Parallelism in Finite Element Programs

ION CÂRSTEA
Department of Computer Engineering and Communication
University of Craiova
Str. Doljului nr. 14, bl. C8c, sc.1, apt.7, Craiova
ROMANIA
DANIELA CÂRSTEA
High-School Group of Railways, Craiova
ROMANIA
ALEXANDRU ADRIAN CÂRSTEA
University of Craiova
ROMANIA

Abstract: –In this paper we present the parallelism facets in the numerical simulation of the distributed-parameter systems using the finite element method. As a parallel implementation of the finite element (FE) programs, the domain decomposition method is the best among three possible decomposition strategies for the parallel solution of partial derivative equations (PDE), namely, operator decomposition, function-space decomposition and domain decomposition. This is one of the motivations to present the inherent parallelism in finite element programs.

The principal objective of the paper is to describe some computational aspects for parallel computers in computer-aided design of the electromagnetic devices using the finite element method. For each stage of FE program there is an inherent parallelism of the algorithm that can be exploited on the new architectures.

Key-Words: - Parallel computing; Finite element method.

1 Introduction

The physical systems have the mathematical models a set of partial-derivatives equations and only in a first approximation we use lumped-parameter models [4]. Although a tremendous variety of parallel numerical methods have been proposed for simulation of these systems, the most recently invented parallel computational strategies are largely based on the finite element (FE) and multigrid methods [3]. The programs for the simulation of the distributed-parameter systems have an inherent parallelism when FE method is used [6].

The technique of dividing a large physical system into a system of components is very old and is still used extensively ([1], [2], [13]). In this way different components are designed in parallel by different groups of researchers or companies. It is obviously that this traditional approach can be used with parallel computers if the complete finite element system is subdivided so that each group of elements within a small domain is assigned to one processor.

In this paper we present several parallel computational strategies for the FE applications [8]. The fact that the finite element method is central to many modern engineering simulations constitutes a real motivation for its consideration in this work. Another motivation is based upon some of the algorithmic issues raised by the FE method in comparison with other methods like the difference finite methods. The FE method can handle discrete meshes with an irregular or complicated distribution of points. The matrix of linear equation coefficients has not a regular, predictable structure characteristic of the finite difference method. Another main motivation to consider the FE method is the existence of a large amount of software developed for conventional computers based on it. The justification of this large amount of software products in this area consists in the facility with which the FE method can be used to handle many physical problems described by partial differential derivative equations.

FE methods for the solution of elliptic partial differential equations using unstructured meshes

have been in use for a large number of physics problems [4]. We shall limit our discussion to the parallel solution of elliptic partial differential equations (PDE) in 2D space using a distributed unstructured mesh of triangles with linear approximations within an element (piecewise linear elements), although the ideas presented here do extend to higher order elements [17].

Problems vary widely in the degree to which parallelism is evident in the problem itself [14]. Thus, we can have some categories of parallelism: obvious, discovered and implicit parallelism.

Obvious parallelism is evident in the problem by its nature and it is encountered in the problems involving large regular data spaces. If the problem is inherently parallel, it is simpler and more natural to follow the parallelism in its solution.

Discovered parallelism. In other problems we must seek the parallelism in an approach to a solution. Typically, the algorithm for problem solution is chosen for specific computer architecture.

The *implicit parallelism* is detected and extracted automatically by the compilers. The user programs are developed in a language without explicit parallel features. The advantage of this type is the portability.

In the multiprocessor systems, the performance of the computation is influenced by some factors as:

- type of problem parallelism;
- methods used for decomposing a problem into tasks;
- allocation of tasks to microprocessors;
- granularity of the tasks;
- possibility of overlapping communication with processing;
- data-access method;
- architecture of the host system;
- speed of processors, memories and interconnection work.

The performance of a computer program depends both on the method used to solve a specific problem - known as the algorithm and on the skill with which the algorithm is implemented on the computer by the programmer or compiler during the operation of coding [11]. The best way to obtain high-performance code is obtained if the parallelism in the algorithm matches the parallelism of the architecture so that the details of programming for any particular architecture are issues that must be considered in an efficient algorithm [5].

