1 EFFICIENT PARTITIONING METHODS FOR 3-D UNSTRUCTURED GRIDS USING GENETIC ALGORITHMS

A.P. Giotis, K.C. Giannakoglou, [†] B. Mantel and J. Periaux [‡]

New partitioning methods for 3-D unstructured grids based on Genetic Algorithms will be presented. They are based on two partitioning kernels enhanced by of acceleration and heuristic techniques to effectively partition grids, with reduced CPU cost. Five variants are presented and assessed using three 3-D unstructured grids.

1.1 INTRODUCTION

"Real" world problems which are modeled through partial differential equations require huge meshes and CPU-demanding computations to reach a converged numerical solution. Multi-processing is a way to overcome the high computing cost. Using distributed memory computing platforms, data need to be partitioned to the processing units, so that evenly work-loaded processors with minimum exchange of data handle them concurrently.

In this paper, we are dealing with 3-D unstructured grids with tetrahedral elements, widely used in Computational Fluid Dynamics (CFD). Since there is always a mapping of any grid to a graph (see fig. 1.1), the proposed partitioners are general. Working with the equivalent graph, grids with other types of elements (quadrangles, hexaedra, etc) or even hybrid grids can be readily partitioned. Apart from CFD, the grid or graph partitioning problem is recurrent in various disciplines, structural analysis, linear programming etc. In the literature, the UGP problem is handled using greedy methods, coordinate-based bisections, inertial methods or recursive spectral bisection techniques, as discussed in [5]. Effective unstructured grid partitioners based on Genetic Algorithms (*GAs* [1]) have been presented in [4] and [3], where two partitioning

[†]National Technical University of Athens, Greece
[‡]Dassault Aviation, France

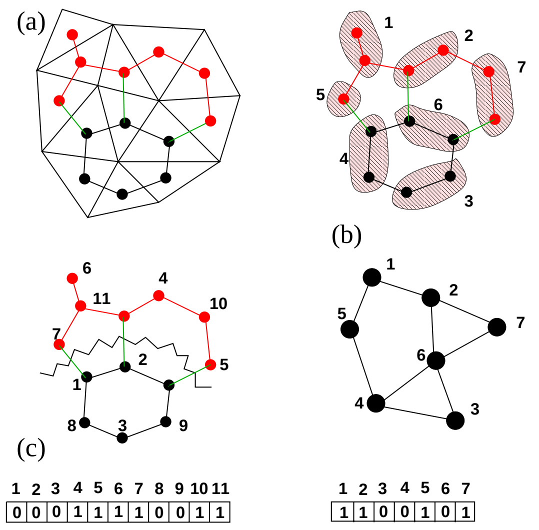

Fig. 1.1
(a) A simple 2-D unstructured grid and its equivalent graph,
(b) a one-step coarsening procedure (and binary encoding at the coarse level),
(c) Bisection encoding through the *BITMAP* method.

kernels (in what follows they will be refered to as *BITMAP* and *FIELD*) have been introduced.

Based on the aforementioned kernels and heuristic methods, new partitioning algorithms have been devised. All of them use the recursive approach and the single-pass multilevel scheme. They employ new coarsening algorithms and, by appropriately scheduling the UGP kernels, they proved to be very effective partitioners. A noticeable advantage of the new partitioners is that they are faster, by at least one order of magnitude, than the methods presented in [4] and [3]. The new UGP methods will be demonstrated in the partitioning of 3D unstructured meshes, where tetrahedron-based partitions will be sought for.

1.2 BASIC PARTITIONING TOOLS

In what follows, we will be dealing only with the equivalent graph of the unstructured grid. So, the terms “nodes” and “edges” will refer to graph nodes (i.e. tetrahedral elements) and graph edges (i.e. triangular faces separating adjacent elements), respectively.

1.2.1 Recursive Bisections and the Multilevel Scheme

Starting from the graph to be partitioned G^0 (Fig. 1.1a), a multilevel partitioner creates a sequence of graphs of smaller size, through the repetitive application of a coarsening procedure. During each coarsening step, graph nodes are clustered into groups, as in Fig.1.1b. The total number M of coarsening steps is determined by the G^0 size and the desirable final graph size. With the same initial and (approximate) final graph sizes, different coarsening algorithms lead to different values of M. It is recommended that G^M should have about 50–150 nodes. During the coarsening procedure, weights are assigned to graph nodes and edges. At G^0, the weights for all graph nodes or edges are set to 1. Proceeding from level G^m to G^{m+1}, the weight of each newly formed graph node equals the sum of the weights of the constituent nodes. For the edges, new weights need to be computed due to the coarsening by taking into account the number of graph edges linking adjacent groups.

