Load Balancing for Parallel Computations with the Finite Element Method

José Luis González García[1], Ramin Yahyapour[1], and Andrei Tchernykh[2]

[1]GWDG - Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen,
Göttingen, Lower Saxony,
Germany

[2]CICESE Research Center, Ensenada, Baja California,
Mexico

{jose-luis.gonzalez-garcia, ramin.yahyapour}@gwdg.de, chernykh@cicese.mx

Abstract. In this paper, we give an overview of efforts to improve current techniques of load-balancing and efficiency of finite element method (FEM) computations on large-scale parallel machines and introduce a multilevel load balancer to improve the local load imbalance. FEM is used to numerically approximate solutions of partial differential equations (PDEs) as well as integral equations. The PDEs domain is discretized into a mesh of information and usually solved using iterative methods. Distributing the mesh among the processors in a parallel computer, also known as the mesh-partitioning problem, was shown to be NP-complete. Many efforts are focused on graph-partitioning to parallelize and distribute the mesh of information. Data partitioning is important to efficiently execute applications in distributed systems. To address this problem, a variety of general-purpose libraries and techniques have been developed providing great effectiveness. But the load-balancing problem is not yet well solved. Today's large simulations require new techniques to scale on clusters of thousands of processors and to be resource aware due the increasing use of heterogeneous computing architectures as found in many-core computer systems. Existing libraries and algorithms need to be enhanced to support more complex applications and hardware architectures. We present trends in this field and discuss new ideas and approaches that take into account the new emerging requirements.

Keywords. Load balancing, FEM, HPC efficiency.

Balanceo de Cargas para Computación en Paralelo con el Método de Elementos Finitos

Resumen. En este artículo damos una vista general de los esfuerzos para mejorar las técnicas actuales de balanceo de cargas y eficiencia en el cómputo con el uso del método de elementos finitos (MEF o FEM por sus siglas en inglés) en máquinas paralelas de gran escala. Introducimos también un balanceo de cargas multinivel para mejorar las diferencias locales. El MEF es usado para aproximar numéricamente las soluciones a ecuaciones diferenciales parciales (EDP o PDE por sus siglas en inglés) o a ecuaciones integrales. El dominio de las EDP se hace discreto convirtiéndolo en una malla de información y usualmente se soluciona utilizando métodos iterativos. La distribución de la malla en los procesadores de una computadora paralela, también conocido como el problema de partición de la malla, es NP-completo. Muchos esfuerzos se enfocan en partición de grafos para paralelizar y distribuir la malla de información. La partición de la información es importante para ejecutar las aplicaciones eficientemente en sistemas distribuidos. Para abordar este problema, una variedad de librerías de propósito general y técnicas se han desarrollado proveyendo gran efectividad. Pero el problema del balanceo de cargas no está del todo solucionado. Las extensas simulaciones de hoy requieren nuevas técnicas para poder ser ejecutadas eficientemente en sistemas de miles de procesadores y para tomar en cuenta los recursos disponibles debido al extenso uso de arquitecturas heterogéneas en la actualidad. Las librerías y algoritmos actuales deben ser adaptados para ser capaces de manejar aplicaciones más complejas y diferentes arquitecturas de hardware. Nosotros presentamos las tendencias en este campo y discutimos nuevas ideas que consideran los requerimientos emergentes.

Palabras clave. Balanceo de cargas, método de elementos finitos, eficiencia en computación de alto desempeño.

1 Introduction

The finite element method (FEM) is a powerful tool widely used for predicting behavior of real-world objects with respect to mechanical stresses, vibrations, heat conductions, etc. [1, 2]. However, applications have large computation, communication and memory costs to be useful in practice in the form of sequential implementations. Parallel systems allow FEM applications to overcome this problem [3], but in turn they create new problems regarding system efficiency, see Section 3 for details.

Partial differential equations (PDEs) are used to describe a given problem. The PDEs domain is discretized into a mesh of information (triangles or rectangles in 2D, tetrahedra or hexahedra in 3D), then the PDEs are transformed into a set of linear equations defined on these elements [4]. In general, iterative methods such as Conjugate Gradient (CG) or Multigrid (MG) are employed to solve the linear systems [1, 5]. The quality of the solution heavily depends on the accuracy of the discretization; the elements of the mesh have to be small enough in order to allow an accurate approximation. An extremely fine discretization may incur in extra computation, communication and memory costs. Adaptive techniques allow the solution error to be kept under control while costs can be minimized [6].

The parallelization of numerical simulation algorithms usually follows the single-program multiple-data (SPMD) paradigm. Hence, the mesh is partitioned and distributed evenly among the parallel system [2, 7]. Distributing the mesh among the processors in a parallel computer, also known as the mesh-partitioning problem, was shown to be NP-complete [8, 9]. The mesh can be easily represented as a graph, so in recent years, much effort has been focused on developing suitable heuristics based on the graph-partitioning problem [10–17].

The most important causes of load imbalance in FEM parallel applications are the dynamic nature of the problem over time (in computational and communication costs) and the adaptive refinement of meshes during the computations. Other causes may include the interference from other users in a time-shared system, among others. Thus, an efficient dynamic load balancing is required. The increasing size of modern parallel or distributed computers requires software libraries to be enhanced to support these new hardware architectures.

In the next section we present a background of FEM computations and basic concepts in the area. Section 3 presents the load balancing problem in parallel FEM computations and relevant previous work. Trends in the field and discussion of new ideas and approaches that consider the new emerging requirements are given in Section 4, while Section 5 finishes the paper with some conclusions and hints for future research.

2 Background

In this section, we give an overview of the FEM. We refer the reader to [5] for a more extensive description. We also describe some available FEM frameworks.

2.1 FEM and PDE

PDEs are often used to model physical phenomena such as the flow of air around a wing, the distribution of temperature on a plate, the propagation of a crack [1, 2], and rarely have an explicit solution.

The most widely used method to solve PDEs is to discretize them into a mesh (triangles or rectangles in 2D, tetrahedra or hexahedra in 3D). There are several ways to do it (e.g., [18]). The simplest method uses finite difference approximations for the partial differential operators. The FEM replaces the original function by a function with some degree of smoothness over the global domain. A structure with a complex geometry is modeled by a number of small connected *cells* (elements, nodes). Thus, the function can be numerically approximated.

The matrices which arise from these discretizations are generally large and sparse. Since the derived matrices are sparse, the equations are typically solved by iterative methods such as CG or MG [1, 5]. The quality of solutions obtained by such numerical approximation algorithms heavily depends on the accuracy of the discretization. The smaller the

elements, the better is the quality of the solution achieved. It is also true that a better solution leads to more intensive computations.

2.2 Solvers and Preconditioners

The FEM solver solves a set of matrix equations which approximate the physical phenomena under study. The first introduced iterative methods were based on relaxation of the coordinates like Jacobi, Gauss-Seidel, and SOR [5]. These methods are rarely used in our days. Other techniques utilize a projection process to approximate the solution of the linear system. The Krylov subspace methods are considered among the most important techniques. We can mention Arnoldi's Method, CG, Lanczos Biorthogonalization, and Transpose-Free Variants [5], among others. MG methods were initially designed for the solution of discretized elliptic PDEs. Later, they were enhanced to handle other PDEs problems as well as problems not described by PDEs. The performance of MG methods is superior to that achieved by Krylov subspace methods, however, they require specific implementations for each problem in contrast with Krylov subspace methods which are for general purpose.

The equations are typically solved by iterative methods such as CG or MG. MG methods are among the fastest numerical algorithms for solving large sparse systems of linear equations [19]. The CG method is an iterative algorithm for realization of an orthogonal projection onto a Krylov subspace and it is suitable only for symmetric positive definite matrices.

As described in [5], a preconditioner is a form of implicit or explicit modification of an original linear system which makes it "easier" to solve by a given iterative method. It conditions a given problem into a form that is more suitable for a numerical solution. A preconditioner can be defined as a subsidiary approximate solver combined with an outer iteration technique. The lack of robustness is a well-known problem of iterative solvers, the reason why preconditioning is a key ingredient for Krylov subspace methods.

