EVOLUTIONARY METHODS FOR DESIGN, OPTIMISATION AND CONTROL
K. Giannakoglou, D. Tsahalis, J. Periaux, K. Papailiou and T. Fogarty (Eds.)
© CIMNE, Barcelona, Spain 2002

A FAST GA-BASED DATA PARTITIONER FOR USE IN PARALLEL CFD.

A. P. Giotis, P.I.K. Liakopoulos, D. G. Koubogiannis and K. C. Giannakoglou
Lab. of Thermal Turbomachines, Mechanical Engineering Department,
National Technical University of Athens,
P.O. Box 64069, 15710 Athens, Greece.
e-mail: kgianna@central.ntua.gr

Abstract. This paper describes the implementation of Genetic Algorithms (GA) for the optimal partitioning of huge data sets (like for instance huge computational grids) into subsets, in order to support parallel Computational Fluid Dynamics (CFD) tools and in particular those involving adaptive grids. In the applications we are dealing with, a GA-based partitioner undertakes the optimal redefinition of subdomains in conformity with a parallel flow solver which involves the regular adaptation of 3D unstructured grids to the developing flow field. Through the proposed GA-based partitioner, the grid repartitioning task, though it runs sequentially, it bears almost negligible computational cost and doesn't sell out the gain from parallelization.

Key words: Genetic Algorithms, Unstructured Grid Partitioner, Parallel CFD, Adaptive grids

1 INTRODUCTION

Nowadays, grid-based methods for the numerical solution of p.d.e.'s in large scale physical problems are fully operational provided that the available computer memory is sufficient and the necessary computing time is affordable.

By interconnecting a great number of processors and decomposing the whole computational task into subtasks associated with each of the processors, engineers may numerically analyse many problems for which the solution on single-processor computers is not feasible. For the grid-based methods under discussion, multiple data subsets corresponding to grid subdomains, should be defined prior to their assignement to each one of the networked processors.

For the partitioning of grids, methods which provide evenly sized subdomains (ensuring evenly loaded processors) and minimum number of grid elements over the interfaces (ensuring minimum communication during the parallel execution) are due. The CPU cost for creating the partitions is in general of lower importance. However, the partitioning cost turns out to be crucial whenever grids change in the course of the numerical solution and this is the case of numerical solvers relying on grid adaptation.

Grid adaptation exists almost in every modern CFD code, especially those using unstructured grids. In this kind of solvers, the grid is regularly adapted to the evolving solution and, consequently, requires grid and data repartitioning in order not to damage the parallel efficiency.

This paper demonstrates the use of GA for optimal grid partitioning into subdomains with almost negligible computing cost. Working with 3-D unstructured grids which periodically readapt by embedding new elements, to the developing transonic flow, the GA-based partitioner is employed in a sequential manner and to redetermines the loading per processor after each adaptation. It will be shown that, even if the partitioning is carried out by a single processor, the parallel efficiency remains very satisfactory, thanks to the low CPU-cost of the partitioner itself.

2 THE GA-BASED GRID PARTITIONER

Though this paper is dealing with 3-D unstructured grids with tetrahedral elements, for a better comprehension of what follows the GA-based partitioner will be presented by using 2D grids with triangular elements. Regardless its dimension(2D OR 3D) any grid can be transformed into an equivalent graph. Since this method works with the equivalent graph, it can be generalized to cover any grid type including structured, unstructured or hybrid ones.

The grid or graph partitioner,introduced in previous works by the same authors[1],[4] may create 2^n subdomains through a tree-like scheme with recursive bisections. Such a scheme relies upon the use of the same basic tool(the one that undertakes the bisection of a graph or grid); the basic tool is employed as many times as implied by the tree-like scheme. On the other hand, each bisection is carried out using a multilevel scheme, where the graph is, at first, coarsened several times, then the coarsest graph is partitioned through the GA-based tool and, finally, there is a reverse procedure for going from the coarsest graph to the finest(initial) one through the refinement of its interface. The latter is carried out through heuristics.

In what follows, the basic tool(i.e the scheme employed for each bisection) will be presented. Let us denote G^0 the graph corresponding to the grid which is to be split into two subdomains. The terms “nodes” and “edges” will refer to graph nodes (i.e. triangles, tetrahedra etc.) and graph edges (i.e. grid edges or triangular faces shared by adjacent tetrahedra that belong to different partitions).

