I.C.A.S. '90 Proceedings pp 1557 – 1568

ICAS-90-6.8.2

PARALLEL IMPLEMENTATION OF AN EXPLICIT FINITE VOLUME EULER SOLVER ON AN ARRAY OF TRANSPUTERS

S.R. Turnock
Department of Ship Science, University of Southampton,
Highfield, Southampton. SO9 5NH. United Kingdom.

ABSTRACT

Transputers are specially developed micro-processors which can be linked together to form parallel computers. This paper looks at the ease with which arrays of transputers can be used to solve numerical fluid dynamic algorithms. An explicit two-dimensional finite volume scheme for solving the compressible Euler equations has been implemented to run across an array of transputers. The method used is based on a cell-vertex scheme and a Multiple-Grid scheme is used to accelerate convergence.

The main part of the work has been the development of a general Harness process to control communications between transputers. Numerical algorithms can be made to run on a variable sized array of transputers without the need to write communications software for each application. The effectiveness of the parallel implementation of the Euler solver was measured. It was found that the speed-up in overall time to convergence obtained by increasing the number of transputers was dependent on the physical size of the problem. The larger the number of Transputers, the more efficiently each Transputer was used. With a small domain of 16 by 16 finite-volume cells the code ran 9.7 times faster on 16 Transputers compared to a single Transputer. For a 48 by 48 cell problem a speed-up of 13.4 was obtained with 16 Transputers. For Transputer arrays with more than 4 Transputers the time to carry out one timestep was inversely proportional to the number of Transputers in the array.

NOMENCLATURE

ρ	= density
x,y	= cartesian coordinates
u,v	= velocity in x,y direction
a	= local speed of sound
p	= pressure
e	= specific energy
γ	= Ratio of Specific Heats
T	= Temperature
C_v	= Specifc Heat capcity
S	= Speed-Up
Ω	= Code Efficiency

INTRODUCTION

The method of solution of large-scale Computational Fluid Dynamic (C.F.D.) problems on concurrent computational machines is an area of research which is rapidly developing as more advanced parallel computers are designed. The actual ability of current C.F.D. codes to take full advantage of these new machines is limited by the difficulties in transporting existing sequential code onto the new machines.

The aim of this research has been to investigate the ease with which an explicit finite volume scheme for solving the compressible Euler equations could be mounted on a parallel computer based on an array of Transputers. Jong[1] investigated performance-cost comparisons for a number of different machines based on computational speed divided by system cost. It was found that Transputer based parallel computational machines were the most cost effective devices.

Transputer based concurrent computers are Multiple-Instruction-Multiple-Data stream (M.I.M.D.) parallel machines. They permit both data transfer and the execution of instructions to be executed concurrently. The advantage of transputer type parallel processing systems is that the same basic processing unit is used for both small-scale and large-scale computational applications. Code can be developed on inexpensive machines with a small number of transputers and then executed on a large array of transputers. The improvement in speed of execution when increasing the number of transputers in an array is a basic measure of the performance of a particular application.

The main component of the work has been the development of a Communication Harness to handle communication between the various processes executing in parallel on different Transputers. The Communications Harness, running in parallel with a numerical fluid dynamic algorithm, transfers information around an array of transputers. The separation of a particular application into a Communications Harness and a numerical component permits a numerical algorithm to be developed independently of any particular transputer system.

An explicit finite volume numerical algorithm for the solution of the compressible Euler equations is a good test of the effectiveness of transputers in solving more general fluid problems. The Euler Solver provides a variable computational load for investigating the performance of the Communications Harness in relation to the numerical algorithm. Altering the number of finite-volume cells changes the computational load on the Transputer array.

Concurrent Algorithms

The method by which a numeric algorithm is sub-divided so that the various components can execute concurrently is dependent on the size of the individual computational tasks and the system upon which it is to be run. Fundamental to this is the independence of the sub-divisions. An independent task requires no additional information from other tasks while it is executing. The parallelism strategy used will be dictated by the ease with which the algorithm can be broken into independent processes or processes which require a minimal exchange of information. There are three approaches to implementing a Parallel algorithm.

