A Web-Based Parallel PDE Solver Generation System for Distributed Memory Computing Environments

Chao-Jen Lee and Yeh-Ching Chung

Department of Information Engineering
Feng Chia University, Taichunh, Taiwan 407, ROC
Tel: 886-4-4517250 x3700
Fax:886-4-4516161
Email: {super,ychung}@iecs.fcu.edu.tw

Abstract-The finite element method is widely applied to many domains, such as engineering, atmology, oceanography, biology, etc. The major drawback of the finite element method is that its execution takes a lot of time and memory spaces. Due to the computation-intensiveness and computation-locality properties, we can use the parallel processing method to improve the performance of the finite element method on distributed memory computing environments. However, it is quite difficult to program the finite element method on a distributed memory computing environment. Therefore, the development of a front-end parallel partial differential equations solver generation system is important. In this paper, we want to develop a front-end parallel partial differential equations solver generation system based on the World Wide Web on a distributed-memory computing environment, such as a PC cluster, a workstation cluster, etc. With the system, users who want to use parallel computers to solver partial differential equations can use web browser to input data and parameters. The system will automatically generate the corresponding parallel codes and execute the codes on the distributed memory computing environment. The execution result will be shown on the web browser. The results can also be download by user.

Index Terms: partitioner, refiner, load balancer, finite element method, solver.

1. Introduction

The finite element method (FEM) has been widely used for the structural modeling of physical systems [2]. To solve problems using FEM on a distributed memory parallel computer, in general, we first need to establish the finite element model of the problem. Usually, the model could be a 2D or 3D finite element graph (FEG), which is a connected and undirected graph that consists of a number of finite elements. Each finite element is composed of a number of nodes. The second step, we need to partition the FEG into N subgraphs such that each subgraph has the same amount of computational load and the communication cost among subgraphs is minimized, where N is the number of processors that we use to solve the problem. The third step is to write the parallel codes for each processor according to the data of subgraphs assigned to it. Then we can get the results by running the program on the machine. Traditionally, for those tasks mentioned above require a lot of manual effort and experiences for a user who is familiar to the parallel programming. However, for those users whose backgrounds are in Electronic Engineering, Mechanical Engineering, Physics, etc., those tasks are very difficult for them to perform since they usually rely on software tools to help them solving finite element modeling problems.

To efficiently execute a finite element application program on a distributed memory multicomputer, we need to map nodes of the corresponding mesh to processors of a distributed memory multicomputer such that each processor has the same amount of computational load and the communication among processors is minimized. Since this mapping problem is known to be NP-completeness [10], many heuristics were proposed to find satisfactory sub-optimal solutions [9,11,13,16-17,19-20,23]. Based on these heuristics, many graph partitioners were developed [16-17,20,23]. Among them, Jostle [25], Metis [16], and Party [20] are considered as the best graph partitioners available up-to-date.

If the number of nodes of a mesh will not be increased during the execution of a finite element application program, the mapping algorithm only needs to be performed once. For an adaptive mesh application program, the number of nodes will be increased discretely due to the

refinement of some finite elements during the execution of an adaptive mesh application program. This will result in load imbalance of processors. A load-balancing algorithm has to be performed many times in order to balance the computational load of processors while keeping the communication cost among processors as low as possible. To deal with the load imbalance problem of an adaptive mesh computation, many load-balancing methods have been proposed in the literature [4-8,12,14-15,19,23,25].

Due to this complicated process and huge amount of efforts involved in parallel programming, currently, programmers often reuse existing codes as far as possible. Many research efforts have already provided solid algorithms heading to the sections of FEG partitioning, load balancing, and MPI code packing in various domains. Unfortunately most of them only offer fragmental assistance. Users still need a lot of manual effort in order to complete the whole process of parallel programming. As a result, the cost of parallel programming is still very high and not cost effective. In this paper, we present a parallel partial differential equations solver generator on World Wide Web. This system is an integrated tool that consists of eight components, a partitioner, a load balancer, a simulator, a visualization tool, a refiner, a generator, a executor and a Web interface. Through the Web interface, other seven components can be operated independently or can be cooperated with others. Besides, the system provides several demonstration examples and their corresponding models that allow beginners to download and experiment. The design of the system is based on the criteria including easy to use, efficiency, and transparency.

