GENETIC ALGORITHMS IN ENGINEERING AND COMPUTER SCIENCE

eurogenddbook 7/10/1995 12:07—PAGE PROOFS for John Wiley & Sons Ltd (jwbook.sty v3.0, 12-1-1995)

GENETIC ALGORITHMS IN ENGINEERING AND COMPUTER SCIENCE

Edited by J. PÉRIAUX and G. WINTER

JOHN WILEY & SONS
Chichester · New York · Brisbane · Toronto · Singapore

Contents

13

Parallel Genetic Algorithms for Optimisation in CFD

DENIS DOORLY[1]

13.1 INTRODUCTION

This chapter is concerned with the use of genetic and parallel genetic algorithms for optimisation in computational fluid dynamics (CFD). It is intended to provide an introduction both to optimisation in this area, as well as to specific genetic algorithm (GA) applications. A brief sketch of the problem will thus be given before discussing issues of GA choice and implementation.

CFD attempts to model and predict fluid behaviour by numerically solving equations which approximate the physical laws governing a flow. We are all familiar with the complex patterns which can occur when a fluid is set in motion, from observations of the water in a bath as we splash about, or from looking at the flow of a river behind the pier of a bridge, for example. It is hardly surprising therefore that the computational effort required to solve many flow problems necessitates the use of a fast computer, to obtain a solution in reasonable time. Apart from obtaining a solution to predict some flow attribute, (such as the drag exerted on a given body), CFD may be used as a tool in designing a particular object. Computationally this is even more demanding, particularly where repetitive analysis, geometry modification, and optimisation are combined to automatically seek some 'optimal design'. Hence CFD has an enormous appetite for computational resources, and the use of parallel algorithms to meet these demands is now commonplace.

To maintain the efficiency of a parallel CFD code, there are strong reasons for ensuring an associated genetic algorithm is also rendered parallel. For a program

[1] Aeronautics Dept., Imperial College, London SW7 2BY U.K. e-mail d.doorly@ic.ac.uk
GENETIC ALGORITHMS IN ENGINEERING AND COMPUTER SCIENCE
Editor J. Périaux and G. Winter

containing parallel and sequential portions, the maximum speedup S gained by using P processors instead of a single one is limited by Amdahl's law to:

$$S = \frac{1}{\frac{a}{P} + (1 - a)}, \qquad 0 \leq a \leq 1$$

where a is the proportion of the program execution which is made parallel. Thus if a sequential genetic algorithm is coupled to a parallel CFD solver (to create an optimal design program for example), the maximum speedup will become limited by the time required for execution of the sequential portion. There is an additional motivation for using a parallel genetic algorithm, in that a particular type of parallelism (distributing the population) can have inherent beneficial effects, as discussed later.

Genetic algorithms can be used either in combination with CFD to optimise a design, or they can be more tightly coupled to CFD, for example to optimise some aspect of the solution process itself. In the former case, since genetic algorithms are general purpose optimisation methods they can be coupled to any flow solution method for optimal design. To be practical however, both how the GA is applied and adapted for CFD, and the type of flow analysis to which it is currently most suited need consideration.

The optimisation of flow solvers designed for parallel computers is an example of the second application. Generally this type of solver divides the computational domain into subdomains, with the flow in each subdomain being advanced concurrently on separate processors; 'domain decomposition' refers to the process. The assignment of computational elements to subregions, and the assignment of subregions to processors are equivalent to a form of graph partitioning; for example each processor can be represented as a vertex of a graph, and links between vertices represent interprocessor communication. The design and partitioning applications are used here as illustrative examples, with the material organised as follows.

After a brief outline of a representative CFD algorithm, genetic algorithms for optimising graph partitioning are discussed, and an example program outlined. Non-genetic algorithms for the partitioning problem are briefly considered as competitors or for hybridisation with the GA. The use of genetic algorithms for optimal design is then discussed. Both the above optimisation problems lead to consideration of parallel genetic algorithms; (preliminary) conclusions are finally summarised.

13.2 CFD ANALYSIS FOR AEROSPACE DESIGN

The choice of the most appropriate analysis method for a given flow problem depends on the requirements of accuracy, ability to represent complex geometry, and computational cost. Many methods which can satisfy the first and last of these are often incapable of dealing with the second requirement, which is essential for design studies on realistic configurations. Finite volume and finite element methods can handle geometrical complexity, and solve the full set of equations, (respectively the Navier Stokes equations for viscous flow, and the Euler equations for inviscid flow), but at a cost which may be prohibitive. Boundary integral or panel methods can likewise handle complex geometries and need at least an order of magnitude less computing time, but do not account for viscous effects or shock waves. Coupled panel and shear layer methods can represent viscous effects adequately in many applications, but the

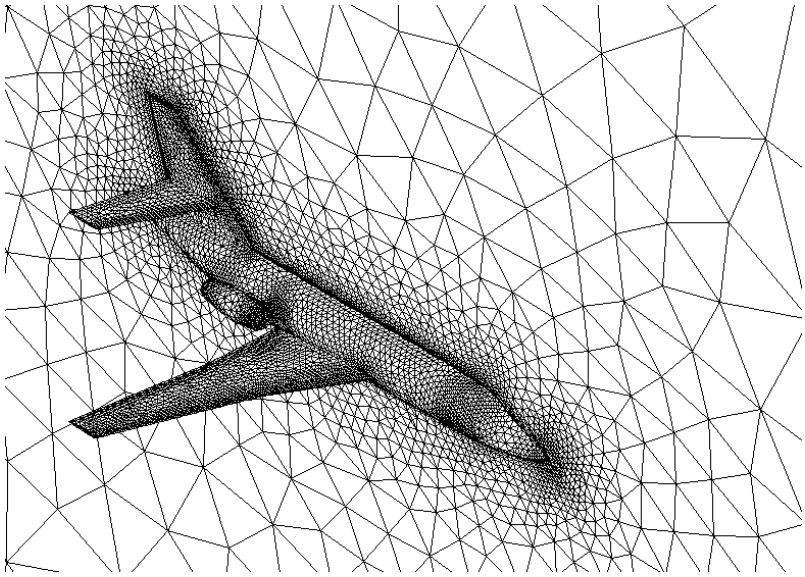


Figure 13.1 Unstructured mesh for aircraft configuration

full equations (considered for the most part here) are required where large separated regions exist, or in transonic flows for example.

