Parallelising FLITE3D – a Multigrid Finite Element Euler Solver

Y. F. Hu*, D. R. Emerson, M. Ashworth,

K. C. F. Maguire, R. J. Blake

CLRC Daresbury Laboratory

Warrington, United Kingdom

Abstract

FLITE3D is a multigrid Euler solver. It is used extensively by British Aerospace in aircraft design and simulation. This paper presents experiences in parallelising this industrial code. Owing to the employment of an agglomeration-based multigrid technique, the communication overhead on the coarser meshes could readily erode any gain from the use of parallel computers. The parallelisation of the code therefore required careful design and implementation. The strategy adopted in the parallelisation of the code, including the use of data structures and communication primitives, is described. Numerical results are presented to demonstrate the efficiency of the resulting parallel code.

Keywords Parallel computing, unstructured mesh, multigrid algorithm.

*Corresponding author. E-mail: y.f.hu@dl.ac.uk. Tel: (44) 1925 603683. Fax: (44) 1925 603634

1 Introduction

CFD simulations are now increasingly being used as a complementary tool or an alternative to wind tunnel experiments in the aerospace industry. The drive towards faster, more accurate simulations inevitably leads to the requirement of parallel super-computers.

The FLITE3D suite of codes [1] is a complete set of CFD routines for obtaining Euler predictions over complex aerodynamic configurations. FLITE3D was developed by Computational Dynamics Research Limited at the University of Wales Swansea and supplied to British Aerospace. It is now under extensive use and development by British Aerospace.

This work is the result of a project in parallelising the FLITE3D codes, funded by British Aerospace. The codes of concern are mainly the flow solver, FLITE3D-FS, and the preprocessor, FLITE3D-PP. The flow solver is an explicit finite element Euler solver based on unstructured meshes, using Runge-Kutta time integration. It employs an agglomeration-based multigrid algorithm [2, 3, 4] to speed up the calculation. Use of a multigrid algorithm makes efficient parallelisation more challenging.

Parallelisation of structured mesh based multigrid algorithms and unstructured mesh based non-nested multigrid algorithms have been extensively documented in previous work (e.g., [5, 6, 7, 8]). However there have been few attempts, so far as the authors are aware, to parallelise unstructured agglomeration-based multigrid algorithms. In [9], an agglomeration-based multigrid algorithm was parallelised, based on independent partitioning of each level of mesh in the multigrid hierarchy. The advantage of this strategy is that by partitioning each level independently, load balance as well as the processor boundary sizes on each mesh level is easier to control. The disadvantage is that the partitions of coarser meshes may

bear no relation to the partition of the corresponding finer meshes. Restriction and prolongation would therefore have to be preceded by a substantial communication step between the two levels. To reduce the inter-mesh communication, the coarse level partition were renumbered to maximize the overlap with the fine level partition. Reasonable scalability has been reported on very large meshes [9]. This approach may be further improved by the use of advanced partitioning tools (e.g., [10]) to force the coarse level partition to be as close to the fine level partition as possible.

In this work we have designed and implemented an alternative approach, which eliminates the need for communication between different mesh-levels during the restriction and prolongation phases. On the finest mesh, following our previous work [11] on FELISA [12, 13] (an inviscid flow solution package employing the same technology as FLITE3D, originally developed by Imperial College in London, University of Wales at Swansea and Massachusetts Institute of Technology for NASA), we have taken the route of using element-based partitioning, with the corresponding data structure to ensure the correct numerical algorithm. For the multigrid part of the code, the partition of a coarser mesh is inherited from the corresponding finer mesh. A concept of "active nodes" is introduced. These ensure that the inter-grid communication during the restriction and prolongation phases of the multigrid algorithm is completely eliminated, and the coarse mesh calculation is reasonably load balanced. Asynchronous communication is also employed which has been found to improve the efficiency of the parallel code significantly. Numerical results show that our approach leads to an efficient parallel multigrid finite element code when the size of the mesh is relatively large.

In Section 2, the equations and numerical algorithms for the flow solver part of FLITE3D are introduced. In Section 3, the methodology of single grid parallelisation is given. Strategies employed in the parallelisation of the multigrid phase

of the code are detailed in Section 4. The performance of the parallel FLITE3D code and the effect of asynchronous communication is discussed in Section 5. Section 6 concludes the paper.

2 Introduction of FLITE3D-FS

2.1 The Euler solver

The Euler algorithm is solved by a Galerkin finite element method using Runge-Kutta time integration, explicit artificial dissipation [14, 15, 16] and a side-based data structure [17]. In the following this solution procedure is briefly summarised. Further details can be found in [17, 12].

The compressible Euler equations can be expressed as

$$\frac{\partial \boldsymbol{U}}{\partial t} + \sum_{j=1}^{3} \frac{\partial \boldsymbol{F}^j}{\partial x_j} = \boldsymbol{0} \tag{1}$$

where $\boldsymbol{U}$, the unknown vector of conserved variables, and $\boldsymbol{F}^j$, the corresponding inviscid flux vectors, are defined by

$$\boldsymbol{U} = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ \rho E \end{pmatrix}$$

and

$$\left(\boldsymbol{F}^1(\boldsymbol{U}), \boldsymbol{F}^2(\boldsymbol{U}), \boldsymbol{F}^3(\boldsymbol{U}) \right) = \begin{pmatrix} \rho u & \rho v & \rho w \\ \rho uu + p & \rho vu & \rho wu \\ \rho uv & \rho vv + p & \rho wv \\ \rho uw & \rho vw & \rho ww + p \\ (\rho E + p)u & (\rho E + p)v & (\rho E + p)w \end{pmatrix}$$

Here (u, v, w) are the components of velocity, p the pressure, E the total energy and ρ the density. The set of equations is completed by the addition of the perfect gas equation of state

$$p = (\gamma - 1)\rho \left(E - \frac{1}{2}(u^2 + v^2 + w^2) \right)$$

where γ is the ratio of the specific heats.

The solution to these equations is obtained over a spatial domain Ω with boundary Γ. The weak formulation for (1) would be to find $\boldsymbol{U}(\boldsymbol{x}, t)$ such that

$$\int_\Omega \frac{\partial \boldsymbol{U}}{\partial t} W d\Omega \;=\; \sum_{j=1}^{3} \int_\Omega \boldsymbol{F}^j \frac{\partial W}{\partial x_j} d\Omega \;-\; \sum_{j=1}^{3} \int_\Gamma \boldsymbol{F}^j n^j W d\Gamma \tag{2}$$

for every suitable weighting function $W(\boldsymbol{x})$, where n^j denotes the j-th component of the unit outward normal vector, $\boldsymbol{n}$, to the boundary Γ.