In many engineering problems we identify an obvious parallelism so that the use the new architectures is a natural approach [10]. More, in the context of the finite element method there is a natural parallelism so that we do no effort to discover it. These are some main motivations to discuss the parallelism in finite element programs.

2 Finite element programs

The FE program may be divided into three distinct stages [4] :

- pre-processing
- solution (or processing)
- post-processing.

The complexity of each phase lead to a modular development of the FE programs. In the first stage the spatial domain is divided in elements (subdomains) of a desired form. The result of this stage is a database for the next stages. This database contains a large amount of data with respect to geometrical and the physical properties of the elements. Interactive programs can do it with a graphical interface for a control of the mesh topology.

In the second stage, called solution, a system of equations is obtained by approximation of the unknowns in each element, system that is solved approximately. This stage involves some phases like: the selection of interpolation functions for each element, computation of the element equations, assembling the element equations in a global system and the solution of the global system by direct or iterative methods.

The third stage has a single goal: to transform the abstract results of the processing stage in data with engineering meanings. It is obviously that there is a limited parallelism in this stage so that we limit our discussion to the first stages.

3 Parallelism facets in FE programs

Parallelism is obviously in every stage of the FE program and these parallelism facets we present in brief.

3.1.Parallelism facets in pre-processing

The pre-processing stage can be done in parallel although the unstructured meshes arise some problems in the parallel implementation of this stage. Mesh generation can be done in parallel using different algorithms [4]. The general principle consists in the division of the original

spatial domain in a number of regions that are assigned at different processors. Each processor generates a submesh (using an algorithm as Delaunay algorithm), and finally the global mesh is obtained by joining the submeshes. Parallel generation of a mesh is a constrained problem. A requirement of the generation algorithm is to minimise the number of generated nodes that lie on the boundary between regions on different processors. A generation algorithm is influenced by the approach used in the solution phase of FE program.

Many software products in this area use the adaptive mesh generators. In the first step, a coarse mesh is generated and the problem is solved with this mesh. Afterwards, an error estimator is run in order to estimate the errors over each element and, if it is the case, a new mesh is generated by a refinement technique. In this direction the following strategies are used: h-strategy (when elements are refined); p-strategy (when the order of the polynomial approximations of the unknowns is increased); h-p strategy (a combination of the previous strategies).

The parallel refinement techniques consists in the following steps [4] :

- Partitioning the coarse mesh using an efficient algorithm
- Distribution of the mesh partitions to the processors. It is possible that each processor to have a number of subregions to deal with.
- Independent refinement of the submeshes by each processor.

In this refinement technique, each processor needs a list of local vertices, which are shared with neighbouring subdomains. The requirement of keeping a constraint join between the different regions, which are meshed independently, is the main constraint.

As a final conclusion, the FE mesh itself can be generated in parallel, with each processor generating a portion of the mesh, which it was allocated during the partitioning phase. Refinement techniques can be used if it is necessary.

3.2.Paralelism facets in the solution stage

For the solution of FE problems, typical processing involve the following model of computation [7]:

- computation of local system of equations (elemental matrices)
- assembly into global matrix
- solution of the global equations system.

The operations involved in calculating the elemental systems for the different elements are independent, and therefore can be done simultaneously. In the global assembly only minor utilisation of parallelism is feasible but with the conjugate-gradient (CG) methods, there is no need for a global assembly step.

In the parallel implementation of FE solution, a large number of efficient solvers were developed, especially for specific problem classes. These solvers essentially rely on a set of basic kernel procedures (or routines) that consists of matrix-vector product, scalar product of two vectors and preconditioning operations. The key to performance of these solvers is the efficiency of the kernel procedures and the communication it requires.

An important problem of any implementation is the database structure and the management of this database [12]. Typical serial FE applications lead to the equations systems with a sparse matrix. This matrix can be stored in one of two general ways: in an element-by-element scheme or, as fully summed equations. In the first approach, each element matrix is stored separately and is not summed with contributions from neighbouring elements. In this way all matrix-vector operations are performed with elemental matrices and the global vector is obtained only after summing over all elements. In a parallel implementation this scheme can be efficient but increases the required storage and floating point operations. The second approach sums the elemental matrices at the beginning of the linear solver so that the drawbacks of the first approach are eliminated.