13.2.1 Unstructured Finite-Volume and Finite-Element Methods

If conservation of mass, momentum and energy are applied to a control volume V bounded by a surface S, the Navier Stokes equations are obtained;

$$\frac{\partial}{\partial t}\int_V \vec{U} dV = -\int_S \vec{F}(U)\cdot d\vec{S}$$

Here $\vec{U}$ represents the vector of unknown flow variables (for a compressible fluid: density, momentum, and energy $[\rho, \rho u, \rho v, \rho w, \rho e]$), and the flux vector $\vec{F}(\vec{U})$ expresses transport of flow quantities across S and boundary acting terms. In the numerical solution procedure, the equations are discretised in space and the solution is advanced forward in time from an initial state, applying appropriate boundary conditions. In the finite volume formulation we consider, the spatial discretisation uses tetrahedral cells, within which values are assumed to vary linearly, and gradients are thus constant over a cell.

An example of an unstructured mesh for computation of the flow about an aircraft configuration is illustrated in fig. 13.1, showing the surface triangulation for a tetrahedral mesh (of approximately 80,000 nodes, and 350,000 elements) generated by the advancing front method, described in [PMP93] (The mesh was produced and kindly provided by Dr. J. Peiró). The discrete equations to update the value of the

vector of flow variables at a node (such as "P" in the 2D mesh below) are then

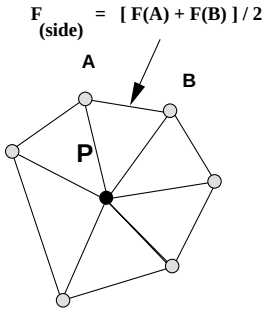


Figure 13.2

$$\frac{d}{dt}\,\vec{U}\Big|_P = \frac{1}{\Delta V}\sum_{face(i)} \vec{F}\Big|_{(i)} \cdot \vec{n}_{(i)} ds$$

where ΔV is a control volume about P, and the value of the flux vector on each face (i) with normal $n_{(i)}$, surrounding the node is obtained by interpolating the values at each extremity of the face. The gradient terms in the flux vector (either arising from viscous stresses, or artificially added to limit spurious oscillations near shock waves) are calculated from nodal values by applying the Green-Gauss relation to a suitable control volume. Details vary somewhat from method to method, but in any event the flux terms for the control volume surrounding each node (on the right of the above equation) may be assembled from the values of the fluxes at the mid points of the edges connecting the nodes. In performing the forward integration in time, explicit or implicit methods may be used, ie. the flux vector can be evaluated either wholly at the previous time level, or at least partly at the current level. For explicit integration, updating each node requires only the previous values of the flow variables at surrounding nodes, and can proceed independently of all the other nodal updates. With an implicit method, all the nodes are updated together, implying simultaneous solution. For this case however, we can assume that the implicitness is treated iteratively, (eg. via multigrid Jacobi iteration), or that the implicit treatment is broken into subdomains, with an iterative treatment of the equations linking subdomains (which are assumed to reside on different processors).

13.2.2 Parallel Computational Fluid Dynamics

Parallelisation of the CFD solution algorithm involves distributing the work (of updating nodal values) among the available processors; clearly for the algorithm to work efficiently the task is to minimise interprocessor communication whilst evenly distributing the computational load of updating the nodal values. In the case of a regular structured mesh, the decision as to how to distribute the mesh nodes among the processors may often be decided with ease in advance, eg. [HD90]. For an unstructured mesh, however, the problem of deciding how to apportion the elements is much more

difficult. The number of nodes or elements assigned to a processor relates to the degree of parallelisation; in medium or coarse grain parallelisation, a given processor is responsible for updating a domain containing many if not thousands of nodes, whereas with fine grain parallelisation, a processor updates either a single node or edge, or possibly just the nodes associated with one cell.

Communication between processors is required to exchange flow variables along edges cut by the partition. For coarse grain parallelisation, this involves the exchange of 'halo data'. In the simplest case, the values at nodes which are excluded from a

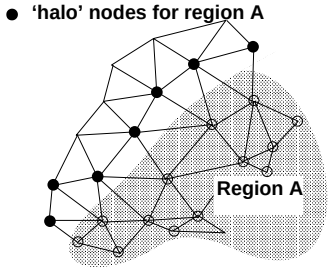


Figure 13.3 Halo data nodes (black) required to update solution in region A

given partition but which are required to update values inside the partition (near the boundary) are transferred from the neighbouring partitions, forming a 'halo' about the boundary. Since the flux associated with a given edge is used in updating the nodal values at each end of the edge, it need only be computed once. The graph connecting the nodes along the cut edges may therefore be treated as a directed graph, where an outward direction from one node implies that the processor which stores that nodal value is responsible for computing the shared flux. In fine grain parallelisation [HB92] each nodal vertex is assigned to a processor, and thus the elimination of redundant flux calculations by directing the edges of the graph is essential. Unfortunately however there can be a large load imbalance if one nodal vertex has many more outward pointing edges than other vertices. For a planar graph, there is a linear time algorithm [CE91] to assure no vertex has more than three outward directed edges, but there is as yet no bound for non planar graphs. For coarse grain parallelisation, the calculation of redundant fluxes is a very small proportion of the total workload, thus the graph is generally treatable as an undirected graph. (If need be, a form of directionality can be implemented by non-symmetric mapping of halo data, but we will not consider this).