2.3 Meshes

The quality of the solution heavily depends on the accuracy of discretization. The elements of a mesh have to be small in order to allow an accurate approximation. Unfortunately, regions with large gradients are not known in advance. Hence, meshes can be unstructured and periodically refined/coarsened in areas where it is required during calculations; or they can be structured with equal connectivity for each node. The main issue with structured meshes is a large number of small elements in regions where they are not needed. Obviously, the first variant is preferred and used for FEM. The solution has the same quality but the time needed is only a fraction of the time required by the structured mesh. Adaptive techniques allow the solution error to be kept under control while computation costs can be minimized [6].

Depending on the problem, some regions of the mesh are refined during computations. Since these areas are not known in advance or can vary over time, the mesh is refined and coarsened several times during the computations. This is a source of imbalance in parallel FEM simulations. Hence, load balancing techniques must be applied to reduce the impact of this refinement/coarsening process on the efficiency of computations. It is necessary to find a new balanced partition with the additional objective not to cause too many elements to change their processor. Migrating elements may be an extremely costly operation since a lot of data has to be sent over communication links. A number of solutions were proposed [20, 21]. Well known graph partitioning libraries, presented later in Section 3, also deal with the adaptive mesh refinement problem.

2.4 Parallelization of Numerical Simulations

Due to a large amount of mesh elements required to obtain an accurate approximation, FEM has become a classical application for parallel computers. Parallel versions of numerical simulation algorithms follow the SPMD paradigm. Each processor executes the same code on a different part of the data. The mesh has to be divided and mapped to processors in order to

minimize the overall computation time [2, 7]. Distributing the mesh among the processors in a parallel computer, also known as the mesh-partitioning problem, was shown to be NP-complete [8, 9]. The parallel efficiency heavily depends on two factors: the distribution of data (mesh) on the processors and the communication overhead of boundary data.

During the computations, the mesh is refined and coarsened several times. Hence, the workload is changed unpredictably and a new distribution of the mesh among the processors is required without causing a change of the location for too many elements. Dynamic load balancing has to be applied. The application has to be interrupted for a load balancing step. This interruption should be as short as possible. Thus, the full advantages of High Performance Computing (HPC) technology will be able to be exploited only when efficient load balancing techniques are applied.

As parallel simulations and environments become more complex, partitioning techniques, used to distribute the load among the processors, must be enhanced to fit the emerging requirements. Partitioning algorithms need to be aware of computer architectures, memory and communication resources. Additionally, FEM simulations must scale linearly with respect to the number of processors and the problem size.

2.5 FEM Frameworks and Simulators

A variety of FEM tools and frameworks have been developed in the last years [22–42]. Ready-to-use software is also available for commercial use [2, 43–45]. While some provide effective results for particular problems, others are more suitable for general purposes. We mention the most relevant tools including their main features.

Charm++ [22] is a parallel framework developed at the University of Illinois. It gives scientists the opportunity to focus on modeling the problem and not on parallelization details. It is based on multi-partition decomposition and data driven execution. The application is decomposed into a large amount of small parts called objects. The objects are then distributed among the processors. The communication pattern is set between the objects and not between processors. The framework separates the numerical algorithms from the parallel implementation. One example of FEM simulators based on the Charm++ framework is NAMD2 described in [46]. The authors give the analysis of its performance on some benchmark applications.

Recent effort is being focused on massively parallel programming due to the increasing use of clusters of thousands of processors. Heister *et al.* [37] focus on a design of efficient data structures and algorithms for these new requirements. They have enhanced the library deal.II [47] to take advantage of the large cluster power. This library uses object-oriented and data encapsulation techniques to divide finite element implementations into smaller blocks. It supports a large number of different applications covering a wide range of scientific areas, programming methodologies, and application-specific algorithms.

Dolfin [28] employs novel techniques for automated code generation. Mathematical notations are used to express Finite Element (FE) variational forms, from which low-level code is automatically generated, compiled, and integrated with implementations of meshes and linear algebra. Dolfin differs from many other projects such as Sundance [31] and Life [34, 35], among others, in that it relies more on code generation. As a result, Dolfin supports a wider range of FE since it may assemble FE variational forms on FE spaces supported by the *form compiler* and FE backend. The *form compiler* automatically generates code from a user-defined high-level description of the FE variational form [28].

FEAST [29] is a FE based solver toolkit for the simulation of PDE problems on parallel HPC systems. It is being developed at the Technical University of Dortmund. It is the successor of the established FE packages FEAT and FEATFLOW [48]. The next version, FEAST2, is currently under development and will include new features such as 3D support.

ESI Group [44] develops a wide selection of software for different applications such as biomechanics, casting, crash, electromagnetic, and fluid dynamics applications, among others.

3 The Load-Balancing Problem in Parallel Computations with FEM

This section presents information related to load-balancing techniques. We mainly focus on load-balancing through graph/mesh partitioning methods. Much work has been done previously in this area.

3.1 Description and Factors Leading to Imbalance

Load-balancing is important in parallel computations; it is an interesting area of research with a vast range of applications. It was first introduced by Shivaratry *et al*. [49] who described and compared some common strategies. Load-balancing maximizes application performance by keeping processor idle time and interprocessor communication overhead as low as possible. To minimize the overall computation time, all processors should contain the same amount of computational work, and data dependencies between processors should be minimized. Thus, the full advantages of HPC technology will be able to be exploited only when efficient load balancing techniques are applied.

Numerous methods for static and dynamic load balancing have been proposed. Some of them will be discussed later. The dynamic problem has not been extensively studied as the static one. Devine *et al*. [50] provide ideas to address the dynamic problem. Willebeek-LeMair and Reeves [51] provide a comparison study of dynamic load-balancing strategies.

The most important causes of load imbalance in FEM parallel application are the dynamism of the problem over time (in computational and communication costs), and the adaptive refinement of meshes during the calculation. Since these areas are not known in advance or can vary over time, the mesh is refined and coarsened several times during the computations. Interference from other users in a time-shared system and heterogeneity in either the computing resources or in the solver can also result in load imbalance and poor performance.

3.2 Multiphase Problems

For certain applications the mesh elements may belong to more than one phase. Typically these applications arise from multiphysics or contact-impact modeling, and geometric partitioners are often preferred to compute the partitions.

As data needs to be communicated between phases, computing a single partition well with respect to all phases would reduce communication. Computing this single partition is more complex as each processor would have multiple workloads corresponding to each phase. In principle, the partitioning is done phase by phase, using the results of the previous phase to influence the partition of the current one [52].

This kind of problems consists of various separate phases interrelated (e.g., crash simulations consist of two phases: computation of forces and contact detection). Often, separate partitions are used for each phase and data communication is required [53].

3.3 Load Balancing through Graph Partitioning

Mesh-based PDE problems are often expressed as graphs. Graph vertices represent the data (or work) to be partitioned. Edges represent relationships (data dependencies) between vertices. The number of boundary edges approximates the volume of communication needed during computation. Vertices and edges can be weighted to reflect associated computation and communication costs, respectively. The goal of graph partitioning, then, is to assign equal total vertex weight to partitions while minimizing the weight of cut edges.

However, this should be considered only as balancing the application if it is known that each graph vertex represents an equal amount of work. In fact, this is usually not true, and the computational work is dominated by the cost of the local subdomain solutions. Another limitation of the use of graphs is the type of systems they can represent [54]. Since edges in the graph model are non-directional, they imply symmetry in all relationships, making them appropriate only for problems represented by square, symmetric matrices. To address this drawback, hypergraphs are used to model PDE problems. As in a graph,

hypergraph vertices represent the data to be partitioned. However, hypergraph edges (hyperedges) are sets of two or more related vertices. The number of hyperedge cuts is an exact representation of communication volume, not merely an approximation [55]. Effectiveness of hypergraph partitioning has been demonstrated in many areas, including VLSI layout [56], sparse matrix decompositions [55, 57], database storage and data mining [58, 59].

3.3.1 The Graph-Partitioning Problem

In a few words, the graph-partitioning problem is to divide the set of vertices of a graph into subsets (subdomains) no larger than a given maximum size, so as to minimize some cost function (e.g., the total cost of the edge cut).

For the purposes of this paper, we use the definition of graph partitioning presented in [60]. Let $G = G(V, E)$ be an undirected graph of V vertices, with E edges which represent the data dependencies in the mesh. We assume that the graph is connected. We also assume that both vertices and edges are weighted (with positive integer values) and that $|v|$ denotes the weight of vertex v. Similarly, $|e|$, $|S_P|$ and $|E_c|$ denote the weights of edge e, subdomain S_P and edge cut E_c, respectively.

