions that are characterized by reasonable subdomain aspect ratios and a relatively small number of interface points. On a few occasions, this algorithm has been misrepresented [10], perhaps, because one statement is unfortunately missing in the corresponding Fortran code given in [3]. This statement is the one which forces every subdomain to start with an element attached to the previously computed interface. The GR algorithm enjoys a relatively large user community because of its high performance/price ratio. For example, it is capable of partitioning a three-dimensional unstructured mesh containing 439 272 tetrahedra and 77 279 vertices into 64 subdomains with 25 906 interface points, in less than 15 seconds on a Crimson Silicon Graphics workstation. Recently, some interesting variants of the basic GR algorithm have been proposed [11, 12].

The Recursive Spectral Bisection (RSB) graph partitioning algorithm was first proposed in [7]. This scheme is at the same time the least intuitive mesh decomposer, and the partitioning algorithm that has most attracted the attention of the parallel computing community. Unlike the Greedy algorithm which is simple and has no underlying theory, the RSB scheme is sophisticated and relies on a relatively well understood graph theory. More specifically, the RSB algorithm is derived from a graph bisection strategy based on the computation of the Fiedler vector—that is, the second eigenvector of the Laplacian matrix of the graph associated with the given problem [7]. Thanks to the multilevel strategy described in [13] for extracting the Fiedler vector, the computational requirements of this partitioning scheme are no longer overwhelming, even on a simple workstation. However, the multilevel RSB algorithm is still more expensive than most other partitioning schemes. For example, when applied to the decomposition into 64 subdomains of the same three-dimensional mesh containing 439 272 tetrahedra and 77 279 vertices, it consumes 707 seconds on a Crimson Silicon Graphics workstation and generates a mesh partition with 21 139 interface points. This mesh partition has 18.40% less interface points than the decomposition generated by the Greedy algorithm, but costs 48.07 times more CPU time to generate. Depending on the target parallel application, such an improvement at such price may or may not be interesting. Recently, a parallel version of the RSB algorithm has been implemented on the CM-5 [15]. This version has certainly improved the computational feasibility of the RSB partitioner.
REMARK 1. Throughout this paper, RSB designates the multilevel Recursive Spectral Bisection algorithm. In particular, all performance results reported for RSB applications correspond to the fast multilevel scheme described in [13], and release 2.1 of the code as integrated in TOP/IDOMDEC [30].

Minimizing interprocessor communication costs in general, and the number of interface points in a mesh partition in particular, is a reasonable objective to 'prioritize' when the target parallel application:

(a) involves communication essentially between neighboring subdomains. This is typically the case for explicit time-integration (or pseudo-integration) schemes, and some basic iterative solvers such as the conjugate gradient or Jacobi preconditioned conjugate gradient methods.

(b) has a computational complexity that can be simply related to mesh entities such as, for example, nodes, and/or edges, and/or elements, and/or cells. In that case, load balancing can be reasonably well achieved by requiring that each subdomain contain the same number of such entities. In the event of heterogeneous meshes, a weighting factor can be attributed to each basic entity and the number of mesh entities per subdomain can be modified accordingly. Most importantly, load balancing in that case does not significantly interfere neither with the minimum edge cut aspect of a graph partitioner, nor with the practical implementation of the corresponding mesh decomposer.

(c) uses a solution methodology whose performance is insensitive to the characteristics of a mesh partition such as, for example, the subdomain aspect ratio or the subdomain interconnectivity.

It is our experience that when conditions (a), (b) and (c) are met, the GR and RSB algorithms generate excellent mesh partitions for parallel processing. Therefore, we have consistently used both algorithms for the subset of our parallel applications that can be described by the above (a), (b) and (c) points.

However, many important parallel applications do not fit the profile implied by the (a), (b) and (c) points. For example, frontal sparse solvers [16–20] require mesh partitions that do not significantly inflate the operation count of their sequential counterparts [17, 18]. This particular issue relates more to the subdomain local frontwidths than to the subdomain interface sizes. Moreover, controlling the load balance of these direct solvers is not in general as simple as distributing equally some basic mesh entities across the desired subdomains.

Optimal domain decomposition based iterative solvers are another class of parallel applications whose scalability is not governed by interprocessor communication costs only [21]. These solution algorithms are interesting on massively parallel processors when their number of iterations for convergence does not grow (or grows weakly) with the number of subdomains. Their effectiveness is determined by their convergence rate and not by their amount of communication. In particular, optimal non-overlapping domain decomposition based iterative solvers require mesh partitions that have as perfect subdomain aspect ratios (close to unity) as possible. Sometimes, fulfilling this requirement leads to mesh partitions with larger interfaces than otherwise possible. This is well demonstrated below for the structural High Speed Civil Transport wing finite element model containing 3150 nodes. For this problem, the 32-subdomain mesh partition generated by the RSB algorithm has 707 interface nodes and an average subdomain aspect ratio $AR = 0.39$ (Fig. 1). The 32-subdomain mesh partition generated by the methodology described in this paper and shown in Fig. 2 has 808 interface nodes, but an average subdomain aspect ratio $AR = 0.62$. The FETI domain composition iterative solver [22] applied to the structural wing problem converges in 47 iterations and 11.93 seconds when using the RSB mesh partition on a 32-processor iPSC-860: it converges in 30 iterations and 7.75 seconds when using the mesh partition with larger interface but improved subdomain aspect ratio [23]. Hence, one should question whether the minimum interface size is not after all an over-emphasized mesh partitioning criterion.

