Multi-Constraint Mesh Partitioning for Contact/Impact Computations*

George Karypis

karypis@cs.umn.edu

Department of Computer Science & Engineering/Army HPC Research Center
University of Minnesota
Technical Report 03-022

Last updated on May 5, 2003 at 11:14am

Abstract

We present a novel approach for decomposing contact/impact computations in which the mesh elements come in contact with each other during the course of the simulation. Effective decomposition of these computations poses a number of challenges as it needs to both balance the computations and minimize the amount of communication that is performed during the finite element and the contact search phase. Our approach achieves the first goal by partitioning the underlying mesh such that it simultaneously balances both the work that is performed during the finite element phase and that performed during contact search phase, while producing subdomains whose boundaries consist of piecewise axes-parallel lines or planes. The second goal is achieved by using a decision tree to decompose the space into rectangular or box-shaped regions that contain contact points from a single partition. Our experimental evaluation on a sequence of 100 meshes, shows that this new approach can significantly reduce the communication overhead over existing algorithms.

1 Introduction

In order for mesh-based scientific simulations to be effectively executed on a wide variety of parallel architectures we need to distribute the underlying finite element meshes among the processors so that (i) the computations are balanced, (ii) the interprocessor communication is minimized, and (iii) the cost of redistributing the mesh (in the context of adaptive mesh computations) is minimized. It has been recognized in recent years that this can be effectively achieved by using graph-partitioning and repartitioning algorithms [28, 9, 8, 34], and a new class of partitioning algorithms has been developed based on the multilevel paradigm [3, 11, 17, 19, 25, 35] that produce high-quality partitionings, are very fast, and can scale to graphs containing several millions of vertices [10, 19, 15, 17]. Moreover, efficient parallel formulations of these algorithms have been developed for distributed-memory parallel computers [22, 33, 18, 20], which are capable of scaling to thousands of processors and partition meshes with billions of elements.

*This work was supported by NSF ACI-0133464, CCR-9972519, EIA-9986042, ACI-9982274, and by the Army High-Performance Computing Research Center under contract number DAAD19-01-2-0014.

In its simplest form, the graph-partitioning problem focuses on computing a k-way partition of a graph such that the edge-cut is minimized and each partition has an equal number of vertices (or in the case of weighted graphs, the sum of the vertex-weights in each partition are the same). The task of minimizing the edge-cut can be considered as the *objective* and the requirement that the partitions will be of the same size can be considered as the *constraint*. In addition, more general *multi-constraint* [16] and *multi-objective* [31] instances of the graph-partitioning problem have also been developed that can compute partitionings that simultaneously balance multiple weights associated with the vertices while minimizing multiple objectives associated with the edges. Single-constraint and single-objective graph partitioning is used to distribute static mesh-based parallel scientific simulations, whereas multi-constraint, multi-objective graph partitioning is used to distribute (adaptive) multi-phase and multi-physics simulations [9, 8, 34].

Despite the success of multilevel partitioning algorithms and the recent advances in multi-constraint and multi-objective partitioning problem formulations, there are still a number of important scientific problems that cannot be effectively parallelized by these algorithms. One such example are the scientific simulations in which the mesh elements come in contact with each other and are routinely performed in the context of simulations that study vehicle crashes, material deformations, and projectile-target penetration. Most of the existing multilevel partitioning algorithms cannot effectively decompose these types of simulations as they ignore the underlying geometry and produce subdomains that incur high-communication overheads during the contact-search phase of the computation. For this reason, an alternative approach has been developed that computes a different, geometry-aware decomposition for the contact-search phase [27, 2]. Even-though, this approach is effective in reducing the communication overhead associated with contact-search, it requires a rather expensive, all-to-all personalized communication step in order to transfer information between the two decompositions.

In this paper we present a new approach for partitioning and performing contact/impact computations on parallel computers that reduces the communication overheads of existing approaches while ensuring that the overall computation remains well-balanced. Our algorithm computes a multi-constraint partitioning that leads to subdomains whose boundaries consist of piecewise axes-parallel lines or planes, and uses a binary tree to partition the space covered by the mesh into disjoint axes-parallel rectangles or boxes whose leafs contain surface nodes from a single subdomain. Our experimental evaluation on a real simulation consisting of a sequence of 100 meshes shows that the resulting algorithm leads to a substantial reduction in the overall amount of data that needs to be communicated.

The rest of this paper is organized as follows. Section 2 provide some definitions and background information on graphs, graph partitioning, and contact/impact computations. Section 3 surveys some related research on partitioning contact/impact computations. Section 4 provides an overview of the partitioning approach that we developed and describes the algorithms used for its different phases. Section 5 provides an experimental evaluation of the resulting algorithm and compares it against previously developed schemes. Finally, Section 6 offers some concluding remarks and describes directions along which our approach can be further improved.

2 Background Information

2.1 Definitions

Given a weighted undirected graph $G = (V, E)$ a decomposition of V into k disjoint subsets $V_1, V_2, \ldots, V_k$, such that $\bigcup_i V_i = V$ is called a *k-way partitioning* of V. We will use the terms *subdomain* or *partition* to refer to each one of these k sets. A k-way partitioning of V is denoted by a vector P such that $P[i]$ indicates the partition number that vertex i belongs to. A partitioning is said to *cut* and edge e, if its incident vertices belong to different partitions. The *edge-cut* of a partitioning P, denoted by EdgeCut(P) is equal to the sum of the weights of the edges that are cut by the partitioning. The *partition weight* of the ith partition, denoted by $w(V_i)$ is equal to the sum of the weights of the vertices assigned to V_i. The *total vertex weight* of a graph, denoted by $w(V)$ is equal to the sum of the weights of all the vertices in the graph. The *load-imbalance* of a k-way partitioning P, denoted by LoadImbalance(P) is the ratio of the highest partition weight over the average partition weight, *i.e.*, $\max_i(w(V_i))/(w(V)/k)$.