Given that it is necessary to distribute the mesh to P processors, we define a partition π to be the mapping of V into P disjoint subdomains S_P such that $\bigcup_P S_p = V$. The partition π induces a *subdomain graph* on G, which we shall refer to as $G_\pi = G_\pi(S, C)$. There is an edge $(S_p, S_q) \in C$ if there are vertices $v_1, v_2 \in V$ with $(v_1, v_2) \in E$, and $v_1 \in S_p$, $v_2 \in S_q$; the weight of a subdomain is just the sum of the weights of the vertices in it, $|S_p| = \sum_{v \in S_p} |v|$. We denote the set of intersubdomains or edge cut (i.e., edges cut by the partition) by E_c (note that $|E_c| = |C|$). Vertices that have an edge in E_c (i.e., $\{v \in V : \text{there esists } v' \in V, \text{with } (v, v') \in E_c\}$) are referred to as *border* or *boundary* vertices.

The definition of the graph-partitioning problem is to find a partition that evenly balances the load (i.e., vertex weight) in each subdomain, while minimizing the communication cost. The optimal subdomain weight is given by $\bar{S} := \lceil |V|/P \rceil$ (where the ceiling function $\lceil x \rceil$ return the smallest integer$\geq x$), and the *imbalance* is then defined as the maximum subdomain weight divided by the optimal weight (since the computational speed of the underlying application is determined by the most heavily weighted processor). Throughout this paper, the communication cost will be estimated by $|E_c|$, the weight of edge cut or cut-weight. A more precise definition of the graph-partitioning problem is therefore to find π such that $|S_p| \leq \bar{S}$ and $|E_c|$ is minimized. Note that a perfect balance is not always possible for graphs with non-unitary vertex weights.

To date, algorithms have been used almost exclusively to minimize the edge cut weight. It is important to note that this metric is only an approximation of communication volume and usually does not model the real costs [54]. Besides, it is known that this is not necessarily the best metric to use. It has been demonstrated that it can be extremely effective to vary the cost function based on the knowledge of the solver [61]. A more appropriate metric is the number of boundary vertices. It models the resulting communication volume more accurately, but unfortunately, it is harder to optimize [54].

However, for many applications, minimizing other metrics may be desirable. We list the most widely used ones here: send volume, receive volume, diameter, outgoing migration, and incoming migration. The send volume is the amount of outgoing information from each sub-domain. The receive volume is the amount of incoming information. The diameter is the longest shortest path between two vertices of the same sub-domain (infinity, if the sub-domain is not connected). The outgoing migration is the number of vertices that have to be migrated to a different sub-domain. And the incoming migration is the number of vertices that have to be migrated from a different sub-domain.

Furthermore, in dynamic load balancing, speed is often more important than quality of the partition (its balance). A less balanced solution does not necessarily cause unbalances during computation, but of course, allows other metrics to improve.

3.3.2 Mesh to Graph/Graph to Mesh Conversion

The mesh is converted into a weighted graph. The vertex weights correspond to calculation costs and edge weights correspond to potential communication costs. Different graph representations can be used. The type of graph should be selected based on the application requirements, the cost function model, and the accuracy with which the cost model should be approximated. We refer the reader to [62] for details.

The output of graph partitioners is an array indicating for each graph vertex to which process (sub-domain) it should be migrated. In the case of a dual graph, this array gives only a new distribution for the mesh elements, while a new distribution for the nodes still has to be determined. A similar situation holds in the case that a nodal graph has been used.

Types of graphs

Dual graph or *element graph*. The weighted graph vertices correspond to mesh elements and the associated calculation costs. The edges represent the potential communication between neighboring elements. Vertices are connected by an edge if the corresponding elements share an edge in 2D or a face in 3D.

Extended dual graph. For meshes with elements of different dimensions, the potential communication cannot be well represented by a dual graph. In an extended dual graph, graph vertices are connected by an edge when the corresponding elements share one or more nodes. Hence, certain connections between sub-domains that are lost in a classical dual graph, including connections between elements of different dimensions, are maintained. However, the extended dual graph may become very complex, requiring a lot of memory, especially for 3D tetrahedral meshes.

Generalized dual graph. This graph lies between the classical dual graph and the extended dual graph. As with the extended dual graph, it is well suited for meshes with different element types. However, not all elements sharing a node are joined by an edge of the graph. An element is connected only to those neighboring elements that share a (local) maximum number of nodes.

Nodal graph. Here graph vertices correspond to mesh nodes, and vertices are connected if they share an element.

Combined graph. In this graph, both elements and nodes are represented by vertices, allowing a good representation of all calculation costs. Since finite element applications often use node lists for inter-process communication, graph edges represent communication requirements between elements and nodes. Hence, this graph is a simplification of a general combined graph that would have all kinds of element-element, node-node, and element-node connections.

3.3.3 Partitioning Algorithms

Many methods have been proposed in the literature to deal with the partitioning problems of FE graphs on distributed memory multicomputers. These methods have been implemented in several graph partitioning libraries. We give an overview of their classification.

Greedy methods

Greedy approaches are based on graph connectivity. Typically, the first subdomain of a partition is initialized with one single vertex and further vertices are added until the required subdomain size is reached. Then, a new subdomain is initialized with an unassigned vertex and it is built up in the same greedy fashion. Several possibilities of choosing new vertices exist: progressing in a breath-first manner [11], choosing vertices which reduces the edge cut [10,15], etc. The greedy approach usually results initially in very compact subdomains, but often the last subdomain consists of all leftover elements and its shape is not smooth. Different methods try to solve this problem [15, 63–65].

A well-known algorithm is Bubble [66]. Its major drawback is the lack of a guarantee for balanced partitions. Although at the end the seeds are spread out evenly over the whole graph, the subdomains could not contain the same number of vertices. To overcome this problem, one may add a local partitioning method to balance the load, trying to further optimize either the edge cut or the aspect ratio (AR).

Geometric methods

Geometric partition methods [67, 68] are quite fast but they often provide worse partitions than those of more expensive methods such as spectral. Furthermore, geometric methods are applicable only if coordinate information for the graph is available. They are effective only when geometric locality is important and/or natural graph connectivity is not available. Geometric partitioners can induce higher communication costs than graph partitioners for some applications because they do not explicitly control communication. However, because of their simplicity, they generally run faster and can be implemented easier than graph partitioners. Examples of this approach are presented in [15, 50, 69–71].

Diffusive methods

This technique for dynamic load balancing has been proposed primarily due to its simplicity and its analogy with the physical process of diffusion. It is the work diffusing in a natural way through the multiprocessor network. Another interpretation of this approach involves analogies with finite Markov chain models. Work distribution can be considered to be an initial probability distribution, and the diffusion of work is mathematically identical to the evolution of state occupation probabilities. Much work has been done in this area [11, 66, 72–78].

Spectral methods

More elaborate methods, called spectral methods, use the connectivity measures based on the second smallest eigenvalue of the graph's Laplacian. These methods [14, 79] are quite expensive, but combined with fast multilevel contraction schemes they belong to the state-of-the-art in graph partitioning software [80, 81]. The Multilevel Spectral Bisection (MSB) algorithm produces partitions that are as good as those produced by the original spectral bisection, but it is one to two orders of magnitude faster, because it computes the Fiedler vector of the graph using a multilevel approach [82]. Other approaches can be found in [66, 73, 77, 83].

Multilevel methods

Recently, a number of researches have investigated a popular and successful class of algorithms that have moderate computational complexity, known as multilevel algorithms. This class of algorithms provide excellent (even better than spectral) graph partitions [13, 79, 84] and the basic idea behind them is very simple.

A graph contraction algorithm creates a series of progressively smaller and coarser graphs, generally until a few hundred of vertices remain in the coarsest graph. A bisection of this much smaller graph is computed. Then this partition is projected back towards the original graph, refining the partition at each graph level. Since the original graph has more degrees of freedom, such refinements usually decrease the edge cut. To date, these algorithms have been used almost exclusively to minimize the edge cut weight, a cost that approximates the total communication volume in the underlying solver.

From the experiments presented in [13, 79], it is clear that multilevel graph partitioning algorithms are able to find high quality partitions for a variety of unstructured graphs. However, there exists little theoretical analysis that could explain the ability of multilevel algorithms to produce good partitions. We briefly describe the various phases of the multilevel algorithm. The reader should refer to [13] for further details.