The main objectives of this paper are: (a) to highlight the decomposition issues introduced above, and (b) to present a general methodology for partitioning unstructured meshes that addresses the minimum edge cut problems as well as several other application specific important problems. This methodology is simple to implement and is computationally efficient even on conventional workstations. To this effect, the remainder of this paper is organized as follows. In Section 3, we overview a two-step partitioning paradigm that was first proposed in [24, 25] and whose key ingredients are a fast mesh decomposer and a computationally feasible interface optimizer. Three optimization algorithms are introduced, namely,
Simulated Annealing, Tabu Search, and Stochastic Evolution. All three algorithms can handle various objective functions at a reasonable computational cost, because their range of search is confined to the subdomain interfaces. In Section 4, we present a mesh contraction procedure for speeding up the optimization step by as much as one order of magnitude. We show that using a simple decomposition algorithm, the two-step partitioning paradigm and the proposed contraction procedure can generate better mesh partitions (where 'better' is application dependent) than the widely adopted multilevel RSB algorithm, and up to 15 times faster. In Section 5, we illustrate the proposed partitioning methodology with several examples from computational structural mechanics and fluid dynamics problems, and highlight its impact on their total parallel solution time on an iPSC-860 computer and a Convex Meta
3. The two-step partitioning and retrofitting paradigm

As stated in the introduction, the main objective of this paper is to present a general purpose and efficient mesh partitioning methodology that can address the minimum edge cut problem as well as other application specific issues. Since the field of computational mechanics is diverse, such a
methodology cannot be based on a large library of mesh splitting algorithms that address the parallel computing needs of every known computational scheme. On the other hand, it would also be unreasonable not to exploit the existence of well-established partitioning schemes such as the Greedy, Recursive Spectral Bisection, and others.

In order to meet our objective, we have adopted the two-step mesh partitioning paradigm that was first introduced in [24, 25] and which consists in:

Step 1. generating an initial mesh decomposition via either a suboptimal but fast partitioning algorithm, or an algorithm that is known to produce mesh partitions that are reasonably well suited for the target parallel application.

Step 2. formulating the application specific requirements as a cost function $C$, and optimizing it by readjusting the initial subdomain interfaces. This step can also be described as a ‘retrofitting’ procedure [26].

The Greedy algorithm is very fast because its complexity grows as $O(N_e \times N_j)$. Moreover, it produces mesh partitions that are reasonably well-suited for most parallel computational methods. Hence, the GR algorithm is ideally suited for generating an initial decomposition in Step 1. However, sophisticated schemes such as the RSB may also be used in Step 1, especially if preprocessing costs are not an issue.

In Step 2, a cost function representing the decomposition requirements of the target parallel application must first be formulated. A sample list of cost functions to optimize is given below:

- **Interface size**
  \[ C_1 = \frac{1}{2} |\{(i, j) \in E | P_i \neq P_j\}| \]
  Here, the size of the interface is defined as the number of edges in $E$ whose vertices belong to two different subdomains. This cost function may not govern all parallel applications but is certainly helpful in all cases.

- **Load imbalance**
  \[ C_2 = \sum_{k=1}^{N_k} \left[ (N^k - N_{\text{best},k})^2 \right] \]
  When a parallel application has a computational complexity that can be simply related to mesh entities such as, for example, nodes, and/or edges, and/or elements, and/or cells, the computational load $L$ can be easily terminated and $N_{\text{best},k}$ can be set prior to the decomposition to $N_{\text{best},k} = L/N_p$. Otherwise, $N_{\text{best},k}$ is unknown a priori. It can have a different value in every subdomain $k$, and is adaptively evaluated by the optimization algorithm. As an example, the load balance of a parallel frontal solver is discussed in Section 5.3.

- **Subdomain aspect ratio**
  \[ C_3 = \sum_{k=1}^{N_k} \left( \sum_{d=1}^{d_{\text{max}}} \sum_{j=1}^{j_{\text{max}}} (x_{ijk} - \bar{x}_d)^2 \right) \]
  This cost function has been shown to play a pivotal role in the convergence rate of optimal domain decomposition based preconditioned conjugate gradient methods [23].

In practice, the performance of a parallel application is often governed by several distinct factors. Therefore, one should consider in general the following weighted cost function:

\[ C = \sum \alpha_i C_i \]

where $C_i$ is a cost function representing one specific issue—for example, $C_i$ could be anyone of the cost functions listed above—and $\alpha_i$ is the weight attributed to that issue. In that case, optimizing $C$ corresponds to finding the best possible ‘compromise’ mesh partition. Unfortunately, we do not have yet an automatic mechanism for selecting the weight coefficients $\alpha_i$. For this task, we rely on our understanding of the focus parallel application, and experience with the target parallel processor.

After a cost function is formulated the decomposition is optimized by readjusting only the subdomain interfaces. More specifically, only the mesh entities that are attached to the interface are examined for possible exchange between the subdomains. Therefore, the computational complexity of the optimization process is proportional to the interface size and not to the number of elements in the mesh. In our work, we have implemented three different schemes for optimizing a given cost function:

- **Simulated Annealing** (SA) [27]. This algorithm uses a monotonically decreasing ‘temperature’ as control variable for the outer iterations. For a fixed temperature, a number of mesh entities are proposed for transfer to a neighboring subdomain—in the sequel, we refer to this step as a ‘move’. The acceptance of a move is dictated by a probabilistic decision which depends on the difference in cost
between making the move or ignoring it. The optimization process ends when the temperature is sufficiently low and no further moves are accepted. In the inner loop, moves are chosen randomly. The probability of acceptance of bad moves decreases with temperature.

