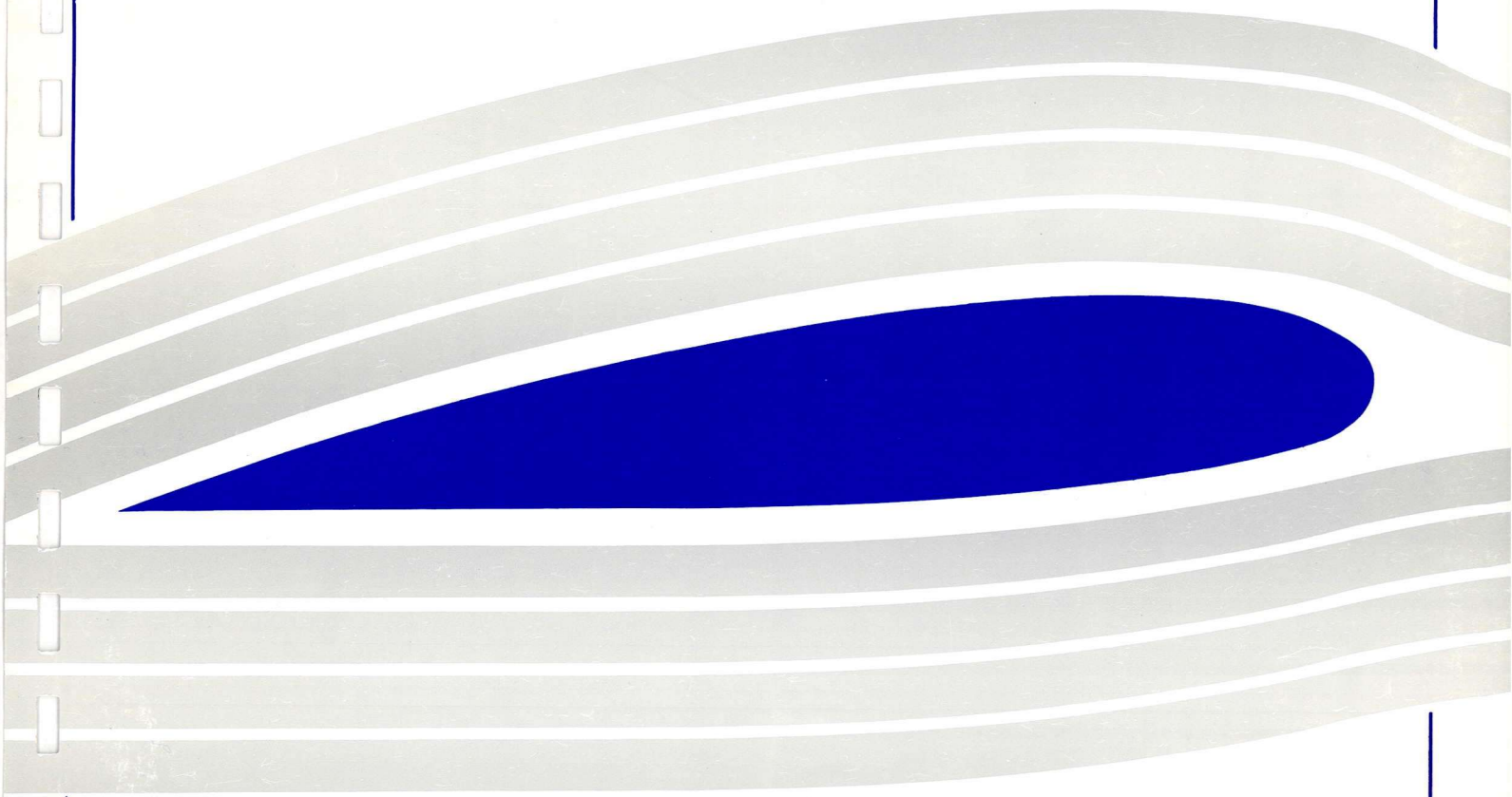
University of Glasgow
DEPARTMENT OF
AEROSPACE
ENGINEERING

Parallel Aerodynamic Simulation on Open Workstation Clusters

B. J. Gribben, K. J. Badcock and B. E. Richards



Parallel Aerodynamic Simulation on Open Workstation Clusters

B. J. Gribben, K. J. Badcock and B. E. Richards

University of Glasgow

Department of Aerospace Engineering

Report Number 9830

October 1998

Abstract

The parallel execution of an aerodynamic simulation code on a non-dedicated, heterogeneous cluster of workstations is examined. This type of facility is commonly available to CFD developers and users in academia, industry and government laboratories and is attractive in terms of cost for CFD simulations. However, practical considerations appear at present to be discouraging widespread adoption of this technology. The main obstacles to achieving an efficient, robust parallel CFD capability in a demanding multi-user environment are investigated. A static load-balancing method, which takes account of varying processor speeds, is described. A dynamic re-allocation method to account for varying processor loads has been developed. Use of proprietary management software has facilitated the implementation of the method.

Contents

1 Introduction

Parallel computing in computational fluid dynamics is a very broad area of current research and development. Parallel computing software and hardware technology is developing very rapidly, and the CFD community is at the forefront in exploiting emerging technology to obtain the high performance computational resource required to solve large CFD problems. The enthusiasm for parallel computing in the CFD community is based on present cost effectiveness compared to conventional computing, and future projections of enormous computing power. The exploitation of parallel computing is considered to be a key to tackling the grand challenges in CFD[1].

To effectively use a parallel computer an intelligent mapping of subsets of the total computational work onto processors must be performed. There are several different levels of parallelism, ranging from job parallelism where processors execute tasks with no interdependency, to arithmetic parallelism where the work of the simplest operations is shared amongst processors. A coarse-grain data parallel approach[2] is usually employed in parallel CFD, where sub-domains of the computational grid are mapped onto the set of processors, with the objective of finding a mapping which results in the fastest overall execution of the parallel task. This approach is commonly referred to as *domain decomposition* in the literature. The principal feature of an efficient domain decomposition is that the load is evenly distributed across the processors. A typical parallel CFD application involves a communication phase where information must be passed between the processors. Communication is necessary periodically, e.g. once every time step. If the load is evenly distributed then the processors arrive at the communication phases simultaneously, minimising processor idle time. For many applications, attempting to minimise the time spent in the communication phase is also necessary for efficient

implementation. The problem of optimal domain decomposition is well known to be NP-complete[3],[4], i.e. a deterministic solution procedure is impractical. The task of achieving a parallel execution via domain decomposition can be viewed as a two-stage process; *mesh partitioning* to form the sub-domains and *allocation* of sub-domains to processors to achieve load balanced execution[3],[5]. A wide variety of methods have been proposed, see for example the proceedings of the Parallel CFD conferences[6],[7], reflecting the variety of problems considered and architectures used. For unstructured grid problems the prevalent approach is to use a mesh partitioning heuristic to obtain equally sized sub-domains and at the same time attempt to minimise the sub-domain interface length to keep down the amount of necessary communication. The resulting partition then consists of the same number of sub-domains as there are processors, and communication has already been considered implicitly in the partitioning stage, so it is sufficient to allocate the sub-domains directly onto the processors. An initially popular method was the 'Greedy' algorithm for mesh partitioning[8], so called because successive 'bites' are taken from the domain. The Greedy algorithm is very fast since it essentially involves only one sweep across the mesh, but is unreliable since the last 'bites' can leave sub-domains of inappropriate size and shape. Most researchers now employ a recursive bisection approach from graph theory, a good review of which is provided in [9]. In recent years some specific methods have become established in the CFD and structural finite-element communities and are available in the public domain[10],[11],[12]. Alternative non-deterministic approaches such as simulated annealing and stochastic evolution have been used for unstructured mesh partitioning, but have the disadvantage of being slow in comparison to recursive bisection methods[4],[5],[13]. Applying the methods of unstructured grid partitioning to multiblock structured grids is often quoted as being possible, but only one example has been found in the literature[14]. This is for two reasons. First, partitioning a multiblock structured grid is easier than

