INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN FLUIDS, VOL. 14, 175–187 (1992)

A FIRE FIELD MODEL IMPLEMENTED IN A PARALLEL COMPUTING ENVIRONMENT

C. S. IEROTHEOU AND E. R. GALEA

Centre for Numerical Modelling and Process Analysis, Thames Polytechnic, London SE18 6PF, U.K.

SUMMARY

This paper describes the implementation of a fire field model in the parallel computing environment offered by multiple transputers. The fire model is built into the general purpose SIMPLE-based CFD code HARWELL-FLOW3D. The technique of domain decomposition has been applied to convert the conventional serial version of FLOW3D into a code capable of efficiently utilizing an arbitrary number of transputers. Fire simulations consisting of up to 24 000 computational cells are performed on parallel systems with up to 15 processors. The run time for this simulation has been reduced from over 4 days on a single processor to just over 8 h on the 15-processor system. An interactive graphics system has also been developed which runs in parallel with the main computations.

KEY WORDS Parallel computing Transputer CFD Fire simulation

1. INTRODUCTION

The mathematical simulation of fire is an extremely demanding application of computational fluid dynamics (CFD). In its simplest form the fire field model[1] involves the simulation of transient, three-dimensional, turbulent, buoyant flows. These simulations have the capability to predict the spread of fire hazards such as heat and smoke within an enclosure subjected to fire. Successful applications of this technology include aircraft cabin,[2] sports stadium,[3] hospital ward,[4] airport terminal[5] and underground station[6] fires.

These models potentially have great utility in assessing the design of inhabited enclosures for safety and in the training of fire-fighting personnel. However, the single most important factor which mitigates against the widespread exploitation of this technology is the CPU and associated elapsed times required to simulate a specific scenario. For example, the simulation of one typical scenario involving a small 57 kW fire burning in a Boeing 737 fuselage currently requires about 64 h on a VAX-like minicomputer with a 0·6 Mflops processor.[7] This case contained a finite volume grid consisting of 21 000 cells and suggests that in excess of 50 000 cells are required to adequately represent 'filled' commercial aircraft bodies.

If physical phenomena such as combustion, radiation, the generation of toxic species and the action of water sprays are to be included, then many hundreds of hours of elapsed time will be expended by each simulation. For example, a two-phase transient fire–sprinkler simulation covering the first 120 s of fire–sprinkler interaction in a simple rectangular room configuration consisting of 2700 computational cells required 236 h.[8]

0271–2091/92/020175–13$06.50

Received December 1990
Revised April 1991

In an attempt to reduce these excessive elapsed times, the computational community first employed vector-processing techniques. The majority of this effort was based on CRAY and CDC supercomputer hardware. Typically, speed-up factors between three and four have been reported for CFD applications.[9–11] These superficially low speed-ups are a result of only a fraction of the total code being efficiently vectorized vis-à-vis the linear equation solver.[12] In an attempt to gauge the potential of vectorization, Ierotheou *et al.*[13] vectorized as much of a purpose-written CFD code as possible. With over 90% of the code vectorized, speed-up factors range from five to 30;[14] these rate amongst the best results quoted in the open literature. Clearly, even in such an ideal situation the speed-up potential from vectorization is limited.

Parallel computing techniques applied to multiprocessor architectures now offer the potential of reducing these considerable elapsed times to a few hours. While it is possible to solve computationally intensive problems using supercomputers, this approach is limited and perceived to be prohibitively expensive for the majority of routine industrial applications.

The Inmos transputer[15] offers an efficient and relatively inexpensive route to achieving this goal. The transputer is a 32 bit RISC microprocessor with 4 Kb of on-chip memory and can access from 1 Mb to 2 Gb of off-chip memory. The T800 series transputer is available with 20 or 25 MHz clock rates. Each transputer has four 20 Mbits s^{-1} communication links which enable it to be linked with up to four other transputers. Using the parallel FORTRAN compiler of 3L Ltd.,[16] the 20 Mhz T800 has a realizable performance of 0·4 Mflops. It is possible to accommodate within a personal computer tower case up to 15 transputers with 56 Mb of memory.

In this paper we discuss the mapping of the general-purpose commercial fluid flow code HARWELL-FLOW3D[17] onto a distributed memory multitransputer architecture and its application to transient, three-dimensional fire simulations. The mapping technique adopted here is that of domain decomposition. While the technique has been applied to the FLOW3D code, it is general and can be equally applied to any other structured control-volume-based code. Furthermore, the technique is not restricted to the transputer and can be applied to any similar parallel computing platform.

Cross *et al.*[18] explored the extent to which the technique could be applied to control-volume-based, three-dimensional, transient solidification-by-conduction problems. Recent applications of this work[19] reveal that efficiencies of 95% can be achieved, resulting in a 17-fold speed-up on an 18-transputer system.

Johnson and Cross[20] later applied the technique to the HARWELL-FLOW3D code. They examined the effectiveness of the mapping on steady state, incompressible, laminar flows. A number of tests were performed, ranging from problems consisting of 4000 to 40 000 computational cells and from one to 50 transputers. They reported efficiencies from 60% to 85%. The general trends suggest that larger problems are more efficient with respect to speed-up than smaller ones. In an attempt to maximize efficiencies, various combinations of solvers were examined. Their findings were highly problem-dependent.