The notions of partition weight, total vertex weight, and load-imbalance can be extended to cases in which each

vertex v is assigned a vector of m weights, that is, $w(v) = (w_1(v), w_2(v), \ldots, w_m(v))$. Specifically, the partition weight of the ith partition with respect to the jth component of the vertex-weight vectors, denoted by $w_j(V_i)$ is equal to $\sum_{v \in V_i} w_j(v)$. The total vertex weight of a graph with respect to the jth component of the vertex-weight vectors, denoted by $w_j(V)$ is equal $\sum_{v \in V} w_j(v)$. The load-imbalance of a k-way partitioning P with respect to the jth component of the vertex-weight vectors, denoted by LoadImbalance(P, j), is $\max_i(w_j(V_i))/(w_j(V)/k)$.

Given a finite-element mesh, its corresponding *nodal graph* is obtained by representing each node of the mesh via a vertex and connecting two vertices via an edge, if there is a corresponding edge in the mesh. Similarly, its corresponding *dual graph* is obtained by representing each element of the mesh by a vertex and connecting two vertices together if their corresponding elements share an edge (in 2D) or a face (in 3D).

2.2 Overview of Different Graph Partitioning Problems

Three distinct graph-partitioning problem formulations have been used to map mesh-based computations onto the processors of a parallel computer. These are the *static graph partitioning*, the *graph repartitioning*, and the *multi-constraint, multi-objective graph-partitioning* [9, 8, 34].

The input to the static graph-partitioning algorithms is a weighted undirected graph $G = (V, E)$. The weight on the vertices correspond to the (relative) amount of computation required by the corresponding mesh node/element, whereas the weight on the edge corresponds to the (relative) amount of data (or communication time) that needs to be exchanged in order for the computation at each mesh node/element to proceed. The goal of the static graph-partitioning problem is to compute a k-way partitioning P, such that for a small positive number ϵ, LoadImbalance(P) $\leq 1 + \epsilon$ and EdgeCut(P) is minimized. Static graph partitioning is used to map traditional static single-phase simulations onto the processors of a parallel computer.

The input to the graph repartitioning algorithms is similar to that for static graph-partitioning algorithms, but as their name suggest, there already exists an initial partitioning of the graph P_0. However, this partitioning may not be balanced or it may have a very high edge-cut. The goal of the graph repartitioning problem is to compute a k-way partitioning P such that LoadImbalance(P) $\leq 1 + \epsilon$, EdgeCut(P) is minimized, and the overlap between the old and the new partitioning is maximized, *i.e.*, maximize the number of vertices v for which $P[v] = P_0[v]$. The second objective of the graph repartitioning problem is to ensure that adaptive computations do not spend a prohibitively large amount of time in redistributing the data in order to adhere to the new partitioning.

The input to the multi-constraint multi-objective graph-partitioning algorithms is a graph whose vertices and edges have a vector of weights associated with them. That is, each vertex v has a weight-vector $w(v)$ of size q, *i.e.*, $w(v) = (w_1(v), w_2(v), \ldots, w_q(v))$, and each edge e has a weight-vector $w(e)$ of size r, *i.e.*, $w(e) = (w_1(e), w_2(e), \ldots, w_r(e))$. The goal of the multi-constraint multi-objective graph partitioning problem is to compute a k-way partitioning of the graph P such that LoadImbalance(P, j) $\leq 1 + \epsilon$ for $j = 1, \ldots, q$, while minimizing an objective function that is defined over the r components of the edge-weight vectors of the edges that are being cut by the partitioning. This multi-constraint multi-objective partitioning problem formulation can be used to balance multi-phase and multi-physics simulations in which during each iteration the actual computation is performed in a number of phases with an explicit synchronization step between each phase, or compute partitionings that simultaneously balance the amount of computations assigned to each partition and the amount of memory that is required by the corresponding elements.

Our discussion on partitioning problem formulations has been primarily focused on edge-cut based objectives. However, other objectives such as the total communication volume [7] can also be used without affecting the algorithmic issues involved, and they can be easily incorporated in existing multilevel partitioning algorithms [15].

2.3 Overview of Contact/Impact Computations

Each iteration of contact/impact simulations is usually composed of two phases. During the first phase, traditional finite difference/element/volume methods are applied on the entire domain and in the second phase, there is a search to

determine whether or not the surface elements of the mesh have come in contact and penetrated each other. Once such contacts have been determined, the positions of the affected mesh elements are corrected, the elements are deformed, and the overall simulation progresses to the next iteration.

The actual contact detection is usually performed in two steps [36, 6, 26, 4, 5]. In the first step, the pairs of surface-elements that are sufficiently close to each other so that they can potentially be in contact are determined, whereas in the second step, the exact locations of the contacts/penetrations (if any) between these candidate contacting surfaces are determined. These two steps are often called *global search* and *local search*. A number of different algorithms have been developed for local search and are in use in different production codes. In this paper we only focus on the global search phase as it is critical for ensuring the overall parallel scalability of these methods. Note that the exact details of the local search phase do not affect the approach used to perform the global search.

For the discussion in the rest of this paper we will use the terms *surface elements* or *contact elements* to refer to the set of elements that need to be searched for contacts and we assume that these elements have been identified as such by the application. We will use the terms *contact nodes*, *contact points*, or *surface nodes* to refer to the set of mesh nodes that belong to surface elements. We will assume that during contact search we are only interested in identifying contacts between surface elements and not contacts between surface and non-surface elements.

3 Related Research

A number of different approaches have been developed for partitioning contact/impact computations. These approaches can be broadly categorized into two groups based on the type of problem that they are solving. The first group contains methods that are designed to address problem instances in which the portions of the meshes that will end-up getting in contact with each other are known a priori (or can be accurately estimated), whereas the second set of approaches are designed to handle cases in which no a priori knowledge about contacting surfaces is known. The second class of contact/impact problems are more general and are the type of problem instances that this paper is focusing on.

Most of the methods for the first type of contact problems, partition the underlying mesh such that the portions of the mesh that contain the to-be-contacting surfaces are assigned to the same (or a small number of) processors. This partitioning is usually done by using a graph to model the mesh and the sets of contacting surfaces by creating additional edges between the surface elements that can potentially come in contact. Using such a graph model, then the desired partitioning is obtained by using a traditional two-constraint graph partitioning algorithm that balances both the mesh- and the surface elements assigned to each processor. Since the resulting partitioning minimizes the edge-cut, such an approach tends to place contacting surface elements on the same processor [12].