1) FARM: A central control process sends independent data packets out to worker processes. Each worker processes the data and sends a processed data packet back to the central controller and waits for another data packet to process. As each worker is an independent unit, increasing the number of workers will increase the speed with which data packets can be processed. The limit to performance is the speed with which the control process can send and receive data.

2) PIPE (Algorithmic): Data packets are sent along a line of parallel processes. Each Parallel process carries out a particular part of the overall algorithm before passing it on to the next process. The speed of processing data is determined by the processing time of the slowest individual process. Increasing the number of workers will not necessarally improve performance unless all workers can process data at a greater rate.

3)GEOMETRIC: The algorithm is divided into a number of sub-regions and each worker processes an allocated sub-region of the data. If necessary, data from other sub-regions are communicated between the individual workers. The addition of extra workers either allows a larger data set to be processed in the same time or allocates smaller amount of data to each processor. This increases the speed of calculation. Geometric parallelism can be used for problems which can be sub-divided into processes requiring minimal exchange of information. For processes with a small ratio of communications to data processing, the speed-up should be roughly proportional to the number of allocated workers.

Geometric parallelism is the obvious method for solving explicit Finite volume type problems. The process of time marching to a steady-state solution requiring information for updating finite-volume cells only from adjacent cells. The only exchange of information necessary is between the boundary cells on adjacent workers. Therefore, as the number of workers (Transputers) are increased the amount of communication remains constant. For a two-dimensional problem each worker only ever has to exchange information with a maximum of four neighbouring workers.

The work described in this paper consists of two parts, the development of a general fluid dynamic communications harness and the implementation of an explicit finite volume algorithm for solving the compressible Euler equations. The numerical algorithm allows the performance of the Harness to be assessed on various sizes of Transputer arrays and on the amount of data assigned to each Transputer. The implementation of the numerical algorithm gives an indication of the ease with which existing algorithms can be rewritten to exploit the parallelism of Transputer based machines.

TRANSPUTERS

Transputers are a range of high-performance VLSI technology devices developed by Inmos Ltd,[2] which consist of local memory, four high speed two-way links and a micro-processor unit all mounted on a single silicon chip. Figure 1 shows a schematic of the layout of a typical Transputer.

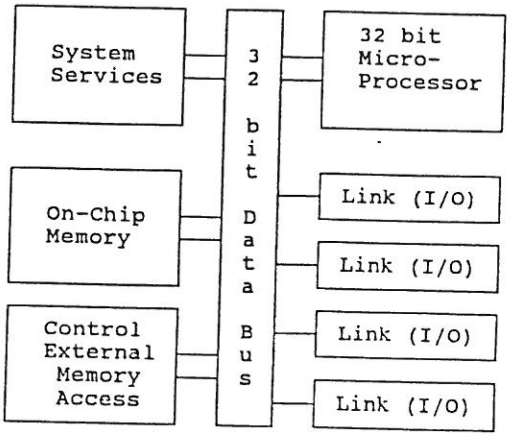


Figure 1. Schematic of Diagram of a Typical Transputer.

The provision of high speed communication links allows Transputers to be connected together to produce a parallel processing computer. There are no limits to the number of Transputers which can be connected together. There are no restrictions to the topology of the

parallel machine as long as each Transputer is connected to no more than four others. Massively parallel machines can be built up from arrays of Transputers. Software developed on small array machines can be easily transferred onto large array machines.

Each Transputer can have external memory assigned to it. Communication occurs by point to point access across the links between individual Transputers. The distributed nature of memory in a Transputer machine does not allow global access to memory except by communication between Transputers.

A range of Transputers has been developed. At present, the T800 Transputer has the best performance, with a peak of 2 million floating point operations per second (2Mflops) and a data transfer rate of 10 or 20 Mbits/sec on each link. Internal memory of 4K of fast access memory is provided, and up 4GBytes of memory can be connected to each Transputer.