A finite element approximation is constructed. The spatial domain Ω is represented by an unstructured assembly of tetrahedral elements. A piecewise linear approximation is sought of the form

$$\boldsymbol{U}^*(\boldsymbol{x}, t) = \sum_J \boldsymbol{U}_J(t) N_J(\boldsymbol{x}) \tag{3}$$

where the summation is over each node J $(1 \leq J \leq n_p)$ of the mesh, $\boldsymbol{U}_J(t)$ denotes the (unknown) value of the approximation U^* at node J at time t, and $N_J(\boldsymbol{x})$ is the finite element shape functions associated with node J. Using the variational statement of (2), the Galerkin approximate solution is constructed as the function $\boldsymbol{U}^*$ such that

$$\int_\Omega \frac{\partial \boldsymbol{U}^*}{\partial t} N_I d\Omega \;=\; \sum_{j=1}^{3} \int_\Omega \boldsymbol{F}^j(\boldsymbol{U}^*) \frac{\partial N_I}{\partial x_j} d\Omega \;-\; \sum_{j=1}^{3} \int_\Gamma \boldsymbol{F}^j(\boldsymbol{U}^*) n^j N_I d\Gamma$$

for each N_I and for all $t > 0$.

The integrals can be evaluated by summing the individual element and boundary face contributions as

$$\sum_{e\in I}\int_{\Omega_e}\frac{\partial \boldsymbol{U}^*}{\partial t}N_I d\Omega \;=\; \sum_{j=1}^{3}\sum_{e\in I}\int_{\Omega_e}\boldsymbol{F}^j(\boldsymbol{U}^*)\frac{\partial N_I}{\partial x_j}d\Omega - \sum_{j=1}^{3}\sum_{e\in I}\int_{\Gamma_e}\boldsymbol{F}^j(\boldsymbol{U}^*)n^j N_I d\Gamma \quad (4)$$

where the summations are over the elements containing node I.

Substituting $\boldsymbol{U}^*$ by equation (3), the left-hand side of the equation (4) can be evaluated to give

$$\begin{aligned}\sum_{e\in I}\int_{\Omega_e}\frac{\partial \boldsymbol{U}^*}{\partial t}N_I d\Omega &= \sum_{e\in I}\int_{\Omega_e}\sum_{J=1}^{n_p}\frac{d\boldsymbol{U}_J}{dt}N_J N_I d\Omega \\ &= \sum_{J=1}^{n_p}\left\{\int_{\Omega}N_J N_I d\Omega\right\}\frac{d\boldsymbol{U}_J}{dt} \approx [M_L]_I\frac{d\boldsymbol{U}_I}{dt} \qquad (5)\end{aligned}$$

Here M_L is the lumped finite element mass matrix, which is an approximation to the true mass matrix M, with

$$M_{IJ} = \int_{\Omega}N_I N_J d\Omega$$

The approximation is valid since the nodal shape functions are non-zero only over those elements containing that node. Hence M is an almost diagonal matrix.

The right hand side of equation (4) can be approximated over a tetrahedron Ω_e, by taking the average of the flux over the four nodes, $I,\ J,\ K$ and L, of the tetrahedron:

$$\int_{\Omega_e}\boldsymbol{F}^j\frac{\partial N_I}{\partial x_j}d\Omega \;\approx\; \frac{\Omega_e}{4}\left.\frac{\partial N_I}{\partial x_j}\right|_e\{\boldsymbol{F}_I^j+\boldsymbol{F}_J^j+\boldsymbol{F}_K^j+\boldsymbol{F}_L^j\} \qquad (6)$$

where Ω_e is the volume of the tetrahedron.

In a side-based data structure, suppose that node I is connected, by sides of the mesh, to nodes $J_1, J_2, \ldots, J_{m_I}$ (assuming here that I is an interior node. For boundary nodes the following procedure needs to be suitably modified), using equations (5) and (6), it can be shown that equation (4) can be written as

$$[M_L]_I\frac{d\boldsymbol{U}_I}{dt} = \sum_{j=1}^{3}\sum_{s=1}^{m_I}C_{IJ_s}^j(\boldsymbol{F}_I^j+\boldsymbol{F}_{J_s}^j) \qquad (7)$$

where

$$C^j_{IJ_s} = \sum_{e \in IJ_s} \frac{\Omega_e}{4} \left. \frac{\partial N_I}{\partial x_j} \right|_e$$

Because the mass matrix is lumped into a diagonal matrix, time integration is explicit via a K-stage Runge-Kutta procedure:

$$\boldsymbol{U}^{n+k/K} = \boldsymbol{U}^n - \alpha_k \Delta t [M_L]^{-1} \left[\boldsymbol{R}\left(\boldsymbol{U}^{n+(k-1)/K}\right) - \boldsymbol{D}\left(\boldsymbol{U}^{n+(k-1)/K}\right) \right], \quad k = 1, \ldots, K \tag{8}$$

where $\boldsymbol{R}$ is the right hand side of (7) and $\boldsymbol{D}$ the explicitly added artificial dissipation. Δt is the maximum allowable local time step, which varies for each node. M_L is the lumped mass matrix. Usually $K = 5$ and $\alpha_1 = 1/4$, $\alpha_2 = 1/6$, $\alpha_3 = 3/8$, $\alpha_4 = 1/2$, and $\alpha_5 = 1$. In the interests of computational efficiency, the dissipation term is only updated at $k = 1$ and kept frozen for the remaining 4 stages of the Runge-Kutta procedure.

Far field boundary conditions are applied through the modification of the boundary flux as the solution of an approximate one-dimensional Riemann problem. Wall boundary condition of zero normal velocity is imposed by projection. Further details for the choice of the time steps, the artificial dissipation scheme, flux calculation and boundary treatment can be found in [12, 18].

2.2 The Multigrid Acceleration

The convergence rate of the aforementioned explicit integration scheme is strongly dependent on the size of the mesh. This is because the largest time step that can be employed is proportional to the cell size. When used for elliptic systems, explicit integration schemes of the kind (8) are known to suffer from an inability to eliminate smooth errors. The idea of multigrid [19] is to employ a series of meshes, each coarser than the its predecessor, such that components of smooth low-frequency errors will appear as high-frequency errors on the coarser mesh

and can therefore be readily damped using the above integration scheme. The increasing cell size on the coarser meshes also allows larger time steps to be used.

Different multigrid strategies have been proposed in the literature [20]. One possibility is to employ a sequence of independently generated coarse meshes [17, 21]. In this approach, the meshes are generally non-nested. It is therefore necessary to construct the prolongation operator through linear interpolation, with the restriction operator taken as the transpose of the prolongation operator. However, in contrast to structured meshes, where coarse meshes can be generated easily and quickly, a non-nested strategy for unstructured meshes are more difficult and time consuming.

An alternative strategy [2, 3, 4] is to form the coarse mesh by agglomerating neighbouring nodes of a fine mesh. In this approach, seed nodes are merged with neighbouring nodes to form coarser mesh nodes. Looking at it from another perspective, control volumes of nodes to be merged are agglomerated to form polyhedra. The resulting coarse mesh is therefore not of conventional tetrahedral type. The coarse mesh equation on each coarse node is formed directly by combining the coefficients of the equations for the fine "leaf" nodes. This is the approach adopted in FLITE3D.

Figure 1 illustrates the procedure of agglomeration on a simple mesh. Figure 2 illustrates a tetrahedral mesh around an M6 wing, the corresponding control volumes and two levels of coarsened meshes. For illustration purposes, the control volumes of the nodes on the two coarse meshes are drawn, instead of the nodes themselves. Further details about this multigrid technique based on agglomeration can be found in [2, 3, 4].