13.2.3 Mesh Partitioning for Parallel CFD

The choice of cost function used to assess partitions, and the constraints imposed on all partitions exert a strong influence on the type of decomposition sought. In turn, the architecture of the target parallel computer and the type of problem guide the choices of cost function and constraints. It is assumed that the computer is of MIMD (Multiple Instruction Multiple Data) type, with equal amounts of distributed memory per processor. At one extreme, one might seek to utilise all the available memory. Exact sharing of the computational work on each processor (perfect load balance) is then enforced, and the problem is to minimise the communication cost. Usually

however the amount of memory required is less than the sum total available, so there is a weaker constraint limiting the memory available for the largest partition. Thus a degree of load imbalance is tolerable provided the overall cost is lower. Suppose there are P processors, N nodes, and the decomposition leads to M domains. Assume M = P; also note that a domain may be the union of disjoint regions, though with a high communication penalty. The execution time is given by the 'worst performing' domain, i.e. the slowest processor and the cost can be written

$$C = max_i \{c_i\} ,$$

with

$$c_i = [n_i * w + \sum_{k=1,K} f(b_{i,k}, d_{i,k})].$$

Here n_i is the number of nodes mapped to region i, w is the average work required to update a node, $b_{i,k}$ is the number of boundary nodes requiring communication between regions $iandk$ and $d_{i,k}$ is the interprocessor distance from processor i to processor k. Alternatively a cost function which expresses the communication cost, number of neighbours, and degree of load imbalance can be used, such as

$$C = \alpha_1 \cdot \sum_i (n_i - N/P)^r + \alpha_2 \cdot {n_b}^s + \alpha_3 \cdot {n_K}^t,$$

where r, s, t and α_1, α_2, α_3 are exponents reflecting machine architecture and problem constraints; n_b, n_K are the number of boundary nodes and domains respectively. The optimisation thus seeks to achieve for each domain:

1. even load balance
2. minimum nodes on domain boundary
3. minimum number of neighbouring domains

It is clear that in the optimal decomposition, the effect of the second term alone is to minimise the surface area of the partition whereas the third term, in minimising the number of interprocessor connections, seeks to pipeline the regions into strips. Thus the communication criteria are in some sense in opposition. Assigning one of M colours to each node, the problem may be restated in terms of colouring the graph of nodal connections, so the complete decomposition is specified by the colour of each element.

13.3 GENETIC ALGORITHMS APPLIED TO CFD

13.3.1 Two Problems

The basis of genetic algorithms has been established by Holland [Hol75], and reference to this, Goldberg [Gol89], Davis [Dav91] and other chapters of this book should provide a comprehensive introduction and many examples. Here the use of genetic algorithms in CFD is introduced by outlining their application for two problems: mesh partitioning and optimal airfoil design. Copies of programs which implement the procedures described are available on e-mail request.

13.3.2 Mesh Partitioning

A simplistic approach to applying a GA to the partitioning problem would be to initialise a population of decompositions, encoded as chromosomes. Each gene comprising an individual chromosome (decomposition) could specify the colour of an associated node. However for 64 processors, 6 bits would be required to specify a unique colour, and a given decomposition (member of the population) would have a genotype more than 60,000 bits long for a 10,000 node mesh. This is simply too large for an affordable GA. Instead the GA may either be applied indirectly to optimise the parameters of an underlying nodal colouring procedure, or the GA may be used to improve an initial solution generated by an alternative procedure. In the former case, the reduced number of parameters of the colouring scheme provides a greatly compressed structure which is amenable to an affordable GA. As an example, the genetic algorithm has been applied to optimise the performance of a low level mesh partitioning scheme based on 'attracting points'; other low level schemes could easily be devised. The procedure is not particularly efficient (as presented here) for mesh decomposition, however it is useful as a simple way of introducing GA and parallel GA approaches.

The basis of the attracting points method is to place a number of points (centres) equal to the number of processors at different locations in the mesh, nodes are then assigned to the nearest point (processor), to produce a given decomposition. (Any node exactly equidistant from two or more centres is arbitrarily assigned to one). Assignment conflicts are rare, so there is the advantage of a nearly unique relation between domains, and positions of the points, however the shape of the allowable regions is restricted, so only a subset of all possible solutions is reachable. The tasks required to assemble a genetic algorithm can be summarised as:

- *define chromosome encoding*
- *set up initial population*
- *evaluate each individual*
- *translate to fitness*
- *regenerate population by reproduction*
- *reproduction operator selection*
- *mutation*
- *crossover*

Defining the chromosome: the parameter set for the partitioning problem corresponds to the position coordinates of a set of attracting centres; the string comprising the successive x, y, and z coordinates of the set of centres makes up an individual or chromosome.
Initial population: an initial population of partitions is be set up by specifying different sets of attracting points. Real or bit string representations may be used; in the program a real position encoding is applied.
Evaluation: the cost of the decomposition specified by each member of the population is evaluated according to the second form of the cost function defined above.
Fitness: rather than simply use the inverse of the cost to define a fitness, the population members are first ranked. Based on rank, a transformed value of fitness is assigned to each member of the population.
Reproduction: the population is regenerated in each generation by reproduction. The

scheme is elitist, without duplication. To select parents the Roulette wheel model is used with selection according to rank-transformed fitness.
Operator selection: the operations of mutation and crossover are separately applied and are selected with specified probabilities.
Mutation: if selected, the mutation operator applies (with low probability) a random displacement to each centre coordinate.
Crossover: two-point crossover is used.

Applications to Unstructured Meshes

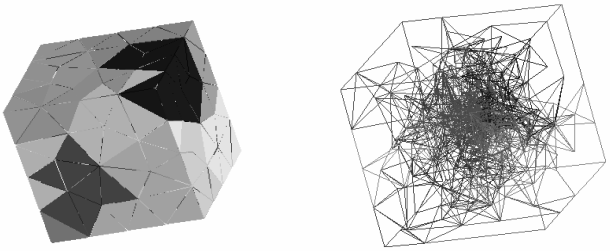
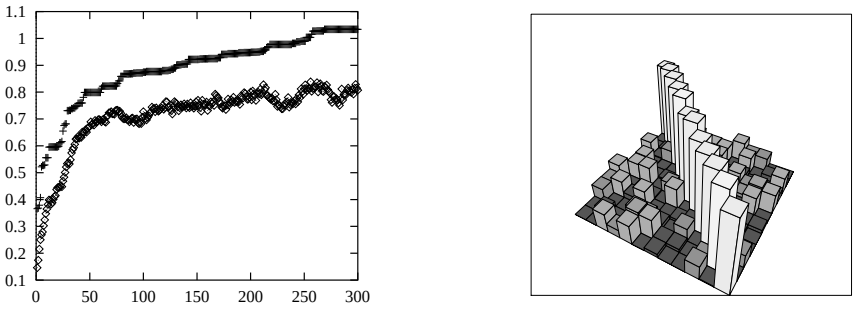