A series of progressively smaller and coarser graphs, $G_i = (V_i, E_i)$, is created from the original graph $G_0 = (V_0, E_0)$ such that $|V_i| > |V_{i+1}|$. A coarser graph G_{i+1} is constructed from graph G_i by finding a maximal matching $M_i \subseteq E_i$ of G_i and collapsing together the vertices that are incident on each edge of the matching. Vertices that are not incident on any edge of the matching are simply copied to G_{i+1}. When vertices $v, u \in V_i$ are collapsed to form a vertex $w \in V_{i+1}$, the weight of the vertex w is $|w| = |v| + |u|$, while the edges incident on w is set equal to the union of the edges incident on v and u minus the edge (v, u). In the case when a vertex z in G_i contains edges to both v and u such that (z, v) and (z, u), then the weight of the resulting edge in G_{i+1} is set to $|(z, v)| + |(z, u)|$. Thus, during successive coarsening levels, the weights of both vertices and edges are increased.

Some authors do not consider the matching as a separate phase, but part of the coarsening one. Maximal matching can be computed in different ways [13, 60, 85]. The method used to compute

the matching greatly affects both the quality of the bisection, and the time required during the uncoarsening phase [86]. Here, we just mention some.

Random matching (RM) computes the maximal matching by using a randomized algorithm [79]. Heavy-edge matching (HEM), computes a matching M_i such that the weight of the edges in M_i is high. The modified heavy-edge matching (HEM*) is a modification of HEM which tries to decrease the average degree of coarser graphs. Walshaw and Cross use a variant of the graph contraction algorithm proposed by Hendrickson and Leland [79].

The third phase of a multilevel algorithm is to compute a balanced partition of the coarsest graph $G_k = (V_k, E_k)$. The *k-way* partitioning problem is most frequently solved by recursive bisection. It is also possible to directly compute a *k-way* partition, but the coarsening phase may become more expensive to perform. Nevertheless, there are advantages such as the entire graph now needs to be coarsened only once, and it is well known that recursive bisection can perform arbitrarily worse partitions than direct *k-way* partitioning [87]. An evaluation of different algorithms for partitioning a coarser graph can be found in [13].

During the uncoarsening phase, the partition of the coarsest graph G_k is projected back towards the original graph G_0 by going through the graphs $G_{k-1}, G_{k-2}, \cdots, G_1$, refining the partition at each graph level. Even if the partition of G_i is at a local minima, the partition of G_{i-1}, obtained by the projection, may not be at a local minima. Hence, local refinement heuristics must be applied to improve the partition of G_{i-1}. A number of refinement algorithms are investigated in [13].

3.3.4 Graph Partitioning Software

Multilevel graph partitioning software is available in the form of public domain libraries, and most of them are free for academic research, such as Chaco [80, 81], METIS [88, 89] and SCOTCH [90–92]. We refer the reader to [93] for a more detailed description of each one. The performance of this software has been compared several times in recent years [21,60, 94, 95]. Due to a large number of configuration parameters of each library, it is hard to achieve a clear conclusion.

Jostle [96–98] is suitable for partitioning unstructured meshes for use on distributed memory parallel computers. It can also repartition and load-balance existing partitions. The refinement algorithm used by Jostle is a multi-way version of the Kernighan-Lin (KL) algorithm [17], which incorporates a balancing flow. The balance flow is calculated either with a diffusive type algorithm, or with an intuitive asynchronous algorithm. Jostle can be used to dynamically repartition a changing series of meshes, both to load-balance and to minimize the amount of data movement, and hence, redistribution costs. Jostle also has a variety of built-in experimental algorithms and modes of operations such as optimizing subdomain AR. Jostle is very suitable for dynamic repartitioning.

METIS is a set of serial programs for partitioning graphs, partitioning FE meshes, and producing fill reducing orderings for sparse matrices. METIS is capable of minimizing the sub-domain connectivity as well as the number of boundary vertices. The implemented algorithms are based on the multilevel recursive-bisection [13], multilevel *k-way* [99], and multi-constraint partitioning schemes. METIS is based in the multilevel paradigm [13, 79, 94, 100] and includes a variety of algorithms for each phase of the partitioning process. Additionally, METIS includes an MPI-based parallel library for the partitioning of unstructured graphs, meshes and computing fill-reducing orderings of sparse matrices.

METIS currently supports four different types of mesh elements: triangles, tetrahedra, hexahedra, and quadrilaterals. The first step is to convert the mesh into a nodal or dual graph. After this is done, the graph is partitioned and converted back into the original mesh including the partition information.

METIS and Jostle are designed to support partitioning and load balancing of adaptive mesh calculations in parallel. Both use the algorithm of [101] in order to determine the balancing flow and use a multilevel strategy for shifting elements, where optionally the coarsening is done only inside subdomains.

Chaco addresses three classes of problems. First, it computes graph partitions using a variety

of approaches with different properties. Second, Chaco intelligently embeds the partitions it generates into several different topologies. The topologies are those matching the common architectures of parallel machines, namely, hypercubes and meshes. Third, Chaco can use spectral methods to sequence graphs in a manner that preserves locality. Chaco implements five partitioning algorithms: linear, inertial, spectral, KL, and multilevel-KL; all described in the user's manual [81]. Each of these algorithms can work on weighted graphs, and can be used to create partitions of two, four, or eight subdomains at each stage of recursive decomposition.

Party [102–104] serves a variety of different partitioning methods in a very simple and easy way. It can be used as stand-alone or as library interface, and provides default settings for an easy and fast start. The PARTY partitioning library provides interfaces to the Chaco library, and the central methods therein can be invoked from the PARTY environment. One of the main advantages of Party is that its general partitioning procedure allows several partitioning methods to be managed at once. Among others, Party generates initial partitions through linear, random and Farhat techniques, and improves the partition with KL or Helpful Sets (HS). In contrast to other implementations, the local refinement algorithm in Party is based on theoretical analysis finding upper bounds for the bisection width of regular graphs [105, 106]. Instead of moving single vertices, the HS heuristic exchanges whole vertex sets between the partitions. However, this approach has been successfully applied only to bisectioning. Complete information can be found in [104].

SCOTCH is a software package for static mapping partitioning and sparse matrix block ordering of graphs and meshes. It is based on the Dual Recursive Bipartitioning (DRB) mapping algorithm and several graph bipartitioning heuristics [107]. The mapper can map any weighted source graph onto any weighted target graph, or even onto disconnected subgraphs of a given target graph, which is very useful in the context of multi-user parallel machines. Recently, the ordering capabilities of SCOTCH have been extended to native mesh structures, thanks to hypergraph partitioning algorithms. It also comprises parallel graph ordering routines.

As mentioned above, a concrete and concise conclusion comparing different graph partitioning software cannot be established. Many comparisons between them have been published [21, 60, 94, 95] with different results. Karypis and Kumar [94] pointed out some differences between Chaco and METIS at the refinement phase, leading to a more expensive partitioning process in the former. Diekmann *et al*. [21] showed that Jostle and METIS are not suitable for use in long periods of time without a complete repartitioning from time to time, when AR is a metric of importance. The experimental results by Walshaw and Cross [60] show a degraded performance of METIS compared to Jostle due the coarsening phase. METIS coarsens to 2000 vertices while Jostle coarsens until the number of vertices equals the number of final subdomains. Usually, METIS is very fast, while Jostle takes longer time but often computes better solutions.

Furthermore, libraries like METIS and Jostle primarily minimize the edge cut and cannot obey constraints like connectivity and straight partition boundaries which are important for some numerical solvers.

3.3.5 Hypergraph Partitioning Software

Serial hypergraph partitioning libraries are available, such as hMETIS [108, 109], PaToH [110, 111], Mondriaan [112]. But for large scale and dynamic applications, parallel hypergraph partitioners are needed. The load balancing library Zoltan [113, 114] also includes a serial hypergraph partitioner which uses multilevel strategies developed for graph partitioning [13, 79]. The hypergraph is coarsened into successively smaller hypergraphs. The smallest hypergraph is partitioned and the coarse decomposition projected back to the larger hypergraphs, using local optimization to reduce hyperedge cuts while maintaining balance at each level.