There are a lot of schemes to storage a sparse matrix, each of them having advantages and drawbacks. These schemes are influenced by the architecture of the host system [4]. In a parallel application on distributed-memory MIMD machines, the matrix may be partitioned [6]. The goal of partitioning is to produce load-balancing partitions. The methods used in this area can be included into one of the following classes: geometric methods and graph methods. Geometric methods use information about the geometric distribution of vertices or elements of meshes that are being partitioned. Graph-based methods are based on the interconnectivity of the vertices and elements to realise partitioning. Both classes of methods can be subdivides in local or global methods. Global methods use information from the entire geometric domain or graph to produce partitions, whereas local methods use only a small

neighbourhood of the vertex under consideration to produce the next candidate partition.

To measure the performance of the partitioning strategy it is necessary to introduce adequate metrics [18]. In accordance with the above classification, the metrics can be local or global. Some possible local metrics are the number of messages per boundary exchanged by the processors and the size of these messages. In a graph-based method the size of message is directly proportional to the number of edges cut by the inter-processor boundary in the connectivity graph. A global metrics can be the total amount of data transmitted between the processors. This sum can be obtained by the number of edges cut by inter-processor boundaries in the graph partition. If the processors are not connected directly, the global metrics can be obtained by the number of data transmitted weighted by the number of links they traverse.

3.2.1.The equations solvers

In the solution of FE equations we can use either direct methods (like Gauss method, LU decomposition, Choleski method etc.), either iterative methods (like the gradient methods). These methods are presented in many books of numerical analysis so that it is not the case to present them in detail [4]. We limit our discussion at a particular one for each class ([6], [12]).

A. Iterative equation solvers

When solving large sparse systems of equations, it is usual to use iterative techniques. These techniques can yield solution in less time than the direct methods. Because there is large amount software based on CG method we present in brief this method. We consider our target example that leads to solving a system Ax=b where A is a **n** ***X*** **n** matrix. The global matrix A and the right -hand side b are assumed to have been constructed, and will remain unchanged by the algorithm. The iteration number will be indicated by a subscript. By **r** we denote the residual vector and p the conjugate search direction.

The sequential algorithm in pseudo-code based on CG method is as follows [6]:

1.Set initial approximation vector x_0
2.Calculate initial residual $r_0=b-A.x_0$.
3.Set initial search direction $p_0=r_0$
4.For i=0 to n-1 do
calculate coefficient

$$\alpha_i = (p_i^T . r_i) / (p_i^T . Ap_i)$$

set new estimate $x_{i+1}=x_i+ \alpha_i (A.p_I)$
evaluate new residual $r_{i+1}=r_i- \alpha_i (A.p_i)$
calculate coefficient
$\beta_I =-(r_{i+1} . A\, p_i)/(p^T_i .A.p_i)$
determine new direction $p_{i+1}=r_{i+1}+ \beta_i p_I$
Convergence test

In this method the direction p_i and p_j are conjugated, that is

$$p_i^T . A. p_j = 0, i \neq j$$

$$r_i^T . r_j = 0, i \neq j$$

By the superscript "T" is denoted the transpose of a vector and by "." is denoted the scalar product of two vectors.

The iterations are terminated when x_i has converged within desired accuracy, as determined by the magnitude of residuals vector **r**. The CG method requires the matrix A to be symmetric and positive definite and these conditions are satisfied by the FE matrix of our target example. The main loop of a sequential program for CG method, using procedures and functions for the basic operations is:

```
repeat
    multiply_matrix_vector(A,p,y)
    suma:=scalar_product(p,y);
    alpha:=scalar_product(p,y);
    update_x(alpha,p,x);
    update_r(alpha,y,r);
    beta:=-scalar_product(r,y)/suma;
    update_p(beta,r,p);
    test_convergence(value_test);
until value_test
```

If we examine the above algorithm, we can identify a number of potentially parallel arithmetic operations. These operations refer to the product of a matrix A and vector p, the scalar product of two vectors and the updating of the vector components for x and r. Note that in CG algorithm the matrix A appears in a matrix-vector product. In order to perform this product it is not necessary to generate the global matrix A since the element contributions to any such product may be computed separately and summed afterward. In the sequential variant this fact is not so important but in a parallel implementation it becomes vital. A given processor may need access to parts of A residing in another processor.