Probably, the most efficient method to deal with the second class of contact problems is that developed by [27, 2]. This approach distributes the overall computations by performing two different partitionings. In the first partitioning, a traditional multilevel graph partitioning algorithm is used to evenly distribute the entire mesh, whereas in the second partitioning, a recursive coordinate bisection (RCB) algorithm is used to evenly distribute only the surface elements. Using these two partitionings, this approach ensures that the overall computation will be balanced. Moreover, since the surface-elements are partitioned using a geometric algorithm, the communication overhead during the contact-search phase of the algorithm is reduced, as it is proportional to the number of elements along the partition boundaries. Also, since during the course of the simulation, the positions of the contact nodes changes, the above algorithm recomputes the RCB-based partitioning of the contact points at each iteration and moves the contact points accordingly. To minimize the cost of this redistribution overhead, these follow-up partitionings are computed by modifying the previous RCB partitioning along the same spirit of the graph repartitioning algorithms described in Section 2.2. In the remainder of this paper, we will refer to the above method as the *ML+RCB* algorithm.

The key to the effectiveness of the ML+RCB algorithm is the fact that it uses the best possible partitioning for each one of the two phases. However, because these two partitionings are de-coupled, the same surface node can reside at two different processors. That is, the processors responsible for performing the finite element-based computations and

the contact-detection-based computations of a particular surface node can be different. As a result, prior to each one of the two computational phases, an all-to-all communication operation is required to send the updated information for each surface node between the two partitionings. Depending on the relative size of the surface mesh to the rest of the subdomain, such a scheme may incur a high communication overhead.

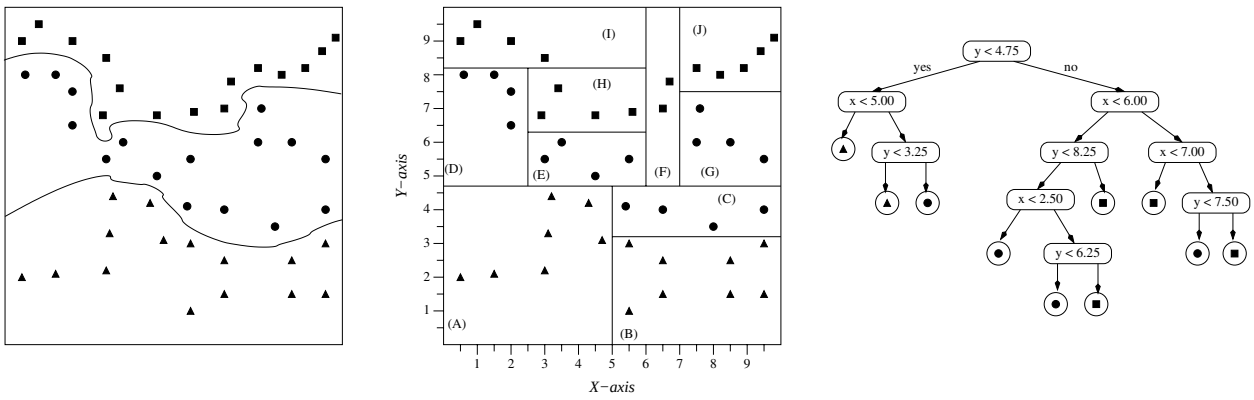
4 Partitioning for Contact/Impact Computations

From the nature of the computations performed in the course of numerical simulations involving contact and impact (described in Section 2.3), we can see that their overall structure is similar to a two-phase computation, in which the first phase involves the entire domain and the second phase involves only the surface elements and their nodes. Consequently, if we model this mesh as a graph with two vertex weights—the first weight modeling the computations performed during the first phase and the second weight modeling the computations performed during the second phase—then the existing multi-constraint partitioning algorithms described in Section 2.2 can be used to load-balance the overall computation. This is the key idea behind our approach for effectively parallelizing contact/impact computations and as such, it eliminates the need to transfer nodal information between the two different partitionings, as is required by ML+RCB.

However, even though this multi-constraint based approach will achieve the desired load balance, care must be taken to ensure that the overall approach does not lead to excessive communication overheads during the global search phase of contact detection. On serial computers, global search is done efficiently by representing each contact surface by its bounding box and using various volume partitioning (or spatial indexing) techniques to quickly narrow down the search space. In the case of a parallel system and a partitioned mesh, the global search is usually accelerated by using a similar technique as follows [23, 24, 2]. Each subdomain is represented by its bounding box and every processor receives a copy of all the bounding boxes of the various subdomains. Then for each surface element that a processor stores, it determines the subdomain bounding-boxes that it intersects with or is fully contained in it, and sends the elements to all of these processors. Finally, each processor proceeds to perform global search using the surface elements that it stores and the surface elements that it received in the previous step. Essentially this approach uses the bounding boxes of the subdomains as a *filter* to determine whether or not a particular surface element can come in contact with the elements that are stored at another subdomain. The overall number of elements that need to be communicated is proportional to the number of surface elements that are at the intersection of the various subdomain bounding-boxes. Since the mesh is partitioned using a two-constraint partitioning algorithm that does not take into account the underlying geometry, there are no guarantees on the degree of overlap of the various subdomains, which can lead to excessive communication. Also note that most of the excessive communication overhead are due to *false-positives*, *i.e.*, a particular surface element is sent to a processor, even though none of the locally stored elements of that processor will identify it as a "hit" for its global search phase.

In principle there are two ways that can be used to address the above problem. First, we can develop multi-constraint graph partitioning algorithms that take into account the underlying geometry and lead to subdomains whose corresponding bounding boxes have as small of an overlap as possible. Second, we can develop better parallel global search algorithms that reduce the number of false-positives. This can be done by using different geometric descriptors for the area/volume that is covered by the elements assigned to a particular subdomain than just its bounding box. In particular, if the area/volume that is covered by these descriptors asymptotically approaches the area/volume of the actual subdomain, then the overlap between the different subdomains will be asymptotically reduced to zero, and thus minimize the number of false-positives.