Here the FLOW3D code is applied to a difficult practical problem and the domain decomposition technique is extended to include transient, compressible, buoyant and turbulent flows.

In addition to enhanced computational performance, the multiprocessor approach allows greater flexibility in the design of the CFD environment. Since all the computations are performed on the transputers, this leaves the host processor free to be engaged in other tasks. This has been exploited by utilizing the host processor to produce interactive graphics without interfering with the progress of computations. This allows the user to keep a detailed check on the computations and, if necessary, alter the convergence path.

2. HARWELL-FLOW3D—AN OVERVIEW

The HARWELL-FLOW3D programme was developed by AEA Technology in the mid-1980s. The software was written to model complex three-dimensional fluid flow problems and uses the control volume approach to discretize the domain. All velocity components are stored at the control volume centres using the non-staggered method of Rhie and Chow.[21] The means by which the resulting coupled equations are solved is based on the SIMPLE solution procedure and its derivatives.[22,23]

Traditionally, CFD problems are compute-intensive and therefore FLOW3D was developed on a CRAY supercomputer to exploit the vector-processing features present.

The code can tackle problems consisting of one-, two- or three-dimensional geometries with either regular or body-fitted grids. The simulations can be either steady-state or transient, involve laminar or turbulent flows (k–ε model) and can also include the solution of scalars, combustion and radiation. There is also a selection of linear equation algorithms to choose from, such as preconditioned conjugate gradient (PCG) methods, a line relaxation algorithm (LSOR) and Stone's strongly implicit procedure (SIP).

FLOW3D was written using FORTRAN 77. The multiple-transputer version is also a totally FORTRAN code. The FORTRAN used was authored by 3L Ltd. and is a FORTRAN 77 standard with extensions for communications.

3. THE HARDWARE

The host hardware used in this work consisted of a 386/387 AT with VGA graphics and monitor. Located within the AT case were two motherboards populated with a total of 15 transputer modules (TRAMS). Each TRAM consists of a T800-20 transputer with either 2 or 8 Mb of off-chip memory.

A single TRAM had 8 MB of off-chip memory and this will be referred to as the *master* processing element (PE). The remaining 14 TRAMS had only 2 MB of off-chip memory and will be referred to as the *slave* PEs. The PEs were arranged in a pipeline with the master at the head as shown in Figure 1.

4. THE PARALLEL IMPLEMENTATION

In deciding on a suitable local memory parallelization strategy, many factors need to be taken into consideration.

One important factor concerns the efficient exploitation of the available raw compute power. Experience within the CFD community on the use of vector processors for compute-intensive simulations has demonstrated that although high efficiencies are possible, they are not easily attainable. For multiple-processor systems this issue is equally relevant.

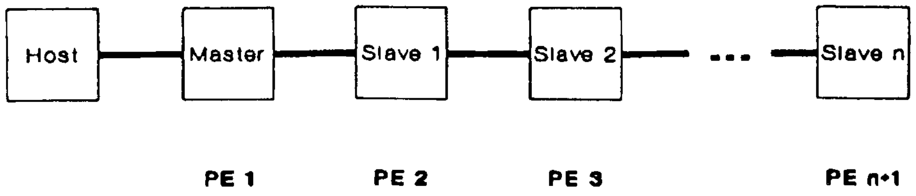

Figure 1. Simple pipeline configuration of PEs showing nearest-neighbour connectivity

If the use of multiple-processor systems is to be successful, then it is essential to keep overheads to a minimum. These can involve data communication between processors or the necessity to perform extra computations. In addition, full use must be made of the PE's local memory so that there is as high a computation-to-communication ratio as possible.

Finally, in order to maintain the original code flexibility, the multiple-processor version must be general enough to permit execution on an arbitrary number of PEs. Furthermore, with the exception of specifying how many PEs are to be used, the appearance of the serial and parallel implementations must be identical.

Structured control-volume-based CFD codes exhibit natural data parallelism. This is exploited here by employing a strategy based on the systematic partitioning of the computational domain. In the control volume approach the information required for a given cell is based solely on the neighbouring cells; thus there are three natural partitioning strategies.

The first is based on blocks of cells, where the domain is subdivided into blocks of $n \times m$ cells and each block is assigned to a corresponding PE. This approach does not allow for nearest-neighbour communications, since the TRAM has only four links and is therefore inefficient. The second is based on lines of cells where each line covers a complete dimension. Nearest-neighbour communications are possible if the PEs are set up in a two-dimensional grid topology. The final approach is based on a slab of cells. For this decomposition, nearest-neighbour communications can be minimized when mapped onto a pipeline topology (Figure 2). While the two-dimensional grid topology has a smaller communications overhead, the slab-partitioning approach requires less restructuring of the code. From the point of view of maintaining load balancing, the two-dimensional grid topology is more restrictive, since load balancing must now be performed in two dimensions. Also, as more PEs are added to the grid, it may become necessary to alter the entire distribution of PEs, thus altering the efficiency characteristics. Therefore, as a first step, the slab-partitioning approach was used in this study. A slab is defined as covering the entire xy-plane for a particular z-position. The data transfers between PEs consist of all xy-slab values represented at the PE overlap areas.