2. Related Work

An environment consisting of a set of tools, internal framework, and the encoding of design knowledge and patterns that can greatly reduce the complexity of programming and increase productivity. The *distributed irregular mesh environment* (DIME) [26] is an environment for doing distributed calculations with unstructured triangular meshes. The mesh covers a 2D manifold, whose boundaries may be defined by straight lines, arcs of circles, or Bezier cubic sections. In also provides functions for creating, manipulating, and refining unstructured triangular meshes. Similarly, Archimedes [24] and SUMAA3D [22] are two environments that support more phases of FEM parallel programming process.

However, DIME is a close system, which can't take input from other systems, and has a limitation on the number of nodes (limited to 10000 nodes), which may be very limited to the analysis of some real world objects. The Archimedes system is missing refinement and load balancing functionality. Although the SUMMAA3D support functionality that cover the entire process except load balancing, it is currently restricted to solving some particular FEM problems and not general enough. It also misses the load balancing function. The DIME, Archimedes and our PPGE are systems that support compiler in their systems.

Most of current tools offer GUI facility but some still do not. However, most of current tools do not support the facilities of load balancing and executor. Only environments like DIME, Archimedes, SUMMAA3D and our system provide an executor. For the offering of solvers, the DIME only support sequential solvers, while the Archimedes and SUMMAA3D systems support both sequential and parallel solvers for either PDEs or linear equation.

Our system is able to collect the history of execution results and generate various performance analysis statistics reports to support the performance analysis from different algorithms. Other tools may only offer very limited analysis reports, which may need more manual efforts in the performance comparison.

From the viewpoint of the integration with existing algorithms, our system is very unique in its adoption with so many existing algorithms. In addition, its capability of dynamic change of different algorithms during the operation has made our system a very efficient tool to help the analysis of FEM. Although several search have been implemented as tools or libraries, none of them has offered its Web interface and high level support to users.

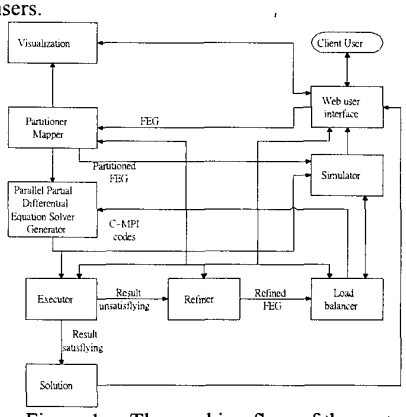


Figure 1. The working flow of the system.

3. The System Structure

The system structure of the web-based parallel PDE solver generation system is show in Figure1. The system consists of eight components, a partitioner, a load balancer, a simulator, a visualization tool, a refiner, a executor, a solver generator and a Web interface. Users can upload the unstructured mesh data and get the running results on any Web browsers. Through the Web interface, other seven components can be operated independently or can be cooperated with others. In the following, we will describe them in details.

3.1 The Partitioner

In the partitioner, we provide three partitioning methods, Jostle/DDM, Metis/DDM, and Party/DDM. Jostle/DDM, Metis/DDM, and Party/DDM were implemented based on the best algorithms provided in Jostle, Metis and Party, respectively, with the dynamic diffusion optimization method (DDM) [7]. The partitioner of the system has the following advantages:

1. In Jostle, Metis, and Party, 3% to 5% load imbalance among partitioned modules is allowed. The dynamic diffusion optimization method can efficiently balance the 3% to 5% load imbalance among partitioned modules allowed by these three methods and improve the total cut-edges of partitioned modules. Therefore, the partition methods provided in the partitioner will perform better than their counterparts.
2. The partition results of Jostle, Metis, and Party depend on the shapes of unstructured meshes. It is difficult to tell that which one performs the best for a given unstructured mesh. If we want to get the best result from these three partitioners for an unstructured mesh, we need to run these three partitioners separately. Since the parameters used in these three partitioners are different, it may take some time to get the desire results. By integrating Jostle/DDM, Metis/DDM, and Party/DDM methods in a partitioner, one can try each method once and take the best partitioning result easily because the parameters for these three methods are uniform in the system.