Figure 13.4 Partitioned unstructured box mesh, left shows surface, right all edge boundaries

The program is first applied to optimise the decomposition of an unstructured tetrahedral mesh of the interior of a cube, containing approximately 9,100 elements and 1700 nodes for a 12-processor computation. The above procedure is slightly modified by replacing the single point attractor with a cluster of three points, which allows greater flexibility in the shape of the decomposed regions though the computational cost increases. The resulting partition at the surface is shown in fig. 13.4. Although it appears to be quite sub-optimal, the problem is made more complex by clustering many of the elements about a point in the interior, (fig.13.4, showing edge boundaries). Comparison of the convergence of the generation best, and population average for the genetic algorithm using a population size of 100 is shown in fig. 13.5; for the same computing effort, a random search method typically produces hardly any improvement over the initial score. The processor loading and communications cost matrix are represented in fig. 13.5, where the diagonal elements represent the computational work per processor, and the off-diagonal terms represent the interprocessor communication load. Although not fully converged, a good decomposition is produced. The next example shows part of the surface decomposition for the aircraft configuration, for a 16 processor calculation after 400 generations. The clustering about the engine



(a) Convergence history

(b) Processor loading (12 processors)

Figure 13.5

nacelle and near the wing leading edge again complicates the procedure, as shown. The decomposition is still far from optimal however, and more efficient GA approaches

Figure 13.6 Surface partitions of unstructured aircraft mesh

are considered later.

Non-Genetic Algorithm Approaches

The genetic algorithm described above, while simple to implement, will not produce a good decomposition in reasonable time if there are more than a few partitions. It is thus necessary to consider alternative approaches, either to initialise a genetic algorithm, or in combination with the genetic algorithm.

In *recursive coordinate bisection*, nodes are sorted into equal groups by position in one coordinate direction in one step; in succeeeding steps the coordinate direction is

alternated and recursive subdivision applied. In *recursive graph bisection* distance is measured in the number of links required to connect nodes. Such techniques are fast, but the surface/volume ratio of some domains may be poor and the parallel efficiency suffers.

With the *greedy algorithm* [FL93], each node is weighted according to the number of connections; after randomly selecting a 'pivot node', the domain grows by adding boundary nodes (of lowest weight), promoting smoothness of the boundary. The procedure has been shown to be very fast [FL93], but it is most effective in minimising the communication cost associated with domain surface/volume ratio.

The *eigenvalue recursive bisection* procedure [BS92] is based on finding an eigenvector of the Laplacian matrix of the graph, and splitting the graph based on the magnitude of the components (corresponding to the nodes) of the eigenvector. At present it seems to be the most efficient sequential decomposition procedure.

Simulated annealing has also been applied to the partitioning problem. Comparative suggest that that given a sufficiently long run, this method produces the best decomposition.
Later in 13.5.3 the decomposition problem is reconsidered as an application of parallel genetic algorithms.

13.3.3 Optimal Airfoil Design

A few points on CFD design are given before considering the use of GAs in this area; further information can be found in [VKI94], [RJ95].

Optimal airfoil design can be categorised according to whether direct or inverse optimisation procedures are followed. In the direct approach, aerodynamic analysis is coupled to a numerical optimisation and geometry modification scheme, to search for an optimal design. In the inverse approach, a favorable pressure distribution is first sought, and the corresponding geometry is then determined by an inverse solution procedure. The former approach is more powerful, but far more demanding computationally; in the latter approach, the computation is greatly eased by restricting the scope of the design. There is a penalty however in that a specified pressure distribution may not be realisable, or may produce unforeseen disadvantageous flows. Furthermore, it usually involves modifying an initial design to approach the specified 'target' pressure distribution by an optimisation process, so that the distinction between methods can be somewhat blurred.

In both cases however, it is necessary to first define a cost function, which expresses the quantity it is required to minimise. Following Jameson [VKI94], write the cost function as $I = I(w, X)$ with w representing the flow, and X the geometry;
examples are: $I = \frac{{C_l}^2}{C_d}$, and $I = \int (P_{target} - P_{design})^2 ds$.

Next consider the component parts of the optimisation:

1. *Flow analysis method:* the flow analysis method is a major impediment to the routine adoption of optimal design; the first decision relates to the type of analysis method to be used, i.e. Navier Stokes (expensive but more capable) or at the other extreme, a panel method (fast but potentially unreliable).

2. *Optimisation procedure:* traditional approaches use gradient search techniques, where in the simple steepest descent method, the geometry is modified in the direction of largest reduction in I. Since

$$\delta I = \frac{\partial I^T}{\partial w}\delta w + \frac{\partial I^T}{\partial X}\delta X$$

evaluation of δI requires δw, which is computed by recalculating the flow after making a small change in each design variable separately; the use of n design variables thus implies n flow evaluations. An adjoint equation approach has been developed [Jam94], in which the governing equations are introduced as a constraint, so a single solution of the adjoint equation yields the gradient. This approach considerably lessens the computational load of gradient-based optimisation procedures, and appears very suitable for combination with a GA.
3. *Geometry representation:* as it is necessary to minimise the number of design variables, the surface is usually parametrised in some way. Commonly used are B-spline representations, and localised shape functions, such as the Hicks-Henne [HH78] bump

$$b(x) = \left\{ \sin(\pi x^{\frac{log(.5)}{log(t_1)}}) \right\}^{t_2}, \qquad 0 < x < 1$$

the width of the bump is controlled by t_2, and the maximum by t_1.