The success of this approach depends on the even balancing of the computational effort amongst the PEs. Inefficiencies will result if some of the PEs remain idle during part of the

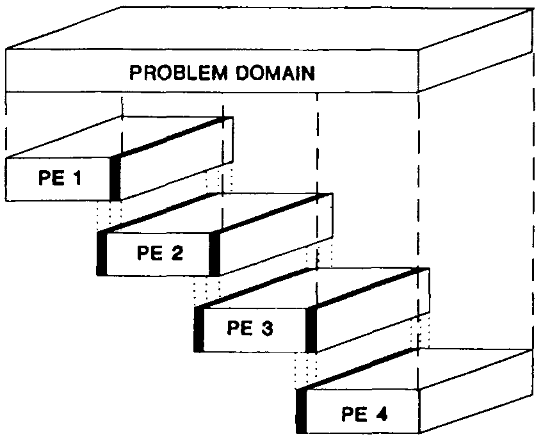

Figure 2. Partioning of problem domain over four PEs (shaded areas showing overlapped region)

computation phase. Since the computation time per cell is roughly constant, PEs will be left idle if the numbers of cells per PE are not identical. In some cases these idle times can be excessive.

In the pipeline topology each PE has its own copy of the FLOW3D executable. Each PE is responsible for the computation of cell information in its assigned area of the domain. When necessary, data are transferred between neighbouring PEs. This is done in order to preserve as far as possible the data dependences present in the original scalar code. The major differences between the FLOW3D executables on the PEs lie in array declarations (which include the overlapped regions) and 'DO loop' ranges which are set to cover the nodes assigned to a given PE.

The FLOW3D code has a variety of different linear equation solvers used to solve the discretized equations. These need to be rewritten to exploit the current parallel architecture. Attention is focused on three solvers: the PCG methods, LSOR and SIP.

The PCG methods are largely explicit in nature; as a result there is minimal data transfer between PEs. The SIP algorithm is an example of an implicit solver and involves recurrences in the forward elimination and back-substitution stages. By re-ordering the indices over which the cells are updated, the algorithm can be made to perform in a pipeline fashion. This introduces an overhead in the form of a start-up time. The SIP and PCG algorithms have similar convergence properties in both their serial and parallel implementations. The LSOR algorithm uses the latest cell approximations and can be made to perform in a similar way in parallel. Unfortunately, large idle times are introduced while the necessary data values are computed on other PEs. An alternative is to use a 'local' PE LSOR algorithm. The LSOR algorithm uses updated values as they become available from assigned cells but uses old approximations for the overalapped cells. The latter effect causes an increase in the total number of iterations needed to achieve the same level of convergence.

It was also necessary to introduce the evaluation of global scalar numbers. An example is the maximum residual required during a solution process. This was determined in two stages. Firstly, local evaluation of the scalar was carried out and all such numbers were then transferred up the pipeline to the master PE. In the second stage the scalar is determined for the whole data set and, if necessary, transferred back down the pipeline to all the slave PEs.

Other alterations to the original code are mainly in the form of data exchanges of overlapped areas between neighbouring PEs. The exchanges are performed whenever there is an update of the overlapped areas.

In this paper we discuss the performance of this code on up to 15 PEs. However, the code has been developed for operation with an arbitrary number of PEs.

Since all computations are performed on the PEs, the host processor remains idle for a large portion of the total time. It can be put to good use by performing other tasks which will aid the CFD modeller. In the current implementation, for example, an interactive graphical display has been included. This is updated in parallel with the transputer calculations of the simulation, incurring only a small overhead in the process.

The graphical tasks are not intended to be used for post-processing, but rather as a monitoring tool of the simulation. They allow a detailed interrogation of the flow field together with other related parameters. Some of the features currently provided in the 386 version include contour plots of a selected variable, e.g. pressure, temperature, etc., and vector plots of a given two-dimensional plane. For the purposes of monitoring convergence behaviour, there are graphs of mass residual and monitoring variables of a given control volume. In addition, relaxation parameters can be changed to modify the convergence behaviour. The direction of view and the selected two-dimensional plane can also be changed during the simulation. It is intended that this facility will be extended to include interactive modifications to boundary conditions and internally defined regions such as heat sources.

5. THE PHYSICAL PROBLEM

The test case considered here is an enclosure fire. The fire compartment represents a closed office of dimensions $2{\cdot}97 \times 2{\cdot}97 \times 6{\cdot}00\ \text{m}^3$. Following accepted practices for the modelling of non-spreading fires,[1-8] the fire—meant to simulate a small heater or the early stages of a waste paper fire—was modelled as a volumetric heat source. It was centrally located on the floor at the back wall. For the purposes of this test case combustion was ignored. The heat source had dimensions of $0{\cdot}54 \times 0{\cdot}27 \times 0{\cdot}4\ \text{m}^3$ and a constant power output of 6 kW. The simulation was concerned with predicting the evolution of the office environment over the first 60 s.

This simulation is not intended to represent state-of-the-art fire field modelling. Rather, it demonstrates the advantages in using a multitransputer environment to perform the calculations necessary to simulate complex fire scenarios.

6. THE MATHEMATICAL PROBLEM

The starting point of the analysis is the set of three-dimensional, partial differential equations that govern the phenomena of interest here. This set consists in general of the following equations: the continuity equation; the three momentum equations that govern the conservation of momentum per unit mass in each of the three space directions (the Navier–Stokes equations); the equation for conservation of energy; and the equations for a turbulence model, in this case the k–ε model. The precise formulations of the differential equations describing the model will not be presented here since they may be found elsewhere;[24,25] however, we shall consider them in their general vector form.