an unstructured grid in that there are less possible boundary path permutations, but harder in terms of programming in that flow solver constraints (e.g. block interface matching) must be considered in the partitioning algorithm. Secondly, often the number of grid blocks naturally arising from the grid generation process is far greater than the number of processors, so this partition can be accepted as long as a heuristic is designed to arrange these blocks onto the processors such that the load is balanced. If there are very large blocks which impede a good load balance then it is a simple matter to split them 'manually', unlike unstructured grids. Hence for structured multiblock grids the emphasis in domain decomposition is much more on the allocation stage. The heuristic techniques employed, often cost function minimisation procedures, are similar to those attempted for unstructured mesh partitioning, but are better suited for this problem due to the reduced size of the state-space[3]; tens or hundreds of blocks are considered rather than tens or hundreds of thousands of grid cells. See references [3],[5] for a summary of the preferred methods.

The domain decomposition methods mentioned above have all considered the *static* problem, where the decomposition is determined before run-time. *Dynamic* re-allocation methods have not been discussed. It is necessary to reconsider the decomposition during run-time to preserve load balance if the solution procedure is adaptive, for example when adaptive grids are used. Also, some researchers seeking the last percentages of parallel performance gains maintain that a static decomposition can never exactly account for actual processor speeds and communication costs, so some degree of dynamic re-allocation is required. For an overview of this type of dynamic problem see references [6],[15]. We are interested here in a different type of dynamic problem where the decomposition may have to respond to varying processor loads; this point is returned to below.

Compared to a decade ago, parallel CFD technology is considerably more advanced. However, as noted by Knight[15], the huge amount of publications devoted

to parallel CFD research is not matched by the amount of CFD research conducted using parallel CFD as a tool. Based on contacts between the CFD group at Glasgow University and the U.K. aerospace industry, this appears to be as much the case for CFD use in industry as in academia. Knight suggests three reasons for this:

- parallel computers are perceived as lacking a decisive performance advantage
- parallel code has portability problems
- parallel code is difficult to program efficiently

Advances in hardware and software have now made the first two points an irrelevancy. Numerous recent projects have demonstrated the enormous potential and cost savings of using workstation clusters or modern commodity processors in parallel, for example[16]. The development of standards in languages (eg. High Performance Fortran[17]) and message passing (eg. MPI[18], PVM[19]) have brought the portability of parallel code almost to the same level as sequential code. The problem lies in the third point; the practical difficulties in making parallel CFD work can be discouraging[20]. To aerodynamicists, there has always been a trade-off between the amount of effort necessary to apply a prediction method and the accuracy of the results that the method produces. In addition to the effort required for a sequential CFD capability, parallel CFD requires the aerodynamicists to:

- obtain and install a message passing library or parallel compiler
- write the parallel code
- write a domain decomposition method or assimilate an 'off the shelf' method
- manage the execution of parallel tasks

The first two points are mitigated by the emergence of standards in parallel programming, as mentioned above, where message passing libraries are in the public

domain, parallel compilers are available from vendors, advances have been made in making parallel programming easier and help on all of these is freely available via the internet. However, it is noted that large organisations are likely to employ specialist programmers and information technologists; small and medium-sized organisations are more likely to be discouraged by the first two points. Chien et al.[21] have made some important observations concerning the third and fourth points. Existing domain decomposition methods are restricted to parallel systems consisting of a homogeneous processor set[1] and which are operated in single-user mode. This typifies a dedicated parallel machine possessed by a large organisation; smaller organisations are likely to make their first steps in parallel processing using a non-dedicated heterogeneous network of workstations. Making use of spare capacity on existing UNIX workstations, originally obtained for other purposes, was pioneered by Pratt & Whitney[23] and McDonnell Douglas. However in these cases a policy of interactive/sequential and batch/parallel use segregation was enforced, the parallel jobs being executed overnight, and all other jobs being suspended. This heavy-handed restriction on activity is unwanted in any environment and practically impossible to enforce in academia. To make parallel CFD more attractive on 'open' networks of workstations, the ideal parallelisation approach should

- include a domain decomposition method for a heterogeneous processor set
- be integrated seamlessly with existing sequential batch queueing
- take account of varying network load

[1] this assertion is perhaps slightly too strong. Varying processor power is occasionally accounted for in a cost function approach, but without the method being actually demonstrated on a heterogeneous network, for example in [3]. In addition, a successful, truly heterogeneous domain decomposition has been demonstrated[22]; however the dynamic re-allocation method used to achieve the load balance appears very communication-intensive and may only be suitable for very compute-intensive problems of the type presented.

	Type 1	Type 2	Type 3	Type 4
No. of machines	3	2	4	7
Processor	R5000	R4400	R4400	R4600
Speed (MHz)	150	150	150	133
Main memory (Mb)	96	160	64	64
Data cache (Kb)	32	16	16	16
Instruction cache (Kb)	32	16	16	16
CPU factor, k	1.9	1.6	1.6	1.2

Table 1: *Specifications of the workstation cluster*

In this report, the integration of a parallel multiblock structured aerodynamic simulation code into an open, heterogeneous workstation cluster environment is examined. The use of clusters of workstations for parallel CFD is of high interest to industry[24]. The expected performance increase is limited but comes essentially free since the workstations have usually already been purchased and installed for either sequential CFD work or other tasks. The workstation cluster used is located in the Department of Aerospace Engineering at the University of Glasgow. The cluster consists of a number of Ethernet-connected Silicon Graphics Indy workstations of four different types, as described in Table 1. The cluster is typical of departmental level computing facilities (albeit larger than usual) and the facilities at the disposal of industry, where often the development of the computing resource over time results in an inevitably heterogeneous computing environment[25]. The focus of the work is to consider the needs of small and medium sized organisations who require a parallel capability to scale up their computing resource but may at present be discouraged by the perceived practical difficulty involved. This differs from the majority of parallel CFD research where the principal or sole aim has been to achieve the high parallel efficiencies necessary for potential or actual massively-parallel applications on dedicated machines. Network load management software services are exploited to facilitate the application of the decomposition method, and assimilating parallel tasks into the overall batch scheduling and queueing sytem for the workstation

cluster is considered.

The flow solver used is PMB2D[26],[27],[28],[29] developed by the CFD group at the University of Glasgow. Overlapping grids are employed with two rows of 'halo' cells associated with each internal block boundary. After each time step the updated solution is copied to these halo cells from the corresponding cells in the adjacent block, such that each block has the necessary information to form the residual vectors and Jacobian matrices for the next time step. If blocks sharing a common boundary reside on different processors, then the copying of data is enabled using message passing.

2 Cost function minimisation

We wish to distribute structured data blocks amongst the processors of a parallel machine. The primary consideration in determining an efficient distribution is that each processor should spend the same amount of time performing calculations between the synchronous communication phases i.e. that the processors are not idle. This is the load balancing problem. Restricting our discussion at present to a homogeneous parallel machine (where all the processors are identical), for CFD applications a balanced load can be obtained, to a good approximation, by assigning an equal number of grid cells to each processor. Sub-domain shape, e.g. block aspect ratio, and boundary conditions can also influence processor load[30] but these are usually ignored as less important effects. The load balancing problem can then be modelled as a minimisation problem for the 'cost' H due to the time spent performing calculations[13]:

$$H = \frac{P^2}{N^2} \sum_{q=1}^{P} N_q^2 \tag{1}$$

where P is the number of processors, N is the total number of grid cells and N_q is the number of grid cells resident on the processor q. As noted in section 1,

non-deterministic procedures are used to solve this allocation problem. No clear consensus on which method is best has appeared in the literature, although simulated annealing (S.A.) is most often cited as reliably producing near-optimal results, for example in [4],[13],[30], although there are some reservations about the relatively long execution time of the S.A. algorithm. An iterative improvement (I.I.) technique is said to often out-perform S.A. if tailored towards the particular application[31]. For these reasons I.I. and S.A. will be evaluated as minimisation procedures for the cost function (1). Their algorithms are described below.