The outputs of the partitioner are a partitioned unstructured mesh file and partitioned results. In a partitioned unstructured mesh file, a number j in line i indicates that node i belongs to processor j. Users can download the partitioned unstructured mesh file for further use and see the partitioned results on a Web browser. The partitioned results include the load balancing degree and the total cut-edges of a partitioned unstructured mesh.

3.2 The Load Balancer

In the load balancer, we provide two load-balancing methods, the prefix code matching parallel load-balancing (PCMPLB) method [5] and the binomial tree based parallel load-balancing (BINOTPLB) method [5]. The inputs of the load balancer are the number of processors and files of the connection model, the element model, and the partitioned model of an unstructured mesh. The number of processors specifies how many processors will be involved in the load balancing process. The data format of the partitioned model file of an unstructured mesh is the same as that of the output file of the partitioner. In the load balancer, users can also use the partitioned unstructured demo mesh model provided by the system. In this case, the inputs are the load imbalance degree and the number of processors. The outputs of the load balancer are a load-balanced unstructured mesh file and the load balancing results. Users can download the load-balanced unstructured mesh file for further use and see the load balancing results on a Web browser. The load balancing results include the load balancing degree and the total cut-edges.

3.3 The Simulator

The simulator provides a simulated distributed memory multicomputer for the performance evaluation of a partitioned unstructured mesh. The execution time of an unstructured mesh on a P-processor distributed memory multicomputer under a particular mapping/load-balancing method L_i can be defined as follows:

$$T_{par}(L_i)=max\{T_{comp}(L_i, P_j)+T_{comm}(L_i, P_j)\}, \quad (1)$$

where $T_{par}(L_i)$ is the execution time of an unstructured mesh on a distributed memory multicomputer under L_i, $T_{comp}(L_i, P_j)$ is the computation cost of processor P_j under L_i, and $T_{comm}(L_i, P_j)$ is the communication cost of processor P_j under L_i, where $j = 0, ..., P-1$.

The cost model used in Equation 1 is assuming a synchronous communication mode in which each processor goes through a computation phase followed by a communication phase. Therefore, the computation cost of processor P_j under a mapping/load-balancing method L_i can be

defined as follows:

$T_{comp}(L_i, P_j) = S \times load_i(P_j) \times T_{task}$, (2)

where S is the number of iterations performed by a finite element method, $load_i(P_j)$ is the number of nodes of an unstructured mesh assigned to processor P_j, and T_{task} is the time for a processor to execute tasks of a node.

For the communication model, we assume a synchronous communication mode and every two processors can communicate with each other in one step. In general, it is possible to overlap communication with computation. In this case, $T_{comm}(L_i, P_j)$ may not always reflect the true communication cost since it would be partially overlapped with computation. However, $T_{comm}(L_i, P_j)$ should provide a good estimate for the communication cost. Since we use a synchronous communication mode, $T_{comm}(L_i, P_j)$ can be defined as follows:

$T_{comm}(L_i, P_j) = S \times (\delta \times T_{setup} + \phi \times T_c)$, (3)

where S is the number of iterations performed by a finite element method, δ is the number of processors that processor P_j has to send data to in each iteration, T_{setup} is the setup time of the I/O channel, ϕ is the total number of data that processor P_j has to send out in each iteration, and T_c is the data transmission time of the I/O channel per byte.