3.4 Load Balancing Libraries

The DRAMA [62, 115] library performs a parallel computation of a mesh re-allocation that will re-balance the costs of the application code based

on the DRAMA cost model. The DRAMA cost model is able to take into account dynamically changing computational and communication requirements. The library provides the application program sufficient information to enable an efficient migration of the data between processes. DRAMA is designed to be called by parallel message-passing (MPI) mesh-based applications.

Different parts of parallel applications that are separated by explicit synchronization points are defined as phases within the DRAMA cost model. The total cost is then given by the sum of the maximum cost over all processes per phase and over all phases. The load imbalance for each phase is the ratio of the maximum to average process costs for that phase.

The Zoltan Parallel Data Services Toolkit [113, 114] is unique in providing dynamic load balancing and related capabilities to a wide range of dynamic, unstructured and/or adaptive applications. Zoltan supports many applications through its data-structure neutral design. Similar libraries, such as DRAMA which supports only mesh-based applications, focus on specific applications; Zoltan does not require applications to have specific data structures. However, with respect to data migration, libraries like DRAMA can provide greater capabilities, as they have knowledge of application data structures.

Zoltan's design is effective for both applications and research. It allows both existing and new applications to easily use Zoltan. New algorithms can be added to the toolkit easily and compared to existing algorithms in real applications using Zoltan.

Dynamic Resource Utilization Model (DRUM) [116, 117] provides applications aggregated information about the computation and communication capabilities of an execution environment. DRUM has been designed to work with Zoltan, but may also be used as a separate library. DRUM encapsulates the details of hardware resources, capabilities and interconnection topology; provides a facility for dynamic, modular, and minimally intrusive monitoring of an execution environment; and provides this information to be used by any load-balancing algorithm as the percentage of overall application load to be assigned to a partition.

UMPAL [118] is an integrated tool consisting of five components: a partitioner, load balancer, simulator, visualization tool, and web interface. The partitioner uses three partitioning libraries: Jostle, Metis and Party. The partitioning results are then optimized by the Dynamic Diffusion Method (DDM) [75], the Directed Diffusion Method (DD) [119] or the Multilevel Diffusion Method (MD) [97]. The load balancer provides two load-balancing methods: the prefix code matching parallel load-balancing method and the binomial tree based parallel load-balancing method, both proposed by Liao [75]. The simulator provides a performance simulation environment for a partitioned mesh. The visualization tool provides a way for users to view a partitioned mesh. The web interface provides a mean for users to use UMPAL via Internet and integrates the other four parts.

4 Current Trends

In this section, we present the current trends related to load-balancing techniques. We also propose a new model for load-balancing and optimizing unstructured mesh partitions based on a multilevel technique.

4.1 New Challenges

Today's large simulations require new techniques to scale on clusters of thousands of processors, and to be resource aware due the increasing use of heterogeneous computing architectures as found in many-core computer systems. Existing libraries and algorithms need to be enhanced to support more complex applications and hardware architectures. Thus, the full advantages of HPC technology will be able to be exploited only when efficient load balancing techniques are applied.

Often, FEM libraries restrict their use to small systems, and this becomes a limitation when thousands of cores are available. This has led to a significant disparity between the current hardware and software running on it. Heister *et al*. [37] propose parallel data structures and algorithms to deal with massively parallel simulations. They enhanced the library deal.II to overcome the problem. They focus on the primary

bottlenecks to parallel scalability: the mesh handling, the distribution and global numbering of the degrees of freedom, and the numerical linear algebra. Another library designed for massively parallel simulations is ALPS [25]. It is based on the p4est library [120], but it lacks the extensive support infrastructure of deal.II, and it is not publicly available.

After the mesh refinement, the workload among processors may become unbalanced. As the load-balancing step could be relatively large, a load-balancing step is necessary only when the degree of imbalance is high. Therefore, it is important to determine the influence of the imbalance on the total cost of a numerical simulation in order to decide if the load-balancing step should be performed or not. Olas *et al*. [27] introduce a new dynamic load balancer to NuscaS [26] based on a performance model. This model estimates the cost of the load-balancing step, as well as the execution time for a computation step performed with either balanced or unbalanced workload.

Many supercomputers are constructed as networks of shared-memory multiprocessors with complex and non-homogeneous interconnection topologies. Grid computing enables the use of geographically distributed systems as a single resource. This paradigm introduces new and difficult problems in resource management due the extreme computational and network heterogeneity. To distribute data effectively on such systems, load-balancers must be resource-aware. That is, they must take into account the heterogeneity in the execution environment. Some attempts to address this issue can be found in [50, 121–124].

4.2 Multilevel load balancer

As previously mentioned, new hardware architectures bring new capabilities and new problems in resource management. New approaches and algorithms have to be developed in order to overcome these issues. To this end, we propose a new multilevel load-balancing model, which aims to reduce the local imbalance, while tries to reduce the global communication overhead. The use of resource information and a cost function is important to achieve a good load balance.

The compute time has to scale linearly with respect to the problem and the number of processors. Additionally, local memory requirements should depend only on the local, not the global problem size. To efficiently distribute data on the underlying system, we need to gather information about the computing environment (e.g., processors, network topology and memory). A perfect balanced partition is worthless if it cannot be efficiently mapped. Such partitions have to be computed based on the knowledge of the system. A non-balanced partition could fit better to specific hardware architectures (e.g., when processor speeds differ between them). The system information is gathered before the actual FEM simulation begins using a configuration step. In case of dynamic resources, this step has to be performed before each computation step within the simulation. There exist libraries, such as LINPACK [125], that can be used for this purpose.

Our model works as follows. The first level is responsible for the main load-balancing steps. It performs the load distribution over the entire system, such as traditional models, before each computation step. We use additional information to compute the mesh partitioning and mapping. A graph is built from the available hardware information which represents the underlying system. Vertices represent processors and edges network links; both can be weighted to mimic the heterogeneity. Therefore, we use two graphs, one representing the mesh, and one the system. With this the extra information, a partition that better fits the system can be found. In this way, we are able to better distribute the load among the processors using well known libraries such as METIS and Jostle.

A similar cost model to the one proposed by Olas *et al*. [27] can be used to determine if a balancing step is required or not. If the time required by the load balancing step is smaller than the time that will be saved with a new distribution, then it is performed. We enhanced the model by adding additional information and handling the system heterogeneity. Instead of computing the communication time only by multiplying the amount of data to be transferred and the network speed, we take into account the

speed of each network link independently. The same is applied to the computing time. In this way, we have a more accurate prediction; therefore, the second level of load-balancing will provide better results.

The second level uses hardware information to perform a local load-balancing. It is not a separate step; instead it is performed during computations. First, we identify clusters of processors (groups of processors joint by high speed network links). This can be done during the configuration step before the FEM simulation (or during each configuration step before each computation step in a dynamic system). Second, we identify the mesh cells with numbers. These numbers represent the gain of moving the cell to a neighbor processor in the case of imbalance. This is done during the last global load-balancing step when the partition is refined. We keep these values and use them to improve local imbalance in this balancing level. As previously mentioned, the graph model does not represent the exact real workload. Thus, the imbalance may become evident during the computation step. According to the progress in solving PDEs by each processor, we can decide to move some mesh cells to a neighboring processor within the cluster of processors with high speed network links. Overloaded processors migrate mesh cells to neighbors during the computation step. This is done only if local predictions assure a gain in performance. As these communications are done concurrently and locally, the performance of the whole system is not degraded.

This approach solves some of the problems we have described previously. We believe that by tuning-up the cost functions used in predictions during the simulation, we can achieve better results. Including more information in the partitioning process may add complexity to the problem; but if used efficiently, a good improvement in performance can be achieved.

5 Conclusions and Future Work

In this paper, we presented an overview of efforts to improve current techniques of load-balancing and efficiency of FEM on large-scale parallel machines. Much work has been done in the field, but requirements of emerging technologies are not met by state-of-the-art libraries. We also introduced a multilevel load balancer to improve the local load imbalance. It is based on graph partitioning algorithms and takes into account the hardware architecture. We have presented an enhancement to the cost function used by Olas *et al*. [27] including new information, which helps to better approximate the computation and load-balancing costs of the next FEM computation step.

Our new model can successfully be used as a starting point for a more complex balancing strategy. It is still under development and comparison data is not available up to date. The research can be extended to a number of directions including the development of a more complex cost function, and prediction model into the multilevel load balancer.

Acknowledgment

This work was partially supported by CONACYT under grant number 309370.