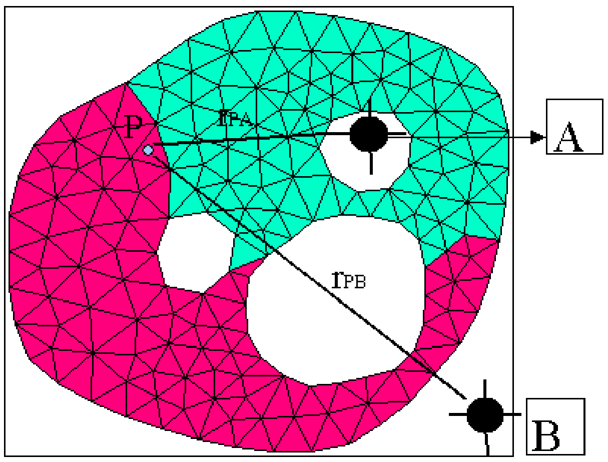

Figure 1: Schematic presentation of the partitioning of a 2D unstructured grid in two subdomains through the control of “two point charges”.

Graph G^0 is first coarsened and this coarsening is of primary importance. Starting from the initial graph G^0 a sequence of graphs of smaller size is created, through the repetitive application of a simple coarsening procedure. During each coarsening step, graph nodes are clustered into groups. The total number M of coarsening steps is an outcome of the whole procedure and it depends on the user-defined final graph size. During the coarsening, weights are assigned to graph nodes and edges. At G^0, all graph nodes or edges are given unit weights. Proceeding from level G^m to G^{m+1}, the weight of each newly formed graph node is set equal to the sum of the weights of the constituent nodes. For the edges, new weights are computed due to the coarsening by taking into account the number of graph edges linking the adjacent groups.

The coarsening procedure works as follow: all G^m nodes are visited one-by-one, in random order. The edges emanating from a node are sorted by their weights. The edge having the maximum weight is selected and the two supporting nodes collapse and form a new G^{m+1} node; its weight equals to the sum of the weights of the two G^m nodes. At the end, the remaining G^m nodes are directly upgraded to G^{m+1} ones, with the same weights. This method tends to maximize the weights of the collapsed graph edges and consequently to minimize the weights of the active G^{m+1} edges.[1]

The graph partitioner operates on a properly defined parametric space. For the initial graph G^0, all nodes are given the cartesian coordinates of the barycenter of the corresponding grid element. At the coarser levels $m > 1$ the graph nodes are relocated to the mean of the coordinates of the constituent nodes.

At the coarsest graph level(G^m) the bisection is cast in the form of an optimizing problem. For this purpose two point-charges A and B are assumed to float within predefined limits creating potential fields $F(x, y)$, or $F(x, y, z)$ in the 3D case, around them. Ue to these changes, at any graph node P the local potential value F_P is computed by, readily, superimposing scalar contributions from both of them. The field created by each point charge may be governed by various mathematical expressions. Among them, in this work an exponentially decaying law will be used, according to which the F_P value is computed as follows:

$$F_P = F(x, y, z) = e^{k_A r_{P_A}} - e^{k_B r_{P_B}} \qquad (1)$$

where r_A and r_B are the distances of the corresponding point-charge from the graph node P.

In general, the minimum limits of the graph space over which A and B are allowed to float are the extreme coordinate components of the graph vertices, but this is of minor importance.

Given the location of charges and the values of k_A and k_B and after having computed the potential values F_P over all of the graph nodes, these values are first rank sorted. Then, the upper half of them are assigned to the first domain whereas the other to the second one. By doing so two evenly loaded subdomains are defined and the quality of the so-created bisection is to be judged upon the interface length, as the first requirement is automatically fulfiled. Among the different partitions that can be created in this way, th optimal one is the one having the minimum inteface. As a consequence an optimization method needs to be employed for the search of the optimal solution. Herein, a standard GA bears the burden of searching for the values $(k_B, x_A, y_A, z_A, x_B, y_B, z_B)$ which minimize the interface of the partition. Note that when dealing with 2D grids, two of the design variables (z_A, z_B) are eliminated and the number of degrees of freedom is reduced by two. Over and above, in either 2D or 3D grids, one of (K_A, K_B) coefficients can be defined arbitrarily; so, we define $k_A = -1$. Consequently, the degrees of freedom of only 5 (for 2D grids) or 7 (for3D grids). Such a reduced number is extremely advantageous in the GA-based opyimization. Using the GA for minimization, a cost function should be defined and this is, of course, equal to the number of graph edgescut by the separator. We recall that the first requirement (evenly loaded subdomains) is automatically satisfied.

3 THE FLOW SOLVER WITH GRID ADAPTATION FEATURES.

A primitive variable flow solver for compressible flows was posted to a multiprocessor system using the SIMD (Single-Instruction-Multiple Data) technique and the PVM[?] message exchange protocol. The solution method employes a time-marching scheme and second order upwinding for the convection scheme, by sweeping basically along the grid edge. The grid partitioning in subdomains is carried out so that the number of triangles (i.e. quantities which were previously reffered to as graph nodes.) be evenly distributed. At their interfaces. grid nodes are shared between two or more subdomains and the interface length is measured in terms of the interfacial grid edges.