To use the simulator, users need to input the partitioned or load-balanced unstructured mesh file and the values of S, T_{setup}, T_c, T_{task}, and the number of bytes sent by a finite element node to its neighbor nodes. The partitioned or load-balanced unstructured mesh file can be uploaded from a user's browser or can be a demo file provided by the system. The outputs of the simulator are the execution time of the unstructured mesh on a simulated distributed memory multicomputer and the total cut-edges of a partitioned unstructured mesh.

3.4 The Visualization Tool

The system also provides a visualization tool for users to visual the partitioned unstructured mesh. The inputs of the visualization tool are files of the coordinate model, the element model, and the partitioned unstructured mesh models of an unstructured mesh, and the size of an image. After rendering, a Web browser displays the unstructured mesh with different colors, and each color represents one processor. Currently, the visualization tool can only display partitioned 2D meshes. The visualization of partitioned 3D meshes is still under development.

3.5 The PDE Solver Generator

The PDE solver generator contains a set of partial differential equations solvers that distinct our system from other tools by enabling the dynamic selection of different solvers. This flexibility makes the selection of appropriate solver algorithms much easier and cost-effective. The PDE solvers generator is responsible to generate the corresponding C+MPI codes of the selected solver. Current the PDE solver generator provides a solver - Laplacs. The methods that help solve PDEs have also been incorporated in PDESG, including Conjugate Gradient, Preconditional Conjugae Gradient, Gauss-Seidel, Direct Gaussian Elemination, Jocobi Iterative, and Successive Over Relaxation.

3.6The Executor

The task of the executor is to compile the generated solver program and automatically load it to distributed parallel machines for execution. Since our solver program is written in C+MPI, the executor uses either MPICH [3] or WinMPI [18] to compile the solver program. The major platform of the system has focused on workstation clusters, PC clusters, or IBM SP2. The execution is based on SPMD (Single Process Multiple Data) model.

3.7 The Refiner

In order to utilize finite element to solve PDE efficiently, we usually refine finite element mesh to mince every element's area to fast make solution converge an approximate answer. So mesh refiner improves rate of finding partial differential equation answer. Since the results by solving the input FEG may not be always acceptable, the refiner is used to further refine the input FEG. The system has adopted and implemented the regular refinement algorithm proposed by Bank *et al.* [1] and the bisection refinement algorithm [21].

3.8 The Implementation of the system

In order to support standard WWW browsers, the front end is coded in HTML with CGI. The CGI interface is implemented in Perl language. The CGI interface receives the data and parameters from the forms of the HTML interface. It then calls external tools to handle requests. The tools of the system, partitioner, balancer, refiner, executor, solver generator, and simulator are coded in C language and MPI+C codes. They receive the parameters from the CGI interface and use the specified methods (functions) to process requests

of users.

To support an interactive visualization tool, the client/server software architecture is used in this system. In the client side, a Java Applet is implemented to display images rendered by server. In the server side, a Java serverlet is implemented as a Java Application. The Java serverlet renders image with specific image size and unstructured mesh models. As the server finishes its rendering work, it sends the final image to client side and users can see the final image from users' Web browsers.

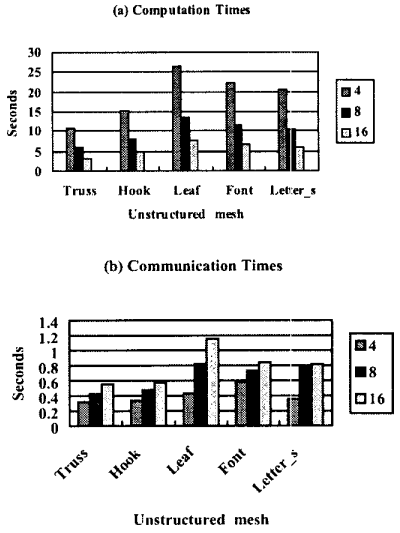
4.Experimental Results

In this section, we will present some experimental results for unstructured meshes by using the solver. To evaluate the performance of solver and five unstructured meshes are used as tested samples. These five unstructured meshes are part of demo meshes provided in the system. The number of nodes, the number of elements, and the number of edges of these five unstructured meshes are given in Table 1. Figure 2 shows the result of test samples run on IBM SP2 parallel machine with 4,8,16 processors. In this experimental test, the execution of a parallel Laplace solver is evaluated. The number of iterations performed by a Laplace solver is set to 100 by using the Jacobi method. Table 2 shows the result of test samples at 4, 8, 16 processors. According to Figure 2, we found that the solver execution time of 4 processors for all test samples. From Table 2, we can say that the parallel codes generated by the system are scaled well for the number of processors used for the experimental test, that is, the system does generate good parallel codes for the corresponding meshes.