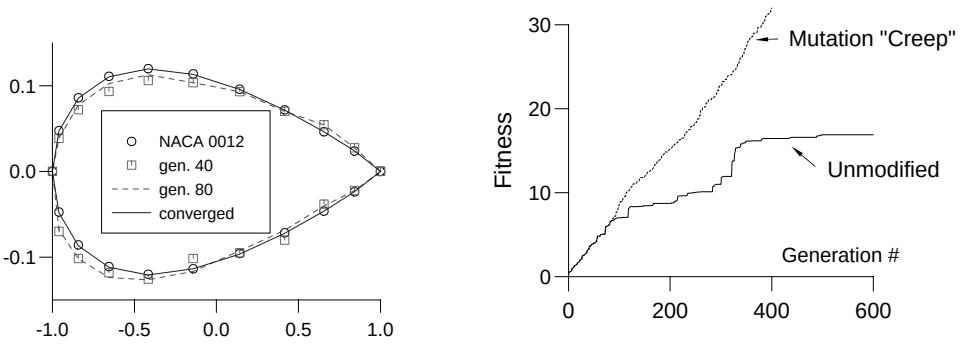
13.3.4 Application of Genetic Algorithms to Airfoil Optimisation

There have been a number of applications of GAs to airfoil optimisation, e.g. see [QD95],[OT95], [YI95]. The search ability and robustness of the GA appears suited to complex aerodynamic problems, where there may be highly nonlinear or discontinuous parameters. To begin, results using a simple GA optimisation program (requiring a few minutes on a fast PC) are outlined; both GA and CFD parts can readily be improved.

Simple GA airfoil optimisation

The surface of the airfoil is defined in the program by specifying the y ordinates of nodal points on upper and lower surfaces; an incompressible panel method solves for the flowfield. With leading and trailing edges fixed at $y = 0$, a string of 18 points defines a 20 panel airfoil, admittedly rather coarsely. A real number encoding is used for node height y, (scaled to an integer range). Mutation resets height to a random value in [-0.15,0.15]; population replacement avoids duplication, and is elitist with *(generation best)* and one *(mutated* [generation best]*)* individuals automatically copied.

Two point crossover is used, with independent mutation and crossover operations, [Dav91]. Given an initial population of airfoils (of random y-ordinates), the GA may be used to search for a profile which matches a particular pressure distribution, as per inverse design. With the computed inviscid Cp distribution for a NACA 0012 airfoil input as a target pressure distribution, the results of applying the genetic algorithm (population 200) are shown in fig. 13.7. Fitness is defined as $\frac{1}{C}$, with



(a) Inverse airfoil design by GA

(b) GA convergence

Figure 13.7

$C = [\int (Cp_{ref} - Cp)^2 ds]^{\frac{1}{2}}$. (Geometry differences are hardly noticeable for $\frac{1}{C} \geq 15$). With the given, constant large amplitude mutation operator, the GA becomes less effective as the solution starts to converge, slowing the improvement rate. After 600 generations, the differences between target and achieved geometries are still noticeable.

If the mutation amplitude is scaled down (by 5,10,40 after generations 80, 160, 320), the convergence rate dramatically improves. Reducing mutation amplitude is a form of 'creep' [Dav91], and is useful near convergence, though in this case, a switch to gradient descent would be more efficient. It is readily demonstrated (by running the program with more panel locations) that the convergence rate drastically reduces, as the dimension of the search space increases. This and the requirement for smoothness (both to be aerodynamically acceptable, and to simplify the topography of the 'fitness landscape') make parametrisation of the geometry desirable.

Transonic airfoil optimisation, surface parametrisation

In [QD95], full potential CFD analysis and GA were combined to modify a baseline airfoil shape, and obtain shockless transonic flow. Shape modification was defined using 28 parameters each encoded in 7 bits, and single-point crossover was implemented. In airfoil optimisation often the optimal geometry may perform rather poorly outside a narrow range; multi-point optimisation (where the cost function combines objectives at several conditions) avoids selecting narrow optima. For example, two point GA optimisation was performed [QD95] to seek $max[1/(\frac{C_{d_1}}{C_{l_1}^2} + \frac{C_{d_2}}{C_{l_2}^2})]$ again for a transonic airfoil. Although effective, it needed nearly eight times the cost (in generated individuals) and more than 3 times the cost (in evaluations) of single-point optimisation.

An interesting way of defining geometry very suited to GA optimisation has been introduced by Yamamoto and Inoue [YI95], inspired by ideas of ontogeny. In their approach, a hierarchical tree structure is used to represent the entire geometry (fig. 13.8, from [YI95]). At the base or 'level 1' the position of the camber line (r_1, θ_1) and

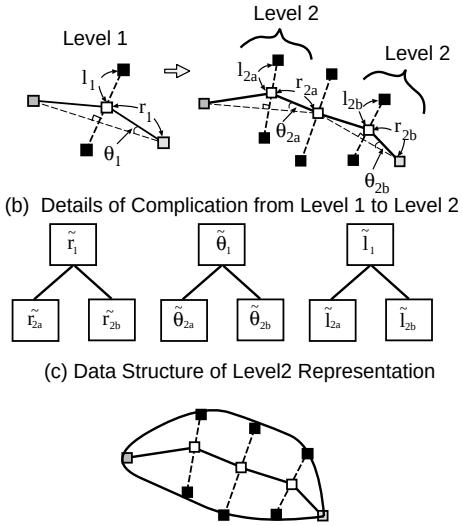


Figure 13.8 Encoding of airfoil geometry. Yamamoto and Inoue, (1995) (Reproduced by permission).

thickness (l_1) are given; finer details, such as the camber and thickness at either end are specified in 'level 2' branches. The branching process continues to lower levels as required to specify the complexity of the geometry, which is thus recursively defined. Crossover involves local branch exchange, providing an effective way of combining schemata. In the work of [YI95], branching takes the form of local structure refinement via thickness and camber. Other of branch refinements could be employed (eg. separate branching for upper and lower surfaces, or final surface representation transferred to outermost branches). Although the same in principle it is worth identifying which form most efficiently provides building blocks for the GA. This work [YI95] also demonstrated the feasibility of full Navier Stokes GA optimisation of lift/drag for a supercritical airfoil, though the evaluations (70 generations, population of 64, and 160x50 grid) required substantial vector-parallel and parallel computer resources.