Edge-based and node-based coarsening procedures have been employed and tested in various test cases. Some of them are described, in detail, in [2]. Some other have been programmed using similar considerations, but they will be omitted in the interest of space. In general, edge-based schemes outperform the node-based ones. The most important features of the best coarsening procedure tested so far will be described below.

All G^m nodes are visited one-by-one, in random order. The edges emanating from a node and pointing to another G^m 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 with weight equal 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. Since the sought for separator will cut active G^{m+1} edges, separators of reduced length will be produced.

After series of tests, a major conclusion was that the coarsening procedure considerably affects the quality of the separator as well as the partitioning cost. Inappropriate coarsening is a serious threat to the partitioner's effectiveness, as it leads to “poor” coarse graphs. The optimum separator of a “poor” G^M is usually far from the optimum separator of G^0 and CPU-time consuming refinements at the finer graph levels should be employed. Unfortunately, the fast partitions of the coarser graphs cannot outweight the slow refinements at the finer ones.

1.2.2 Cost Function

The cost function used for a single bisection should combine the requirements of equally loaded subdomains and of minimum interface. Assume that the bisection of a graph with N nodes gives rise to two subdomains with N_1 and N_2 nodes ($N_1 + N_2 = N$), respectively. Let N_{ie} be the number of graph edges cut by the separator. A cost function that scalarizes two objectives into a single one, reads

$$C_f(N_1, N_2, N_{ie}) = \frac{|N_1 - N_2|}{\sqrt{N_1 N_2}} + w \frac{N_{ie}}{N_{ge}} \tag{1.1}$$

where N_{ge} is the total number of graph edges and w is a weighting parameter. The optimum partition is, of course, the one that minimizes $C_f(N_1, N_2, N_{ie})$.

1.2.3 The *BITMAP* Bisection Method

The *BITMAP* method [4] is fully compatible with the binary encoding often used in *GAs*. It is based on the direct mapping of each graph node to a single gene or bit in the chromosome, as depicted in fig. 1.1c. Genes marked with 0 or 1 denote graph nodes that belong to the one or the other partition. Concerning the genetic operations, a standard two-point crossover and a modified mutation operator [4] are employed. In each generation, the modified mutation operator takes into account the graph connectivity and the actual shape of the separator by assigning increased mutation probabilities to the graph nodes which are closer to the separator. Tests have shown that a one-way mutation (only 0's turning to 1's, or vice-versa, are allowed during each generation) with variable probability enhance the method characteristics. Elitism is also used along with the correction scheme that applies whenever small groups of graph nodes fully surrounded by nodes belonging to another subdomain are found. The parent selection operations are carried out in part by linear fitness ranking and in part by probabilistic tournament. The cost function used in *BITMAP* is the one described in eq. 1.1. For the computation of the cost function at any level $m \neq 0$, the graph nodes' and edges' weights are considered.

1.2.4 The *FIELD* Bisection Method

The so-called *FIELD* method requires the mapping of the graph onto a parametric space, which could be carried out using different techniques. Two of them are described below.

Method P1: In the parametric space, each G^0 graph node is given the Cartesian coordinates of the barycenter of the corresponding grid element. At the coarser levels $m, (m \neq 0)$ the average Cartesian coordinates of the "constituent" elements are assigned to any graph node.

Method P2: Random $0 \leq \Phi, \Psi, \Omega \leq 1$ coordinates are firstly assigned to each graph node. Then, a Laplacian filter, is applied to smooth down the random mapping, without enforcing fixed (Φ, Ψ, Ω) values at the "boundary" nodes. The filter is iterative and each node is given the average of the adjacent nodes' (Φ, Ψ, Ω) values. Only a few iterations should be carried out, since the full convergence leads to a uniform (Φ, Ψ, Ω) field for the entire graph. The number of iterations required should be carefully chosen. Insufficient smoothing often results to multiply connected partitions.

In both parameterizations, the (Φ, Ψ, Ω) coordinates of any node are such that $0 \leq \Phi, \Psi, \Omega \leq 1$ defining thus a cubic space which encloses the graph.