Our algorithm uses both of these approaches to reduce the amount of communication that is required during the global search phase of contact detection. It computes a multi-constraint partitioning that leads to subdomains whose boundaries consist of piece-wise axes-parallel lines or planes, and uses a binary tree to partition the space covered by the entire mesh into disjoint axes parallel rectangles or boxes whose leafs contain surface elements from a single subdomain. Details on how the geometric descriptors are constructed, how the multi-constraint partitioning is computed,



(a) Partitioning of the contact points.

(b) Partitioning of the space.

(c) Associated Decision Tree

Figure 1: An example three-way partitioning of 45 contact points. (a) Shows the partitioning of the 45 contact points into three partitions. (b) Shows the description of the various subdomains as a set of axes-parallel rectangles. (c) Shows the underlying decision tree description of these descriptors.

and various issues involved with incrementally updating this information are provided in the rest of this section.

Note that our partitioning algorithm operates on the nodal graph of the mesh and for this reason our discussion throughout this section will focus on the problem of partitioning the nodal graph of the mesh. However, the actual global search is done with respect to the surface elements of the mesh and details on how the nodal partitioning is used to perform that search is also provided.

4.1 Subdomain Geometric Descriptors

The best way to describe the method that we use to represent the area/volume occupied by each subdomain is via an example. Figure 1(a) shows such a hypothetical example of a 2D problem in which 45 contact points have been partitioned into three subdomains. The subdomain that each point belongs to is represented by the shape of each point (*i.e.*, triangle, circle, and square). Given this partitioning, our algorithm then proceeds to partition the underline space into rectangular regions as depicted in Figure 1(b). The property of these regions is that each one of them contains points from only a single partition. Given such a space partitioning, then for contact search purposes, we assume that each subdomain occupies the area consisting of the rectangles containing its points. For example, in the case of the "triangle" subdomain, its geometric description will consist of the (A) and (B) rectangles, whereas for the "square" subdomain, its corresponding description contains rectangles (F), (H), (I), and (J).

Note that this space partitioning is not arbitrary but is obtained by performing a sequence of recursive bisections along either the x or the y axis. This sequence is depicted in Figure 1(c) using a binary tree. The interior nodes of this tree represent bisections along a particular point of the x or y axis and the leaf nodes represent rectangles that contain only points from a single partition. For example, the root corresponds to the bisection that is performed along the y-axis at point 4.75. This tree can be also used to locate the rectangles that a particular point (x_i, y_i) belongs to by starting from the root and going to the left or to the right, depending on whether or not x_i or y_i satisfies the test that is depicted at that node. In particular, if the point satisfies the test, then it will traverse the left subtree (*i.e.*, the *yes* branch), otherwise it will traverse the right subtree (*i.e.*, the *no* branch). These types of trees are extensively used in the machine learning community, and they are called *decision trees*; whereas, the splitting points at each node are called the *decision hyperplanes*.

The resulting decision tree can also be used to perform the global search and identify the partitions that contain contact points with which each surface element can potentially come in contact with. In that case, the input is the surface element's geometry (or a bounding-box approximation of it), and the global search algorithm will start from the root of the tree and traverse either the left, right, or both branches depending on whether or not the surface element is on the left, right, or intersects the decision hyperplane. In general, the average complexity of each search will be

proportional to the average height of the tree. Since each subdomain will be in general described by more than one leaf node, the complexity of the search will tend to be somewhat higher than the complexity of the search required by ML+RCB. However, since for each surface element, the decision tree based approach identifies a subset of the contact points stored by a processor that it can come in contact with, this information can be used to speedup the follow up contact search that is performed by each processor. Thus, it does not significantly affect the overall complexity of contact search.

Note that the above space partition only focuses on the contact points and it entirely ignores the non-contact points. As a result, depending on the complexity of the underlying geometry, contact surfaces, and partitioning, each of these rectangular boxes may actually contain non-contact points from multiple partitions. However, since the contact search involves only surface elements and contact points, this does not create any correctness or completeness problems.

4.1.1 Constructing the Decision Tree

Our discussion so far has focused on the properties of the space partitioning obtained by the decision trees but we have not yet described how the decision tree is obtained in the first place. Specifically, the problem that we need to address is the following. Given a k-way partitioning of a set of points in either 2D or 3D, construct a decision tree that partitions the space into rectangles or boxes each of which contain points from only one partition. Moreover, since the decision tree needs to be built in parallel and communicated to all the processors (in order to perform the contact search), reducing the overall number of nodes in the resulting tree is an additional objective that needs to be taken into account while building the tree.

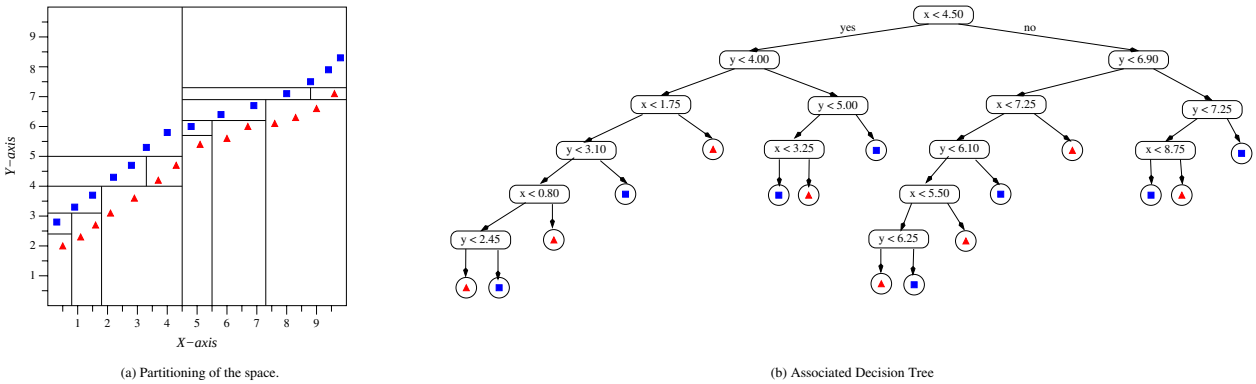
Fortunately, the problem of building a decision tree that has the above characteristics has been extensively studied by the machine learning community (under the name of *tree induction*) and a number of different heuristic algorithms have been developed [1, 29]. For our problem, we decided to use the C4.5 algorithm [30] as it leads to small trees, is computationally efficient, and as part of our earlier work, have developed efficient and scalable parallel formulations for it [14].