**Tabu Search** (TS) [28]. This scheme stores in a *tabu* list a specified number of recently accepted moves. In the inner loop, several moves outside the tabu list are proposed, and the move with the highest positive or negative gain is accepted. In the outer loop, the last accepted move replaces the oldest move in the tabu list. Therefore, if this algorithm escapes a local minimum, it cannot use the same path in the solution space to reach this minimum again.

**Stochastic Evolution** (SE) [29]. The main difference between this algorithm and Simulated Annealing is in the evolution of the control variable and the selection of the moves. At each outer iteration, all interface elements are proposed for a move in a predefined order. The temperature decreases rapidly, thereby decreasing the probability of accepting bad moves, until the solution reaches a local minimum of the cost function. At this point, the temperature is reset to its initial value. In general, this algorithm behaves as a series of fast SA processes where the solution jumps from one local minimum to another. A quality/speed trade-off can be applied to each of the above optimization schemes by ‘tuning’ a few control parameters [25].

There is at least one compelling reason for having more than one optimization algorithm at hand. In some cases, the initial mesh partition generated in Step 1 can get entrapped in a local minimum at the first step of an optimization scheme, in which case Step 2 does not improve the original decomposition. One can hope that switching to another optimization algorithm pulls the solution out of that local minimum. Everytime we have encountered this problem for SA, we were able to resolve it by switching to TS.

In order to illustrate the two-step methodology described above and highlight its potential, we consider the partitioning of two three-dimensional fluid dynamics unstructured meshes into 64 and 128 subdomains. The first mesh, FALC, is designed for the simulation of external Euler flows around a Falcon aircraft (Fig. 3). It contains 439777 tetrahedra and 77779 vertices. The second mesh, MUFF, is designed for the simulation of internal viscous flows inside a car muffler (Fig. 4). It contains 237963 tetrahedra and 43592 vertices. Here, we assume that the objective is to generate mesh partitions with equal number of tetrahedra and minimum number of interface points. Hence, the load balance factor can be written in this case as follows:

\[
LBF = \frac{\text{average}_k N^k_t}{\max_k N^k_t} \tag{2}
\]

More complex objectives are deferred to Section 5.

First, the GR and multilevel RSB algorithms are used to partition the FALC mesh into 64 subdomains, and the MUFF mesh into 128 subdomains. Following the recommendation given in [13], the computational size of the lowest level is set to 300 for the RSB scheme. Next, the two-step methodology is applied to generate similar mesh partitions. The GR algorithm is selected for Step 1, and the SA optimization scheme for Step 2. For both meshes, the cost function is defined as \( C = 0.5 \times C_1 + 0.5 \times C_2 \), and the parameters \( N^k \) and \( N^{\text{best},k} \) are set to \( N^k = N^k_t \) and \( N^{\text{best},k} = N^k_t/N_p \). The characteristics of the resulting mesh partitions are summarized in Tables 1 and 2. All computations are performed on a Crimson Silicon Graphics workstation.

The results reported in Table 1 show that for the FALC mesh, RSB outperforms GR for the imposed objective. The mesh partition produced by RSB has 18.40% less interface points than that delivered by GR, but costs 48.07 times more CPU time to generate. On the other hand, the two-step partitioning methodology with GR as an initial decomposer outperforms RSB for the same objective. The mesh partition generated by GR and optimized by SA has 9.83% less interface points than that delivered by RSB and costs 1.97 times less CPU time to produce.

For the MUFF mesh, the results reported in Table 2 show that GR outperforms RSB for the imposed objective. More specifically, GR produces a mesh partition with 2.53% less interface points than RSB does and 115.40 times faster. The two-step partitioning methodology with GR as an initial decomposer outperforms both RSB and GR, is significantly cheaper than RSB, but is also significantly more expensive than GR.
For the above two examples, we have used GR as an initial decomposer in order to keep the total partitioning costs as low as possible. However, if preprocessing costs are not an issue, RSB can also be used in Step 1. For the MUHF mesh, the two-step method with RSB as an initial decomposer generates
Table 1
Partitioning of the FALC mesh: \( N_s = 439,272 \) - \( N_i = 64 \)
\[ C = 0.5 \times C_1 + 0.5 \times C_2 \]
SGI/Crimson

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Optimizer</th>
<th>( N_i )</th>
<th>LBF</th>
<th>CPU (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Step 1</td>
</tr>
<tr>
<td>RSB</td>
<td>None</td>
<td>21,139</td>
<td>0.999</td>
<td>707.10</td>
</tr>
<tr>
<td>GR</td>
<td>None</td>
<td>25,906</td>
<td>0.999</td>
<td>14.71</td>
</tr>
<tr>
<td>GR</td>
<td>SA</td>
<td>19,080</td>
<td>0.999</td>
<td>14.71</td>
</tr>
</tbody>
</table>

Fig. 4. Three-dimensional discretization of the flow domain inside a car muffler.

We are particularly interested in fast and good partitioning algorithms because we would like to be able to inspect—possibly interactively—a few mesh decompositions before selecting one for a target parallel application. The examples reported above highlight the potential of the two-step methodology.
for generating excellent mesh partitions. However, the optimization step is not as fast as we would like it to be. Next, we present a contraction procedure for speeding up the optimization process in Step 2.