All the equations can be expressed in a general form[26] as follows.

The continuity equation

The conservation of mass is expressed as

$$\frac{\partial \rho}{\partial t} + \operatorname{div}(\rho V) = S. \tag{1}$$

The general Φ-equation

The general source balance equation for Φ is

$$\underbrace{\frac{\partial}{\partial t}(\rho\Phi)}_{\text{transient}} + \underbrace{\operatorname{div}(\rho V\Phi}_{\text{convection}} - \underbrace{\Gamma_\Phi \operatorname{grad}\Phi)}_{\text{diffusion}} = \underbrace{S_\Phi}_{\text{source}}. \tag{2}$$

Equations for the various quantities (e.g. velocity components, energy, pressure, etc.) differ primarily in the way in which the terms Γ and S_Φ are connected with other variables. They are derived from equation (2) simply by replacing Φ, Γ and S_Φ with the appropriate expressions.

The boundary conditions

The initial temperature within the room was set to 24 °C. For all walls of the compartment the no-slip condition was used for velocities and both isothermal (at 24 °C) and adiabatic conditions for temperature. The usual 'wall functions'[27] were used to compute shear stresses and heat fluxes at the walls.

7. THE NUMERICAL EXPERIMENTS

The volume of the fire enclosure was discretized using a Cartesian framework. For convenience the grid spacing was uniform in each space direction. A series of grids were employed ranging from $10 \times 10 \times 15$ (1500 cells) through $10 \times 20 \times 30$, $20 \times 10 \times 30$ and $20 \times 20 \times 15$ (6000 cells) up to $20 \times 20 \times 30$ (12 000 cells). These grids were selected for convenience, since this allowed the model to be implemented comfortably on five TRAMS. A further grid consisting of $20 \times 20 \times 60$ (24 000) cells was also used; the execution of this problem was restricted to 15 TRAMS.

The number of sweeps used within a given time step was set to a maximum of 25. Tests revealed that setting this number to 50 had very little effect on the overall solution. Convergence is assumed within a given time step if either the maximum number of sweeps is reached or the mass source residual falls below 1×10^{-4}. It is worth noting here that converged solutions obtained from the serial and parallel codes display no significant differences. The hybrid differencing scheme is used throughout and the Stone method is used to solve the momentum and enthalpy equations. The pressure correction equation was solved using the ICCG method and the turbulence quantities were solved using the LSOR method. This corresponds to the default selection of solvers as recommended by the FLOW3D developers. Experience has shown these solvers to be efficient and robust for most applications. They were not selected on the basis of demonstrating peak efficiencies in parallel. The fully implicit backward differencing scheme was used for the discretization of time. Finally, the simulation was run for a total of 60 time steps, where each time step represented 1 s.

8. RESULTS AND DISCUSSION

Before discussing the findings of the numerical experiments, it is appropriate to briefly discuss the evolution of the room environment as predicted by the numerical model. The heat source creates a buoyancy-driven flow with large-scale turbulent motion which controls the diffusion of momentum. The non-uniform buoyancy forces not only drive this flow but also increase the turbulent mixing in the rising plume and inhibit it in hot stratified layers.

Within 5 s the rising plume above the heat source impinges on the ceiling, creating a jet of hot gases which travels along the length of the room. This results in vigorous turbulent and a thickening of the hot layer. In order to feed the rising thermal plume, the heat source entrains cooler ambient air at floor level. As a result a large recirculation current is set up within the room. Within 25 s the ceiling jet encounters the far wall. As the simulation progresses, the hot ceiling layer thickens and there is a further increase in temperature. After 60 s a multilayered thermal stratification has developed within the room (Figure 3).

Results from the numerical experiments concerning run times, speed-ups and efficiencies are summarized in Table I. Run times refer to the elapsed wall clock time required to perform the simulation calculations. Initial set-up and screen I/O times, while small in relation to the time involved in performing the calculations, are not included in these figures. In all but the 24 000-cell case, quoted speed-ups refer to the ratio of run times on a single PE to the respective multiple-PE case. For the purposes of comparing the serial and parallel FLOW3D implementations, the serial times quoted refer to the performance of the solvers in their original form. The simulation involving 24 000 cells could not be performed on a single PE because of memory limitations. In this case the run time was estimated to be simply twice the time required to perform the 12 000-cell simulation on a single PE. In this way the quoted speed-up represents a minimum expected value. The efficiency is simply a percentage expression which represents the speed-up ratio divided by the number of PEs implemented.