A parallel implementation of a CG to solve the equation can be obtained easily. We can assign at every processor the computations of the entries of

the matrix belonging to an element (a fine-grain parallelism), or a subdomain (more elements) in a coarse-grain parallelism. In each iteration it is necessary to exchange information only when global scalar quantities are computed.

In the steps of CG algorithm we meet two basic operations: the vector inner product and the matrix-vector product. In the case of vector inner product, the parallel implementation is easily. Each processor is given responsibility for a subset of the elements which comprises a vector such r or p. Each processor calculates its contribution to the inner product and the final result is obtained by summing the processors contributions via a combination phase. For load balancing, the processors are given equal numbers of elements.

Let us consider the computation of the product **A.p**. A given row (or column) of the matrix A contains nonzero entries only for the points that share a common element. Consequently, a given entry of the vector **A.p** makes use of the information that is spatially local to the corresponding mesh point. Therefore it must group the elements in processors according to spatial grouping of elements. In the computation of the matrix-vector product it is not necessary to assemble the matrix A in its global form but it is necessary to exchange data between processors that contain points on the boundary of a processor's subdomain. So that the calculation of the **A.p** product involves two steps:

- computations for all elements internal to a given processor
- communication which brings in the boundary **A.p** contributions

The communication must be designed in such a way to additively accumulate all element contributions to a given point and to distribute the result to all processors, which share that nodal point. In order to obtain an efficient implementation, the communicating processors must agree on the number and kind of shared boundary nodes along their common boundaries.

The design of the procedures, which incorporate the inter-processor communication step, is very important for the efficiency of the algorithm [18]. These procedures refer to the inner product of two vectors and the procedure that combines neighbouring processor parts of **A.p**.

B.Direct equation solvers - Gauss algorithm

The Gauss algorithm is well-known in the solution of the FE equations among the direct methods [16]. It is useful to start by briefly describing the basic approach of Gauss method before detailing a particular one. Let A.x=b be a linear system of equations, where A is a dense square matrix of order n, and b a vector with N components. For simplicity of discussion we consider the matrix A augmented with the right-hand side b, that is, we let A:=(A, b) be a matrix of dimension $n \times (n+1)$.

The Gauss algorithm involves two phases: forward reduction and backward substitution. The initial system is transformed into an equivalent triangular system T.x=c (this is the forward reduction). This transformation consists in n-1 steps. At each step k, we zero out the elements of column k, which are below the main diagonal. For this we add multiple of row k to each row i, $k+1 \le i \le n$. In the backward substitution, we solve the triangular system, starting with the last component of x and terminating with the first component.

In the following discussion we consider only the forward reduction. The generic formulation of the Gauss algorithm is

For k:=1 to n-1 do
For i:=k+1 to n do
$a_{ik} := a_{ik}/a_{kk}$
For j:=k+1 to n+1 do
$a_{ij} := a_{ij} - a_{ik} * a_{kj}$

If we examine the above algorithm for a parallel implementation, we find a number of potentially parallel arithmetic operations. The first of these operations is the dividing all elements of the column elements a_{ik} by the pivot element a_{kk}. On a given entry there are m<n operations implied in this transformation so that if we have p processors available and p>m, all these operations can take place simultaneously. The second operation that candidates at parallelism is the updating of the matrix columns.

A practical implementation of the Gauss algorithm depends by the communication network of the host computer so that we limit our discussion to general aspects. There is a rich literature for particular implementations of the algorithm on a specific architecture ([3], [16]).

3.3. A parallel implementation based on domain decomposition

As we have mentioned above, the FE method includes several phases, such as generation of the input data, direct or iterative solution of system of equations and post-processing, each of them having an inherent parallelism, and can be implemented on SIMD or MIMD platforms.