4. An efficient contraction procedure

The idea of contracting a mesh before partitioning it is not new. It was first proposed in [13] for reducing the costs of the RSB partitioning scheme, and recently in [14] for storage optimization purposes. The coarsening approach presented in [13] is based on the concept of maximal independent sets. The contraction approach proposed herein is based on the Greedy algorithm and our experience with this heuristic. Our main objective is to speed up the optimization process in Step 2 of the partitioning methodology described in Section 3. Our main strategy goes as follows.

First, the mesh is recursively coarsened using an $O(N)$ Greedy-based contraction procedure until its size reaches a user specified value, say $N_{me} = 5000$ macro-elements. An initial decomposition is performed on the coarse mesh using preferably a fast mesh partitioning algorithm. This decomposition is followed by a few smoothing iterations using one of the three optimization schemes introduced in Section 3. Next, the obtained coarse partition is mapped onto the original and finer mesh, and another optimization is performed on the fine level. When more than one level of contractions is needed to reach the specified number of macro-elements $N_{me}$, coarse-to-fine mapping and optimization are performed at every intermediate level.

More specifically, the contraction step is implemented as follows. Given a starting element, a fixed-size cluster is constructed by agglomerating neighboring elements in a recursive manner. This cluster defines a macro-element in the contracted mesh. At the beginning, the starting element is selected among the peripheral elements. Later, it is selected among those elements which neighbor existing clusters. The contraction ends when all elements are attributed to a cluster. In practice, we have found that 5 elements is a good choice for the size of a cluster. However, fewer or more elements can sometimes define a cluster for connexity purposes.

The impact of the contraction procedure described above on the two-step partitioning methodology is highlighted in Tables 3 and 4 for the FALC and MUFF meshes, respectively.

For the FALC mesh and 64 subdomains, the contraction procedure is shown to reduce the cost of Step 2 by a full order of magnitude. In that case, the two-step partitioning method with GR as an initial...
Table 4
Partitioning of the MUFF mesh: \( N_i = 237,963 - N_i = 128 \)
\[ C = 0.5 \times C_1 + 0.5 \times C_2 \]

Effects of the contraction procedure
SGI/Crimson

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Optimizer</th>
<th>( N_i )</th>
<th>LBF</th>
<th>CPU (s)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RSB</td>
<td>None</td>
<td>17,810</td>
<td>0.999</td>
<td>0.00</td>
<td>791.69</td>
</tr>
<tr>
<td>GR</td>
<td>None</td>
<td>17,358</td>
<td>0.999</td>
<td>0.00</td>
<td>6.86</td>
</tr>
<tr>
<td>GR SA</td>
<td>SA</td>
<td>14,934</td>
<td>0.996</td>
<td>0.00</td>
<td>551.72</td>
</tr>
<tr>
<td>Contr. + GR</td>
<td>Contr. + SA</td>
<td>12,792</td>
<td>0.999</td>
<td>3.02</td>
<td>143.70</td>
</tr>
</tbody>
</table>

decomposer produces a mesh partition with 23.55% less interface nodes than that generated by RSB, and is 15.68 times faster than RSB.

For the MUFF mesh and 128 subdomains, the two-step partitioning method with graph contraction and GR as an initial decomposer produces a mesh partition that has 28.17% less interface nodes than the RSB partition, and is 5.39 times faster than RSB.

The performance results reported in Tables 3 and 4 also show that the proposed contraction procedure not only speeds up the two-step partitioning method, but also results in better mesh decompositions. Indeed, the contracted mesh represents the structure of the original grid, and the optimization of its decomposition tends to improve the global structure of the desired mesh partition by moving several elements simultaneously. When the mesh is not contracted, the global structure of the mesh partition remains identical to that of the initial decomposition because the probability of transferring large amounts of elements between the initial subdomains is usually low.

As mentioned earlier, a quality/speed trade-off can be applied to each of the three optimization schemes by ‘tuning’ some of their control parameters [25]. An example of such trade-off is illustrated in Table 5 for the FALC mesh and various number of subdomains. From the results reported in Table 5, it follows that, for the cost function \( D = 0.5 \times C_1 + 0.5 \times C_2 \), the two-step partitioning method with contraction can generate even better mesh partitions when the optimization algorithm is allowed to run longer in Step 2. Note that even in that case, the two-step method is still significantly cheaper than the multilevel RSB algorithm. For example, it can generate a 64-subdomain partition for the FALC mesh with 14,613 interface points only in 161.04 s whereas the RSB scheme consumes 707.10 s to generate a 64-subdomain mesh partition with 21,139 interface points (see Table 3). This amounts to an almost twice better mesh partition at a quarter of the price. The performance results summarized in Table 5 also show that the complexity of the two-step partitioning method with contraction is sublinear with the number of subdomains.

In the remainder of this paper, we use exclusively GR for all initial decompositions. We show that in
all cases, the two-step partitioning methodology with the contraction procedure described herein is a cheaper and better alternative to the multilevel RSB algorithm. However, we remind the reader that if preprocessing costs are not an issue, the two-step partitioning methodology with contraction can also be used with RSB as initial decomposer, and may produce even better results than with GR as initial partitioner.

5. Applications

The two-step decomposition methodology and the contraction procedure described in this paper have been implemented in the TOP/DOMDEC [30] interactive software package for mesh partitioning and parallel processing. In this section, we illustrate these two methodologies with several examples from computational structural mechanics and fluid dynamics, and highlight their impact on the parallel solution time of these problems on an iPSC-860 multiprocessor and a Convex Meta Series system.