Given a set of points A, each belonging to one of the k partitions, the C4.5 algorithm identifies the hyperplane that bisects the points into two sets A_1 and A_2 such that both A_1 and A_2 are as *pure* as possible. A set is considered to be *completely pure*, if it contains only points from one of the k partitions. Since A can contain points from up to k partitions, it may be impossible to find a bisecting hyperplane that results in both A_1 and A_2 being pure. In such cases, C4.5 tries to find a hyperplane such that the points of each one of the k partitions is assigned primarily to either A_1 or A_2. This is usually done by computing a *splitting index* that measures the purity of the resulting bisection, and selecting the hyperplane that maximizes the value of that index. For our purposes, we used a modified version of the gini index [1] given by

$$\textit{splitting index} = \sqrt{\sum_{i=1}^{k} |A_{1,i}|^2} + \sqrt{\sum_{i=1}^{k} |A_{2,i}|^2}, \tag{1}$$

where $A_{1,i}$ and $A_{2,i}$ are the number of points of partition i that were assigned to sets A_1 and A_2, respectively. In the simple case in which $k = 2$, and A contains an equal number of points from the two partitions, the above function achieves its maximum value when each set is assigned points from a single partition. Analyzing the properties of this function for a general distribution is beyond the scope of this paper, and the reader should refer to [1, 30].

Once the best splitting hyperplane has been identified, the original set is partitioned into A_1 and A_2, and each set that contains points from more than one partition is further split by applying the same procedure recursively. The reader should refer to [30] for further details. The computational complexity for splitting a set A is linear on the size of the set, assuming that the points in A have been sorted along each one of their dimensions. This is because the algorithm needs to try only each hyperplane that passes between successive points along each of the dimensions (a total of $3|A|$ possible points), and at each successive point, Equation 1 can be computed incrementally in $O(1)$ time. Also, the required sorting can be done once for the the entire set and maintained by properly splitting the various sets during the bisection steps.



(a) Partitioning of the space.

(b) Associated Decision Tree

Figure 2: An example two-way partitioning of 28 contact points. (a) Shows the description of the various subdomains as a set of axes parallel rectangles. (b) Shows the underlying decision tree description of these descriptors.

4.2 Computing a Contact-Friendly Multi-Constraint Partitioning

We partition the mesh using a multi-constraint graph partitioning algorithm [16] whose goal is to decompose the mesh among the k processors such that it balances the computations that are performed during each one of the two phases. Specifically, let $G = (V, E)$ be the nodal graph of the underlying mesh. Let $w(v) = (w_1(v), w_2(v))$ be a two-element vector assigned to each vertex v, such that $w_1(v)$ is set equal to a value that reflects the (relative) amount of computation that it performs during the first phase of the calculation, and $w_2(v)$ is set to a value that reflects the (relative) amount of computation that it performs during the contact search phase. In general, $w_2(v)$ will be non-zero when v corresponds to a contact point, otherwise it will be equal to zero. Note that our above weight definitions reflect the most general case in which the computations associated with the different nodes are different. In cases in which this is not true, then all of the non-zero values will be set to one.

The above graph model can be further improved by adding weights to the different edges. The are two inherently different types of edges in the above graph. These are (i) the edges whose incident vertices correspond to contact points, and (ii) the remaining edges. If a partition cuts an edge of the first type, then it will lead to a communication step both during the first phase and (in most cases) during the second phase of the computation. On the other hand, cutting edges of the second type will in general lead to communication only during the first phase. Thus, we can reduce the overall communication cost by assigning a higher weight to the edges between contact points.

Using this graph G, we can then use the multi-constraint partitioning algorithm [16, 32] to obtain the k-way partitioning P that balances the two computational phases. In principle, this partitioning P can then be given directly to the decision tree induction algorithm described in Section 4.1.1 to build the necessary geometric descriptors for contact search. However, depending on the geometry of the boundaries of the various subdomains, the decision tree induced consistent with P may have a large number of nodes. For instance, consider the very simple case of a two-way partitioning of a 2D mesh in which the contact points of the two subdomains are along a diagonal line, as illustrated in Figure 2(a). Then, as shown in Figure 2(b), the resulting decision tree will need to compute a fine-grain space partitioning in order to ensure that each leaf node contains points from only one subdomain. This happens because there is a miss-match between the geometry of the subdomain boundaries that decision trees can model concisely and the geometry produced by the actual partitioning. Since decision trees partition the space by performing axes parallel bisections, they are ideal for boundaries that consist of axes-parallel lines or planes, and they are less effective the further the subdomain boundaries deviate from this model.

For this reason, we developed an algorithm that modifies the initial partitioning obtained from the traditional multi-constraint partitioning algorithm to generate a decision-tree friendly partition. We achieve that by inducing a decision tree on the entire set of vertices of the graph (*i.e.*, both those corresponding to contact points and those that do not), and then using this tree to guide the modification. The decision-tree induction algorithm that we used is similar to that described in Section 4.1.1 but instead of terminating the tree induction when it reaches a pure node, it terminates

when (i) it reaches a node that is pure and contains less than $\max_p$ points, or when (ii) it reaches a node that is impure and contains less than $\max_i$ points. The first condition forces the tree induction to continue even for pure tree-nodes when these nodes contain a large number of points, whereas the second condition terminates the tree induction when an impure tree-node has a small number of points.

Now, given this tree, our algorithm computes a new partitioning P' from P, by assigning all the points that are covered by a particular leaf node to the majority partition of these points. As a result of this policy, the points that belong to pure leaf nodes will not change partitions; however, the points that belong to impure leaf nodes may be assigned to a different partition. By construction, this vertex reassignment leads to a partitioning P' whose boundaries consist of piecewise axes-parallel lines or planes. However, P' may not necessarily satisfy the balancing constraints and as such, it is not an acceptable solution.

To correct this we perform a multi-constraint k-way partitioning refinement operation whose goal is to modify P' such that the resulting partition P'' does satisfy the constraints. However, in order to ensure that subdomain boundaries of P'' retain their nice geometric properties, we do not perform that refinement on the original graph G, but on a much smaller graph G' that is obtained by collapsing together all the vertices belonging to each leaf-node of the decision tree into a single vertex. Thus, the refinement algorithm moves between partitions these rectangular- or box-shaped regions, and as a result P'' is guaranteed to retain its nice geometric characteristics. Note that both the initial multi-constraint partitioning, the construction of G', and the multi-constraint k-way partitioning refinement can be done effectively in parallel [32], and these algorithms are already available in the latest release of ParMETIS [21].