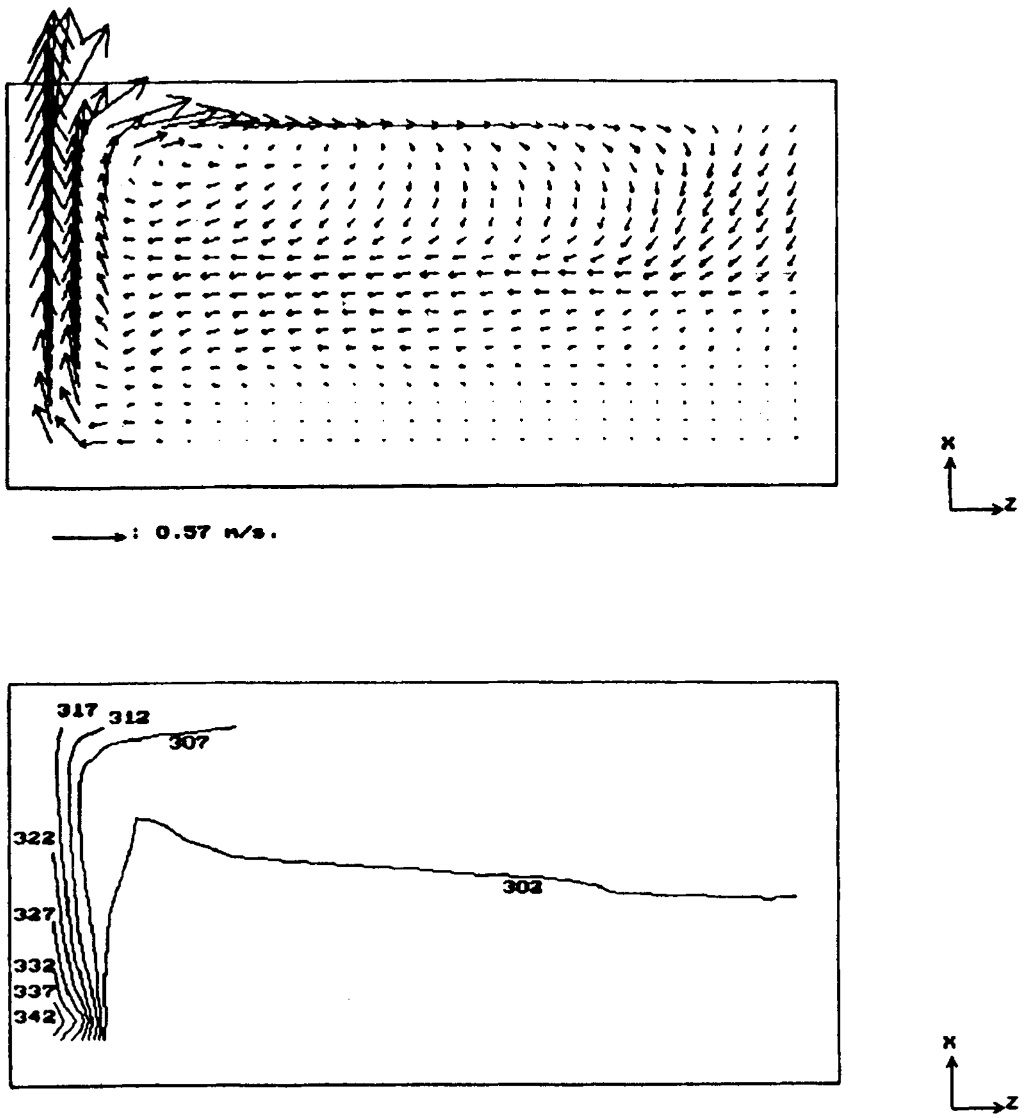

Figure 3. Vertical centre plane depicting velocity vectors and temperature (K) contours after 60 s

Figures 4 and 5 depict speed-up and wall-clock times for the 12 000-cell (20 x 20 x 30 geometry) simulation utilizing up to 15 PEs. Also indicated in Figure 4 (dotted line) is the ideal linear speed-up curve. It is clear from this figure that as the number of PEs involved in the solution procedure increases while the number of cells remains constant, the efficiency of the system deteriorates. This is primarily due to a reduction in the proportion of computation to communication times. As more PEs are utilized, the time spent by each in performing the calculations is reduced while the time spent on communications remains constant. In addition, the parallel LSOR algorithm incurs a penalty in the form of an increase in the total number of iterations needed to achieve convergence. This is likely to become significant as additional PEs are included.

Table I. Results of fire simulation numerical experiments performed on multiple transputers

Grid size	Number of PEs	Wall clock time (h)	Speed-up	Efficiency (%)
10 x 10 x 15 (1500)	1	4·13	—	—
10 x 10 x 15 (1500)	5	1·34	3·1	62
10 x 20 x 30 (6000)	1	18·94	—	—
10 x 20 x 30 (6000)	5	5·64	3·36	67
20 x 10 x 30 (6000)	1	18·76	—	—
20 x 10 x 30 (6000)	5	5·44	3·45	69
20 x 20 x 15 (6000)	1	19·32	—	—
20 x 20 x 15 (6000)	5	6.06	3·19	64
20 x 20 x 30 (12000)	1	49·22	—	—
20 x 20 x 30 (12000)	5	11·61	4·24	85
20 x 20 x 30 (12000)	10	6·22	7·91	79
20 x 20 x 30 (12000)	15	4·42	11·14	74
20 x 20 x 60 (24000)	1	98·43*	—	—
20 x 20 x 60 (24000)	15	8·31	11·85*	79*

*Indicates minimum estimate.

Despite these difficulties, speed-up factors range from 4·3 on five PEs (85% efficiency) to 11·1 on 15 PEs (74% efficiency). In terms of run times, this means that the 12 000-node fire simulation which requires in excess of 49 h to complete on a single PE can be performed in under 4·5 h using 15 PEs (see Figure 5).