Mesh partitioning algorithms are often evaluated and/or benchmarked by simply assessing and/or comparing the characteristics of the mesh partitions they generate (interface size, theoretical load balance factors, ...). Such an approach is at best incomplete. The ultimate goal of a mesh partitioning algorithm is to reduce, if possible, the parallel CPU time of the target parallel application. Hence, mesh partitioning algorithms should be benchmarked by comparing their impact on problem solving. Here, we consider three classes of applications: the solution of a set of sparse linear equations via a domain decomposition based iterative algorithm, and the solution of a system of sparse linear equations via a frontal method.

5.1. Explicit time-marching

First, we consider a stress wave propagation problem in a line-pinched plate with a circular hole. The plate is discretized using 47,680 4-node shell elements and 48,235 nodes (Fig. 5). The corresponding number of equations is 233,939. For this problem, the semi-discrete finite element equations of dynamic equilibrium are time-integrated using the explicit central difference scheme. Four different mesh partitions are generated for parallel computations on a 64-processor iPSC-860 system. The characteristics of these decompositions are summarized in Table 6 where $N_{\text{min}}^{k}$, $N_{\text{average}}^{k}$, $N_{\text{max}}^{k}$ and $N_{i}$ denote, respectively, the minimum, average and maximum number of interface nodes per subdomain, and the total number of interface nodes in the mesh partition. Given that the parallel performance of the central difference scheme—and most explicit time-integration algorithms—is governed by load balancing and communication costs, the cost function $C = 0.5 \times C_{\text{comm}} + 0.5 \times C_{\text{sol}}$ and the parameters $N_{i}^{k} = N_{i}$ and $N_{\text{best}}^{k} = \sum_{i=1}^{k} N_{i}^{k} / N_{p}$ are used for this application. For the above problem and 64 subdomains, the interface size of the mesh partition generated by the RSB scheme is 14% smaller than that of the mesh partition produced by the GR algorithm. On the other hand, the two-step partitioning methodology without contraction reduces the interface size of the GR decomposition by 29%, and with contraction it reduces it by 36%.

The performance results on a 64 processor iPSC-860 system of the transient analysis of the plate problem are reported in Table 7 for 2000 integration time-steps, and the four generated 64-subdomain mesh partitions. Throughout the remainder of this paper, $T_{\text{comm}}$ and $T_{\text{sol}}$, denoted, respectively, the communication time, and the total solution time for the target parallel application. Clearly, the results reported in Table 7 show that the communication costs of the explicit central difference time-integration algorithm are directly related to the number of interface nodes (for this problem, it turns out that all generated mesh partitions have a similar average number of neighboring subdomains). However, these results also indicate that for this class of parallel applications, there is little to gain by searching for the 'perfect' mesh partition with the least number of interface nodes. For example, the two-step mesh decomposition algorithm with contraction reduces the interface size and communication costs of the GR partition by factors equal to 1.36 and 1.40, respectively, but improves the total CPU time corresponding to the GR partition by 5% only. Hence, it would seem that the applications for which one has legitimate reasons to prioritize the minimization of the interface nodes are the least sensitive to the size
Fig. 5. Finite element discretization of a plate with a circular hole.

Table 6
Partitioning of the plate mesh: \( N = 47,680 - N_s = 48,235 - N_d = 233,939 \) — \( \frac{N}{N} = 64 \)

\[ C = 0.5 \times C_1 + 0.5 \times C_2 \]

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Optimizer</th>
<th>Contraction</th>
<th>( N_{\text{inv.}} )</th>
<th>( N_{\text{per step}} )</th>
<th>( N_{\text{max.}} )</th>
<th>( N )</th>
<th>( N_{\text{GR}} / N )</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSB</td>
<td>None</td>
<td>No</td>
<td>74</td>
<td>108</td>
<td>199</td>
<td>3435</td>
<td>1.14</td>
</tr>
<tr>
<td>GR</td>
<td>None</td>
<td>No</td>
<td>56</td>
<td>124</td>
<td>286</td>
<td>3912</td>
<td>1.00</td>
</tr>
<tr>
<td>GR</td>
<td>SA</td>
<td>No</td>
<td>53</td>
<td>101</td>
<td>149</td>
<td>3039</td>
<td>1.29</td>
</tr>
<tr>
<td>GR</td>
<td>SA</td>
<td>Yes</td>
<td>52</td>
<td>98</td>
<td>144</td>
<td>2876</td>
<td>1.36</td>
</tr>
</tbody>
</table>

Table 7
Explicit central difference
Plate mesh: \( N = 47,680 - N_s = 48,235 - N_d = 233,939 \) — \( \frac{N}{N} = 64 \)

Solution time for 2,000 time-steps on an [PSC-860/64]

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Optimizer</th>
<th>Contraction</th>
<th>( T_{\text{comm}} ) (s)</th>
<th>( T_{\text{sol}} ) (s)</th>
<th>( N_{\text{GR}} / N )</th>
<th>( T_{\text{comm}} / T_{\text{comm}} )</th>
<th>( T_{\text{sol}} / T_{\text{sol}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSB</td>
<td>None</td>
<td>No</td>
<td>115.28</td>
<td>706.91</td>
<td>1.14</td>
<td>1.20</td>
<td>1.03</td>
</tr>
<tr>
<td>GR</td>
<td>None</td>
<td>No</td>
<td>138.34</td>
<td>728.12</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>GR</td>
<td>SA</td>
<td>No</td>
<td>101.72</td>
<td>693.45</td>
<td>1.29</td>
<td>1.36</td>
<td>1.05</td>
</tr>
<tr>
<td>GR</td>
<td>SA</td>
<td>No</td>
<td>101.72</td>
<td>693.45</td>
<td>1.29</td>
<td>1.36</td>
<td>1.05</td>
</tr>
<tr>
<td>GR</td>
<td>SA</td>
<td>Yes</td>
<td>98.81</td>
<td>693.41</td>
<td>1.36</td>
<td>1.40</td>
<td>1.05</td>
</tr>
</tbody>
</table>

of the subdomain interfaces. Of course, such a statement assumes that the given parallel processor is reasonably fast in communication, and that the size of the problem to be solved justifies the chosen number of subdomains or processors.