Iterative Improvement

An algorithm based on iterative improvement[31],[32] is very straightforward to program. Some initial configuration of the state (which can be generated at random if necessary) is required, along with a cost function definition. In an iterative manner, a small change based on a random selection is made to the system and this 'basic move' is either accepted or rejected. The acceptance criterion is as follows: if the move causes the cost to decrease, the move is accepted, otherwise the move is rejected. The process is terminated when a large pre-determined number of consecutive attempts are unsuccessful. Note that careful selection of the basic move is crucial to the success of the method. The method is sometimes referred to as 'hill-climbing'.

Simulated Annealing

The method of simulated annealing[31],[32],[33] is a relatively new method for the minimization of objective functions. It is particularly suited to discrete, very large configuration spaces i.e. for combinatorial optimization problems. The title of the method is due to an analogy with the slow cooling of metals. The simulated annealing algorithm is straightforward to program, and has as its kernel the iterative improvement algorithm. Again an initial configuration of the state and a cost func-

tion definition are required. The acceptance criterion is as follows: if a proposed basic move reduces the cost, then the move is accepted. If the cost is increased, then the move is only rejected with a certain probability, called the Metropolis criterion[34]. Included in this criterion is an artificial system 'temperature' such that at high temperatures almost any basic move is accepted, however costly, and at low temperatures effectively zero 'bad' moves are accepted i.e. the algorithm becomes one of iterative improvement. A high starting temperature is used, and the temperature is periodically forced down by some factor after a large number of basic moves have been proposed. The intention is to explore the entire state-space with the Metropolis criterion providing a means of escape from local minima.

Two structured multiblock grids were considered to evaluate I.I. and S.A. for the allocation problem. Details of grid dimensions are shown in Table 2. Note that both grids consist of a large number of blocks with widely varying block sizes. To evaluate the effectiveness of the cost minimisation procedures, an efficiency measure E_1 is defined as

$$E_1 = \frac{N/P}{N_q^{max}} \tag{2}$$

	Grid 1	Grid 2
Total number of cells (N)	48,425	43,417
Number of blocks	81	21
Average block size	598	2067
Biggest block size	2349	6642
Smallest block size	104	319

Table 2: *Details of multiblock grids used in allocation test problems*

where N_q^{max} is the greatest number of grid cells on any one processor in the final allocation. Note that for an ideal allocation E_1 is unity. Two basic moves were used for both I.I. and S.A.; a 'simple' move where two randomly chosen processors swap randomly chosen blocks, and a 'complex' move where clusters rather than single

blocks are swapped. The clusters begin as randomly chosen blocks, then collect blocks on the same processor with a probability of 0.2 for each possible collection[13]. The values of E_1 obtained for each minimisation method and different numbers of processors are shown in Figures 1 and 2. In each case S.A. out-performs I.I. for the 'simple' move. S.A. provides a mechanism for avoiding local minima which can trap the I.I. procedure. However, there is negligible difference in the final result for the 'complex' move. This basic move is designed to enable larger jumps in the state-space of the type required to avoid local minima (for this problem), and has had the desired effect. Note that the complex S.A. has also out-performed the simple S.A. method. A very good discussion of the importance of choosing an appropriate basic move is included in [13]. Note that for both test problems, the efficiency of the final allocation begins to decrease when an allocation over a large number of processes is attempted. This occurs when the number of cells in the largest block becomes larger than the ideal number of cells per processor N/P. If it were desired to use a large number of processors, this problem could be avoided by manually splitting the biggest grid block.

For the remainder of this work the complex I.I. minimisation procedure will be used. More detailed cost functions will be employed, but the nature of the minimisation problem will remain the same. It is preferred to the complex S.A. procedure since it requires less execution time, less than one second compared to about four seconds, and provides equally high quality results.

3 Communication cost

The majority of parallel applications of structured multiblock codes which have appeared in the literature consider only the criterion of load balancing to achieve good parallel performance. A good example is [14] where impressive results are demonstrated on a number of parallel machines, including a dedicated workstation

cluster. However, a number of researchers have also stressed the need to take into account communication overhead. The simplest way to take into account the cost of communication as well as computation is to introduce a communication cost element into the cost function, and use a balance coefficient μ to scale the relative importance of the cost elements. The cost function for the allocation problem then becomes

$$H = \frac{P^2}{N^2}\sum_{q=1}^{P} N_q^2 + \mu\left(\frac{P}{N}\right)^{\frac{1}{2}}\sum_{e\leftrightarrow f} 1 - \delta_{q(e),q(f)} \qquad (3)$$

The first term in this equation is the load balancing term of equation (1). The second term is the communication overhead term. For all the cells e on block edges which must communicate with cells f in other blocks, a cost is incurred if e and f do not reside on the same processor q. The choice of scaling constants for each cost element is designed to keep their relative magnitudes constant regardless of the problem size, as discussed in [13]. For codes with a great deal of calculation compared to communication μ should be small, and vice-versa. This explains why communication cost may be disregarded for some flow solvers. According to De Keyser and Roose[3], it is only important to determine approximately the relative magnitude of computational and communication cost, rather than a precise value for μ. Hence we seek a value for μ where the resulting final allocation may differ from that obtained with $\mu = 0$, indicating that 'physical' adjacency of blocks is being taken into account to a degree, but where the load balancing problem is not being overwhelmed, i.e. E_1 does not become too small. Trial allocations with varying values of balance coefficient μ and numbers of processors P for Grids 1 and 2 indicated the range $10^{-7} < \mu < 10^{-2}$. To be more certain of obtaining an appropriate value for μ, trial runs of 50 implicit time steps using Grid 1 on two processors of Type 3 were performed for various μ. The results are shown in Figure 3. The timings shown are averages of ten runs performed overnight when the workstation cluster was very lightly loaded. The parallel efficiency E_p shown is

defined as

$$E_p = \frac{sequential\ execution\ time}{(parallel\ execution\ time) * (no.\ of\ processors\ used,\ P)} \quad (4)$$

The single processor run was perfomed on a processor of Type 2 which has the same speed as Type 3 processors but enough memory for a sequential execution. From the figure the communication model has had a small effect on execution times. The shortest execution times were obtained for $10^{-5} < \mu < 10^{-2}$; note that for these cases the allocations found by the minimisation procedure were identical. For high values of μ the communication cost begins to dominate, to the detriment of the load balance. For $\mu = 1.0$ all of the blocks were allocated to one processor. The maximum parallel efficiency achieved was 82%. This indicates that communication costs for the flow solver on the workstation cluster are high. That a greater parallel efficiency was not achieved is not an indication of a failure in the cost function allocation method; regardless of which allocation is determined, communication must always occur between processors. To achieve a higher parallel efficiency without resorting to changing the flow solver algorithm, the way in which the message passing is programmed could be examined for possible improvement, or the communication network upgraded. Far greater parallel efficiency has been obtained for the same code on a dedicated parallel machine[16]. However, in the present work the objective is to achieve a scaling-up of computing power, accepting that performance gains are limited. In this context the parallel efficiencies obtained are acceptable. For subsequent results presented in this report, a value of $\mu = 10^{-3}$ will be used where a communication cost model is employed. The same problem was also calculated using 3 to 6 processors (of Types 2 and 3), with and without the communication overhead term in the cost function. The averaged execution times, parallel efficiencies and parallel speedups are shown in Figures 4, 5 and 6 respectively. The parallel speedup

S is defined simply as

$$S = E_p P \tag{5}$$

Note that for all cases the inclusion of the communication cost element has led to improved parallel performance.