Samples	#node	#element	#edges
Hook	80494	158979	239471
Letter_S	106215	126569	316221
Truss	57081	91968	169518
Font	116043	230444	346486
Leaf	138292	274816	413107

Table 1. The number of nodes, elements, and edges of the test samples.

Figure 2. The result of execution of the unstructured meshes in IBM SP2.



5.Conclusions

In this paper, we have presented a Wed-based PDE solver generation system for distributed memory computing environment. The system consists of eight components, a partitioner, a load balancer, a simulator, a visualization tool, a refiner, a executor, a solver generator, and a Web interface. The design of the system is based on the criteria including easy to use, efficiency, and transparency. By using our system through WWW, the boundary of using tools has been released. Users can use this tool without going through the tedious steps of installation on their machines provided that they are permitted to own a copy. The further maintenance and upgrade of this tool become much cost effective and the inconsistency of the usage of this tool can be minimized because only one version of this tool needs to be maintained. The integration of different methods into our system has made the experiments and simulations of parallel programs very simple and cost effective. The system offers a very high level and user friendly interface. Besides, the demonstration examples can educate beginners on how to apply FEM to solve parallel problems.

References

[1]. R.E. Bank, A. H. Sherman, and A. Weiser, "Refinement Algorithms and Data Structures for Regular Local Mesh Refinement, " in *Scientific Computing*, R. Stepleman et al., ed., Amsterdam: IMACS/North-Holland Publishing Company, pp. 3-17, 1983.

[2]. R. Biswas and L. Oliker, "Load Balancing Unstructured Adaptive Grids for CFD Problems," *Proceedings of the 8th SIAM Conference on*

Parallel Processing for Scientific Computing, Minneapolis, MN, March 14-17, 1997.

[3]. P. Bridges, N. Doss, W. Gropp, E. Karrels, E. Lusk, and A. Skjellum, *"User's Guide to mpich, a Portable Implementation of MPI"*, Oct. 1995.

[4]. Yeh-Ching Chung and Ching-Jung Liao, "Tree-Based Parallel Load-Balancing Methods for Solution-Adaptive Unstructured Finite Element Models on Distributed Memory Multicomputers, " *Lecture Notes in Computer Science 1457*, Springer, pp. 92-103, 1998.

[5]. Yeh-Ching Chung and Ching-Jung Liao, "A Prefix Code Matching Parallel Load-Balancing Method for Solution-Adaptive Unstructured Finite Element Graphs on Distributed Memory Multicomputers, " *IEEE Proceedings of the 1998 International Conference on Parallel Processing*, pp.510-517, Minneapolis, Minnesota, 1998.

[6]. Yeh-Ching Chung and Ching-Jung Liao, "A Binomial Tree-Based Parallel Load-Balancing Methods for Solution-Adaptive Unstructured Finite Element Graphs on Distributed Memory Multicomputers, " *Proceedings of 1998 International Conference on Parallel CFD.*

[7]. Yeh-Ching Chung, Ching-Jung Liao, Chih-Chang Chen, and Don-Lin Yang "A Dynamic Diffusion Optimization Method for Irregular FEG Partitioning, " *to appear in IASTED Proceedings of the 1998 International Conference on Parallel and Distributed Computing and Networks.*

[8]. G. Cybenko, "Dynamic Load Balancing for Distributed Memory Multiprocessors," *Journal of Parallel and Distributed Computing*, Vol. 7, No. 2, pp. 279-301, Oct. 1989.