13.4 PARALLEL GENETIC ALGORITHMS

13.4.1 Classification of Parallel Genetic Algorithms

When implemented on a sequential machine, the GA normally operates on a single (global) population. On a parallel machine there is a choice between:
a) preserving the use of a global population whilst parallelising the GA operations,
b) dividing the global population into several sub-populations and evolving a sub-population on each processor.
Nang and Matsuo [NM94] provide a formal definition of the parallel GA with sub classifications of types a) and b). Essentially however, parallelisation may be used to:

- evaluate a single or several chromosomes in parallel

- reproduce population in parallel
- distribute population and run complete GA concurrently

Choice of single (panmictic) population or multiple demes

The first two of the above routes to parallelisation can be implemented using a single population (type a)), whilst the last uses sub-populations (type b)) is also referred to as the (parallel) distributed genetic algorithm (DGA). The issue of whether it is better to utilise a single, well-mixed ie. *panmictic* population or a number of smaller subpopulations, with some restrictions imposed on the mixing of these semi-isolated subpopulations (called *demes*) does not appear to be altogether settled, and may be problem dependent. Early DGA model studies [Tan89] demonstrated superior performance of the DGA using multiple small populations over that of a sequential GA with a single large population for a certain class of functions. Other studies have likewise shown improvements gained by use of demes; however [Bel95] found that the DGA does not always produce better results.

Amongst other factors the deme size migration rate and migration mode can have large effects, and there is as yet little to guide these choices. A recent study [GKHCP95] addresses the efficiency of multiple deme GAs for the extreme cases of ideal perfect mixing after deme convergence, and isolated non-mixing demes. The study supports the use of multiple demes, although it showed that for isolated demes, population sizes much below the single population optimum were inefficient on a serial computer. It also demonstrated GA success even with very small populations.

It has been suggested that the success of the DGA is alternately due to the 'shifting balance' theory of Wright [Wri32] or to 'species formation' and subsequent migration leading to 'speciation events', from the work of [EG72]. Premature convergence of the GA is at least partly avoided, in a sense either dynamically (by subpopulation 'drift' in the former case), and by preserving diversity in 'species' in the latter. Experience has shown that it can be difficult to avoid premature convergence of the GA, and 'nicheing' methods have been developed to preserve diversity, eg. the sequential niche technique [BBM93], restricted tournament selection [Har95]. The former is unsuitable for parallelisation, whereas the latter could be implemented in a parallel panmictic GA. Although the DGA may promote species formation it may also be useful to consider some form of niche protection, but such issues will not be considered here.

13.4.2 Constructing a Parallel Genetic Algorithm

The DGA is in a sense naturally parallel; it has an inherent advantage over the parallel single panmictic GA in that the restricting interactions between individuals in different subpopulations limits the interprocessor communication cost.

Parallel single panmictic population GA

The major issue relates to efficiency of parallel implementation, since the basic GA properties are unchanged. Assuming a total population of N, and P processors, several methods [NM94] make each processor responsible for replacing N/P individuals. However, rather than broadcasting complete information on each partial population at

every generation, the parallel implementation of Tajima [Taj95], requires only fitness values to be broadcast. Local selection from the entire population fitness can be performed and then only requested chromosomes are broadcast.

Parallel DGA

There are alternative parallel DGA implementations, according to migration strategy. In the *island parallel* approach, each processor exchanges individuals with randomly chosen processors, (usually at random time intervals). It is thus suited to asynchronous MIMD implementation. For the *stepping stone* model, migration exchange only occurs between geographical neighbours (eg. North, South, East, West in a 2D processor array). Migration frequency may vary from

- never - effectively a partitioned GA
- only after local convergence [CMR91]
- randomly or at regular intervals

13.5 PARALLEL GENETIC ALGORITHMS IN CFD

13.5.1 Overview

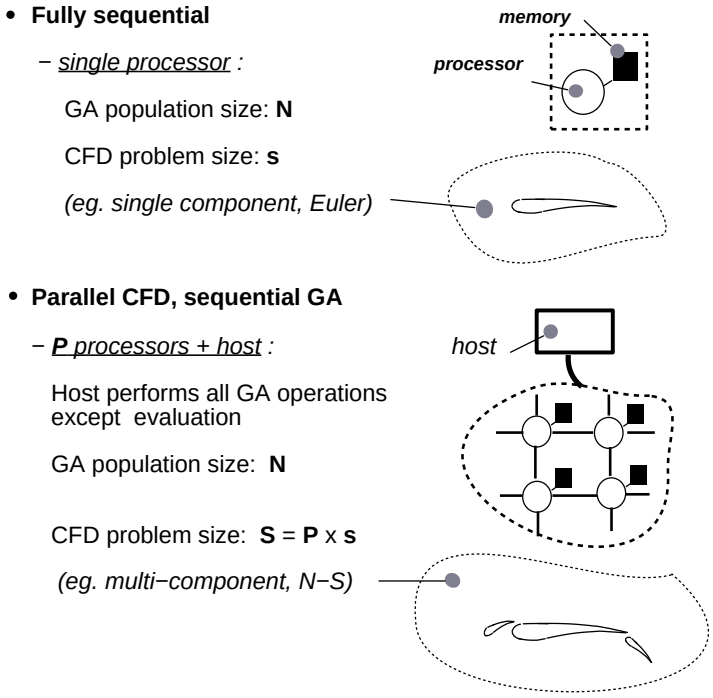


Figure 13.9

A program combining a GA and CFD analysis can be structured so execution is purely sequential, either one operates in parallel, or both do so. Figs. 13.9 and 13.10