When communication overhead is taken into account, the most popular approach is to approximately account for the relative importance of computational and communication costs, as described above for the allocation problem, and also for the mesh partitioning problem. For particular applications a direct mapping of the computational domain onto processors can be visualised and exploited, as discussed in [3]. A good example of this is included in [35] where a large single block problem is decomposed into a two-dimensional array of rectangular patches to exploit the processor connectivity of a massively parallel machine where the processors are arranged in a two-dimensional array. However, this type of approach lacks generality, few computational domains decompose easily to topologies which match the target machine topology. The obvious next step in developing a communication cost model is to explicitly predict or measure the communication time, rather than approximately accounting for it. However, communication time is a function of message size, message-passing method, processor type, processor loading, network type and network loading which means creating a predictive model is prohibitively complex[3],[36]. Some researchers have attempted to measure inter-processor communication costs during run-time[36],[37] which removes some of the difficulties but the implementation of such an approach is still an order of magnitude more difficult than using the simpler method employed here, and a commensurate improvement in performance has not been demonstrated.

4 Heterogeneous load balancing

The computational cost is a function of the processor speed as well as the number of cells allocated to the processor. As discussed in Section 1, research in parallel CFD has almost exclusively concentrated on homogeneous parallel computers consisting of identical processors. However, if the parallel computer consists of a non-dedicated workstation cluster, for example that considered in this work (see Table 1), then the varying processor speeds of the heterogeneous computer must be included in the cost function to efficiently use the resource. Extending equation (3) to include different processor speeds gives the new cost function

$$H = \frac{P^2}{N^2}\sum_{q=1}^{P}\left(\frac{N_q}{k_q}\right)^2 + \mu\left(\frac{P}{N}\right)^{\frac{1}{2}}\sum_{e\leftrightarrow f} 1-\delta_{q(e),q(f)} \qquad (6)$$

where k_q is a coefficent which varies directly with the processing speed of processor q. Hence if an allocation is attempted onto two processors, the first with twice the speed of the second (i.e. $k_1/k_2 = 2$), then to minimise the computational cost two thirds of the cells would be allocated to the first processor and one third to the second. The most reliable way to determine values for k is to compare execution times on each of the processors for a standard sequential problem[25]. Vendor information concerning processing speed is unreliable for this purpose, especially when processors from more than one vendor are used. On the workstation cluster considered the commercial management software LSF is used for job control and batch scheduling. Use of such management software enables efficient use of distributed computing networks[38] and is becoming widespread in industry. LSF also provides numerous functions for interrogation of processor configuration and loading that can be simply included in user programs. An LSF function for ascertaining directly the coefficents k_q (termed 'CPU factors' in LSF) was employed in the static allocation method. Values for k from the workstation cluster used are included in Table 1. The computational cost model could be further refined. The processor speed is in-

fluenced by the proportion of accessed memory which resides in the memory cache rather than the main memory[24], although most researchers ignore or disregard this effect as insignificant.

In order to examine the effectiveness of the new cost function (6) in exploiting a heterogeneous processor set, the trial problem of 50 time steps using Grid 1 was repeated using various heterogeneous workstation sets for the parallel machine. Ideally the execution time will vary inversely with the sum k_{total} of the CPU factors k of the processors used. The execution times are plotted against $1/k_{total}$ in Figure 7. The serial execution time is included in the figure, and is joined by a straight line through the origin to indicate optimal performance. The parallel timings are presented in three groupings, results for 2, 4 and 6 processors. For each grouping, the result with the largest $1/k_{total}$ is the result for execution on a homogeneous set of Type 4 processors (the slowest available grouping). For each group, if the results formed a straight line passing through this end point, with gradient equal to the optimal gradient, then the usage of available processing power would be as efficient as the homogeneous case. We could expect the gradient to be slightly less than optimal since the communication time remains approximately constant for increasing processor speed. The results demonstrate a general trend of decreasing execution time with increasing processor power as required, except at a few points where the execution time has increased with increasing processor power. This is disappointing since a heterogeneous execution should always be at least as quick as a homogeneous execution using processors of the slowest type in the heterogeneous set. However it is perhaps unrealistic to expect a non-deterministic allocation method to always produce a near-optimal result; none of the timings are unacceptable and the general trend clearly indicates that heterogeneity is being reasonably accounted for. Examining some particular results helps to indicate the worth of employing the heterogeneous allocation model. Using two processors of Type 3 ($k_{total} = 3.2$,

$1/k_{total} = 0.3125$) the execution time is 838 seconds. Replacing one of these processors with the faster Type 1 processor (now $k_{total} = 3.5$, $1/k_{total} = 0.2857$) would not result in a faster execution if a homogeneous allocation method were employed, the faster processor having to wait while the slower computes its half of the load. With the heterogeneous allocation model the execution time was 761 seconds, a reduction of 9.2% for a 9.4% increase in computing power.

Note that in Figure 8 results are again presented for allocations determined both with and without communication cost modelling. In every case the executions were faster when the communication cost element was included.

5 Dynamic load balancing

In Section 1 it was described how the available parallel computing resource often takes the form of a non-dedicated heterogeneous network of workstations. Most parallel CFD work is performed at present using dedicated, single-user parallel computers. The presence of other users' tasks causes a serious problem for parallel applications. Even if the subset of processors to be used for the parallel task is carefully selected before run-time, either manually or using management software, the load during run-time on these processors can vary dramatically and unpredictably. A load histogram for the workstation cluster considered in this work is included in [38]. One sequential task running interactively on a processor already being used for a parallel task can double the execution time of the parallel task. Inexperienced use of a workstation can lead to disk space/main memory 'swapping' which can easily reduce the effective processing speed by an order of magnitude and have an even worse effect on the parallel task. Even seemingly benign activities such as using an internet browser or a graphical electronic mail tool can have a significant effect. This *dynamic load balancing* problem must be tackled if the objective of reliable, robust parallel execution is to be achieved.

The recorded execution times for twenty trial runs of the test problem described in Section 3 using four processors of Type 4 are shown in Figure 8 for various network conditions. The timings denoted 'quiet' are the results of overnight runs when the cluster was lightly loaded. The variation in execution time from the fastest possible is small. The timings denoted 'busy' are the results of day-time runs when the workstation cluster was moderately to heavily loaded. The timings are far less predictable, some taking 30% longer than the fastest possible. From experience, some of these timings could have been even greater. The longest execution times shown are probably due to interactive use of internet browsers and mail tools on one or more processors. Other common workstation cluster activities which would have a greater impact are the use of graphical grid generation and solution visualisation software. A 'worst case' timing is also included in the figure. After initialising the parallel task, an interactive serial task was deliberately started on one of the processors. This has increased the execution time by approximately 55%. The present dynamic load balancing problem is then to bring the 'busy' and 'worst case' timings down to the 'quiet' level. The averaged parallel efficiencies are 64%, 57% and 42% for the 'quiet', 'busy' and 'worst case' situations respectively.