References

1. **Blazy, S., Borchers, W., & Dralle, U. (1996).** Parallelization methods for a characteristic's pressure correction scheme. *Flow Simulation with High-Performance Computers: II*, *Notes on Numerical Fluid Mechanics*, 48, 305–321.
2. **Diekmann, R., Dralle, U., Neugebauer, F., & Römke, T. (1996).** PadFEM: A portable parallel FEM-tool. *High-Performance Computing and Networking*, *Lecture Notes in Computer Science,* 1067, 580–585.
3. **Olas, T., Karczewski, K., Tomas, A., & Wyrzykowski, R. (2002).** FEM computations on clusters using different models of parallel programming. *Parallel Processing and Applied Mathematics*, *Lecture Notes in Computer Science,* 2328, 170–182.
4. **Zienkiewicz, O.C. & Taylor, R.L. (2000).** *The finite element method* (5th ed.), vol.1. Oxford; Boston: Butterworth-Heinemann.
5. **Saad, Y. (2003).** *Iterative methods for sparse linear systems* (2nd ed.) Philadelphia: Society for Industrial and Applied Mathematics.
6. **Verfürth, R. (1994).** A posteriori error estimation

and adaptive mesh-refinement techniques. *Journal of Computational and Applied Mathematics*, 50(1-3), 67–83.

7. **Diekmann, R., Meyer, D., & Monien, B. (1995).** Parallel decomposition of unstructured FEM-meshes. *Parallel Algorithms for Irregularly Structured Problems*, *Lecture Notes in Computer Science,* 980, 199–215.
8. **Garey, M.R., Johnson, D.S., & Stockmeyer, L. (1976).** Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1(3), 237–267.
9. **Garey, M.R. & Johnson, D.S. (1979).** *Computers and intractability: A guide to the theory of NP-completeness*. San Francisco: W.H. Freeman.
10. **Diekmann, R., Monien, B., & Preis, R. (1995).** Using helpful sets to improve graph bisections. *Interconnection networks and mapping and scheduling parallel computations*, 21, 57–73.
11. **Farhat, C. (1988).** A simple and efficient automatic FEM domain decomposer. *Computers & Structures*, 28(5), 579–602.
12. **Hendrickson, B. & Leland, R. (1995).** An improved spectral graph partitioning algorithm for mapping parallel computations. *SIAM Journal on Scientific Computing*, 16(2), 452–469.
13. **Karypis, G. & Kumar, V. (1998).** A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1), 359–392.
14. **Pothen, A., Simon, H.D., & Liou, K.P. (1990).** Partitioning sparse matrices with eigenvectors of graphs. *SIAM Journal on Matrix Analysis and Applications*, 11(3), 430–452.
15. **Simon, H.D. (1991).** Partitioning of unstructured problems for parallel processing. *Computing Systems in Engineering*, 2(2–3), 135–148.
16. **Fiduccia, C.M. & Mattheyses, R.M. (1982).** A linear-time heuristic for improving network partitions. *19th Design Automation Conference*, Las Vegas, Nevada, 175–181.
17. **Kernighan B.W. & Lin, S. (1970).** An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 49(1), 291–307.
18. **Fox, G.C., Williams, R.D., & Messina, G.C. (1994).** *Parallel computing works!*. San Francisco, CA: Morgan Kaufmann Publishers, Inc.,
19. **Hülsemann, F., Kowarschik, M., Mohr, M., & Rüde, U. (2005).** Parallel geometric multigrid. *Numerical Solution of Partial Differential Equations on Parallel Computers*, *Lecture Notes in Computational Science and Engineering*, 51, 165–208.
20. **Oliker, L. & Biswas, R. (1998).** PLUM : Parallel load balancing for adaptive unstructured meshes. *Journal of Parallel and Distributed Computing*, 52(2), 50–177.
21. **Diekmann, R., Preis, R., Schlimbach, F., & Walshaw, C.H. (2000).** Shape-optimized mesh partitioning and load balancing for parallel adaptive FEM. *Parallel Computing*, 26(12), 1555–1581.
22. **Bhandarkar, M.A. & Kalé, L.V. (2000).** A parallel framework for explicit FEM. *7th International Conference on High Performance Computing*, Bangalore, India, 385–394.
23. **Stewart, J.R. & Edwards, H.C. (2003).** The SIERRA framework for developing advanced parallel mechanics applications. *Large-Scale PDE-Constrained Optimization*, *Lecture Notes in Computational Science and Engineering,* 30, 301–315.
24. Sandia National Laboratories, Trilinos. Retrieved from http://trilinos.sandia.gov/.
25. **Burstedde, C., Burtscher, M., Ghattas, O., Stadler, G., Tu, T., & Wilcox, L.C. (2009).** ALPS: A framework for parallel adaptive PDE solution. *Journal of Physics: Conference Series*, San Diego, California, 180.
26. **Wyrzykowski, R., Olas, T., & Sczygiol, N. (2001).** Object-oriented approach to finite element modeling on clusters. *Applied Parallel Computing. New Paradigms for HPC in Industry and Academia, Lecture Notes in Computer Science,* 1947, 250–257.
27. **Olas, T., Leśniak, R., Wyrzykowski, R., & Gepner, P. (2010).** Parallel adaptive finite element package with dynamic load balancing for 3D thermo-mechanical problems. *Parallel Processing and Applied Mathematics*, *Lecture Notes in Computer Science,* 6067, 299–311.
28. **Logg, A. & Wells, G.N. (2010).** DOLFIN: Automated finite element computing. *ACM Transactions on Mathematical Software*, 37(2), Article no. 20.
29. **Turek, S., Göddeke, D., Becker, C., Buijssen, S.H.M., & Wobker, H. (2010).** FEAST - Realization of hardware-oriented numerics for HPC simulations with finite elements. *Concurrency and Computation: Practice and Experience*, 22(16), 2247–2265.
30. **Langtangen, H.P. (2003).** *Computational partial differential equations: Numerical methods and Diffpack programming* (2nd ed.), 1, Berlin: Springer.
31. Sundance. Retrieved from http://www.math.ttu.edu/~kelong/Sundance/html/.
32. **Dular, P. & Geuzaine, C. (s.f.).** GetDP: A general

environment for the treatment of discrete problems. Retrieved from http://geuz.org/getdp/.

33. FreeFEM.org Retrieved from http://www.freefem.org/.

34. Prud'homme, C., Chabannes, V., & Feel++ Group (2011). Retrieved from https://forge.imag.fr/projects/life/.

35. Prud'homme, C. (2007). Life: Overview of a unified C++ implementation of the finite and spectral element methods in 1D, 2D and 3D. *Applied Parallel Computing. State of the Art in Scientific Computing*, *Lecture Notes in Computer Science*, 4699, 712–721.

36. Jiao, X., Li, X.Y., & Ma, X. (1999). SIFFEA: Scalable integrated framework for finite element analysis. *Computing in Object-Oriented Parallel Environments*, *Lecture Notes in Computer Science*, 1732, 84–95.

37. Heister, T., Kronbichler, M., & Bangerth, W. (2010). Massively parallel finite element programming. *Recent Advances in the Message Passing Interface*, *Lecture Notes in Computer Science*, 6305, 122–131.

38. Bruaset, A.M. & Langtangen, H.P. (1997). A comprehensive set of tools for solving partial differential equations; Diffpack. *Numerical Methods and Software Tools in Industrial Mathematics* (61–90), Boston, Mass: Birkhäuser.

39. Kirk, B.S., Peterson, J.W., Stogner, R.H., & Carey, G.F. (2006). libMesh: a C++ library for parallel adaptive mesh refinement/coarsening simulations. *Engineering with Computers*, 22(3-4), 237–254.

40. Renard, Y. & Pommier, J. (2004-2013). GetFEM++. Retrieved from http://download.gna.org/getfem/html/homepage/index.html.

41. Patzák, B. & Bittnar, Z. (2001). Design of object oriented finite element code. *Advances in Engineering Software*, 32(10-11), 759–767.

42. Logg, A. (2007). Automating the finite element method. *Archives of Computational Methods in Engineering*, 14(2), 93–138.