The nonlinearity in the right hand side of (8) is treated with the full multigrid scheme [19]. The details of the multigrid strategy employed are best described in a two-grid setting. Throughout the rest of this paper, the subscripts f and c are

used to denote fine and coarse mesh quantities, respectively. Denote the right hand side of (8) as $\boldsymbol{r}$, and simplify the notation $n + k/K$ as k, then at the last stage of the Runge-Kutta procedure, equation (8) on a fine mesh can be written as

$$\boldsymbol{U}_f^k = \boldsymbol{U}_f^0 - \alpha_k \Delta t [M_L]^{-1} \boldsymbol{r}(\boldsymbol{U}_f^{k-1}), \quad k = K$$

At this point the flow field and the corresponding residual are restricted to the coarse mesh to give the initial flow field of

$$\boldsymbol{U}_c^0 = I_f^c \; \boldsymbol{U}_f^K \tag{9}$$

and initial residual of

$$\boldsymbol{r}_c^0 = I_f^c \; \boldsymbol{r}(\boldsymbol{U}_f^K) \tag{10}$$

where I_f^c in the two equations are restriction operators which may not necessarily be the same. The calculation then proceeds on the coarse mesh with

$$\boldsymbol{U}_c^k = \boldsymbol{U}_c^0 - \alpha_k \Delta t [M_L]^{-1} (\boldsymbol{r}(\boldsymbol{U}_c^{k-1}) + \boldsymbol{r}_c^0 - \boldsymbol{r}(\boldsymbol{U}_c^0)), \quad k = 1, \ldots, K \tag{11}$$

The two extra terms on the right hand side of (11) ensure that, when the fine mesh has fully converged ($\boldsymbol{r}_c^0 = I_f^c \; \boldsymbol{r}(\boldsymbol{U}_f^K) = 0$), the coarse mesh equation will be automatically satisfied by the initial field (9). Thus it is the fine mesh residual which is driving the coarse grid solution. The solution of the coarse mesh will be prolongated and added to the fine mesh as a correction, to form the initial field for the next fine mesh integration:

$$\boldsymbol{U}_f^0 = \boldsymbol{U}_f^K + I_c^f (\boldsymbol{U}_c^K - \boldsymbol{U}_c^0)$$

where I_c^f is the prolongation operator.

In FLITE3D residuals on the fine mesh are lumped to form the residuals on the coarse mesh. The fine mesh fields are volume averaged to give the initial coarse

mesh field. The prolongation operation is simply carried out through injection. For the Euler equation this combination of the restriction and prolongation is sufficient to satisfy the fundamental multigrid principle [19], which states that the sum of order of restriction and prolongation operators used should be greater than the order of the equation solved. Figure 3 illustrates the performance of the multigrid algorithm compared with the single grid algorithm.

3 Single-grid Parallelisation

The single grid parallelisation is carried out by domain decomposition. The finite element mesh is partitioned using the graph partitioning package METIS [22]. There are a number of approaches to the partitioning and the subsequent parallelisation of finite element algorithms. The partitioning could either be element-based, or node-based. Following earlier work in parallelising a single grid finite element solver FELISA [11], the element-based approach is adopted. Here, the dual graph of the tetrahedral mesh is used by METIS to generate a partition of elements. As illustrated by Figure 4, after the partitioning, each element belongs uniquely to a processor. Each side also belongs uniquely to a single processor. On the other hand, nodes of the mesh at the boundary of the partition are duplicated. Figure 5 shows a mesh around a Dassault Aviation FALCON aircraft, partitioned into 4 subdomains using METIS [22].

The main operations of FLITE3D-FS are either side-based, or face-based for boundary calculations. For example, the right-hand-side term $\boldsymbol{R}$ in equation (8) is calculated by the pseudo-code in Figure 6. Because of this, in the parallel calculation, it is important that the sides are unique to processors (faces on the physical boundary are unique to processors anyway). This is shown in Figure 4, where sides on the processor boundary of processor 1 are drawn with broken lines

to illustrate that these sides do not exist on processor 1. The parallel pseudo-code corresponding to the sequential code in Figure 6 is given in Figure 7. The parallel code is essentially the same as the sequential code, except that the loops are over the number of sides and boundary faces residing locally on each processor. An exchange and summation operation is also carried out at the end of the local summations so as to collect the partial sums on each processor to give the correct $\boldsymbol{R}$ values. For instance, in Figure 4, after the local summations on processor 1, node v will have contributions from edges (sides) $e_1,\ e_2,\ e_3$ and e_4. Likewise, after the local summations on processor 2, node v will have a contribution from edge $e5$. By exchanging the $\boldsymbol{R}$ terms between the nodes on the processor boundary and summing them, node v on both processors will have the correct contribution from all the five edges $e_1, \ldots, e_5$. This exchange and sum operation will also be used later in parallelising the multigrid algorithm, and shall be referred to as the ExSum operation.

4 Strategies for Multigrid Parallelisation

During the multigrid process, the restriction of residuals is achieved by simply summing the residual values of the fine nodes which agglomerate into a coarse node. The restriction for the flow variables is performed similarly, except that the sum is weighted by the volumes controlled by the fine nodes (normalised). The prolongation is carried out by injecting the correction on the coarse nodes to the constituting fine nodes.

4.1 Partition of the coarse meshes

It can be seen that the data dependence during the multigrid process is between the coarse mesh nodes and their corresponding fine mesh nodes. Define the

location of a node to be the processor it is on, and the *leaves* to be the fine mesh nodes that corresponds to a coarse node (the *root*). In a parallel setting, the leaves of a coarse root may be scattered around several processors.

One way of decomposing a coarse mesh is to partition it independently. This approach has been adopted in [9]. The advantage of this approach is that, by partitioning each level independently, load balance as well as interprocessor communication on each level is easier to control. The disadvantage is that the partition of a coarse mesh may bear no relation to the partition of its corresponding finer mesh. Restriction and prolongation have to, therefore, be preceded by a substantial communication step.

An alternative approach, which we have proposed for this work, is for the coarse mesh to inherit its partition from that of the corresponding fine mesh. Through careful implementation, the need for communication between different levels of meshes is eliminated. This approach also makes the coding relatively straightforward with almost no extra code necessary, other than what is required for parallelising the single grid algorithm.

Roots inherit their locations from their leaves. In others words, if the leaves of a root exist on three processors, then the root will also be duplicated on three processors. The main reason for this duplication of root nodes is that, through this arrangement, communication between different levels of mesh are eliminated completely. The only communication related to the multigrid procedure is that of the ExSum operation seen in Figure 7, performed once on the coarse meshes after local restriction, as shall be explained in Sections 4.2 and 4.3.