Chien et al.[21],[39] present an advanced dynamic load balancing method. The effective speed of each processor is continually monitored by measuring and comparing the waiting time for the communication phase to complete on each processor, adjusting coefficients in the cost function if necessary, and re-allocating the mesh partitions if necessary. The method is very efficient for re-allocating a dynamically adapted grid, and enables eventual complete migration of the parallel task from a heavily loaded processor if necessary. The approach of Chien et al. produces impressive results but at the expense of considerable complexity and programming effort. Furthermore, the group have themselves asserted[2] that the dynamic load balancing

[2]during their ECCOMAS conference presentation[21]

problem has been over-elaborated in recent years, with very complex methods being developed to achieve increasingly small performance gains, and that the only real problem in dynamic load balancing on open workstation clusters can be presented simply as

- recognise when processor A is being heavily used by another task
- identify a lightly loaded processor B
- migrate the work of processor A onto processor B
- do all of this as quickly and simply as possible

These are also the objectives of the present work. From Figure 8, some interference of the parallel task can be tolerated (where the 'busy' timings are only slightly longer than the 'quiet'), any performance gains in sending a subset of the blocks on the 'busy' processor to other processors are likely to be small and would not justify the programming effort. The only real problem arises when a processor becomes heavily loaded, and the entire load from that processor should be migrated. Note that this also protects the interests of the interactive user, who then becomes the sole user of the processor. A dynamic re-allocation method was implemented as follows, using native LSF and PVM functions called from within the flow solver code for simplicitly rather than creating custom software:

- periodically monitor processor loadings (LSF)
- if a processor is too heavily loaded, find a candidate alternative (LSF)
- initiate a new task on the new processor (PVM), pass all the necessary information including the solution and the grid to the new task (PVM)
- stop the old task and proceed with the calculation

Note that the frequency of load monitoring and the threshold for deciding whether a processor is overloaded are decided before run-time by the user. The major part of the information passed to the new process consists of the solution and grid for the partition allocated to that processor. For the present test problem this is approximately 500Kb for one migration, which is a manageable figure for an Ethernet network. It is not necessary to pass the Jacobian matrices (for the implicit scheme) which form the major part of the total memory usage for the flow solver. The 20 trial runs in the 'worst case' scenario were repeated, but this time using the dynamic re-allocation method. The results are denoted 'dynamic' in Figure 8. The load monitoring frequency was set at every 10 time steps, recalling that 50 time steps are executed in total. It is detected that one processor is over-loaded at the first call of the load monitoring function (i.e. after 10 time steps) and the load from that processor is transferred to a lightly loaded processor. There is therefore a clear improvement over the 'worst case' execution time. An average parallel efficiency of 57% was achieved in the 'dynamic' case, as opposed to 42% for the 'worst case'. The 'dynamic' execution times could be further reduced by increasing the load monitoring frequency. Note that this model parallel CFD task has a lower associated parallel efficiency than would be the case for a real problem. It is unlikely that an engineer would use four processors for a problem which comfortably executes on two of the same processors, as in this case. It is well known that larger problems have greater parallel efficiencies (since the communication cost to computational cost ratio decreases), so since the dynamic re-allocation method effectively reduces the computational cost the performance gains for real problems would be larger. In addition, dynamic re-allocation would be of greater use for typical CFD jobs with longer execution times than the ten minutes in the current test problem.

In the event of no suitable alternative processor being available, the present method proceeds with the calculation on the same processor. This could be improved

by first attempting to contract the problem onto one less processor, or if this is not possible by automatically re-submitting the parallel task to the batch queue, re-starting from the latest checkpoint files. The present method includes periodic checkpointing to local and main disks to enable re-starting in the event of a network failure.

Most organisations with a distributed computing network employ a batch scheduling and queueing system, either developed in-house or proprietary, to enable transparent load management and achieve high productivity. Users are becoming accustomed to the convenience of high performance environments, where the system does the work of prioritizing batch jobs and selecting resources, and the user must only submit the (sequential) job and can depend on the timely arrival of the results. Ideally executing parallel tasks should be as simple and reliable. The dynamic reallocation method presented here coupled with management software such as LSF which fully supports sequential and parallel applications alike makes this possible.

6 Discussion

A domain decomposition method for a parallel, structured multiblock flow solver has been presented. The method is suitable for use on a non-dedicated parallel computer consisting of a heterogeneous workstation cluster. It has been noted that the majority of work concerning parallel CFD considers dedicated, homogeneous parallel computers. The additional difficulties encountered in a non-dedicated heterogeneous environment have been discussed. The parallel computing resource available to many engineers in small and medium-sized enterprises is of this type, although widespread use of parallel CFD to achieve a scaling-up in computational resource appears to be hindered by the perceived complexity involved. With this in mind, the domain decomposition strategy presented here attempts to deliver an effective resource in as straightforward a manner as possible.

The method employs a cost function minimisation approach. It is assumed that the multiblock grid consists of enough small blocks to enable a reasonably balanced distribution. The cost function consists of computational and communication cost elements. The time required for a processor to compute its share of the load is assumed to vary directly with the number of grid cells assigned to that processor. The time required for inter-processor communication is assumed to vary directly with the number of cells on the block boundaries which must communicate with blocks which reside on different processors. The relative importance of the cost elements is defined by a coefficient, a value for which is determined from timing experiments. The various processor speeds are ascertained using the management software LSF and are accounted for in the cost function. LSF is also used to monitor interference of other users' tasks with parallel execution, and to select a lightly loaded processor as a target for migration. The method enables effective parallel execution in the demanding environment of an open heterogeneous workstation cluster. Implementation is straightforward, facilitated by modern management software and message-passing libraries, and does not require a specialist programming or information technology effort.