The efficiency obtained from a particular PE array is dependent on the number of computational cells involved in the simulation. Figure 6 shows that a five-PE system can deliver efficiencies varying from 62% (3·1 speed-up) through 69% (3·45 speed-up) up to 85% (4·3 speed-up) by changing the problem size from 1500 through 6000 to 12 000 cells respectively. On doubling the number of cells to 24 000 (20 x 20 x 60), the efficiency of the 15-PE system increases from 74% to an estimated minimum of 79%. Therefore, to achieve the maximum practical efficiency, it is essential to solve the largest problem that can be accommodated within the PE's memory.

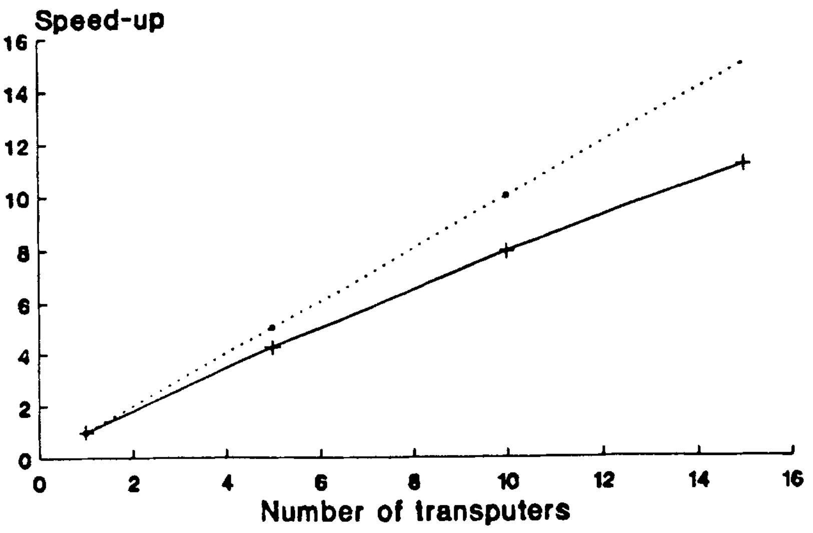

Figure 4. Speed-up factors for 20 × 20 × 30 problem utilizing up to 15 transputers (ideal speed-up shown as dotted line)

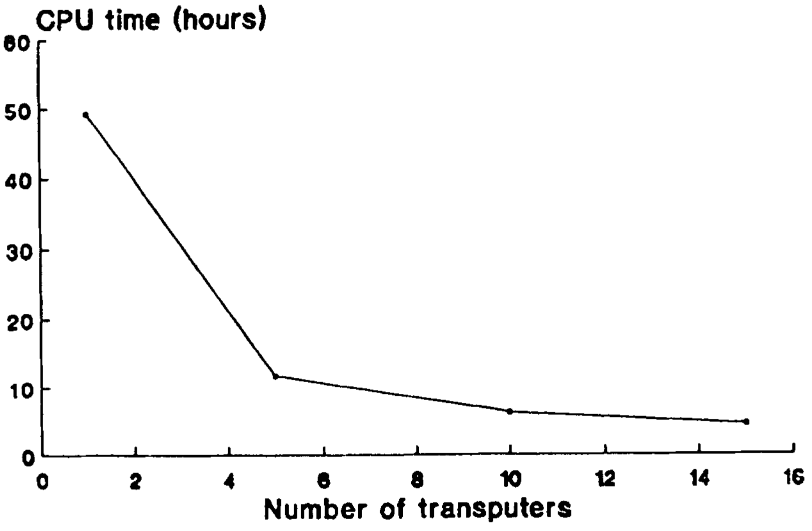

Figure 5. Wall clock times for 20 × 20 × 30 grid utilizing up to 15 transputers

These results suggest that while keeping the problem size fixed, a point will eventually be reached where no further gain may be expected by adding additional PEs. However, the onset of this cut-off point can be forestalled by increasing the problem size.

It is also apparent that for a given problem size the efficiency is dependent on the manner in which the cells are distributed within the solution domain. This is illustrated by the solution of the 6000-cell problem using five PEs. Three cell distributions were considered, 10 × 20 × 30, 20 × 10 × 30 and 20 × 20 × 15, resulting in efficiencies of 67%, 69% and 64% respectively.