The inheritance relationship between roots and leaves is illustrated in Figure 8. In the top part of the figure, a simple one-dimensional mesh with seven nodes and six sides is shown agglomerated into a mesh with three nodes and two sides. In the bottom half of the figure, the mesh at level i was partitioned into two as

shown. Next, the locations of the three root nodes $v1$, $v2$ and $v3$ at level $i+1$ is decided. Root $v1$ has all its leaves ($u1$ and $u2$) on processor one, so it will reside only on processor one. Root $v3$ has all its leaves ($u6$ and $u7$) on processor two, and will therefore reside only on processor two. However, because root $v2$ has leaves existing on both processor one ($v3$ and $v4$) and processor two ($v4$ and $v5$), it is duplicated on both processors.

Coarse mesh sides remain unique among processors, as in the single grid case.

4.2 Parallelisation of the restriction operation

As mentioned at the beginning of Section 4, restriction of residuals is through simple summation over the fine grid nodes which agglomerate into a coarse node, while restriction of flow variables is through summation of flow variables over the corresponding fine grid nodes, weighted by the volumes controlled by these fine nodes, with the weights normalised for conservation of flow fields. The pseudo-code for the sequential operation on a flow variable U is given in Figure 9.

To parallelise this operation, the restriction can be performed by summing the contributions from the local leaves, followed by an ExSum operation. However for roots on the processor boundary, there is a possibility of over-counting. For example, in Figure 8, after local restrictions, on processor one:

$$\boldsymbol{U}_c(v2) = w_f(u3)\boldsymbol{U}_f(u3) + w_f(u4)\boldsymbol{U}_f(u4) \quad \text{(on processor one)}$$

and on processor two:

$$\boldsymbol{U}_c(v2) = w_f(u4)\boldsymbol{U}_f(u4) + w_f(u5)\boldsymbol{U}_f(u5) \quad \text{(on processor two)}$$

After an ExSum operation, which exchanges and sums the field $\boldsymbol{U}_c$ on the processor boundary (in this case only node $v2$ is on the processor boundary),

$$\boldsymbol{U}_c(v2) = w_f(u3)\boldsymbol{U}_f(u3) + 2w_f(u4)\boldsymbol{U}_f(u4) + w_f(u5)\boldsymbol{U}_f(u5) \quad \text{(on both processors)}$$

Therefore the contribution from the fine node $u4$ is double counted. In general this over-counting will happen to the nodal values of all fine nodes that are duplicated among processors.

The concept of an *active* node is therefore introduced to avoid this over-counting. A fine mesh node is active on one and only one processor, even if it is duplicated among many processors. For example, in Figure 8 node $v4$ is active only on processor one. The node $v4$ on processor two is thus marked with an empty circle, rather than a solid one. A broken line is used to link this node with its root $v2$ on processor two to illustrate that there is no contribution from the non-active leaf node $v4$ to its root node $v2$.

By taking contributions only from active fine nodes residing locally, followed by an ExSum operation, the correct parallel restriction operation is achieved. The pseudo-code for parallel restriction is given in Figure 10.

4.3 Parallelisation of the prolongation operation

The prolongation is through a simple injection of the coarse mesh correction to the leaf nodes on the finer mesh. Because the locations of root nodes are inherited from their leaf nodes, a root node and its leaf nodes always exist on the same processor. Therefore, the parallel prolongation operation employs exactly the same code as the sequential prolongation operation, and the injection of the coarse mesh correction is carried out locally on each processor. The pseudo-code for the parallel prolongation is given in Figure 11.

5 Performance of the Parallel FLITE3D Code

The parallel FLITE3D-FS code has been implemented using the MPI message passing standard. The code has been tested on a Cray T3E-900 parallel super-

computer, with 450 MHz Alpha (EV5/6) processors, each having a peak performance of 900 MFlop/s and 128 Mbytes of memory. The observed latency and peak bandwidth for point to point communication using MPI is around 30 μs and 180 MB/s respectively. It is possible to achieve better communication performance by using the Cray specific SHMEM facility, although we have not implemented SHMEM in FLITE3D-FS for reasons of portability.

Initially the message passing in the ExSum was implemented using MPI_SEND and MPI_RECV, as shown in Figure 12. These are blocking operations, in the sense that the calls do not return until the message to be sent is copied into a buffer, or the message to be received is copied into the receiving array. Although blocking send and receive have the advantage that the sending and receiving array *array_send* and *array_recv* can be re-used immediately, the performance in using the blocking communication primitives is not satisfactory. To illustrate the problem, we have applied the FLITE3D-FS on a small mesh around the M6 wing, with 29784 nodes, 208880 sides and 173176 tetrahedra. On this small problem, the speedup (Figure 14) of the multigrid code with 4 mesh levels on 64 processors is only 12. This is partly due to the fact that this case is quite small. As illustrated in Table 1, on 64 processors, each processor has, on average, only $208880/64 \approx 3264$ sides on the finest level. This reduces rapidly to $1199/64 \approx 19$ sides per processor on the coarsest mesh. However, the total length of messages in the ExSum routine reduces more slowly from 27254 words to 8482 words per field variable. Thus on the coarse meshes, more time is spent in the communication rather than computation. This slow reduction (or even increase from level 3 to level 4) in the total message length is an unfortunate consequence of the fact that a coarse root that has leaves on P processors is duplicated on these P processors. Therefore more and more processors share nodes as one moves from the fine to the coarser meshes. Two processors are said to be *neighbours* if they share more

than one node. The average number of neighbours per processor for this small case increases from 11 on the finest mesh to 38 on the 4th level of mesh.

By using a non-blocking MPI implementation of the ExSum routine, illustrated in Figure 13, it is possible to reduce the communication time, thus improving the speedup from 11 to 17. The disadvantage is that now a separate array is needed to pack the processor boundary data relating to each neighbouring processor, thus increasing the memory requirement slightly.

On a moderately large case, which involves a mesh around an F18 fighter aircraft with 518519 nodes, 3786836 sides and 3227966 tetrahedra (Mach number 0.8, angle of attack 1.0), the speedup is much improved. Due to memory requirements, at least 8 processors are needed to be able to run this case. Figure 14 gives the speedup using the non-blocking communication. On 128 processors, which is the largest number of processors available in one partition on the Cray T3E machine, the speedup is 80, assuming a perfect speedup for 8 processors. Tables 2 and 3 give some detailed statistics of the code on 64 and 128 processors. It is seen that the average number of neighbours per processor on the coarsest mesh is 21 on 64 processors and 26 on 128 processors. These are much smaller than the M6 case (Table 1), where on average there were 38 neighbours on 64 processors.

The fact that smaller meshes tend to give a larger numbers of neighbours on the coarsest mesh can be explained as follows. A root node on the coarsest mesh has many leaf nodes on the finest mesh. The smaller the number of nodes on the fine mesh, the more likely it is that these leaf nodes may be spread over many processors. As a result, on the coarsest mesh, these processors becomes neighbours to each other since they all share the same root node.