43. ANSYS. Retrieved from http://www.ansys.com/.

44. ESI Group. Retrieved from http://www.esi-group.com/.

45. TRANSVALOR Material forming simulation. (1984-2013). Retrieved from http://www.transvalor.com/.

46. Kalé, L., Skeel, R., Bhandarkar, M., Brunner, R., Gursoy, A., Krawetz, N., Phillips, J., Shinozaki, A., Varadarajan, K., & Schulten, K. (1999). NAMD2: Greater scalability for parallel molecular dynamics. *Journal of Computational Physics*, 151(1), 283–312.

47. Bangerth, W., Hartmann, R., & Kanschat, G. (2007). deal.II - A general-purpose object-oriented finite element library. *ACM Transactions on Mathematical Software*, 33(4).

48. Turek, S. (1999). *Efficient solvers for incompressible flow problems: An algorithmic and computational approach*. Berlin, Germany: Springer-Verlag.

49. Shivaratri, N.G., Krueger, P., & Singhal, M. (1992). Load distributing for locally distributed systems. *Computer*, 25(12), 33–44.

50. Devine, K.D., et al. (2005). New challenges in dynamic load balancing. *Applied Numerical Mathematics*, 52(2-3), 133–152.

51. Willebeek-LeMair, M.H. & Reeves, A.P. (1993). Strategies for dynamic load balancing on highly parallel computers. *IEEE Transactions on Parallel and Distributed Systems*, 4(9), 979–993.

52. Walshaw, C.H., Cross, M., & McManus, K. (2000). Multiphase mesh partitioning. *Applied Mathematical Modelling*. 25(2), 123–140.

53. Plimpton, S., Attaway, S., Hendrickson, B.A., Swegle, J., Vaughan, C., & Gardner, D. (1998). Parallel transient dynamics simulations: Algorithms for contact detection and smoothed particle hydrodynamics. *Journal of Parallel and Distributed Computing*, 50(1-2), 104–122.

54. Hendrickson, B.A. (1998). Graph partitioning and parallel solvers: Has the emperor no clothes?. *Proceedings of the 5th International Symposium on Solving Irregularly Structured Problems in Parallel*, 218-225.

55. Çatalyürek, Ü.V. & Aykanat, C. (1999). Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication. *IEEE Transactions on Parallel and Distributed System*, 10(7), 673–693.

56. Caldwell, A.E., Kahng, A.B., & Markov, I.L. (2000). Design and implementation of move-based heuristics for VLSI hypergraph partitioning. *Journal of Experimental Algorithmics*, 5.

57. Vastenhouw, R.H., & Bisseling, B. (2005). A two-dimensional data distribution method for parallel sparse matrix-vector multiplication. *SIAM Review*, 47(1), 67–95.

58. Chang, C., Kurc, T., Sussman, A., Çatalyürek, Ü.V., & Saltz, J. (2001). A hypergraph-based workload partitioning strategy for parallel data aggregation. *Proceedings of the Tenth SIAM Conference on Parallel Processing for Scientific Computing*.

59. Ozdal, M.M. & Aykanat, C. (2004). Hypergraph

models and algorithms for data-pattern-based clustering. *Data Mining and Knowledge Discovery*, 9(1), 29–57.

60. **Walshaw, C.H. & Cross, M. (2000).** Mesh partitioning: A multilevel balancing and refinement algorithm. *SIAM Journal on Scientific Computing*, 22(1), 63–80.
61. **Vanderstraeten, D. & Keunings, R. (1995).** Optimized partitioning of unstructured finite element meshes. *International Journal for Numerical Methods in Engineering*, 38(3), 433–450.
62. **Basermann, A., et al. (2000).** Dynamic load-balancing of finite element applications with the DRAMA library. *Applied Mathematical Modelling*, 25(2), 83–98.
63. **Goehring, T. & Saad, Y. (1994).** *Heuristic algorithms for automatic graph partitioning*. Minneapolis, U.S.A.
64. **Linde, Y., Buzo, A., & Gray, R.M. (1980).** An algorithm for vector quantizer design. *IEEE Transactions on Communications*, 28(1), 84–95.
65. **Walshaw, C.H., Cross, M., & Everett, M.G. (1995).** A localized algorithm for optimizing unstructured mesh partitions. *International Journal of High Performance Computing Applications*, 9(4), 280–295.
66. **Meyerhenke, H. & Schamberger, S. (2005).** Balancing parallel adaptive FEM computations by solving systems of linear equations. *Euro-Par 2005 Parallel Processing*, 3648, 624–624.
67. **Heath, M.T. & Raghavan, P. (1995).** A cartesian parallel nested dissection algorithm. *SIAM Journal on Matrix Analysis and Applications*, 16(1), 235–253.
68. **Miller, G.L., Teng, S.H., Thurston, W., & Vavasis, S.A. (1993).** Automatic mesh partitioning. *Graphs Theory and Sparse Matrix Computation*, 56, 57–84.
69. **Berger, M.J. & Bokhari, S.H. (1987).** A partitioning strategy for nonuniform problems on multiprocessors. *IEEE Transactions on Computers*, 36(5), 570–580.
70. **Taylor, V.E. & Nour-Omid, B. (1994).** A study of the factorization fill-in for a parallel implementation of the finite element method. *International Journal for Numerical Methods in Engineering*, 37(22), 3809–3823.
71. **Farhat, C., Lanteri, S., & Simon, H.D. (1995).** TOP/DOMDEC - A software tool for mesh partitioning and parallel processing. *Computing Systems in Engineering*, 6(1), 13–26.
72. **Horton, G. (1993).** A multi-level diffusion method for dynamic load balancing. *Parallel Computing*, 19(2), 209–218.
73. **Schamberger, S. (2005).** A shape optimizing load distribution heuristic for parallel adaptive FEM computations. *Parallel Computing Technologies*, 3606, 263–277.
74. **Cybenko, G. (1989).** Dynamic load balancing for distributed memory multiprocessors. *Journal of Parallel and Distributed Computing*, 7(2), 279–301.
75. **Liao, C.J. (1999).** *Efficient partitioning and load-balancing methods for finite element graphs on distributed memory multicomputers*. Feng Chia University, Seatwen, Taiwan.
76. **Elsässer, R., Monien, B., & Preis, R. (2002).** Diffusion schemes for load balancing on heterogeneous networks. *Theory of Computing Systems*, 35(3), 305–320.
77. **Schamberger, S. (2004).** On partitioning FEM graphs using diffusion. *Proceedings of the 18th International Parallel and Distributed Processing Symposium*, 277.
78. **Heirich, A. & Taylor, S. (1994).** *A parabolic load balancing method*. Pasadena, USA.
79. **Hendrickson, B.A. & Leland, R. (1995).** A multilevel algorithm for partitioning graphs. *Proceedings of the 1995 ACM/IEEE conference on Supercomputing (CDROM)*.
80. **Hendrickson, B.A. & Leland, R. (2012).** Chaco: Software for partitioning graphs. Retrieved from http://www.sandia.gov/~bahendr/chaco.html.
81. **Hendrickson, B.A. & Leland, R. (1995).** The Chaco user's guide: Version 2.0. Albuquerque, USA.
82. **Barnard, S.T. & Simon, H.D. (1994).** Fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems. *Concurrency: Practice and Experience*, 6(2), 101–117.
83. **Meyerhenke, H., Monien, B., & Schamberger, S. (2006).** Accelerating shape optimizing load balancing for parallel FEM simulations by algebraic multigrid. *Proceedings of the 20th IEEE International Parallel & Distributed Processing Symposium*, 10.
84. **Karypis, G. & Kumar, V. (1995).** Analysis of multilevel graph partitioning. *Proceedings of the 1995 ACM/IEEE conference on Supercomputing (CDROM)*, 29.
85. **Abou-Rjeili, A. & Karypis, G. (2006).** Multilevel algorithms for partitioning power-law graphs. *International Parallel and Distributed Processing Symposium*, 10.
86. **Karypis, G. & Kumar, V. (1995).** *Analysis of multilevel graph partitioning*. Minneapolis, USA.