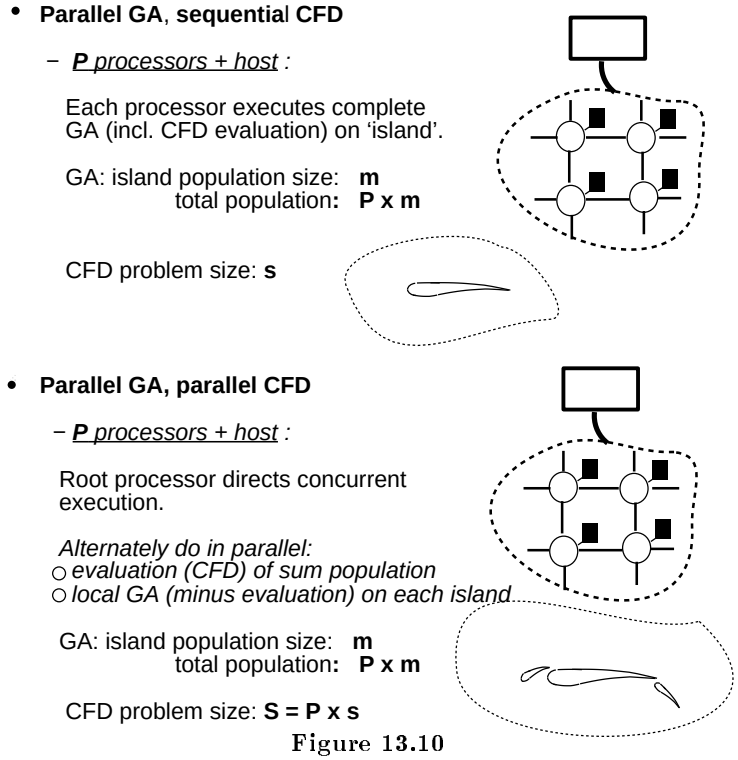


Figure 13.10

illustrate the possibilities, for the example of GA-based airfoil or wing optimisation. In the simplest case (fig. 13.9), parallelism is used merely to accelerate the CFD and/or allow larger problems to be attempted, for example the analysis may change from an Euler computation of a single airfoil, to a Navier Stokes analysis of a multi-component airfoil. Whilst the efficiency will be limited by Amdahl's law, this may not be much of a problem if the CFD analysis greatly dominates the computing time. In the third case, (top of fig. 13.10) the execution is fully parallel, although in effect a distributed sequential CFD analysis is performed on each processor. This form of implementation is ideally suited to a parallel DGA; results using the program of section 13.3.3 adapted to a stepping-stone population exchange DGA will be discussed below. Finally, both CFD and GA can be implemented in parallel. There are several possibilities: e.g. evaluation (CFD), and GA operations may alternately execute with either panmictic or distributed populations, or processors may be grouped into clusters, with each cluster running a DGA.

13.5.2 Parallel Genetic Algorithms for Design Optimisation

Most of the applications of GAs to airfoil design have used a single population, with parallelism restricted to the evaluation by CFD, as in the lower part of fig. 13.9. It is probably simplest to implement a parallel GA-CFD optimisation using the DGA model with multiple sequential CFD evaluations as for the top example in fig. 13.10.

To do so, requires sufficient memory to allow each processor to execute independently. For panel method analyses it is unlikely that there will be insufficient memory on each MIMD processor, but there may not be sufficient to permit independent Navier-Stokes or even Euler (in 3D flow) execution. In that case, a more complex fully parallel GA-CFD implementation is necessary, at least for direct optimisation.

If however, an inverse design method of the form used by [OT95] is employed, the GA is solely used to design the optimal target pressure distribution. The method finds the shape of the pressure distribution curve which best provides a specified integrated lift, and meets a set of semi-empirical criteria; the CFD solution is merely used to recover the designed shape. For this method, it is easy to adapt the GA to be parallel, and there is no restriction on the type of CFD solver, as it is only used after completing the GA optimisation.

Optimisation using parallel DGA panel program

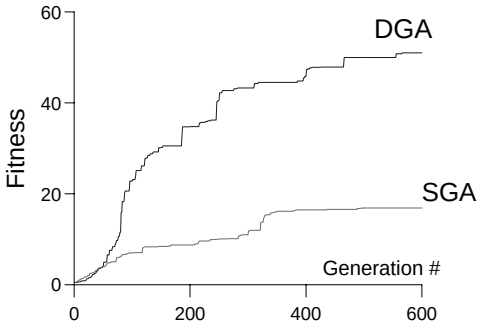


Figure 13.11 Performance of DGA

The program of 13.3.3 has been adapted to a parallel DGA with stepping-stone migrant exchange. To compare sequential panmictic and DGA results, the NACA 0012 inverse design problem is again considered. The DGA uses the same total population size of 200, which is split into 20 demes of 10 individuals, each mapped to a separate processor. After every five generations the fittest individuals from each island migrate to north, south, east and west neighbours, replacing the least fit locals. Fig. 13.11 compares convergence rates for the DGA and single population (labelled SGA) applied to the problem (both without using creep). In a series of such runs, the DGA consistently outperformed the SGA. Given that the parallel DGA also requires a fraction of the execution time, to process the same number of generations, the improvement is very large.

13.5.3 Parallel Genetic Algorithms for Mesh Partitioning

Post-generation mesh partitioning

The mesh partitioning methods described earlier are intended for an existing generated mesh. For optimisation of any basic method by a parallel DGA which uses independent

evaluation on each processor, (analagous to the top paradigm in fig. 13.10) there is the disadvantage that each processor needs sufficient memory to hold the entire mesh, which is often impractical. An advantage of the attracting centres method of 13.3.2 however, is that the evaluation can easily be performed in parallel. Since the assignment is purely based on a distance metric, the elements may be distributed in any fashion among the processors; each processor being responsible for 'colouring' a portion of the elements. Either form of parallel GA may be implemented, but the basic procedure is still inefficient. It may be improved by recursively dividing the domain among two or three centres at a time; the GA can be applied to optimise each stage of the process.

Other methods are based on improving fast approximate partitions by exchanging interface nodes or elements, until the partition is optimal; simulated annealing (SA) has for example been used [FMB95] for the element exchange optimisation. Combining the parallel GA with SA, as in [MG95] appears to be an attractive procedure for accelerating such a process, but remains to be tried.

Parallel generation and partitioning

The above approach to mesh partitioning for parallel CFD computations suffers two serious deficiencies. In the first place, it is assumed that there is sufficient memory available to perform the decomposition on a single processor, the host or root processor in fig. 13.9, 13.10, whereas for very large computations on a MIMD machine, the complete mesh may not fit on any single processor. Secondly, the time required to produce the decomposition must be greatly reduced, particularly if adaptivity is employed, since remeshing and thus re-balancing may be required frequently. Both issues were addressed by Khan and Topping [KT93], who proposed a combined GA-optimised recursive bisection strategy and neural net mesh density predictor, termed the subdomain generation method or SGM [TK93], [BTK94].

The basic idea is to partition a coarse background mesh into subdomains containing relatively small numbers of elements, and subsequently generate a fine mesh in each subdomain. The partitioning of the background mesh must be such that each subdomain will contain approximately equal numbers of elements and interfacing boundary edges will be minimised. In their approach, a trained neural network was used to estimate the number of elements of the fine mesh generated per coarse mesh element by an adaptive mesh generator, given the nodal coordinates and mesh parameter values for each background mesh element. The coarse mesh decomposition is by recursive bisection, accomplished by a GA-optimized greedy algorithm. They report difficulties in training the neural net for a wide range of mesh densities, but achieved encouraging results. Clearly there is considerable scope for the application of parallel GAs to the partitioning problem.