In the parametric space (Φ, Ψ, Ω), two point-charges A and B are allowed to float within predefined limits, creating potential fields $F(\Phi, \Psi, \Omega)$ around them. At any point P in the (Φ, Ψ, Ω) space, the local potential value F_P is computed by superimposing scalar contributions from the charges. Different laws can be used. Among them, two laws with good physical reasoning are

$$F_P = F(\Phi_P, \Psi_P, \Omega_P) = e^{k_A r_{P_A}} - e^{k_B r_{P_B}} \tag{1.2}$$

or

$$F_P = F(\Phi_P, \Psi_P, \Omega_P) = \frac{k_A}{r_{P_A}^2} - \frac{k_B}{r_{P_B}^2} \tag{1.3}$$

where

$$r_{P_M} = \sqrt{(\Phi_P - \Phi_M)^2 + (\Psi_P - \Psi_M)^2 + (\Omega_P - \Omega_M)^2}, \quad M = A, B \tag{1.4}$$

Either k_A or k_B can be defined arbitrarily. So, $k_A = -1$ and a negative k_B value is sought for. Summarizing, the free-parameters are $(\Phi_A, \Psi_A, \Omega_A, \Phi_B, \Psi_B, \Omega_B, k_B)$ and these are controlled by the *GAs*. Standard parent selection, crossover and dynamic mutation operators are employed. Elitism is also used.

For any pair of point charges, a single potential value is computed at any graph node. Potential values are sorted and the graph nodes above and below the median are assigned to the first and second subdomain, respectively. At G^0, this is straightforward and satisfies the load balance requirement. In this case, the first term in the r.h.s. member of eq. 1.1 is automatically zeroed, but at any other level, graph nodes' weights are involved in the computations, and eq. 1.1 should be considered in its full expression.

1.3 ACCELERATION TECHNIQUES

Acceleration-refinement techniques have been devised in order to diminish the partitioning cost. Profound economy in CPU time is achieved when huge meshes are to be partitioned. In the context of the single-pass multilevel scheme, the acceleration tools are necessary during the bisection of the G^0 graph and, depending on the case, during some of the $G^m (m \ll)$ bisections.

The random initialization inherent in the *BITMAP* method inevitably creates badly organized distributions of 0's and 1's over the graph nodes. The modified genetic operators used in *BITMAP* help reduce the computing cost but the gain is not enough. For further acceleration, a technique which is well suited to *BITMAP* is the zonal refinement. This is activated whenever provisory, though distinct, subdomains have been formed and only local changes to the separator shape are expected. The concept is to encode and handle only a narrow zone of the closer to the interface graph nodes. Only these nodes are binary encoded, by forming chromosomes of reduced length. The economy in CPU time reflects the much lower CPU cost per generation (chromosomes are much shorter) and the small-scale alterations this method tolerates. Changes close to the separator are only allowed, without affecting the remaining parts of the graph which have already been assigned to subdomains.

In the *FIELD* method, each bisection is based on a potential field created by point charges. The potential field is always continuous and defines more or less (depending on the parameterization) distinct subdomains, even during the very first generations. Consequently, compared to *BITMAP* fewer generations are required to reach the converged solution but the cost of a single generation increases by the cost for computing the potential field and sorting nodal values. The number of generations required for convergence in the *FIELD*-based methods should be reduced through the regular shrinkage of the search space for all free-parameters. The new search space is always centered upon the location of the best current solution and extends symmetrically in each direction. It is smaller than the previous one by a user-defined factor.

An alternative, less effective way to accelerate a *FIELD*-based partition is through zonal refinement, similar to that used in *BITMAP*. The concept underlying this technique is based on the identification of the "negotiable" graph nodes. These are nodes which in the sorted list of nodal potential values, lie around its median. Only the negotiable nodes are allowed to change owner during the forthcoming generations. Their number should be either constant or a fixed percentage of the total number of nodes at this level.

1.3.1 Heuristic Partitioning techniques

Apart from the acceleration-refinement techniques , a heuristic refinement method is additionally used. It can be used at any graph level, either as a supplement to the genetic optimization or as a substitute for it. It cannot create a partition from scratch, but effectively refines existing partitions. The proposed heuristic method in the form of a pseudocode is given below.

```
* Create a list with the graph nodes in contact with the separator
* According to the current imbalance, identify the preferable direction
   of node movement
iter = 0
do { changes = 0
       for (each list entry) {
             Extract a node from the tail
             if (node movement:
                   is towards desired direction
                   and the imbalance excess does not increase
                   and the interface length does not increase
             ) then
                   * Proceed with the node movement
                   * Update desired direction
                   * Update balance and interface
                   * Add new interfacial nodes to the head
                   changes = changes + 1
              else put node back to the head of the list
              endif
       }
       iter = iter+1
} while (list not empty and iter<maxiter and changes>0)
```