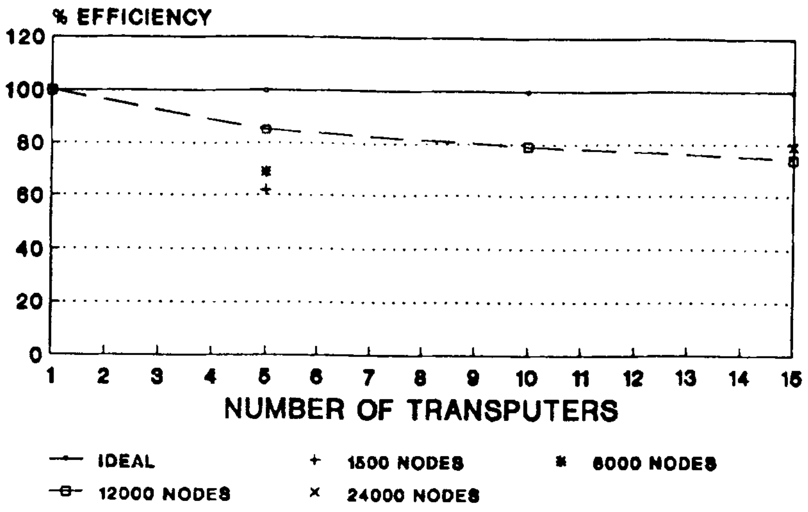

Figure 6. Efficiency produced with a variety of different grids

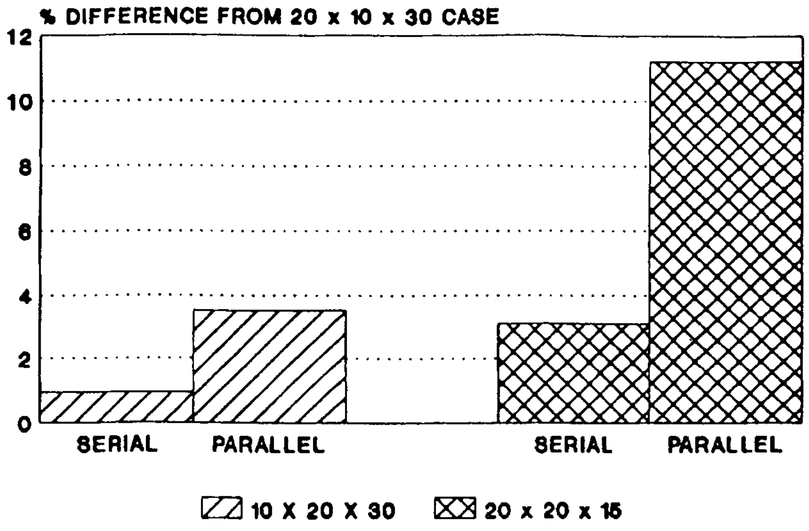

Figure 7. Differences in CPU time between various distributions of 6000 cells

Table I reveals that the 20 × 10 × 30 configuration proved to be the fastest in both serial and parallel cases. The 20 × 20 × 15 case is the most inefficient, with the parallel and serial implementations running 11·2% and 3·1% slower than their respective fastest counterparts (Figure 7).

However, there was little difference between the 20 × 10 × 30 and the 10 × 20 × 30 cases, less than 1% in scalar and 3·5% in parallel (Figure 7). The relatively small difference between these parallel cases is expected, because the primary overhead in each case—that of data transfers between PEs—is identical, since the number of cells in the *xy*-slabs are the same. This also means that the same number of calculations per *xy*-slab are performed, resulting in similar computation times for both parallel and scalar comparisons.

The marked differences observed between the $20 \times 10 \times 30$ and $20 \times 20 \times 15$ configurations can be explained in a similar manner. For the parallel case the data transfers between PEs are not identical. The first configuration involves 20×10 data elements while the second involves 20×20. Hence there is a greater communication overhead associated with the $20 \times 20 \times 15$ case. This is coupled with the fact that algorithms which solve in an *xy*-slab fashion will work more effectively on a 20×10 rather than a 20×20 slab.

9. FUTURE DEVELOPMENTS

The development of the transputer (and parallel FORTRAN) has released the power of inexpensive parallel computing to computationally demanding applications such as fire simulations. Ironically, it is also the transputer which is limiting its further development. Delivering only 0·4 Mflops, hundreds of T800-20 transputers are required to achieve 'supercomputer' performance. For example, the 64 h aircraft cabin fire simulation would require 112 trasnputers (working at 80% efficiency) in order to complete the simulation in 1 h. However, if each PE could achieve 10 Mflops, only five would be required.

This performance level is currently being pursued with a new-generation TRAM which incorporates both a T800 transputer and an Intel i860 microprocessor. The i860, capable of performing vector operations, has a peak performance of 80 Mflops. The T800 is used for communication purposes while the i860 performs the compute-intensive calculations. Use of the i860 will mean that, with the exception of inserting vector calls, minimum alterations to the parallel version of FLOW3D will be necessary. Our early experiences with the i860 reveal that the $20 \times 20 \times 30$ case when run on a single i860 required under 3·6 h. of processing. This compares with a time of 4·42 h when run on 15 transputers in parallel.

The next-generation transputer—known as the T9000 and due for release in 1991—also offers the possibility of enhanced performance. This is expected to be of the order of a 10-fold improvement in processing speed and communication rates.

10. CONCLUSIONS

The pipeline TRAM architecture allows for the efficient solution of 'large' CFD problems. In order to achieve maximum performance from a given array of PEs, it is essential that they be configured with sufficient memory to allow the solution of massive problems involving hundreds of thousands of computational cells. The memory issue becomes critical when simulating two-phase phenomena such as fire–sprinkler interaction. These compute-intensive simulations involve nearly twice as many variables and hence will involve twice as much memory. In addition, it is desirable to orientate the geometry of the problem such that the direction containing the maximum number of cells is conicident with the partition direction. This must be coupled with a good balance of computation amongst PEs to achieve maximum efficiency.

On fire field models the technique has achieved efficiencies of 85%. Performed on a 15-PE system, run times for fire simulations involving transient, three-dimensional, turbulent, buoyant flows on a mesh of 24 000 computational cells have been reduced from more than 4 days to 8 h. It is expected that parallel/vector PEs will aid this development further.

ACKNOWLEDGEMENTS

The authors are indebted to AEA Technology Harwell for making available the source code of FLOW3D and to the U.K. Civil Aviation Authority and the SERC for funding. They also wish to acknowledge the contributions of Stephen Johnson, Mark Cross and Peter Chow.