One could argue that the above conclusions hold only for two-dimensional problems where the subdomain interfaces are topologically one-dimensional, but not necessarily for three-dimensional
problems where the subdomain interfaces are topologically two-dimensional, and the average number of neighbors for a given subdomain is higher. For this reason, we investigate next the parallel performance of the explicit central difference algorithm applied to the evaluation of the linear transient response of a three-dimensional engine nozzle subjected to a sudden pressure burst. The nozzle is discretized into 12,800 8-node brick elements, 15,579 nodes and 46,701 active degrees of freedom (Fig. 6). Four different mesh partitions are generated for parallel computations on a 64-processor iPSC-860 system. The characteristics of these decompositions are summarized in Table 8. As for the previous

Fig. 6. Three-dimensional finite element discretization of a nozzle.
example, the cost function is set to $C = 0.5 \times C_1 + 0.5 \times C_2$, $N^k$ is set to $N^k = N_{f}^k$, and $N_{best}^k = \sum_{k=1}^{K} N^k / N_p$ is adopted. For the nozzle problem and 64 subdomains, the mesh partition generated by the RSB scheme has 1.12 times less interface nodes than that produced by the GR algorithm. The two-step partitioning methodology with contraction reduces the interface size of the GR decomposition by a factor equal to 1.19. Note that reducing the total number of interface nodes also seems to improve the interface load balancing factor $ILBF = N_{average,k} / N_{max,k}$. For example, $ILBF = 0.67$ only for the 64-subdomain mesh partition generated by the GR algorithm, while $ILBF = 0.80$ for that produced by the two-step decomposition methodology.

Table 9 reports the CPU time on a 64-processor iPSC-860 system of a 2000 time-step transient analysis of the engine nozzle using the various 64-subdomain mesh partitions. Before commenting on the performance results summarized in Table 9, it is worthwhile noting that the iPSC-860 computer used for this application has only 8 Mbytes of memory per processor. The smallest number of processors on this machine that is a power of two and can need the storage requirements of this three-dimensional dynamics application is 64. From Table 8, it follows that 32 to 39% of the nodes of a 64-subdomain mesh partition of the nozzle mesh are interface nodes. Hence, the hardware configuration of this iPSC-860 and the memory requirements of the nozzle dynamics problem are such that the computational and communication requirements of this target parallel application are not particularly well balanced. This is reflected in the performance results summarized in Table 9 which show that 38 to 41% of the total CPU time is spent in communication. To some extent, this situation is typical of three-dimensional finite element problems solved on small memory massively parallel processors. In Table 9, it is shown that RSB improves the communication time over GR by a factor equal to 1.09, and the two-step partitioning methodology with contraction improves the communication time over GR by a factor equal to 1.15. These factors are consistent with those describing the reduction of the number of interface nodes. However, for the enhanced mesh partitions, the total CPU time is only 7 to 8% better than that corresponding to the GR partition, which is also consistent with the distribution of the total simulation time between computation and communication. In summary, minimizing the number of interface nodes of a mesh partition does improve the total CPU time of this class of parallel applications, but not by impressive factors. Stated differently, unless communication costs are overwhelming—in which case parallel processing is not necessarily attractive—any reasonable mesh partition is suitable for this type of parallel applications. This fact is rarely recognized in the parallel processing literature.

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Optimizer</th>
<th>Contraction</th>
<th>$N_{min,k}$</th>
<th>$N_{average,k}$</th>
<th>$N_{max,k}$</th>
<th>$N_i$</th>
<th>$N_{GR}/N_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSB</td>
<td>None</td>
<td>No</td>
<td>116</td>
<td>185</td>
<td>272</td>
<td>5401</td>
<td>1.12</td>
</tr>
<tr>
<td>GR</td>
<td>None</td>
<td>No</td>
<td>129</td>
<td>212</td>
<td>316</td>
<td>6068</td>
<td>1.00</td>
</tr>
<tr>
<td>GR</td>
<td>TS</td>
<td>No</td>
<td>134</td>
<td>191</td>
<td>259</td>
<td>5494</td>
<td>1.10</td>
</tr>
<tr>
<td>GR</td>
<td>TS</td>
<td>Yes</td>
<td>120</td>
<td>177</td>
<td>220</td>
<td>5079</td>
<td>1.19</td>
</tr>
</tbody>
</table>

Table 9

Explicit central difference
Nozzle mesh: $N_i = 12 800 - N_e = 15 579 - N_g = 46701 - N_i = 64$

<table>
<thead>
<tr>
<th>Solution time for 2 000 time-steps on an iPSC-860/64</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scheme</td>
</tr>
<tr>
<td>--------</td>
</tr>
<tr>
<td>RSB</td>
</tr>
<tr>
<td>GR</td>
</tr>
<tr>
<td>GR</td>
</tr>
<tr>
<td>GR</td>
</tr>
</tbody>
</table>
5.2. Domain decomposition based iterative solvers