1.4 THE PROPOSED *UGP* ALGORITHMS

Fig. 1.2 shows a general flowchart for any UGP algorithm based on a single-pass multilevel scheme. This flowchart is representative of any partitioner described below, regardless the specific tools used to effect partitioning at each graph level. The first two proposed methods, already discussed in [3], are based on the standard kernels (*BITMAP* and *FIELD*) indiscriminately applied at any level. They will be referred to as *S–BITMAP* and *S–FIELD*. Unfortunately, both methods become too slow at the

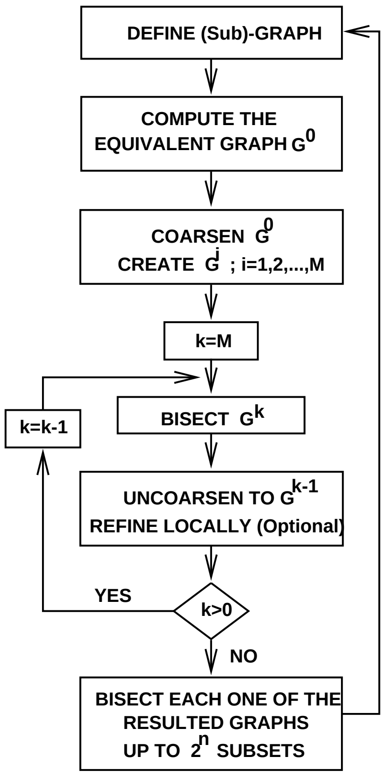
DEFINE (Sub)-GRAPH
COMPUTE THE EQUIVALENT GRAPH G^0
COARSEN G^0
CREATE G^i ; i=1,2,...,M
k=M
k=k-1
BISECT G^k
UNCOARSEN TO G^{k-1}
REFINE LOCALLY (Optional)
YES
k>0
NO
BISECT EACH ONE OF THE RESULTED GRAPHS UP TO 2^n SUBSETS

Fig. 1.2 Flowchart: Recursive bisections and the multilevel scheme

finest graph levels, and especially at G^0. The CPU cost per generation in *S–FIELD* is higher than in *S–BITMAP* though the latter requires less generations to reach the same fitness score.

A hybrid partitioner (*HYBRID*) was also presented in [3], where the *FIELD* kernel was applied for the partitioning of the coarser graphs, followed by the *BITMAP* one at the finer graph levels. Since G^M is handled by the *FIELD* method, good initial guesses are obtained that facilitate a lot the use of *BITMAP* in the subsequent finer graphs. The *HYBRID* method exploits the capabilities of *FIELD* to instantly create almost distinct subdomains and the excellent refinement properties of *BITMAP*.

The new two variants proposed in this paper will be referred to as *M–BITMAP* and *M–FIELD*, where *M* stands for Modified. In *M–BITMAP* the *BITMAP* kernel undertakes the partitioning of G^M. On completion of the G^M bisection, the best partition is directly uncoarsened to G^{M-1} where it is further improved using the aforementioned heuristic refinement. Its role is to smooth down the “wavy” parts of the interface. The same procedure is repeated up to G^0. A remark concering the *M–BITMAP* is that it may sometimes create multiple-connected subdomains. This occurs during the partitioning of G^M through the *BITMAP* method and cannot be recovered afterwards if heuristics are only applied.

Likely *M–BITMAP*, the *M–FIELD* was devised and tested. *M–FIELD* uses the *FIELD* method to partition G^M and then uncoarsens to the subsequent levels using heuristic refinement. In general, *M–FIELD* overcomes the problem of multiple-connected subdomains. Tests have shown that the use of the P2 parameterization in *M–FIELD* is favored.

1.5 APPLICATIONS - ASSESSMENT IN 3-D PROBLEMS

The five UGP variants discussed are evaluated in the partition of 3-D unstructured grids into 2^n subdomains. In all cases, the population size was 50 and the algorithm was considered as converged after 8 non-improving generations. The probabilities of mutation and crossover were equal to 0.15–0.5% and 90% respectively. The correction task applied on the 10% of the population and the upper bound for this correction was 20% of the G^0 nodes.