The $\max_p$ and $\max_i$ parameters of the above scheme play an important role in determining the extent to which this scheme leads to effective solutions. Specifically, if $\max_p$ and $\max_i$ are set to very small values, then it will be relatively easy for the post-refinement step to correct any load-imbalances and lead to high-quality solutions in terms of the cut. However, the resulting subdomains may consist of a large number of regions, and as such they can still lead to large decision trees. If $\max_p$ and/or $\max_i$ is set too high, then it may be difficult for the post-refinement algorithm to balance the constraints, as G' will contain vertices with high vertex weights (*i.e.*, will correspond to a large number of vertices of G), that cannot be moved for balancing purposes. Moreover, high values of $\max_i$ may lead to an intermediate partition P' that significantly violates the balancing constraints and has a high cut—both of which will make the post-refinement task quite difficult.

Our experimental study on the sensitivity of our approach to these parameters has shown that if n is the total number of vertices in the graph and k is the number of partitions, then good choices for $\max_p$ and $\max_i$ are within the following ranges:

$$\frac{n}{k^{1.5}} \leq \max_p \leq \frac{n}{k} \quad \text{and} \quad \frac{n}{k^{2.5}} \leq \max_i \leq \frac{n}{k^2}$$

Note that the fact that $\max_i < \max_p$, does make an intuitive sense, since high values of $\max_i$ degrades the P' partitioning solution both in terms of balance and in terms of cut.

4.3 Updating the Information

A key element of contact/impact simulations is that during successive time-steps, as parts of the mesh come in contact with each other, the position of the nodes in the underlying mesh change, and depending on the actual characteristics of the numerical simulation algorithm, some existing elements disappear; thus, changing the topology of the underlying nodal graph. As a result, the partitioning of the mesh and the information used to search for contacts need to be updated periodically.

Within the context of our algorithm, there are two different ways of performing these updates. One approach will recompute a multi-constraint partitioning of the graph and set up the associated geometric subdomain descriptors during each time-step of the simulation. To ensure that there is a high degree of overlap between successive partitionings, the updated multi-constraint partitioning will be computed using a multi-constraint repartitioning algorithm [32]. The second approach will leave the multi-constraint partitioning unchanged but just use the tree-induction algorithm to compute the new geometric subdomain descriptors that take into account the new location of the contact points.

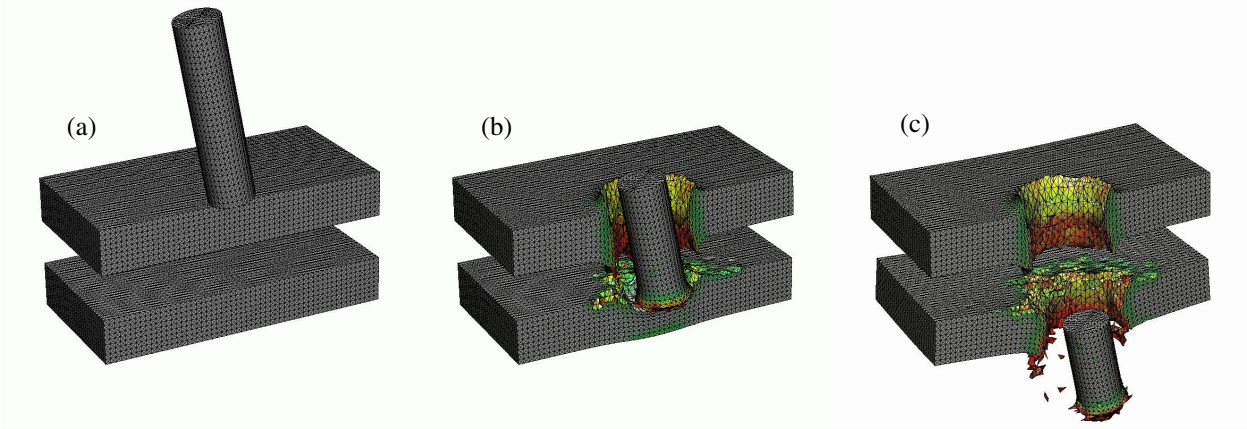


Figure 3: Various stages of the simulation.

The second approach has the advantage of being faster, as it does not perform the multi-constraint partitioning, and there is no need for dynamic data redistribution in order to adhere to the new partitioning. Moreover, as long as the underlying topology and the contact points do not change dramatically, this approach still ensures that the overall computation is load-balanced. However, its drawback is the fact that as the simulation progresses, the algorithm is required to build the decision tree on a partitioning whose subdomain boundaries are not any more piecewise axes-parallel lines or plane. As a result, the number of nodes in the induced decision tree may increase.

The above discussion suggests that a hybrid approach may be the optimal choice. That is, in the course of the simulation, the mesh will be infrequently repartitioned using the first approach (so that to ensure that the work remains load-balanced and that the geometry of each subdomain is "nice"), and between these repartitioning steps, it will be updated by simply inducing a new decision tree on the contact points.

5 Experimental Evaluation

We experimentally evaluated the performance of our multi-constraint-based partitioning algorithm for contact/impact computations on a sequence of meshes corresponding to an actual numerical simulation. The simulation corresponds to a projectile penetration through two plates, as illustrated in Figure 3. The initial mesh contains 156,601 nodes, 701,952 elements, 40,512 contact surfaces, and 20,262 contact nodes. The simulation was performed using the EPIC code [13] and required a total of 3,768 time steps to finish. Due to the relative large number of time steps, we instrumented the code to output the mesh and the associated contact surface information approximately every 37 time steps, resulting in a total of 100 successive snapshots of the mesh. We then used this sequence of meshes to evaluate our algorithm and compare it against ML+RCB. In order to simplify the presentation, we will refer to our approach for partitioning contact/impact computations as the *MCML+DT algorithm*.