This smaller number of neighbours on larger meshes help to control the communication time on the coarsest level. Unlike in the M6 case, where the communication time on the coarsest mesh (level 4) is higher than that on level 3, in both

Tables 2 and 3, the communication time on the coarsest mesh is less than that on other levels. This is more clearly shown in Figure 16, where the communication time on each level of the mesh has been plotted against the number of processors. It can be seen that the communication time on the level 1 mesh reduces as more processors are used. On the coarser meshes, particularly on the 4th level, the communication time stays relatively flat. Nonetheless, the communication time on the coarse meshes is always smaller than that on the finer meshes. Figure 17 shows the overall communication and computation time. It is seen that the computation time reduces proportionally to the number of processors used. The communication over all mesh levels stays relatively flat. On 128 processors, the ratio between the computation time and the communication time is about 1.2. If the size of the mesh increases further, the computation time, which is proportional to the number of sides, should increase faster than the communication time, which is proportional to the length of processor boundary and the number of neighbours. Therefore, for larger meshes, the scalability of the code is expected to be even better.

6 Conclusions

In this paper the approach to parallelise the FLITE3D industrial code has been discussed and numerical results illustrating its performance have been given. The strategy employed in the parallelisation of an agglomeration-based multigrid algorithm has eliminated the need for communication between different levels of meshes. Numerical results have shown that scalability on moderately large meshes is satisfactory, proving that this approach is a viable alternative to that of [9]. The code was delivered to British Aerospace in 1998. It has been used by British Aerospace routinely and has since proved to be robust and scalable on real in-

dustrial problems.

Element-based partitioning has been used in this work because this was the approach taken in our previous successful parallelisation of the single grid FELISA code, a code employing similar technology as FLITE3D. With hindsight, it may have been beneficial to adopt a node-based partitioning instead. The coarsest mesh could then be partitioned first by utilising the edge weight (the number of original edges an edge on the coarsest mesh represents) and node weight (the number of original nodes a node on the coarsest mesh represents). Communication between different levels of mesh would again be eliminated because all leaf nodes reside on the same processor as the root node. Communication time on the coarse meshes may also be under better control because the partitioning would have been carried out on the coarsest mesh. This remains an interesting topic for future work.

Acknowledgments

The authors would like to thank Dr. Robert Allan for reading a draft of this paper. This work was supported by British Aerospace plc under contract number 97/527. The authors would like to thank the Computational Fluid Dynamics Group at Sowerby Research Center, and particularly Phillip Woods, for many helpful discussions.

References

[1] Appa J. FLITE3D Version 4.0 User Guide. British Aerospace Plc. Sowerby Research Centre, Filton, Bristol, UK. August 1997.

[2] Lallemand MH, Steve H, Dervieux A. Unstructured multigridding by volume agglomeration: current status. *Computers Fluids.* 1992; **21**: 397-433.

[3] Venkatakrishnan V, Mavriplis DJ. Agglomeration multigrid for the 3-dimensional euler equations. *AIAA JOURNAL.* 1995; **33**: 633-640.

[4] Mavriplis DJ, Venkatakrishnan V. A 3D agglomeration multigrid solver for the Reynolds-averaged-Navier-Stokes equations on unstructured meshes. *International Journal for Numerical Methods in Fluids.* 1996; **23**: 527-544.

[5] Alef M. Concepts for efficient multigrid implementation on SUPRENUM-like architectures. *Parallel Computing.* 1991; **17**: 1-16

[6] Venkatakrishnan V, Simon HD, Barth TJ. A MIMD implementation of a parallel Euler solver for unstructured grids. *Journal of Supercomputing.* 1992; **6**: 117-137.

[7] Morgan K, Weatherill NP, Hassen O, Manzari MT, Bayne LB, Brookes PJ. Parallel processing for large scale aerospace engineering simulations. In *Parallel Computational Fluid Dynamics: Recent Developments and Advances Using Parallel Computers,* Emerson DR, Ecer A, Periaux J, Satofuka N (eds); Elsevier Science, 1998; pp. 15-22.

[8] Crumpton PI, Giles MB. Multigrid aircraft computations using the OPlus parallel library. In *Parallel Computational Fluid Dynamics: Implementations and Results Using Parallel Computers,* Ecer A, Périaux J, Satofuka N, Taylor S (eds); Elsevier Science, 1996; pp.339-346.

[9] Mavriplis DJ, Pirzadeh S. Large-scale parallel unstructured mesh computation for 3D high-lift analysis. ICASE Report No. 99-9. Institute for Computer Applications in Science and Engineering, 1999.

[10] Schloegel K, Karypis G, Kumar V. Multilevel diffusion schemes for repartitioning of adaptive meshes. *Journal of Parallel and Distributed Computing*. 1997; **47**: 109-124.

[11] Hu YF, Blake RJ. Algorithms for Scheduling with Applications to Parallel Computing. *Advances in Engineering Software*. 1997; **28**: 563-572.

[12] Peiró J, Peraire J, Morgan K. FELISA SYSTEM Reference Manual. Version 1.1. August 1994. Available at *http://abftp.larc.nasa.gov/ftpsite/felisa/*.

[13] Peiró J. A finite element procedure for the solution of the Euler equations using unstructured meshes. PhD Thesis. University of Swansea, 1989.

[14] Roe PL. Approximate Riemann solvers, parameter vectors and difference schemes. *Journal of Computational Physics*. 1981; **43**: 357-372.

[15] Yee HC. A class of high-resolution explicit and implicit shock capturing methods. Lecture Series 1989-04. Von Karman Institute for Fluid Dynamics, 1989.

[16] Jameson A. Artificial diffusion, upwind biasing, limiters and their effect on accuracy and multigrid convergence in transonic and hyperbolic flows. *AIAA* paper 93-3359, 1993.

[17] Peraire J, Peiró J, Morgan K. Multigrid solution of the 3-D compressible Euler equations on unstructured Tetrahedral grids. *International Journal for Numerical Methods in Engineering*. 1993; **36**: 1029-1044.

[18] Peraire J, Peiró J, Morgan K. Finite element multigrid solution of Euler flows past installed aero-engines. *Computational Mechanics*. 1993; **11**: 433-451.

[19] Wesseling P. *An introduction to multigrid methods*. Wiley: Chichester, 1992.

[20] Marvriplis DJ. Multigrid techniques for unstructured meshes. ICASE Report No. 95-27, NASA Langley Research Center. 1995.

[21] Marvriplis DJ. Three-dimensional multigrid for Euler equations. *AIAA Journal*. 1992; **30**: 1753-1761.

[22] Karypis G, Kumar V. Multilevel k-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed Computing*. 1998; **48**: 96-129.

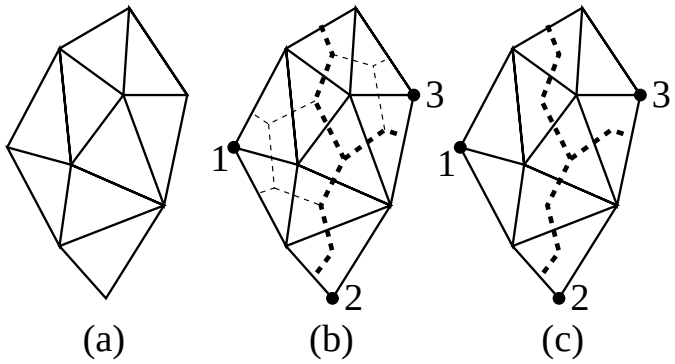

Figure 1: (a) A simple triangular mesh. (b) The control volumes around each nodes. Three seed nodes are chosen. (c) The neighbouring nodes of seed node 1 are agglomerated with it to form the first coarse grid node; the unagglomerated neighbouring nodes of seed node 2 are agglomerated with it to form the second coarse grid node; and finally the unagglomerated neighbouring nodes of seed node 3 are agglomerated with it to form the third coarse grid node.