Here, we focus on the solution of the system of equations arising from the finite element static analysis of an elastic bearing under a distributed surface load. The finite element model of this three-dimensional structure contains 9600 8-node brick elements and 33,075 degrees of freedom (Fig. 7). The optimal domain decomposition based FETI iterative solver [21, 22] is used for parallel computations on a 64-processor iPSC-860 system. Three 64-subdomain mesh partitions are generated using RSB, GR, and the two-step mesh partitioning method with $C = 0.5 \times C_2 + 0.5 \times C_3$, $N^k = N_0^k$ and $N^{new,k} = N_0^c / N_p$. The characteristics of these mesh partitions and the corresponding performance results of the FETI solver are reported in Table 10 where $AR$ and $N_{it}$ denote, respectively, the average subdomain aspect ratio and the number of FETI iterations for convergence. For this application, it is clear that the size of the interface problem does not control neither the communication time nor the total CPU time of the domain decomposition solver. In particular, note that for the above problem there is no correlation between $N_i$ and the communication costs per FETI iteration. This is essentially because the communication costs of this application are dominated by those associated with global dot products and some other full matrix linear algebra on a coarse grid problem. On the other hand, the results reported in Table 10 clearly demonstrate the importance of the subdomain aspect ratio for this

Fig. 7. Finite element discretization of an elastic bearing.
Table 10
Optimal FETI solver
Bearing mesh: \( N_1 = 9600 \) \(-\) \( N_4 = 33075 \) \(-\) \( N = 64 \)
Effect of the subdomain aspect ratio

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Optimizer</th>
<th>Contraction</th>
<th>( N_1 )</th>
<th>( AR )</th>
<th>( T_{comm}/N_{itr} )</th>
<th>( N_{itr} )</th>
<th>( T_{sol} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSB</td>
<td>None</td>
<td>No</td>
<td>5426</td>
<td>0.50</td>
<td>0.37</td>
<td>45</td>
<td>36.09</td>
</tr>
<tr>
<td>GR</td>
<td>None</td>
<td>No</td>
<td>5032</td>
<td>0.32</td>
<td>0.37</td>
<td>43</td>
<td>35.27</td>
</tr>
<tr>
<td>GR</td>
<td>SA</td>
<td>Yes</td>
<td>4430</td>
<td>0.84</td>
<td>0.40</td>
<td>30</td>
<td>25.77</td>
</tr>
</tbody>
</table>

class of applications. The two-step mesh decomposition method with contraction improves the subdomain aspect ratio of the mesh partitions generated by GR and RSB by a factor equal to 1.7, which reduces the number of FETI iterations by a factor equal to 1.5, and the total solution time by a factor equal to 1.4.

5.3. Parallel frontal solvers

The problem of computing the steady-state flow of an incompressible Oldroyd fluid in a two-cam mixing apparatus arises in polymer processing applications. This problem is governed by a set of mixed elliptic/hyperbolic nonlinear partial differential equations [31]. Here, we consider such a problem and the flow domain depicted in Fig. 8. Its finite element discretization contains 1217 elements only, but generates 26082 equations. At each Newton iteration, these equations are solved with the frontal direct solver described in [19].

Among all parallel applications, the frontal direct solver is perhaps the most challenging one for mesh partitioning. Ideally, this algorithm requires a mesh partition where: (a) each subdomain frontwidth is smaller or equal to the frontwidth of the global problem, (b) the computational load is perfectly balanced, and (c) the subdomain interfaces have a minimum and equal number of nodes. Criterion (a) should be emphasized, because trading computational efficiency for parallelism is not always a winning strategy. Enforcing criterion (b) is a seriously difficult task, because the computational load per subdomain cannot be derived a priori from the computational complexity of the global problem. Criterion (c) attempts at minimizing the communication and storage requirements associated with the elimination of the interface unknowns.

Fig. 8. Discretization of the flow domain in a two-cam mixing apparatus.
Table 11

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Opt.</th>
<th>(N_i)</th>
<th>(\text{FR}^{\text{average}})</th>
<th>EFRLBF</th>
<th>(T^{\text{average}}<em>{\text{internal}} / T^{\text{internal}}</em>{\text{max}})</th>
<th>(T_{\text{sol}}(\text{s}))</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSB</td>
<td>None</td>
<td>97</td>
<td>282.87</td>
<td>0.53</td>
<td>0.60</td>
<td>135.96</td>
</tr>
<tr>
<td>RSB</td>
<td>SA</td>
<td>88</td>
<td>209.38</td>
<td>0.83</td>
<td>0.85</td>
<td>80.01</td>
</tr>
<tr>
<td>GR</td>
<td>None</td>
<td>139</td>
<td>393.75</td>
<td>0.47</td>
<td>0.63</td>
<td>230.53</td>
</tr>
<tr>
<td>GR</td>
<td>SA</td>
<td>85</td>
<td>252.50</td>
<td>0.67</td>
<td>0.66</td>
<td>102.48</td>
</tr>
</tbody>
</table>

Here, four 8-subdomain mesh partitions are generated for parallel computations on an 8-processor Convex Meta Series system, using GR, RSB, and the two-step mesh partitioning method with both GR and RSB as initial decomposers. For this application, the cost function to be optimized is set to \(0.5 \times C_1 + 0.5 \times C_2\). However, note that in this case \(N^{\text{best},k}\) cannot be determined a priori. Let \(\text{FR}^{k}\) and \(\text{FR}^{\text{max},k}\) denote, respectively, the variable subdomain frontwidth and its maximum value. During the optimization (or retrofitting) process, \(N^{\text{best},k}\) is computed so that \(N^{\text{best},k} \times \text{FR}^{\text{max},k}\) is the same in all subdomains.