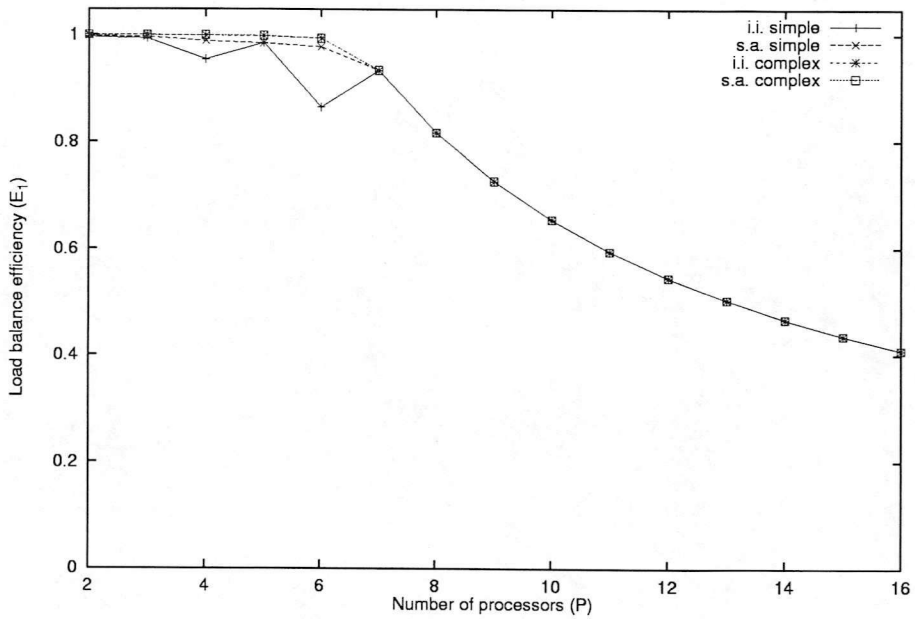


Figure 1: *Cost function minimisation test, Grid 1*

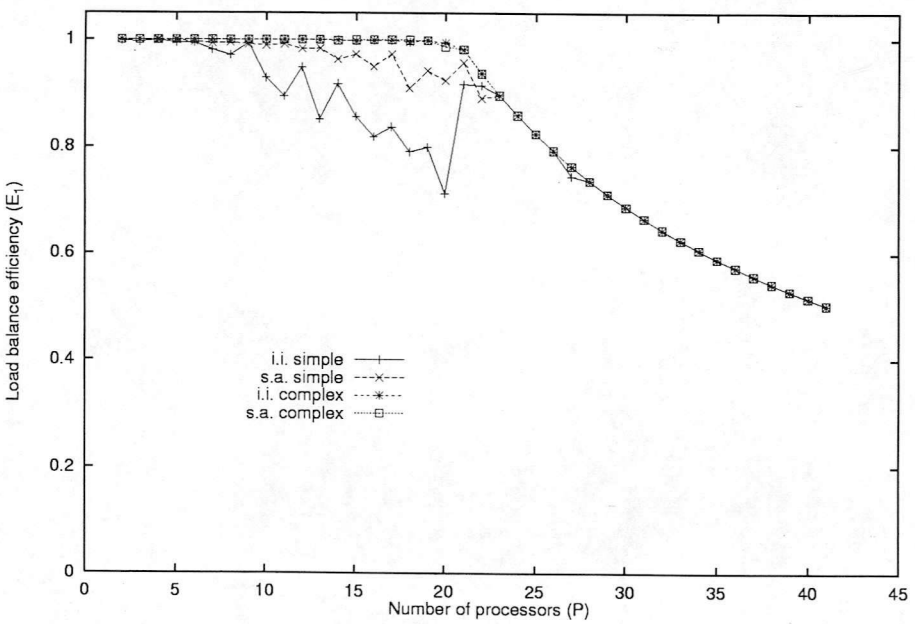


Figure 2: *Cost function minimisation test, Grid 2*

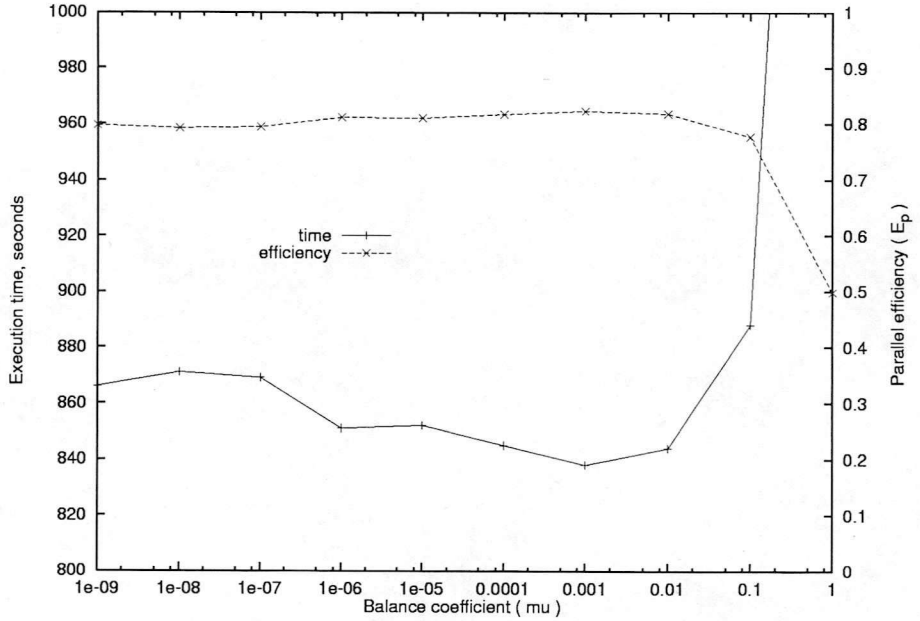


Figure 3: *Influence of balance coefficient* μ

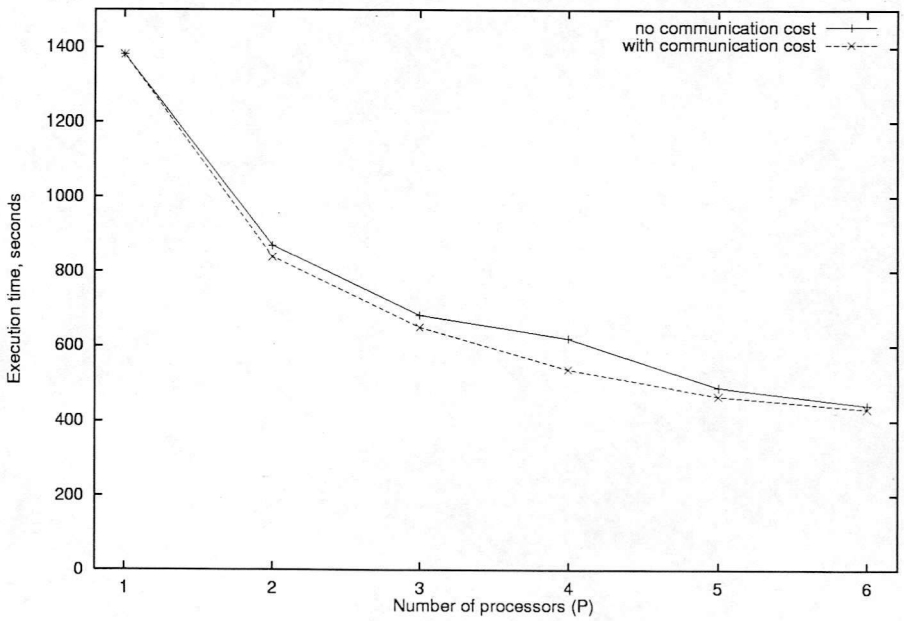


Figure 4: *Execution times, homogeneous network*

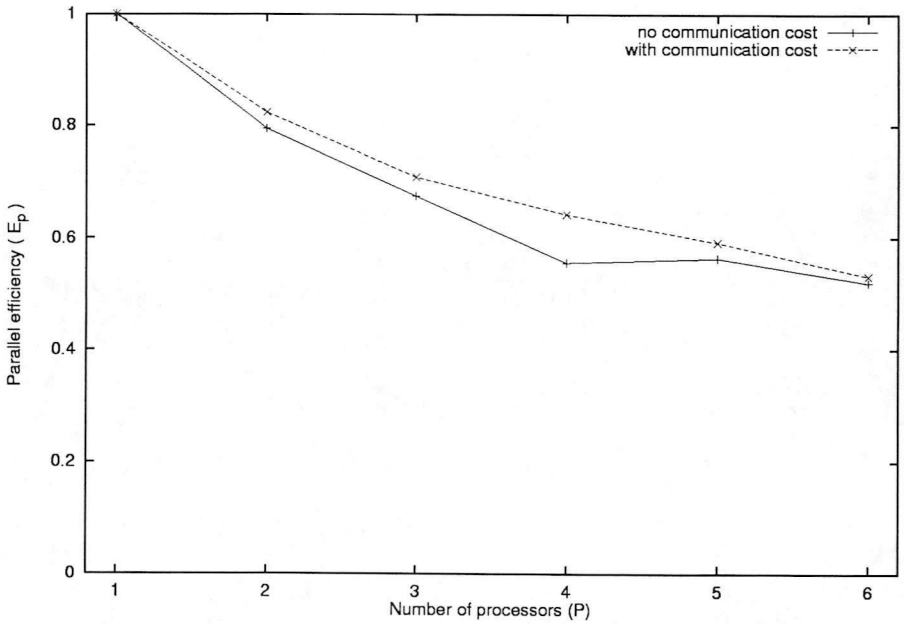


Figure 5: *Parallel efficiencies, homogeneous network*

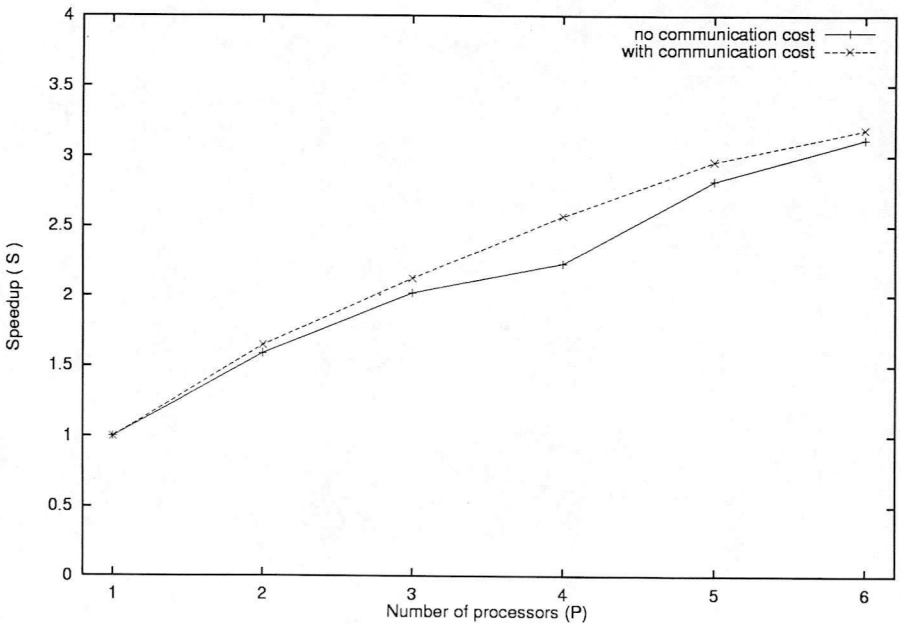


Figure 6: *Parallel speedup, homogeneous network*

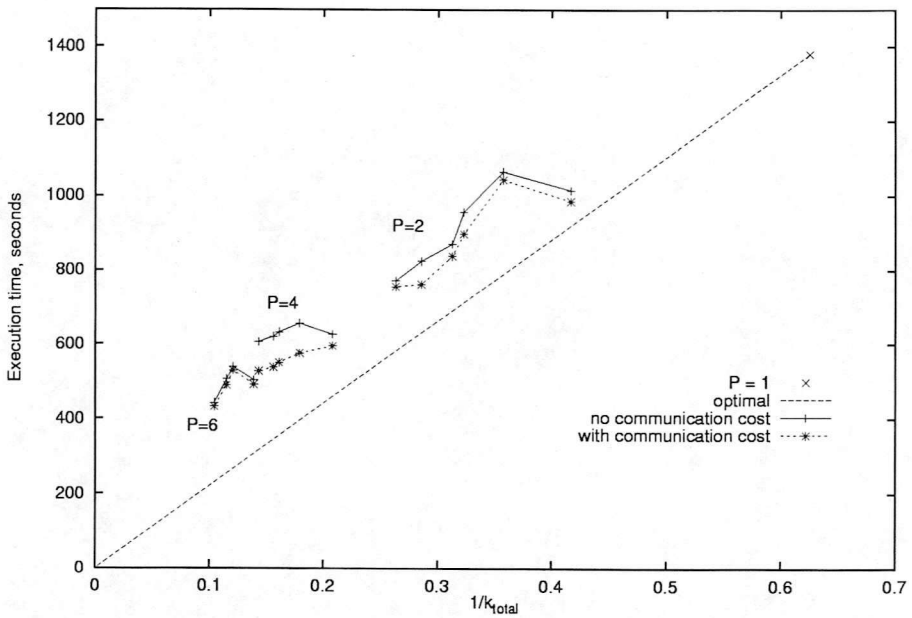


Figure 7: *Execution times, heterogeneous network*

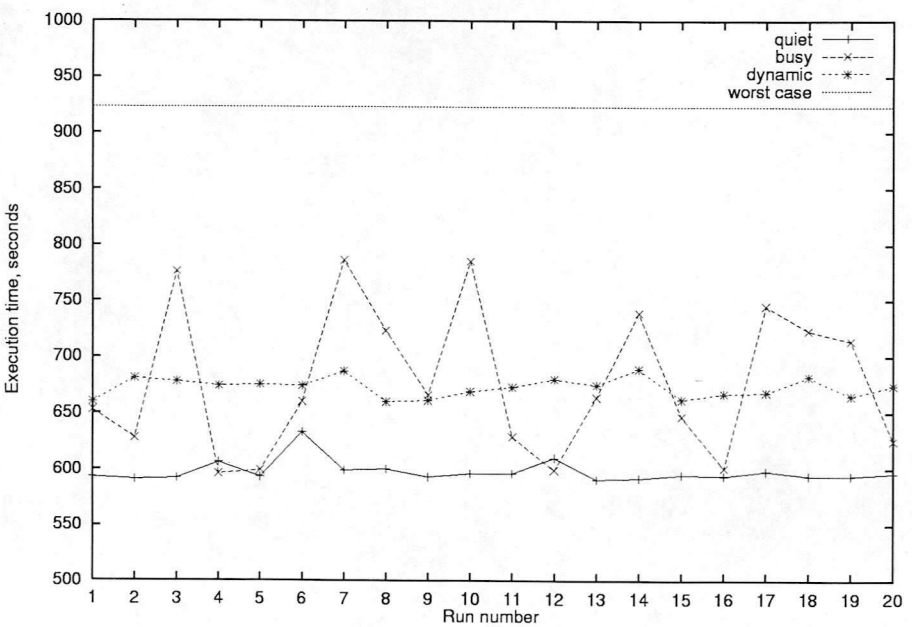


Figure 8: *Dynamic re-allocation performance*

Acknowledgements

The authors would like to thank the other members of the CFD group within the Aerospace Engineering Department, Bill McMillan in particular, for their contributions and fruitful discussions.

This work is supported by a University of Glasgow scholarship and sponsorship from DERA Bedford.

References

[1] N. Kroll. Technical Evaluation Report. In *AGARD-CP-578, Progress and Challenges in CFD Methods and Algorithms*, 1996.