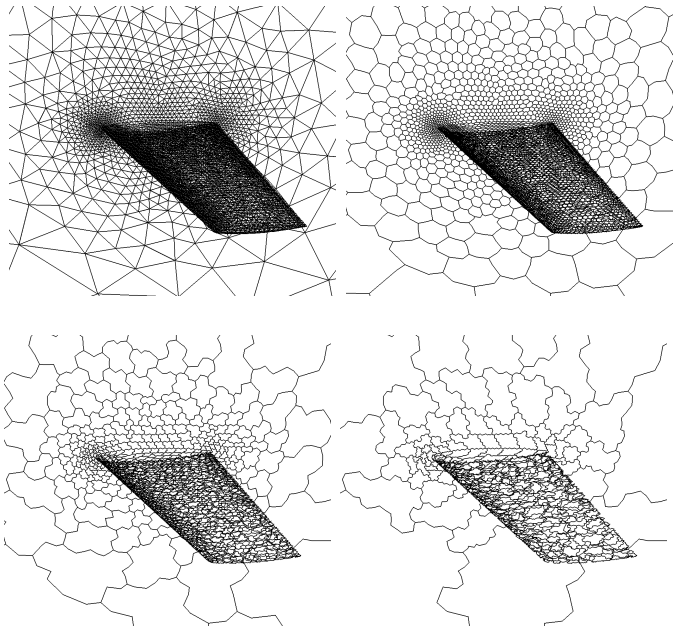

Figure 2: The original mesh (top left), the control volumes (top right), after one level of coarsening (bottom left) and after two levels of coarsening (bottom right).

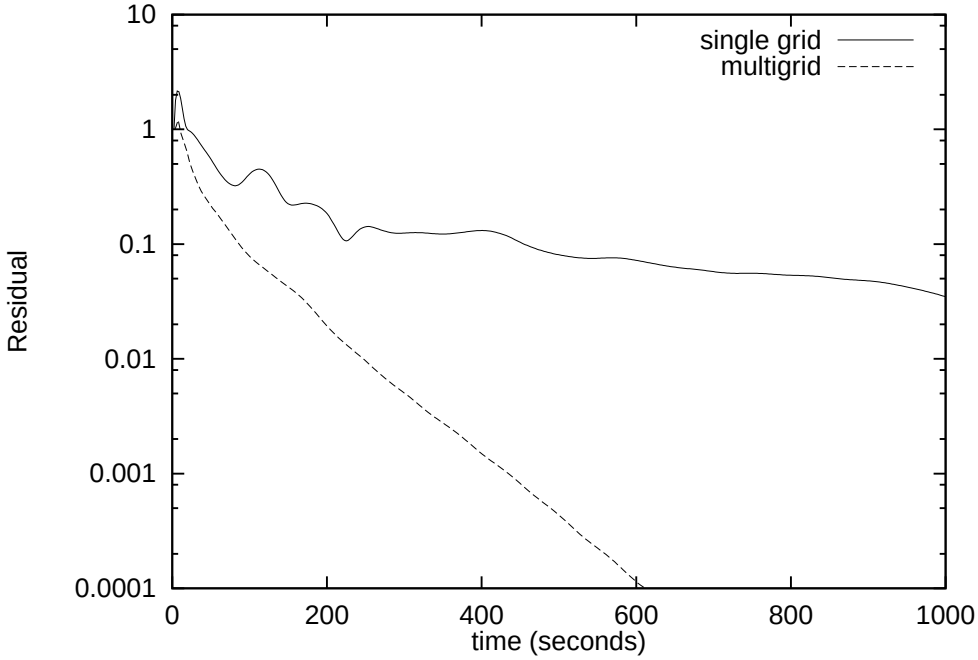

Figure 3: The convergence of multigrid (with 4 levels) and single grid algorithms on a mesh around M6 wing with 29784 nodes, 208880 sides and 173176 tetrahedra, at 3.06^o angle of attack and Mach number 0.84.

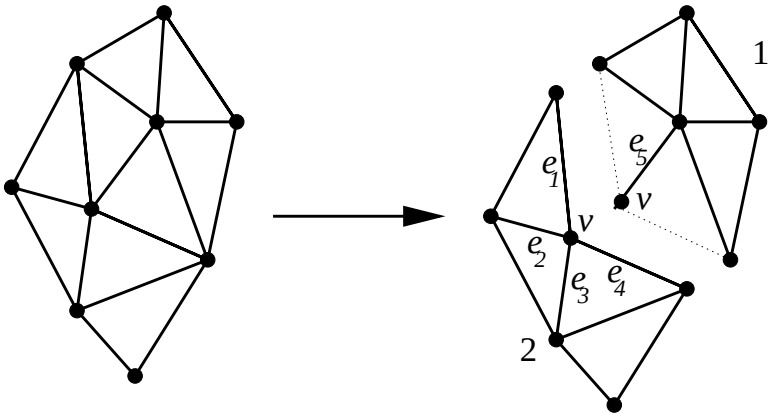

Figure 4: An illustration of how a mesh is partitioned.

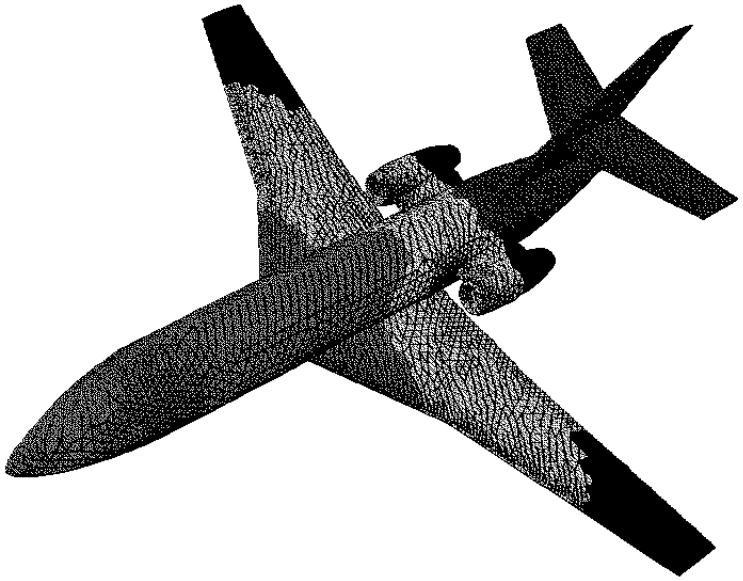

Figure 5: A mesh around FALCON aircraft partitioned into 4 subdomains (only surface mesh on the body of the aircraft shown).