MCML+DT has a number of tunable parameters that can affect its performance. However, due to space constraints, we only present the results that we obtained by using a single set of parameters that was kept constant over all 100 meshes. Specifically, the multi-constraint nodal graph was obtained by setting the various vertex weights to one (*i.e.*, we assume that all the mesh nodes and all the contact points perform the same amount of computation during the first and the second phase of the computation, respectively), and we set the weights of the edges connecting contact points to five while we kept the weight of the remaining edges to one. For updating the contact search information in successive iterations we followed the approach that keeps the partitioning of the mesh fixed but only updates the geometric descriptors of the various subdomains by inducing a new decision tree. Finally, the contact search for both MCML+DT and ML+RCB was performed by approximating each surface element by its bounding box.

	MCML+DT Algorithm			ML+RCB Algorithm			
	FEComm	NTNodes	NRemote	FEComm	M2MComm	UpdComm	NRemote
25-way	28101	1206	5103	23961	12205	553	4972
100-way	65979	2144	9915	59688	12582	1125	11078

Table 1: The performance achieved by the MCML+DT and ML+RCB algorithms for partitioning the sequence of 100 meshes. These results correspond to averages over the 100 meshes.

5.1 Performance Metrics

We evaluated the performance of MCML+DT and ML+RCB using six different metrics whose meaning is as follows.

FEComm is the total communication volume resulted from partitioning the entire mesh and represents the communication overhead of the first phase of the computations performed during each time-step. Note that for ML+RCB, we used METIS's [15] multilevel algorithm to compute the k-way partitioning of the mesh.

NTNodes is the total number of nodes in the decision tree induced by MCML+DT that is used to obtain the geometric descriptors of each subdomain. It represents the cost of setting up the contact-search data structures.

NRemote is the total number of surface elements that need to be sent to the different partitions so that they can be searched for contact. It represents the communication cost that is incurred during the global search phase of contact detection.

M2MComm is the total number of contact points that belong to partitions that are different from the partitions that were assigned for the first phase. It represents the communication cost associated with mapping information between the two meshes in the ML+RCB algorithm. Note, in order to ensure that the communication overhead of transferring information between the two partitions is minimized as much as possible we used a maximal weight matching algorithm to optimized the mapping between the two partitions.

UpdComm is the number of contact points that end up being assigned to different partitions as a result of the mesh-updating strategy used by ML+RCB.

5.2 Results

Table 1 shows the performance of the two algorithms in terms of the different metrics for 25 and 100 partitions. These values were obtained by averaging the various matrices over the entire set of 100 meshes.

From these results we can see that the ML+RCB algorithm leads to partitionings whose FEComm cost is smaller than that of MCML+DT. This should not be surprising, as the k-way partitioning computed by MCML+DT has to balance two different constraints, whereas ML+RCB's partitioning needs to balance only one. However, this savings in terms of FEComm, comes at the expense of the M2MComm cost that ML+RCB has to perform but MCML+DT does not. Since in general, information needs to be transfered both from the first to the second partitioning and then back to the first, the communication cost incurred by ML+RCB will be twice that of the M2MComm value shown in the table. Thus, if we take this into account and if we assume that each communication operation involves data elements of the same size, ML+RCB requires 72% and 29% more communication than MCML+DT for the 25- and 100-way partitions, respectively. Note that these figures correspond to comparisons that do not include any contact search related operations.

Comparing the performance of the two approaches in terms of the NRemote metric, we can see that for the 25-way partitioning, MCML+DT and ML+RCB lead to comparable performance, with MCML+DT doing 2.6% worse, whereas for the 100-way partitioning, the MCML+DT scheme outperforms ML+RCB as the latter needs to communicate 12% more surface elements. These results suggest that MCML+DT's approach of describing each subdomain

as a set of box-shaped regions is quite effective in eliminating most of the false positive contacts, and that the initial multi-constraint partitioning was quite effective in reducing the number of adjacent contact points that span partition boundaries.

Finally, comparing the remaining two performance metrics, NTNodes and UpdComm we can see that both of them increase with the number of processors, but they are relatively small compared to the other overheads.

6 Conclusions and Directions for Future Research

In this paper we presented a new approach for partitioning contact/impact numerical simulations in the context of parallel processing. This approach combines recent advances in multi-constraint graph partitioning with ideas borrowed from the machine learning community and leads to an algorithm that has a lower communication overhead than the current state-of-the-art ML+RCB approach. Moreover, its overall simpler structure makes its incorporation in various production contact/impact codes easier.

The approach presented in this paper can be improved in a number of ways. We believe that better tree induction methods can be developed that besides the purity of the bisection they also take into account how far away the various contact points are from the decision hyperplane. Specifically, hyperplanes that go through a sparsely populated region of the space or are far away from their nearest points should be preferred, as they will tend to reduce the number of false-positives during contact search. In addition, the development of better geometry-aware multi-constraint partitioning algorithm can greatly improve the performance of this approach. Finally, before this approach can be incorporated in simulation codes, a parallel version of it needs to be developed. Fortunately, efficient parallel formulations of the multi-constraint graph partitioning, multi-constraint partitioning refinement, and decision tree induction already exist, making the parallelization task straightforward.

Acknowledgments

I will like to thank Andrew Johnson from making available the EPIC datasets.

References

[1] Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. *Classification and Regression Trees*. Chapman & Hall, New York, 1984.

[2] K. Brown, S. Attaway, S. Plimpton, and B. Hendrickson. Parallel strategies for crash and impact simulations. *Computational Methods in Applied Mechanics & Engineering*, 184:375–390, 2000.

[3] T. Bui and C. Jones. A heuristic for reducing fill in sparse matrix factorization. In *6th SIAM Conf. Parallel Processing for Scientific Computing*, pages 445–452, 1993.

[4] G. Camacho and M. Ortiz. Adaptive langrangian modeling of ballistic penetration of metalic targets. *Computational Methods in Applied Mechanical Engineering*, 147:269–301, 1997.

[5] R. Diekmann, J. Hungershofer, M. Lux, L. Taenzer, and J. Wierum. Efficient contact search for finite element analysis. In *European Congress on Computational Methods in Applied Sciences and Engineering*, 2000.