[2] M.L. Sawley and J.K. Tegnér. A Comparison of Parallel Programming Models for Multi-Block Flow Computations. *Fluid Mechanics Laboratory, École Polytechnique Fédérale de Lausanne*, T-94-23, October 1994.

[3] J. De Keyser and D. Roose. Load Balancing Data Parallel Programs on Distributed Memory Computers. *Parallel Computing*, 19:1199–1219, 1993.

[4] D. Vanderstraeten and R. Keunings. Optimized Partitioning of Unstructured Finite Element Meshes. *International Journal for Numerical Methods in Engineering*, 38:433–450, 1995.

[5] N.P. Chrisochoides, E.N. Houstis and C.E. Houstis. Geometry Based Mapping Strategies for PDE Computations. In *ACM Supercomputing Conference '91*, pages 115–127, Association for Computing Machinery, New York, 1991.

[6] D.R. Emerson, A. Ecer, J. Periaux, N. Satofuka and P. Fox (editors). *Parallel Computational Fluid Dynamics, Recent Developments and Advances Using Parallel Computers.* Elsevier Science B.V., The Netherlands., 1997.

[7] P. Schiano, A. Ecer, J. Periaux and N. Satofuka (editors). *Parallel Computational Fluid Dynamics, Algorithms and Results Using Parallel Computers.* Elsevier Science B.V., The Netherlands., 1996.

[8] C. Farhat. A Simple and Efficient Automatic FEM Domain Decomposer. *Computers and Structures*, 28:579–602, 1988.