- Initialise $\boldsymbol{R}$ to zero
- Primary: Do $s = 1, N_{\text{sides}}$
  - Add the contribution relating to side s to the $\boldsymbol{R}$ terms of the two end nodes.
- Enddo Primary
- BoundaryCondition: Do $\Delta = 1, N_{\text{faces}}$
  - Add the contribution relating to surface triangle Δ to the $\boldsymbol{R}$ terms of the three nodes of the triangle.
- Enddo BoundaryCondition

Figure 6: Pseudo-code for two typical sequential FLITE3D-FS loops.

- Initialise $\boldsymbol{R}$ to zero
- Primary: Do $s = 1, N_{\text{sides\_on\_p}}$
  - Add the contribution relating to side s to the $\boldsymbol{R}$ terms of the two end nodes.
- Enddo Primary
- BoundaryCondition: Do $\Delta = 1, N_{\text{faces\_on\_p}}$
  - Add the contribution relating to surface triangle Δ to the $\boldsymbol{R}$ terms of the three nodes of the triangle.
- Enddo BoundaryCondition
- ExSum: Exchange and sum the $\boldsymbol{R}$ terms for all the processor boundary nodes.

Figure 7: Pseudo-code for the parallel FLITE3D-FS loops on a processor p, corresponding to Figure 6.

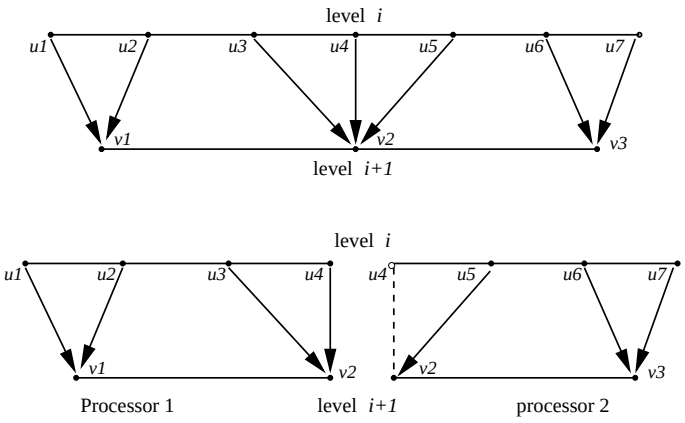

Figure 8: An illustration of the sequential (top) and parallel (bottom) restriction phase.

- Initialise $\boldsymbol{U}_c$ to zero.
- Restriction: Do $v = 1, N_{CoarseNodes}$
  - $\boldsymbol{U}_c(v) = \boldsymbol{U}_c(v) + w_f(u)\boldsymbol{U}_f(u)$ for each leaf u of v, where $w_f(u)$ is the weight associated with leaf u.
- Enddo Restriction

Figure 9: Pseudo-code for the sequential restriction operation.

- Initialise $\boldsymbol{U}_c$ to zero on processor p.
- Restriction: Do $v = 1, N_{CoarseNodes\_on\_p}$
  - $\boldsymbol{U}_c(v) = \boldsymbol{U}_c(v) + w_f(u)\boldsymbol{U}_f(u)$ for each *active* and *local* leaf u of v, where $w_f(u)$ is the weight associated with leaf u.
- Enddo Restriction
- ExSum: Exchange and sum the $\boldsymbol{U}_c$ term for all the processor boundary nodes on the coarse mesh.

Figure 10: Pseudo-code for the parallel restriction operation on a processor p.

Parallel Prolongation

- Initialise $\boldsymbol{U}_f$ to zero
- Prolongation: Do $v = 1, N_{CoarseNodes\_on\_p}$
  - $\boldsymbol{U}_f(u) = \boldsymbol{U}_f(u) + \Delta\boldsymbol{U}_c(v)$ for each *local* leaf u of coarse node v. Here $\Delta\boldsymbol{U}_c(v)$ is the correction to the flow field variable at coarse node v.
- Enddo Prolongation

Figure 11: Pseudo-code for the parallel prolongation operation on a processor p.

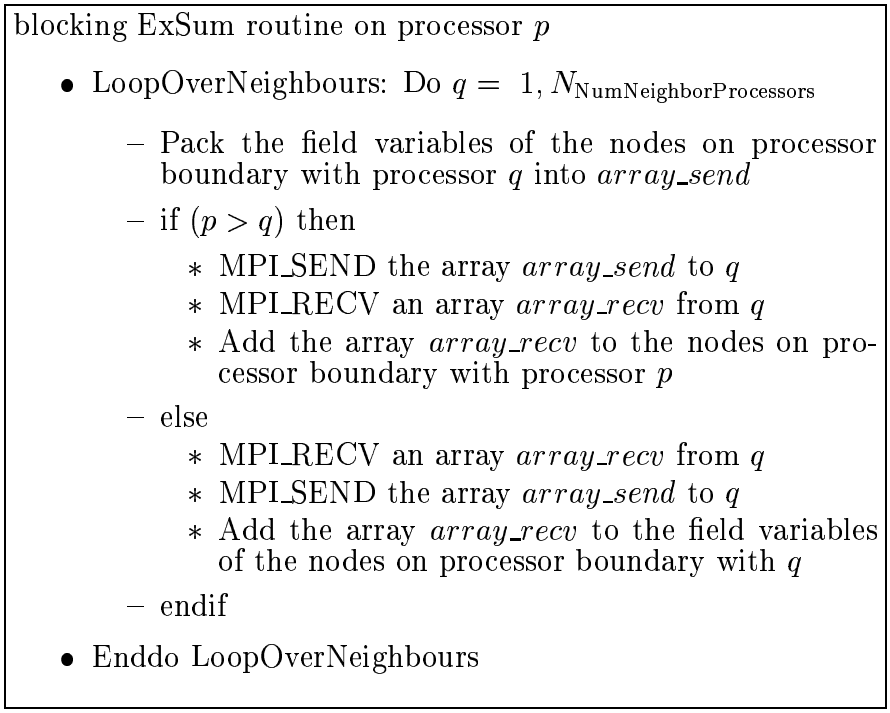

Figure 12: The implementation of the ExSum using blocking MPI calls.

non-blocking ExSum routine on processor p

- LoopOverNeighbours: Do $q = 1, N_{\text{NumNeighborProcessors}}$
  - Pack the field variables of the nodes on processor boundary with neighbour processor q into $array\_send\_q$
  - MPI_IRECV an array $array\_recv\_q$ from q
  - MPI_ISEND the array $array\_send\_q$ to q
- Enddo LoopOverNeighbours
- WaitEachNeighbour: Do $q = 1, N_{\text{NumNeighborProcessors}}$
  - MPI_WAIT for the MPI_IRECV from neighbour q to complete
  - Add the array $array\_recv\_q$ to the nodes on processor boundary with neighbour processor q
  - MPI_WAIT for the MPI_ISEND to neighbour q to complete
- Enddo WaitEachNeighbour

Figure 13: The implementation of the ExSum using non-blocking MPI calls.