The grid adaptation to the evolving flow solution is based on the grid edges' marking for refinement according to predifined sensor functions and a couple of criteria, based on predefined threshold values. Each tetrahedron is allowed to yield either two, four or eight offspring tetrahedra depending on the number of its marked edges (1,3 or 6). Several rules have been laid down in order to avoid modifying the initial grid, creating hanging nodes or going through an endless refinement-derefinement procedure, etc.

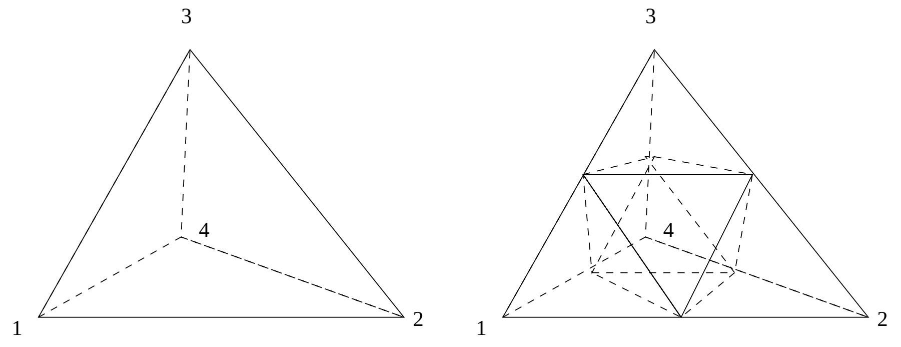

Figure 2: An example: A tetrahedron with all of its edges being marked for refinement produces eight offspring tetrahedra.

Though the numerical intergration of the discretized governing equation is carried out in parallel, both grid adaptation and grid repartitioning are performed by a single processor (the so called master processor, i.e. the same which undertakes data I/O or pre- and post- processing). Thus, from the algorithmic point of view, the master processor recollects the computed values of the flow quantities from all the processors, Then, the same processor employes the refinement criteria. edges are marked and new tetrahedra are defined. Before splitting up the new topology again over the available processors, the master processor uses the GA-based partitioner for the optimal repartitioning of data. The scheme is employed regularly during the iterative solution scheme.

4 METHOD APPLICATION

Parallel CFD tool with grid adaptation and the GA-based grid partitioner was employed for the computation of the supersonic flow within a channel. This channel had an area reduction by a fifteen degree compression corner followed by an equal expansion corner. The Mach number of the flow was equal to two (2). A shock was formed at the compression corner and continued by bouncing off the channel walls. Due to these shock features, there was a need for grid adaptation.[?]

The oveall procedure consists of both sequential and parallel parts. At the beginning of algorithm, a master process takes over the partitioning of the initial grid in a sequential manner and distributes the data to the slave processes. The latter carry out the solution of the flow equations in a parallel mode. When adaptation is going to take place, the slave processes provide the appropriate data to the master process in order to refine the grid and re-partition it sequentially. The data are again redeployed among the processes and the whole procedure is iterated. The end of the cyclic procedure depends on the desired accuracy of the numerical solution.

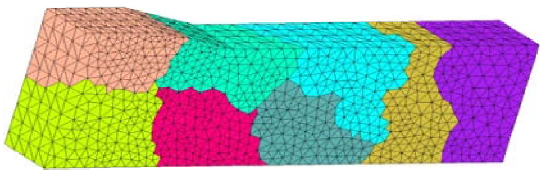
Figure 3: The Intitial Grid after Partitioning(16128 tet.).

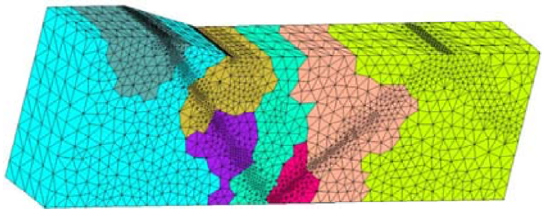
Figure 4: Partitioning after 1st adaptation step(90463 tet.).

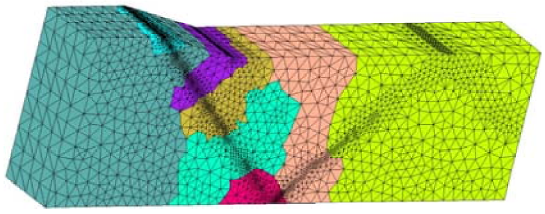
Figure 5: Partitioning after 3rd adaptation step(167959 tet.).