[9]. F. Ercal, J. Ramanujam, and P. Sadayappan, "Task Allocation onto a Hypercube by Recursive Mincut Bipartitioning," *Journal of Parallel and Distributed Computing*, Vol. 10, pp. 35-44, 1990.

[10]. M.R. Garey and D.S. Johnson, Computers and Intractability, A Guide to Theory of NP-Completeness. San Francisco, CA: Freeman, 1979.

[11]. J.R. Gilbert, G.L. Miller, and S.H. Teng, "Geometric Mesh Partitioning: Implementation and Experiments," *Proceeding of 9th International Parallel Processing Symposium*, Santa Barbara, California, pp. 418-427, Apr. 1995.

[12]. A. Heirich and S. Taylor, "A Parabolic Load Balancing Method," *Proceeding of ICPP'95*, pp. 192-202, 1995.

[13]. B. Hendrickson and R. Leland, "An Improved Spectral Graph Partitioning Algorithm for Mapping Parallel Computations," *SIAM Journal on Scientific Computing*, Vol. 16, No.2, pp. 452-469, 1995.

[14]. G. Horton, "A Multi-level Diffusion Method for Dynamic Load Balancing," *Parallel Computing*, Vol. 19, pp. 209-218, 1993.

[15]. Y. F. Hu and R. J. Blake, *An Optimal Dynamic Load Balancing Algorithm*, Technical Report DL-P-95-011, Daresbury Laboratory, Warrington, UK, 1995.

[16]. G. Karypis and V. Kumar, "Multilevel *k*-way Partitioning Scheme for Irregular Graphs," *Joulrnal of Parallel and Distributed Computing*, Vol. 48, No. 1, pp. 96-129, Jan. 1998.

[17]. G. Karypis and V. Kumar, "A Parallel Algorithm for Multilevel Graph Partitioning and Sparse Matrix Ordering," *Journal of Parallel and Distributed Computing*, Vol. 48, No. 1, pp. 71-95, Jan. 1998.

[18]. J. Meyer, "Message-Passing Interface for Microsoft Windows 3.1," Master Thesis of Department of Computer Science, University of Nebraska at Omaha, Dec. 1994.

[19]. C.W. Ou and S. Ranka, "Parallel Incremental Graph Partitioning," *IEEE Trans. Parallel and Distributed Systems*, Vol. 8, No. 8, pp. 884-896, Aug. 1997.

[20]. R. Preis and R. Diekmann, *The PARTY Partitioning – Library User Guide – Version 1.1*, HENIZ NIXDORF INSTITUTE Universität Paderborn, Germany, Sep. 1996.

[21]. M.C. Rivara, "Mesh Rrefinement Processes Based on The Generalized Bisection of Simplicies, " *SIAM Journal of Numerical Analysis*, Vol. 21, pp. 604-613, 1984.

[22]. *Scalable Unstructured Mesh Algorithms and Application*, Argonne National Laboratory, http://www.mcs.anl.gov/sumaa3d.

[23]. K. Schloegel, G. Karypis, and V. Kumar, "Multilevel Diffusion Schemes for Repartitioning of Adaptive Meshes," *Journal of Parallel and Distributed Computing*, Vol. 47, No. 2, pp. 109-124, Dec. 1997.

[24]. J.R. Shewchuk and D.R. O'Hallaron, *Archimedes: A system for unstructured PDE problems on parallel computers*, Carnegie Mellon University, Pittsburgh, PA, http://www.cs.cmu.edu/afs/cs.cmu.edu/project/quake/public/www/archimedes.html.

[25]. C.H. Walshaw, M. Cross, and M.G. Everett, "Parallel Dynamic Graph Partitioning for Adaptive Unstructured Meshes," *Journal of Parallel and Distributed Computing*, Vol. 47, No. 2, pp. 102-108, Dec. 1997.

[26]. R.D. Williams, *DIME: Distributed Irregular Mesh Environment*, California Institute of Technology, 1990.