[9] Yufeng Yao. *High Order Resolution and Parallel Implementation on Unstructured Grids.* PhD thesis, Department of Aerospace Engineering, University of Glasgow, 1996.

[10] G. Karypis and V. Kumar. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *University of Minnesota, Department of Computer Science*, Technical Report 95-035, 1995.

[11] B. Hendrickson and R. Leland. The CHACO User's Guide, Version 1.0. *Sandia National Laboratories*, Technical Report SAND93-1301, 1993.

[12] C. Walshaw, M. Cross and M. Everett. Mesh Partitioning and Load Balancing for Distributed Memory Parallel Systems. In B.H.V. Topping, editor, *Advances in Computational Mechanics for Parallel Distributed Processing*, pages 97–104. Saxe-Coburg Publications, Edinburgh, 1997.

[13] R.D. Williams. Performance of Dynamic Load Balancing Algorithms for Unstructured Mesh Calculations. *Concurrency: Practice and Experience*, 3(5):457–481, 1991.

[14] J.B. Vos, V. Van Kemenade, A. Ytterström and A.W. Rizzi. Parallel NSMB: An Industrialized Aerospace Code for Complete Aircraft Simulations. In P. Schiano et al., editor, *Parallel Computational Fluid Dynamics: Algorithms and Results using Advanced Computers*, pages 49–58. Elsevier Science B.V. Amsterdam, 1996.

[15] D.D. Knight. Parallel Computing in Computational Fluid Dynamics. In *AGARD-CP-578, Progress and Challenges in CFD Methods and Algorithms*, 1996.

[16] C.A. Masson, M. Woodgate, W. McMillan, F. Cantariti, K.J. Badcock and B.E. Richards. The Development of an Affordable High Powered Clustered Computer for Parallel Applications. *submitted for publication*, 1998.

[17] H.P.F. Forum. High Performance Fortran Language Specification. *Scientific Programming*, 2:1–170, 1993.

[18] M.P.I. Forum. MPI: A Message-Passing Interface Standard. *Journal of Supercomputer Applications*, 8, 1994.

[19] V. Sunderam, G. Giest, J. Dongarra and R. Manchek. PVM: A Framework for Parallel Distributed Computing. *Journal of Concurrency: Practice and Experience*, 2:315–339, 1990.

[20] B. Eisfeld, H. Ritzdorf, H. Bleeke and N. Kroll. Portable Parallelization of a 3D Euler/Navier-Stokes Solver for Complex Flows. In *AGARD-CP-578, Progress and Challenges in CFD Methods and Algorithms*, 1996.

[21] Y.P. Chien, J. Huang and A. Ecer. A Two-Stage Computer Load Balancing Method for Parallel Finite Element Analysis Using the Domain Decomposition Approach. In K.D. Papailiou, D. Tsahalis, J. Périaux and D. Knörzer, editor, *Invited Lectures, Minisymposia and Special Technological Sessions of the Fourth European Computational Fluid Dynamics Conference*, pages 164–170. John Wiley and Sons, U.K., 1998.

[22] P. Henriksen and R. Keunings. Parallel Computation of the Flow of Integral Viscoelastic Fluids on a Heterogeneous Network of Workstations. *International Journal for Numerical Methods in Fluids*, 18:1167–1183, 1994.

[23] C. Fischberg, C. Rhie, R. Zacharias, P. Bradley and T. DesSureault. Using Hundreds of Workstations for Production Running of Parallel CFD Applications.

In A. Ecer et al., editor, *Parallel Computational Fluid Dynamics: Implementations and Results using Parallel Computers*, pages 9–22. Elsevier Science B.V. Amsterdam, 1995.

[24] K. Stüben, H. Mierendorff, C.-A. Thole and O. Thomas. EUROPORT: Parallel CFD for Industrial Applications. In P. Schiano et al., editor, *Parallel Computational Fluid Dynamics: Algorithms and Results using Advanced Computers*, pages 39–48. Elsevier Science B.V. Amsterdam, 1996.

[25] Y.P. Chien, F. Carpenter, A. Ecer and H.U. Akay. Load-Balancing for Parallel Computation of Fluid Dynamic Problems. *Computer Methods in Applied Mechanics and Engineering*, 120:119–130, 1995.

[26] K.J. Badcock, W. McMillan, M.A. Woodgate, B.J. Gribben, S. Porter and B.E. Richards. Integration of an Implicit Multiblock Code into a Workstation Cluster Environment. In P. Schiano et al., editor, *Parallel Computational Fluid Dynamics: Algorithms and Results using Advanced Computers*, pages 408–415. Elsevier Science B.V. Amsterdam, 1996.

[27] F. Cantariti, L. Dubuc, B.J. Gribben, M. Woodgate, K.J. Badcock and B.E. Richards. Approximate Jacobians for the Euler and Navier-Stokes Equations. *University of Glasgow, Aero Report 9705*, 1997.

[28] L. Dubuc, F. Cantariti, M. Woodgate, B.J. Gribben, K.J. Badcock and B.E. Richards. Solution of the Euler unsteady equations using deforming grids. *University of Glasgow, Aero Report 9704*, 1997.

[29] L. Dubuc, F. Cantariti, M. Woodgate, B. Gribben, K.J. Badcock and B.E. Richards. Solution of the Unsteady Euler Equations Using an Implicit Dual-Time Method. *AIAA Journal*, 36:1417–1424, 1998.

[30] C. Farhat and M. Lesoinne. Automatic Partitioning of Unstructured Meshes for the Parallel Solution of Problems in Computational Mechanics. *International Journal for Numerical Methods in Engineering*, 36:745–764, 1993.

[31] R.A. Rutenbar. Simulated Annealing Algorithms: An Overview. *IEEE Circuits and Devices Magazine*, pages 19–26, Jan. 1989.

[32] R.H.J.M. Otten and L.P.P.P. van Ginneken. *The Annealing Algorithm.* Kluwer Academic Publishers, 1989.

[33] W.H. Press, S.A. Teukolsky, W.T. Vetterling and B.P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge University Press, 1992.

[34] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller and E. Teller. Equation of State Calculations by Fast Computing Machines. *Journal of Chemical Physics*, 21:1087–1092, 1953.

[35] C.C. Wong, F.G. Blottner and J.L. Payne. Implementation of a Parallel Algorithm for Thermo-Chemical Nonequilibrium Flow Simulation. *AIAA paper*, 95-0152, 1995.

[36] Y.P. Chien, A. Ecer, H.U. Akay and S.Secer. Communication Cost Function for Parallel CFD in a Heterogeneous Environment Using Ethernet. In P. Schiano et al., editor, *Parallel Computational Fluid Dynamics: Algorithms and Results using Advanced Computers*, pages 1–10. Elsevier Science B.V. Amsterdam, 1996.

[37] A. Ålund, P. Lötsdedt and M. Sillén. Parallel Solution of Industrial Compressible Flow Problems with Static Load Balancing. In P. Schiano et al., editor, *Parallel Computational Fluid Dynamics: Algorithms and Results using Advanced Computers*, pages 336–343. Elsevier Science B.V. Amsterdam, 1996.

[38] B. McMillan. Parallel Computing on Workstation Clusters. *Informal Symposium on Implicit Methods, Parallel Computing and Aerospace Applications*, Department of Aerospace Engineering, University of Glasgow, `http://www.aero.gla.ac.uk/Research/ASCU/Index.html`, 14th June 1995.

[39] Y.P. Chien, A. Ecer, H.U. Akay, F. Carpenter and R.A. Blech. Dynamic Load Balancing on a Network of Workstations for Solving Computational Fluid Dynamic Problems. *Computer Methods in Applied Mechanics and Engineering*, 119:17–33, 1994.