The characteristics of all four mesh partitions and the corresponding performances of the parallel frontal solver are reported in Table 11 where EFRLBF = \(\text{average}_k(N^k \times \text{FR}^k) / \text{max}_k(N^k \times \text{FR}^k)\) is the estimated computational load balance factor, \(T^{\text{internal}}_{\text{internal}}\) is the parallel CPU time associated with the elimination of the subdomain internal unknowns, and \(T_{\text{sol}}\) is the total parallel solution time. An internal renumbering scheme [32] is used in every subdomain for minimizing fill-in. The ability of EFRLBF to predict the computational load balance of the parallel frontal solver is well illustrated in Table 11. Also, the suitability of the selected cost function and the effectiveness of the optimization algorithm are well demonstrated. For example, the run-time load balance factor for the RSB mesh partition is equal to 0.53, while that of the optimized RSB partition is equal to 0.83. The net result of the optimization process is a speedup factor in the solution time equal to 1.69. For the GR partition, the net result of the retrofitting step is a speedup factor equal to 2.25. Note also that for the above problem, the mesh partition that leads to the fastest parallel solution of the linearized equations is neither the one with the minimum number of interface nodes, nor that with the minimum subdomain frontwidth, but the mesh partition with the best predicted load balance factor—and it also turns out to be the mesh partition with the best run-time load balance factor.

6. Adaptive refinement

When an application involves some form of h, or p, or h/p adaptive refinement, it may become necessary to continuously repartition the refined mesh because:

- a previously accomplished load balance is usually destroyed after selective refinement.
- local memory capacities can be exceeded if the refinement is strongly localized.

Rather than repartitioning the mesh at every refinement level, it would seem natural to apply the retrofit procedure described in this paper as follows: the mesh partition generated at level \(n - 1\) is treated as an initial decomposition for level \(n\), a cost function \(C\)—usually representing a load balance issue—is formulated, and the interfaces of the initial decomposition are readjusted to optimize \(C\).

As an example, we consider the plane stress analysis of the L-shaped domain depicted in Fig. 9. The initial discretization of this domain includes 2250 triangular elements (Fig. 9(a)). Its decomposition in 32 subdomains using the GR algorithm and the SA optimization scheme is depicted in Fig. 9(b).

After one step of adaptive refinement dictated by the error criterion described in [33], the finite element mesh contains 3671 triangular elements (Fig. 10(a)). The retrofitting of the initial decomposition using the SA optimization scheme consumes 2.55 s CPU on a Silicon Graphics workstation, and produces 422 interface nodes (Fig. 10(b)). On the other hand, the two-step partitioning methodology
applied directly to the 3671 element mesh consumes 2.32 s CPU on the same workstation and generates 398 interface nodes. This disappointing result can be explained as follows. The refined mesh has 63% more elements than the original one, and refinement is essentially confined to two regions. Consequently, the partition generated for the original mesh is not an excellent initial decomposition for the refined one because it is seriously unbalanced. Moreover, we note that during the optimization step, 2016 elements are exchanged between the subdomains when the partition of the original mesh is used as an initial decomposition for the refined one, while 545 elements only are exchanged when the refined mesh is re-decomposed and optimized.

Hence, it seems that the retrofitting approach is interesting for adaptive refinement only when a small percentage of the elements are refined, and more or less across all the previously generated subdomains. Unfortunately, this is seldom the case in practice. However, considering the speed and efficiency of the two-step partitioning methodology with contraction presented in this paper, we do not believe that mesh decomposition is the main challenge for parallel adaptive refinement. Our experience
is that the reshuffling of a distributed data database across the given processors and the readjustment of the data structures describing the subdomain interconnectivity, both induced by adaptive refinement, are by far more challenging issues for parallel processing.

7. Conclusion

In this paper, we have presented a two-step methodology for the fast generation of large-scale mesh partitions. Its ingredients are: (a) a fast initial decomposer, (b) an efficiency contraction scheme, and (c) an economical optimization procedure. We have shown that this methodology can produce better mesh partitions than the well celebrated multilevel Recursive Spectral Bisection (RSB) algorithm, and yet be an order of magnitude faster. If needed, the multilevel RSB scheme can be incorporated in this methodology as an initial decomposer. We have demonstrated the benefits of our approach to several parallel applications running on an iPSC-860 massively parallel processor and a Convex Meta Series systems, including an explicit time-integration scheme, a frontal sparse solver, and a domain decomposition based iterative algorithm. More importantly, we have shown that the over-emphasized minimum interface size criterion does not have a significant impact on a reasonably well parallelized application, and highlighted several other factors which can have a significant impact. These include the subdomain aspect ratio for non-overlapping domain decomposition based iterative methods, and load balance for applications where it cannot be determined a priori, as it is the case for parallel frontal solvers. Finally, we stress that the two most important conclusions of this work are: (1) the quality of a mesh partition cannot be assessed without considering the target parallel application, and (2) communication costs are not always the biggest obstacle for efficient parallel computing. From the first conclusion, it follows that the two-step partitioning methodology described herein will always be superior to a conventional one-step approach that attempts exclusively at minimizing the interface size.

Acknowledgments

The US authors acknowledge partial support by the National Science Foundation under Grant ASC-9217394. The Belgian authors acknowledge partial support by the European Commission (BRITÉ/EURAM programme), the Belgian Services Fédéraux des Affaires Scientifiques Techniques et Culturelles, and the Fonds National de la Recherche Scientifique.

References