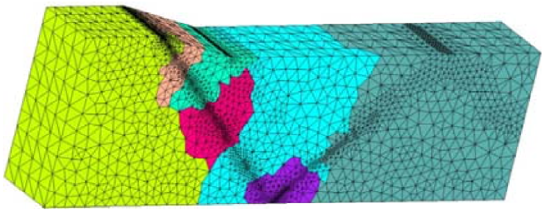
Figure 6: Partitioning after 4th adaptation step(192370 tet.).

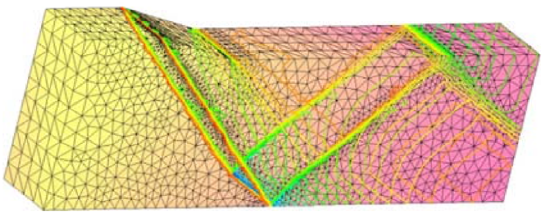
Figure 7: Mach Isolines after the 3rd adaptation step.

Figures 3 to 6 show four "instants" of the adapted grid as this evolves during the numerical solution of the 3D Euler equation. In the same figures, the eight subdomains created by the GA-partitioner are also shown in colours (grey scales). All ths partitions ensure optimal work loading between processors. Figure 7 shows the computed Mach isolines at the end of the 3rd adaptation cycle.

5 CONCLUSIONS

This paper demonstrates the use of Genetic Algorithm as an optimization tool for grid partitioning purposes, in order to support parallel CFD codes with grid adaptation. The concept of the proposed partitioner defines a small number of free parameters and, consequently, the GA is proved to undertake the search for their optimal set of values noticeably low computing cost. Despite the fact that the GA-based grid partitioner is used by a single processor, this does not damage the parallel speed-up values thanks to its low computing cost.

Although the partitioner tool was presented in this paper through its use with CFD tools it can also be used for various applications in Computational Electromagnetics (CEM), Computational Structural Mechanics (CSM) etc.

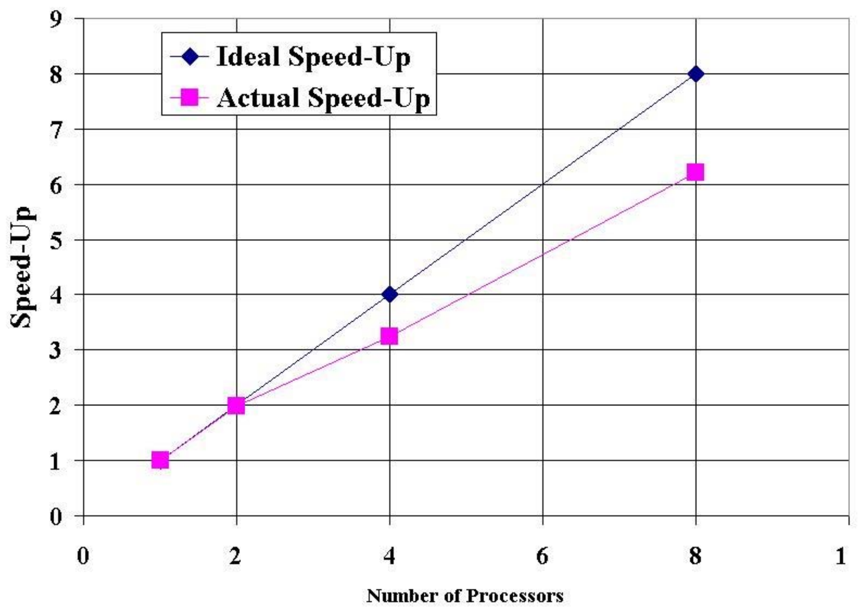

Figure 8: Speed-Up of the parallel Euler solver, with adaptation.

Acknowledgments. Parts of the present research have been separately funded by Dassault Aviation, France and the General Secretariat for Research and Technology, Greece (Program Velos).

REFERENCES

[1] A.P. Giotis and K.C. Giannakoglou, Advances in Engineering Software 29 No. 2 (1998) 128.

[2] D.G. Koubogiannis, K.C. Giannakoglou and K.D. Papailiou, ECCOMAS 98, Proccedings of the European Computational Fluid Dynamics Conference, John Wiley & Sons, 2 (1998) 171.

[3] D.G. Koubogiannis, L.C. Poussoulidis, D.V. Rovas and Giannakoglou K.C., Comp. Meth. in Appl. Mech. and Eng. 160 (1998) 89.

[4] K.C. Giannakoglou and A.P Giotis, ECCOMAS 98, Proccedings of the European Computational Fluid Dynamics Conference, John Wiley & Sons, 2 (1998) 186.