The first stage in developing algorithms for parallel systems is to identify the source of potential parallelism in problem. In most cases this task is the responsibility of the programmer. The parallel algorithm must include mechanisms for performing the division of processing, allocating the sub-problems to the available processors and the collecting the results in order to obtain the final results.

For the advanced architectures, new algorithms must be developed and the domain decomposition techniques are powerful iterative methods that are promising for parallel computation ([9], [10], [15]). Ideal numerical models are those that can be divided into independent tasks, each of which can be executed independently on a processor. Obviously, it is impossible to define totally independent tasks because the tasks are so inter-coupled that it is not known how to break them apart. However, algorithmic skeletons were developed in this direction that enables the problem to be decomposed among different processors. The mathematical relationship between the computed sub-domain solutions and the global solution is difficult to be defined in a general approach.

4 Conclusions

There is no general design method for developing the parallel algorithms to any specific problem. The act of algorithm design is considered an art and may never be fully automated. Every possible algorithm provides a clear specification of the architecture used, the data structures chosen and the mode of synchronisation and communication as well as the computation and communication complexities.

In principle each stage of FE has an inherent parallelism and might require different parallel architectures for optimal efficiency. The various existing parallel computers differ with respect a lot of elements (the number of processors, network topology, memory management), so that the choice of the architecture is a difficult task. The communication in a parallel computer plays an important role for the global performance of the system. Which the architecture is good for a specific problem is an open problem.

Problem size and complexity, architecture and algorithm designs are also interrelated and do this choice a main problem. For this reason we limited our discussion to architectures MIMD. The algorithms depend on the communication network topology so that we present the general aspects of the parallelism for the two stages of a FE program: pre-processing and processing.

References:

[1]. Alonso, A., Valli, A. "A domain decomposition approach for heterogeneous time harmonic Maxwell equations." In: *Computer methods in applied mechanics and engineering,* 143 (1997), pg. 97-112

[2]. Adeli H. (Editor), *Parallel Processing in Computational Mechanics*, Marcel Dekker Inc.,N.Y.

[3]. Bertsekas D.P., Tsitsiklis J.N., *Parallel and Distributed computation. Numerical Methods*. Prentice-Hall Inc., New Jersey 1989.

[4]. Cârstea, D., Cârstea, I. *"CAD of the electromagnetic devices. The finite element method."* Editor: Sitech, 2000. Romania.

[5]. Cole M. *Algorithmic Skeletons: Structured Management of Parallel Computation* , MIT Press , 1989.

[6]. Fox, G., Johnson, M., *Solving problems on concurrent processors ,* Prentice Hall, 1988.

[7]. Golub G., Ortega M.J., Scientific *Computing. An Introduction with Parallel Computing.* Academic Press Inc., 1993.

[8]. Hinton, E., Jowen, D.R., *An introduction to finite element computations.* Pineridge Press Limited, Swansea, 1980, UK.

[9]. Hodgson, D.C., Jimack, P.K., "A domain decomposition preconditioner for a parallel finite element solver on distributed unstructured grids". *Parallel Computing* (23), 1997 1157-1181. NH Elsevier.

[10]. Jaja, J., *An introduction to parallel algorithms*. Addison-Wesley Publishing Company, Inc., 1992.

[11]. Krishnamurthy, E.V., *Parallel Processing*, Addison Wesley Pub. Company, 1988.

[12]. Ortega, J.M., *Introduction to parallel and vector solution of linear systems*. Plenum Press, New York, 1988.

[13]. Paglieri, L., and others, "Parallel computation for shallow water flow: A domain decomposition approach". *Parallel computing* 23 (1997) 1261-1277, NH Elsevier.

[14]. Perrot R.H. (Editor), *Software for Parallel Computers*, Chapman & Hall, London, 1992.

[15]. Quarteroni, A., Valli, A. *"Domain Decomposition Methods for Partial Differential Equations"*. Oxford Science Publication. 1999

[16]. Robert, Y., *The impact of vector and parallel architectures on the Gaussian elimination algorithm.* Halstad Press, 1992.

[17]. Segerlind.L.J., *Applied Element Analysis*, John Wiley and Sons, 1984, USA.
[18]. Tabak, D., *Multiprocessors* , Prentice Hall Inc. 1990.