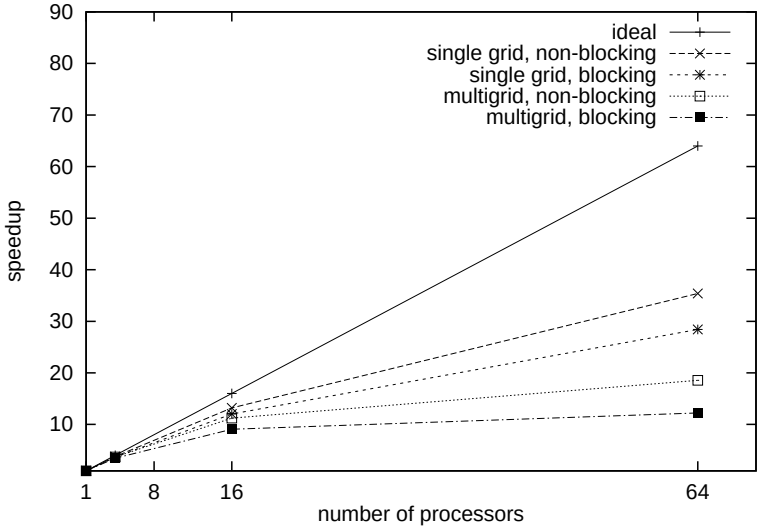

Figure 14: The speedup of the FLITE3D-FS on a mesh around M6 wing with 29215 nodes. Running on a Cray T3E-900.

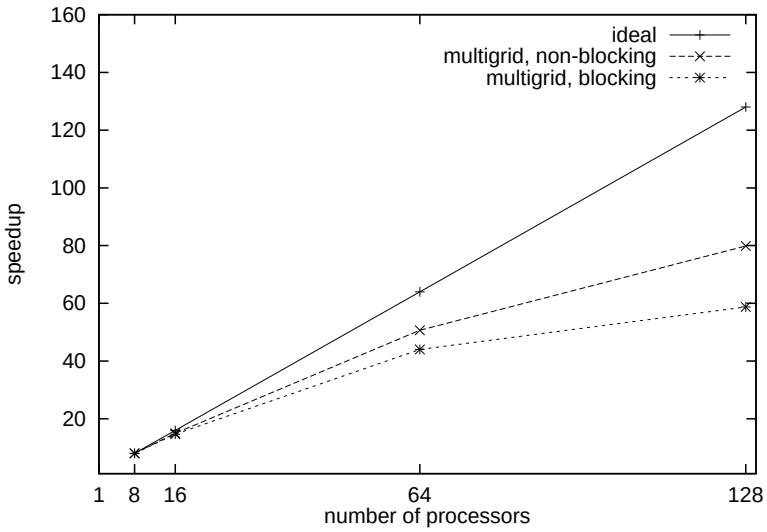

Figure 15: The speedup of the FLITE3D-FS on a mesh around F18 fighter aircraft with 518519 nodes. Running on a Cray T3E-900.

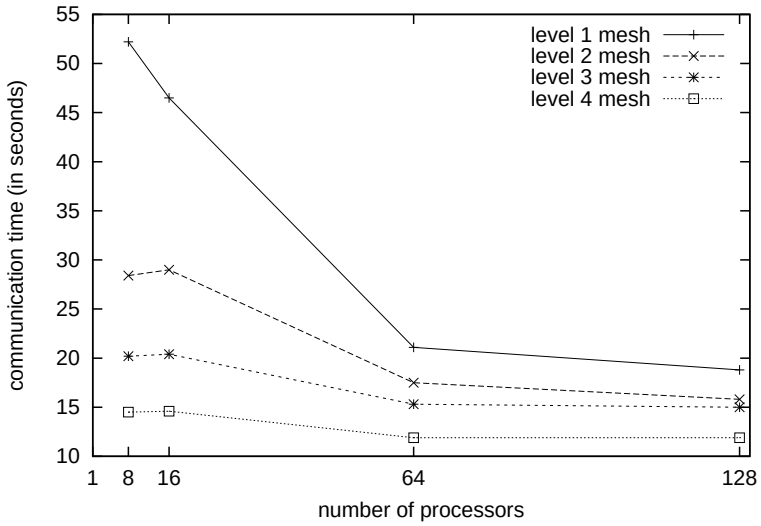

Figure 16: The communication time on each level of mesh. Mesh around F18 fighter aircraft with 518519 nodes. Running on a Cray T3E-900.

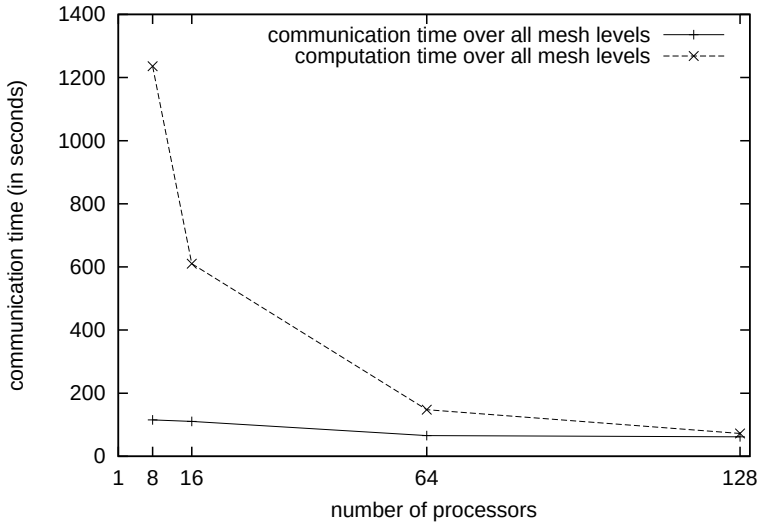

Figure 17: The overall communication time and the computation time. Mesh around F18 fighter aircraft with 518519 nodes. Running on a Cray T3E-900.

Table 1: Detailed breakdown statistics on 64 processors. M6 mesh, runs to convergence after 170 iterations (tolerance 10^{-4}).

mesh level	1	2	3	4
total no. sides	208880	36877	6593	1199
average no. sides	3264	576	103	19
comp. time	12.2	2.5	1.1	0.7
average no. neighbours	11	13	20	38
total length of messages	27254	14202	8824	8482
comm. time (blocking)	20.8	17.5	16.0	14.8
comm. time (non-blocking)	9.7	9.1	9.6	10.9

Table 2: Detailed breakdown statistics on 64 processors. F18 mesh, 100 iterations.

mesh level	1	2	3	4
total no. sides	3786836	630692	104863	17642
average no. sides	59169	9855	1638	276
comp. time	122.4	19.2	4.6	2.1
average no. neighbours	11	12	14	21
total length of messages	154898	66258	28512	14828
comm. time (blocking)	45.0	25.9	19.6	16.4
comm. time (non-blocking)	21.1	17.5	15.3	11.9

Table 3: Detailed breakdown statistics on 128 processors. F18 mesh, 100 iterations.

mesh level	1	2	3	4
total no. sides	3786836	630692	104863	17642
average no. sides	29585	4927	819	138
comp. time	59.8	9.5	2.5	1.1
average no. neighbours	12	13	16	26
total length of messages	219636	96294	44172	25846
comm. time (blocking)	50.6	30.8	21.1	16.2
comm. time (non-blocking)	18.8	15.8	15.0	11.9