[6] M. Heinstein, S. Attaway, F. Mello, and J. Swegle. A general-purpose contact detection algorithm for nonlinear structural analysis codes. Technical Report SAND92-2141, Sandia National Laboratories, 1993.

[7] B. Hendrickson. Graph partitioning and parallel solvers: Has the emperor no clothes? In *Proc. Irregular'98*, pages 218–225, 1998.

[8] B. Hendrickson and K. Devine. Dynamic load balancing in computational mechanics. *Computational Methods in Applied Mechanics & Engineering*, 184:485–500, 2000.

[9] B. Hendrickson and T. Kolda. Graph partitioning models for parallel computing. *Parallel Computing (to appear)*, 2000.

[10] Bruce Hendrickson and Robert Leland. The chaco user's guide, version 1.0. Technical Report SAND93-2339, Sandia National Laboratories, 1993.

[11] Bruce Hendrickson and Robert Leland. A multilevel algorithm for partitioning graphs. Technical Report SAND93-1301, Sandia National Laboratories, 1993.

[12] C. Hoover, A. DeGroot, J. Maltby, and R. Procassini. Paradyn: Dyna3d for massively parallel computers, 1995. Presentation at Tri-Laboratory engineering Conference on Computational Modeling.

[13] G. Johnson, R. Stryk, and S. Beissel. *User Instructions for the 2001 Version of the EPIC code*. Alliant Techsystems, Inc., Hopkins, Minnesota, April 2001.

[14] M.V. Joshi, G. Karypis, and V. Kumar. ScalParC: A new scalable and efficient parallel classification algorithm for mining large datasets. In *Proc. of the International Parallel Processing Symposium*, 1998.

[15] G. Karypis and V. Kumar. METIS 4.0: Unstructured graph partitioning and sparse matrix ordering system. Technical report, Department of Computer Science, University of Minnesota, 1998. Available on the WWW at URL *http://www.cs.umn.edu/~metis*.

[16] G. Karypis and V. Kumar. Multilevel algorithms for multi-constraint graph partitioning. In *Proceedings of Supercomputing*, 1998. Also available on WWW at URL http://www.cs.umn.edu/~karypis.

[17] G. Karypis and V. Kumar. Multilevel k-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed Computing*, 48(1):96–129, 1998. Also available on WWW at URL http://www.cs.umn.edu/~karypis.

[18] G. Karypis and V. Kumar. A parallel algorithm for multilevel graph partitioning and sparse matrix ordering. *Journal of Parallel and Distributed Computing*, 48(1):71–95, 1998. Also available on WWW at URL http://www.cs.umn.edu/~karypis. A short version appears in Intl. Parallel Processing Symposium 1996.

[19] G. Karypis and V. Kumar. A fast and highly quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1), 1999. Also available on WWW at URL http://www.cs.umn.edu/~karypis. A short version appears in Intl. Conf. on Parallel Processing 1995.

[20] G. Karypis and V. Kumar. Parallel multilevel k-way partitioning for irregular graphs. *SIAM Review*, 41(2):278–300, 1999.

[21] G. Karypis, Kirk Schloegel, and V. Kumar. PARMETIS 3.0: Parallel graph partitioning and sparse matrix ordering library. Technical report, Department of Computer Science, University of Minnesota, 2002. Available on the WWW at URL *http://www.cs.umn.edu/~metis*.

[22] George Karypis and Vipin Kumar. A coarse-grain parallel multilevel k-way partitioning algorithm. In *Proceedings of the eighth SIAM conference on Parallel Processing for Scientific Computing*, 1997.

[23] G. Lonsdale, J. Clinckemaillie, S. Vlachoutsis, and J. Dubois. Communication requirements in parallel crashwirthiness simulation. In *Proceedings of the HPCN 94*, pages 55–61, 1994.

[24] J. Malone and N. Johnson. A parallel finite element contact/impact algorithms for non-linear explicit transient analysis: Part II – parallel implementation. *Intl. J. Num. Methods Eng.*, 37:591–603, 1994.

[25] B. Monien, R. Preis, and R. Diekmann. Quality matching and local improvement for multilevel graph-partitioning. Technical report, University of Paderborn, 1999.

[26] M. Oldenburg and L. Nilsson. The position code algorithm for contact searching. *International Journal for Numerical Methods in Engineering*, 37:359–386, 1994.

[27] S. Plimpton, S. Attaway, B. Hendrickson, J. Swegle, C. Vaughan, and D. Gardner. Transient dynamics simulations: Parallel algorithms for contact detection and smoothed particle hydrodynamics. *Journal of Parallel and Distributed Computing*, 50:104–122, 1998.

[28] A. Pothen. Graph partitioning algorithms with applications to scientific computing. In D. Keyes, A. Sameh, and V. Venkatakrishnan, editors, *Parallel Numerical Algorithms*. Kluwer Academic Press, 1996.

[29] J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[30] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, 1993.

[31] K. Schloegel, G. Karypis, and V. Kumar. A new algorithm for multi-objective graph partitioning. In *Proceedings of Europar 1999*, September 1999.

[32] K. Schloegel, G. Karypis, and V. Kumar. Parallel multilevel algorithms for multi-constraint graph partitioning. In *Proceedings of Europar 2000*, September 2000. Distinguished Paper Award.

[33] Kirk Schloegel, George Karypis, and Vipin Kumar. Multilevel diffusion algorithms for repartitioning of adaptive meshes. *Journal of Parallel and Distributed Computing*, 47(2):109–124, 1997. Also available on WWW at URL http://www.cs.umn.edu/~karypis.

[34] Kirk Schloegel, George Karypis, and Vipin Kumar. Graph partitioning for high-performance scientific simulations. In Jack Dongara, Ian Foster, Geoffrey Fox, William Gropp, Ken Kennedy, Linda Torczon, and Andy White, editors, *CRPC PArallel Computing Handbook*, chapter 18, pages 491–541. Morgan Kaufmann, San Francisco, CA, 2002.

[35] C. Walshaw and M. Cross. Parallel optimisation algorithms for multilevel mesh partitioning. Technical Report 99/IM/44, University of Greenwich, London, UK, 1999.

[36] Z. Zhong and L. Nilsson. A contact searching algorithm for general contact problems. *Comp. & Struct.*, 33:197–209, 1989.