In the *FIELD*-based partitioners, eight bits per variable are used. The generations done per bisection are in the range of 150–500 for the *BITMAP*-based tools and 10–200 for the *FIELD*-based ones. The weighting factor w depends on the size of the graph to be partitioned. A recommended expression for w, which is the outcome of many *UGP* tests using graphs of various sizes, is given below

$$w \approx 9\ln(N_1 + N_2) - 55 \tag{1.5}$$

The first case deals with the unstructured grid modeling a 3-D compressor cascade. It consists of 111440 tetrahedra and was partitioned into $2^5 = 32$ subdomains. Tabulated results rather than views of the partitioned grids will be shown. For 2^n partitions, the parameter BAL is defined as $BAL = 2^n \cdot (larger\_partition\_size)/(graph\_size)$ while DD measures the number of D̲istict D̲omains that have been created. Here BAL, DD and the CPU cost in sec (on an Intel Pentium II 400MHz processor) are given. By examining the results shown in Table 1.1a, the superiority of the M̲odified versions of *BITMAP* and *FIELD* is obvious. They all provide simple-connected partitions (the fact that *M–BITMAP* and *M–FIELD*/P2 give 33 rather than 32 subdomains can be neglected) with a considerably smaller interface length than any other partitioner. Comparing *M–BITMAP* with *M–FIELD*, *M–FIELD* gives an interface length that is about 15-20% shorter than that of *M–BITMAP*. In this case, the graph space parameterizations P1 and P2 lead to separators of about the same size. Both *M–FIELD* variants are extremely fast. The standard versions result to much longer interfaces; besides *FIELD* provides a lengthy interface that defines 83 instead of 32 distinct domains. It is interesting also to note that the interface provided by *FIELD* is quite lengthy even during the first bisection. *HYBRID* outperforms both standard versions, giving a better interface with reduced CPU cost but its performance is still far from that of the M̲odified variants.

The second grid examined is formed around a 3-D isolated airfoil. This grid occupies a much larger physical space than the previous grid, the major part of which is filled in a very coarse manner. This is of interest as far as *FIELD*-based methods using the P1 parameterization are of concern. The grid consists of 257408 tetrahedra and has also been partitioned into 32 subdomains. Results are given in Table 1.1b. All of the partitions were balanced, so relevant information has been omitted. In general, the conclusions drawn from this study are similar to previous ones. The Modified variants perform better and faster than the standard or the *HYBRID* version. In this case, the difference between *M–FIELD* and *M–BITMAP* is more pronounced, not only due to the lengthy interface *M–BITMAP* produces but also due to the fact that *M–BITMAP* creates 43 instead of 32 distinct subdomains. It is also interesting to note that *M–BITMAP* provides more unbalanced partitions than any other method used. Finally, *HYBRID* is still better than the standard variants but, in this case, the qualities of the separators obtained by both methods are not far apart.

The third 3-D grid examined is formed around a complete aircraft. It consists of 255944 tetrahedra and a close view is shown in fig.1.3. Compared to the grid used in the previous case which was of about the same size, this grid is a pure unstructured grid, while the other was formed by stacking a real 2-D unstructured one. Results obtained by partitioning this grid through various methods are shown in 1.1c.

	Interface for 2^n subdomains					For n=5			
Partitioner	n=1	n=2	n=3	n=4	n=5	BAL	sec	DD	
A–BITMAP	248	1146	1883	3082	6605	1.00	100.6	35	
A–FIELD/P1	545	1764	3205	4870	7199	1.00	246.8	83	
HYBRID	300	1139	2005	3295	5728	1.00	80.5	34	
M–BITMAP	194	818	1533	2734	4733	1.11	13.4	33	
M–FIELD/P1	180	719	1425	2361	4056	1.00	10.2	32	
M–FIELD/P2	194	752	1484	2428	4064	1.03	8.7	33	(a)
A–BITMAP	2499	4420	7688	10759	14672	1.00	494	71	
HYBRID/P1	1502	4081	7305	11073	14802	1.00	360	72	
HYBRID/P2	1492	3190	6708	9951	13596	1.00	322	71	
M–BITMAP	928	2233	3751	5767	8135	1.19	29.8	43	
M–FIELD/P1	884	1875	2994	4544	6858	1.00	22.8	32	
M–FIELD/P2	750	1642	2749	4497	6774	1.00	19.4	32	(b)
HYBRID/P2	3045	9594	15291	21215	29056	1.00	1688	71	
M–BITMAP	1975	5720	92673	13408	18809	1.12	33.9	37	
M–FIELD/P1	2021	4707	7365	11345	16278	1.00	22.4	32	
M–FIELD/P2	1790	4519	7446	11639	16597	1.00	21.7	32	(c)

Table 1.1 Partitioning of (a) the 3-D compressor cascade grid, (b) the 3-D isolated airfoil grid and (c) the grid around an aircraft

From Table 1.1c we conclude that *M–FIELD* is better than any other separator, gives exactly 32 distinct domains with low CPU cost. *M–BITMAP* gives a slightly larger interface, slightly unbalanced partitions and slightly increased number of distinct subdomains. In general, both *M–BITMAP* and *M–FIELD* perform much better than *HYBRID*. The CPU cost of *HYBRID* is about 50 times that of the Modified versions.