87. **Simon, H.D. & Teng, S.H. (1993).** *How good is recursive bisection*. Moffett Field, USA.
88. **Karypis, G. & Kumar, V. (2012).** *METIS-Serial graph partitioning and fill-reducing matrix ordering*. Retrieved from http://glaros.dtc.umn.edu/gkhome/views/metis.
89. **Karypis, G. (2011).** *METIS A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices*. Minneapolis, USA.
90. **Pellegrini, F. (2012).** *SCOTCH: Static mapping, graph, mesh and hypergraph partitioning, and parallel and sequential sparse matrix ordering package*. Retrieved from http://www.labri.u-bordeaux.fr/perso/pelegrin/scotch/.
91. **Pellegrini, F. (2010).** *Scotch and libScotch 5.1 user's guide.* Talence, France,
92. **Pellegrini, F. & Roman, J. (1996).** SCOTCH: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs. *High-Performance Computing and Networking*, 1067, 493–498.
93. **Baños, R. & Gil, C. (2007).** Graph and mesh partitioning: An overview of the current state-of-the-art. *Mesh Partitioning Techniques and Domain Decomposition Methods (*1-26). Stirlingshire, U.K.: Saxe-Coburg Publications.
94. **Karypis, G. & Kumar, V. (1998).** Multilevel k-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed Computing*, 48(1), 96–29.
95. **Battiti, R. & Bertossi, A.A. (1999).** Greedy, prohibition, and reactive heuristics for graph partitioning. *IEEE Transactions on Computers*, 48(4), 361–385.
96. **Walshaw, C.H. (2012).** *JOSTLE - Graph partitioning software*. Retrieved from http://staffweb.cms.gre.ac.uk/~c.walshaw/jostle/.
97. **Walshaw, C.H. (2002).** *The serial JOSTLE library user guide : Version 3.0*. London, U.K.,
98. **Walshaw, C.H. & Cross, M. (2007).** JOSTLE: Parallel multilevel graph-partitioning software - an overview. *Mesh partitioning techniques and domain decomposition methods* (27-58). Stirlingshire, U.K.: Saxe-Coburg Publications.
99. **Karypis, G. & Kumar, V. (1998).** A parallel algorithm for multilevel graph partitioning and sparse matrix ordering. *Journal of Parallel and Distributed Computing*, 48, 71–85.
100. **Karypis, G. & Kumar, V. (1998).** Multilevel algorithms for multi-constraint graph partitioning. *Proceedings of the 1998 ACM/IEEE conference on Supercomputing*.
101. **Hu, Y.F., Blake, R.J., & Emerson, D.R. (1998).** An optimal migration algorithm for dynamic load balancing. *Concurrency: Practice and Experience*, 10(6), 467–483.
102. **Preis, R. (2012).** *PARTY Partitioning library*. Retrieved from http://www2.cs.uni-paderborn.de/cs/ag-monien/PERSONAL/ROBSY/party.html.
103. **Preis, R. (1998).** *The PARTY Graphpartitioning - Library - User manual - Version 1.99.* Paderborn, Germany.
104. **Preis, R. & Diekmann, R. (1997).** PARTY - A software library for graph partitioning. *Advances in Computational Mechanics with Parallel and Distributed Processing*, 63–71.
105. **Hromkovič, J. & Monien, B. (1991).** The bisection problem for graphs of degree 4 (configuring transputer systems). *Mathematical Foundations of Computer Science 1991*, 520, 211–220.
106. **Monien, B. & Preis, R. (2001).** Upper bounds on the bisection width of 3- and 4-regular graphs. *Mathematical Foundations of Computer Science 2001*, 2136, 524–536.
107. **Pellegrini, F. (1994).** Static mapping by dual recursive bipartitioning of process and architecture graphs. *Proceedings of the 1994 Scalable High-Performance Computing Conference*, 486–493.
108. **Karypis, G. (2012).** *hMETIS - Hypergraph & circuit partitioning.* Retrieved from http://glaros.dtc.umn.edu/gkhome/metis/hmetis/overview.
109. **Karypis, G. & Kumar, V. (1998).** *hMETIS - A hypergraph partitioning package - Version 1.5.3.* Minneapolis, U.S.A.,
110. **Çatalyürek, Ü.V. (2012).** PaToH v3.2. Retrieved from http://bmi.osu.edu/~umit/software.html.
111. **Çatalyürek, Ü.V. & Aykanat, C. (2011).** PaToH: Partitioning tool for hypergraphs. Columbus, USA.
112. **Bisseling, R. (2012).** *Mondriaan for sparse matrix partitioning*. Retrieved from http://www.staff.science.uu.nl/~bisse101/Mondriaan/mondriaan.html.
113. **Sandia National Laboratories (2012).** Zoltan: Parallel partitioning, load balancing and data-management services. Retrieved from http://www.cs.sandia.gov/Zoltan/.
114. **Devine, K.D., Boman, E.G., Heaphy, R.T., Hendrickson, B.A., & Vaughan, C. (2002).** Zoltan data management services for parallel dynamic applications. *Computing in Science Engineering*, 4(2), 90–96.
115. **Maerten, B., Roose, D., Basermann, A., Fingberg, J., & Lonsdale, G. (1999).** *DRAMA: A library for parallel dynamic load balancing of Finite element applications*. *Euro-Par 1999 Parallel*

Processing, 1685, 313–316.

116. **Faik, J., Flaherty, J. E., Gervasio, L.G., & Teresco, J.D. (2012).** *DRUM: The dynamic resource utilization model.* Retrieved from http://j.teresco.org/research/drum/.
117. **Faik, J. (2005).** *A model for resource-aware load balancing on heterogeneous and non-dedicated clusters*. Rensselaer Polytechnic Institute, Troy, USA.
118. **Chu, W.C., Yang, D.L., Yu, J.C., & Chung, Y.C. (2001).** UMPAL An unstructured mesh partitioner and load balancer on world wide web. *Journal of Information Science and Engineering*, 17(4), 595–614.
119. **Hu, Y.F. & Blake, R.J. (1995).** *An Optimal dynamic load balancing algorithm*. Daresbury, U. K.
120. **Burstedde, C., Wilcox, L.C., & Ghattas, O. (2011).** p4est: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees. *SIAM Journal on Scientific Computing*, 33(3), 1103–1133.
121. **Sinha, S. & Parashar, M. (2002).** Adaptive system sensitive partitioning of AMR applications on heterogeneous clusters. *Cluster Computing*, 5(4), 343–352.
122. **Walshaw, C.H. & Cross, M. (2001).** Multilevel mesh partitioning for heterogeneous communication networks. *Future Generation Computer Systems*, 17(5), 601–623.
123. **Minyard, T. & Kallinderis, Y. (2000).** Parallel load balancing for dynamic execution environments. *Computer Methods in Applied Mechanics and Engineering*, 189(4), 1295–1309.
124. **Teresco, J.D., Beall, M.W., Flaherty, J.E., & Shephard, M.S. (2000).** A hierarchical partition model for adaptive finite element computation. *Computer Methods in Applied Mechanics and Engineering*, 184(2-4), 269–285.
125. **Dongarra, J.J., Moler, C.B., Bunch, J.R., & Stewart, G.W. (1979).** *LINPACK User's guide*. Philadelphia, USA: Society for Industrial and Applied Mathematics.

José Luis González García is currently a Ph.D. student at Georg-August Göttingen University. He received his M.S. degree in Computer Science from CICESE Research Center (Centro de Investigación Científica y de Educación Superior de Ensenada, Baja California) in 2009.

Ramin Yahyapour is the executive director of the GWDG University of Göttingen. He has done research on Clouds, Grid and Service-oriented Infrastructures for several years. His research interest lies in resource management. He is a steering group member and on the Board of Directors in the Open Grid Forum. He has participated in several national and European research projects. Also, he is a scientific coordinator of the FP7 IP SLA@SOI and was a steering group member in the CoreGRID Network of Excellence.

Andrei Tchernykh is a researcher in the Computer Science Department, CICESE Research Center, Ensenada, Baja California, Mexico. From 1975 to 1990 he was with the Institute of Precise Mechanics and Computer Technology of the Russian Academy of Sciences (Moscow, Russia). He received his Ph.D. in Computer Science in 1986. In CICESE, he is a coordinator of the Parallel Computing Laboratory. He is a current member of the National System of Researchers of Mexico (SNI), Level II. He leads a number of national and international research projects. He is active in grid-cloud research with a focus on resource and energy optimization. He served as a program committee member of several professional conferences and a general co-chair for International Conferences on Parallel Computing Systems. His main interests include scheduling, load balancing, adaptive resource allocation, scalable energy-aware algorithms, green grid and cloud computing, eco-friendly P2P scheduling, multi-objective optimization, scheduling in real time systems, computational intelligence, heuristics and meta-heuristic, and incomplete information processing.

Article received on 25/02/2013; accepted on 27/07/2013.