REFERENCES

1. E. R. Galea, 'On the field modelling approach to the simulation of enclosure fires', *J. Fire Protect. Eng.*, **1**, 11–22 (1989).
2. E. R. Galea and N. C. Markatos, 'Modelling of aircraft cabin fires', *Fire Safety Science Proc. 2nd Int. Symp.*, Hemisphere, Washington DC, 1989, pp. 801–810.
3. K. A. Pericleous, D. R. E. Worthington and G. Cox, 'The field modelling of fire in an air-supported structure', *2nd Int. Symp. on Fire Safety Science*, Tokyo, Hemisphere, Washington DC, January 1988, pp. 871–880.
4. S. Kumar, N. Hoffmann and G. Cox, 'Some validation of Jasmine for fires in hospital wards', in *Numerical Simulation of Fluid Flow and Heat/Mass Transfer Processes*, Springer, Berlin, 1986, p. 159.
5. R. Waters, 'Air and smoke movement within a large enclosure', in *Numerical Simulation of Fluid Flow and Heat/Mass Transfer Process*, Springer, Berlin, 1986, pp. 135–147.
6. S. Simcox, N. S. Wilkes and I. P. Jones, 'Fire at Kings Cross Underground Station, 18th November 1987; numerical simulation of the buoyant flow and heat transfer', *U.K. Atomic Energy Authority Harwell Report AERE-G 4677*, 1988.
7. E. R. Galea and N. C. Markatos, 'The modelling and computer simulation of fire development in aircraft', *Int. J. Heat Mass Transfer*, **34**, 181–197 (1991).
8. N. A. Hoffmann, E. R. Galea and N. C. Markatos, 'Transient Two-phase fire–sprinkler simulation', *IASTED Int. Conf. on Modeling, Simulation and Optimisation*, Montreal, May 1990.
9. J. R. Kightley and I. P. Jones, 'A comparison of conjugate gradient preconditionings for three-dimensional problems on a CRAY-1', *Comput. Phys. Commun.*, **37**, 205–214 (1985).
10. J. R. Kightley and C. P. Thompson, 'On the performance of some rapid elliptic solvers on a vector processor', *SIAM J. Sci. Stat. Comput.*, **8**, 701–714 (1987).
11. H. A. Van der Vorst, 'The performance of FORTRAN implementations for preconditioned conjugate gradients on vector computers', *Parallel Comput.*, **3**, 49–58 (1986).
12. C. S. Ierotheou, C. W. Richards and M. Cross, 'Vectorization of the SIMPLE solution procedure for CFD problems—Part I: A basic assessment', *Appl. Math. Modell.*, **13**, 524–529 (1989).
13. C. S. Ierotheou, C. W. Richards and M. Cross, 'Vectorization of the SIMPLE solution procedure for CFD problems—Part II: The impact of using a multigrid method', *Appl. Math. Modell.*, **13**, 530–536 (1989).
14. C. S. Ierotheou, 'The simulation of fluid flow processes using vector processors', *Ph.D. Thesis*, Thames Polytechnic, London, 1990.
15. *IMS T800 Architecture, Technical Note 6*, Inmos Ltd., Bristol, January 1988.
16. *Parallel FORTRAN User Guide*, 3L Ltd., Livingstone, 1990.
17. A. D. Burns and N. S. Wilkes, 'A finite-difference method for the computation of fluid flows in complex three dimensional geometries', *U.K. Atomic Energy Authority Harwell Report AERE-R 12342*, 1987.
18. M. Cross, S. Johnson and P. Chow, 'Mapping enthalpy-based solidification algorithms onto vector and parallel architectures', *Appl. Math. Modell.*, **13**, 702–709 (1989).
19. S. Johnson, M. Cross and P. Leggett, 'Casting simulation on highly parallel computer architectures', *Proc. Int. Conf. on Modelling of Casting, Welding and Advanced Solidification Processes*, Davos, September 1990.
20. S. Johnson and M. Cross, 'Mapping CFD algorithm onto fine grained parallel architecture', *Appl. Math. Modell.*, in press.
21. C. M. Rhie and W. L. Chow, 'Numerical study of the turbulent flow past an airfoil with trailing edge separation', *AIAA J.*, **21**, 1525–1532 (1983).
22. J. P. Van Doormaal and G. D. Raithby, 'Enhancements of the SIMPLE method for predicting incompressible fluid flows', *Numer. Heat Transfer*, **7**, 147–163 (1984).
23. R. A. Issa, 'Solution of the implicitly discretised fluid flow equations by operator-splitting', *J. Comput. Phys.*, **62**, 40–65 (1985).
24. N. C. Markatos, M. R. Malin and G. Cox, 'Mathematical modelling of buoyancy induced smoke in enclosures', *Int. J. Heat Mass Transfer*, **25**, 63–75 (1982).
25. N. A. Hoffmann, 'Computer simulation of fire–sprinkler interaction', *Ph.D. Thesis*, Thames Polytechnic, London, 1990.
26. S. V. Patankar, *Numerical Heat Transfer and Fluid Flow*, McGraw-Hill, New York, 1980.
27. S. V. Patankar and D. B. Spalding, 'A calculation procedure for heat, mass and momentum transfer in three-dimensional parabolic flows', *Int. J. Heat Mass Transfer*, **15**, 1787–1806 (1972).