13.6 CONCLUSION

In summary:

- for parallel CFD, a parallel GA implementation maximises combined efficiency,

- parallel computing makes the GA more affordable,
- the parallel distributed GA (parallel DGA) is simple to implement,
- the parallel DGA may outperform the GA in optimal airfoil design,
- to maximise parallel CFD efficiency, the GA may be used in optimising the parallel partitioning and generation of subdomain meshes.

References

[BBM93] Beasley D., Bull D. R., and Martin R. R. (1993) A sequential niche technique for multimodal function optimization. *Evolutionary Computation* 1(2): 101–125.

[Bel95] Belding T. C. (1995) The distributed genetic algorithm revisited. To appear in: *Proceedings of Sixth Int'l. Conference on Genetic Algorithms.* Morgan Kaufmann.

[BS92] Barnard S. T. and Simon H. D. (1992) A fast multilevel implementation of recursive spectral bisefction for partitioning unstructured problems. NAS Systems Division. Moffett Field, CA.

[BTK94] Bahreininejad A., Topping B. H. V., and Kahn A. I. (1994) Subdomain generation using multiple neural networks models. Heriot-Watt University, Edinburgh, UK.

[CE91] Chrobak M. and Eppstein D. (1991) Planar orientations with low out-degree and compaction of adjacency matrices. *Theoretical Computer Science* 84: 243–266.

[CMR91] Cohoon J. P., Martin W. N., and Richards D. S. (1991) Genetic algorithms and punctuated equilibria in vlsi. LNCS-496, pages 134–144. Springer-Verlag.

[Dav91] Davis L. (1991) *Handbook of Genetic Algorithms.* Van Nostrand Reinhold.

[EG72] Eldredge N. and Gould S. J. (1972) Punctuated equilibria: an alternative to phyletic gradualism. in Schopf, T. J. (Ed.) Models of paleobiology, pages 82–115.

[FL93] Farhat C. and Lesoinne M. (1993) Automatic partitioning of unstructured meshes for the parallel solution of problems in computational mechanics. *International Journal for Numerical Methods in Engineering* 36: 745–764.

[FMB95] Farhat C., Maman N., and Brown G. W. (1995) Mesh partitioning for implicit computations via iterative domain decomposition: impact and optimisation of the subdomain aspect ratio. *Int. J. Num. Methd. Engng.* 38: 989–1000.

[GKHCP95] Goldberg D. E., Kargupta H., Horn J., and Cantu-Paz E. (1995) Critical deme size for serial and parallel genetic algorithms. IlliGAL Report No. 95002. Illinois Genetic Algorithms Laboratory, University of Illinois.

[Gol89] Goldberg D. (1989) *Genetic Algorithms in Search Optimization and Machine Learning.* Addison-Wesley.

[Har95] Harik G. (1995) Finding multimodal solutions using restricted tournament selection. Proceedings of the International Conference of Genetic Algorithms. ICGA95.

[HB92] Hammond S. W. and Barth T. J. (1992) Efficient massively parallel euler solver for two-dimensional unstructured grids. *AIAA Journal* 30(4): 947–952.

[HD90] Hall R. C. and Doorly D. J. (April 1990) Parallel solution techniques for compressible transonic flow. Proc. 12th Int. Conf. on Numerical Methods in Fluid Dynamics. Oxford.

[HH78] Hicks R. M. and Henne P. A. (1978) Wing design by numerical optimization. *J. Aircraft* 15(7): 407–412.

[Hol75] Holland J. H. (1975) *Adaptation in Natural and Artificial Systems.* Ann Arbor: Univ. Michigan Press.

[Jam94] Jameson A. (April 1994) Optimum aerodynamic design via boundary control. AGARD-VKI Lecture Series. Von Karman Institute, Belgium.

[KT93] Kahn A. I. and Topping B. H. V. (1993) Subdomain generation for parallel finite element analysis. *Computing Systems in Engineering Vol. 4* 4: 473–488.

[MG95] Mahfoud S. W. and Goldberg D. E. (1995) Parallel recombinative simulated annealing: a genetic algorithm. *Parallel Computing* 21: 1–28.

[NM94] Nang J. and Matsuo K. (1994) A survey on the parallel genetic algorithm. *J. SICE* 33(6): 500–509.

[OT95] Obayashi S. and Takanashi S. (1995) Genetic optimization of target pressure distributions for inverse design methods. AIAA-95-1649-CP.

[PMP93] Peraire J., Morgan K., and Peiró J. (1993) Multigrid solution of the 3d compressible euler equations on unstructured tetrahedral grids. *Int. J. Num. Meth. Engng.* 36: 1029–1044.

[QD95] Quagliarella D. and DellaCioppa A. (1995) Genetic algorithms applied to the aerodynamic design of transonic airfoils. *J. Aircraft* 32: 889–891.

[RJ95] Reuther J. and Jameson A. (1995) A comparison of design variables for control theory based airfoil optimization. Proc. 6th ISCFD Conf., pages 101–107. Lake Tahoe NV USA.

[Taj95] Tajima K. (September 1995) Improvement of interprocessor communication in a parallel genetic algorithm retaining sequential behaviour. Fourth Parallel Computing Workshop. Imperial College London.

[Tan89] Tanese R. (1989) *Distributed Genetic Algorithms for Function Optimization.* PhD thesis, University of Michigan.

[TK93] Topping B. H. V. and Kahn A. I. (1993) Subdomain generation method for non-convex domains. Information Technology for Civil and Structural Engineers. Heriot-Watt University, Edinburgh, UK.

[VKI94] (April 1994) Optimum design methods in aerodynamics. AGARD-VKI Lecture Series. Von Karman Institute, Belgium.

[Wri32] Wright S. (1932) The roles of mutation,inbreeding,crossbreeding and selection in evolution. volume 1 of *Proceedings of the Sixth International Congress of genetics.*

[YI95] Yamamoto K. and Inoue O. (1995) Applications of genetic algorithm to aerodynamic shape optimization. Paper AIAA-95-1650-CP.