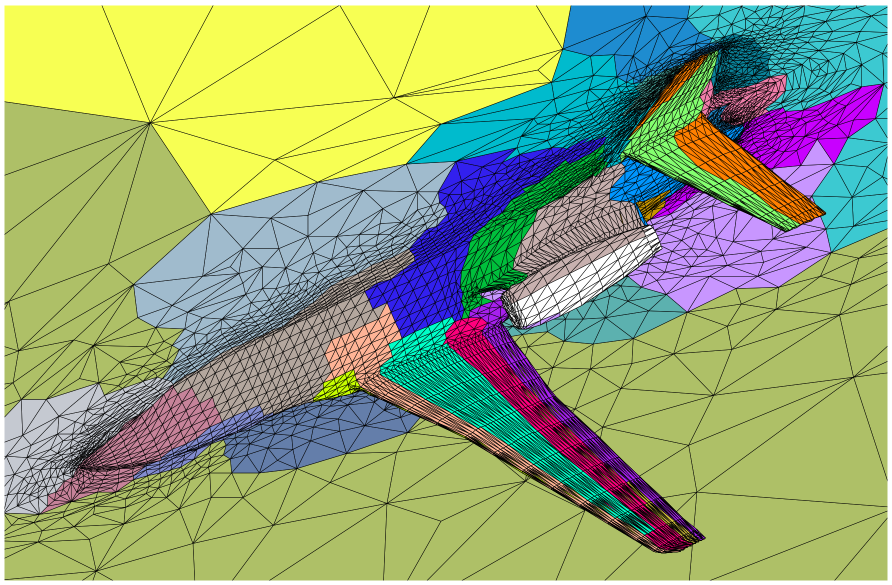

Fig. 1.3 A close view of the grid around an aircraft (Courtesy of Dassault Aviation)

1.6 CONCLUSIONS

Partitioning methods for 3-D unstructured grids, based on two standard kernels, have been presented. They all operate on the equivalent graph of the grid, so they are in fact graph partitioners appropriate for any grid type. They use Genetic Algorithms to minimize a properly defined cost function, either through direct (*BITMAP*) or indirect (*FIELD*) modeling.

The multi-level scheme is necessary for the proposed methods to efficiently undertake the partitioning of heavy industrial meshes. It uses a coarsening procedure which should be carefully chosen as it affects the quality of the partitions. It is recommended that the coarser graph G^M size be of about 50–150 nodes.

Some more remarks on the standard kernels follow. In general, *BITMAP* tends to create disjoint or slightly unbalanced partitions, requiring more generations than *FIELD* to converge. Even if *FIELD* converges faster, a *FIELD* generation requires more CPU time than a *BITMAP* one. The local refinement of the interpartition boundary created by *FIELD* is really necessary in order to improve its shape; this should be carried out at the post-processing level through heuristics. *M–FIELD* and *M–BITMAP* outperform any other partitioner. They produce short interface length, balanced partitions and avoid the formation of disjoint subsets. Among *M–FIELD* and *M–BITMAP*, tests have shown that *M–FIELD* is slightly better. Typical grids of the order of 250.000 tetrahedra can be partitioned within 20–25 secs on an Intel Pentium II 400MHz processor.

References

1. Goldberg D.E. *Genetic Algorithms in search, optimization & machine learning.* Addison-Wesley, 1989.

2. Karypis G. and Kumar V. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 1998.

3. K.C. Giannakoglou and A.P. Giotis. Unstructured 3-d grid partitioning methods based on genetic algorithms. ECCOMAS 98, John Wiley & Sons, 1998.

4. A.P. Giotis and K.C. Giannakoglou. An unstructured grid partitioning method based on genetic algorithms. *Advances in Engineering Software*, 1998.

5. Van Driessche R. and Roose D. Load balancing computational fluid dynamics calculations on unstructured grids. *AGARD*, R-807, 1995.