Communications Harness

The passing of data messages between parallel processes needs to be carefully controlled. The means by which messages pass between Transputers should be made independent of the processes running on each transputer. It was decided to produce a communications harness to deal with all types of messages and control their routeing through a transputer array. This allows numerical algorithms to be developed independently of the specific transputer machine upon which they are to be run.

OCCAM2

OCCAM2 is a high level language which exploits the concurrency of the transputer[2,3]. Designed to express concurrent algorithms as communicating parallel processes, OCCAM2 allows the programmer to produce programs which explicitly deal with communication. Communication takes place along one-way channels. Message protocols define the type of message to be transferred along these channels. When processes on different Transputers communicate, the channels are mapped onto the joining transputer links.

Parallel versions of high-level programming languages such as FORTRAN77 and C do exist. However, it is more efficient to use OCCAM2 code to deal with interprocess communication. This can then act as a harness for embedded FORTRAN77 or C code. The communication harness provides all the necessary input and output routines for the embedded code.

For this work it was decided to write all the software in OCCAM2. This allows the communications model and concurrency of OCCAM2 to be fully exploited in both the design of the communications harness and in the numerical algorithm. The translation of existing FORTRAN77 code into OCCAM2 was found to be a straightforward task.

Available Parallel Transputer Machines

At Southampton University there are a number of commercial and development transputer machines which have been used for this work. The independence of the OCCAM2 code from the transputer machine upon which it runs allowed the majority of the code to be developed on the Ship Science Transputer System and then transferred onto larger machines to investigate speed-up performance. Table 1. lists the capabilities of the machines used in this work.

Table 1. Transputer Systems Available at the University of Southampton.

Machine	Number of Transputer	Memory Kbyte	Speed MFlop
Ship Science System	4* T800	1000	8
Parsys SuperNode	16* T800	256	32
Parsys MegaNode	128* T800	256	256

The links between individual Transputers on the Ship Science Transputer System have to be manually connected to alter topology of the transputer array. The two commercial machines (SuperNode and MegaNode) allow the individual transputer link settings to be made from within the software. All three machines are accessed via a control(Host) Transputer linked to a terminal providing keyboard, screen and disk storage support.

DEVELOPMENT OF A GENERAL COMMUNICATIONS HARNESS

The aim in producing the harness was to allow various strategies for solving concurrent algorithms to be tested. The harness is designed to remove the need for rewriting the inter-process communications when either the topology of the transputer array or the specific numerical algorithm is altered.

Design of Communications Harness

The communications requirements of distributed fluid dynamic algorithms are of a similar nature and can be broadly classed into the following categories:

1) Nearest neighbour communication: Information sent from one transputer to another, e.g. the updating of node information across a common physical boundary.

2) Global message passing: Information to be sent or collected from all the Transputers in an array, e.g. Examining the convergence to solution of a series of processes.

3) Process Information: The user can only directly access information on the control transputer. Therefore, status messages, graphical output and information from other Transputers generated when verifying code have to be sent via the communications harness.

The maximum length of an individual message and the number of messages the harness can deal with at any one time are important design parameters. Large numbers of small messages are liable to increase the ratio of communication time to calculation time, and may overwhelm the harness. However, large messages require greater memory storage at all stages within the harness which reduces the size of data an individual transputer can operate on.

Modular Approach

To aid in the development of the C.F.D. harness the individual components of the harness were written as independent processes. These could be separately tested and if necessary upgraded without affecting the overall structure of the harness. The use of this approach saves considerable development time and is an advantage of the use of OCCAM2. The inherent flexibility in a modular approach allows rapid changes to be made to the overall structure of the harness.

Control of Communications

The main purpose of the Harness process is to control all interprocess communication. The user processes simply input or output messages when necessary (similar to READ/WRITE functions in FORTRAN77) and the Harness ensures that the information is sent to the correct destination.

Figure 2 shows a typical transputer array. One process (Host) is used to control all the others (Guests). The Host process is mounted on the Transputer connected to the PC interface allowing keyboard input, screen display and disk access. The individual Host and Guest processes can communicate with each other only via a Harness process which runs in parallel with the application process on each transputer. The Harness determines the route each message will take through the transputer array to get to its destination.

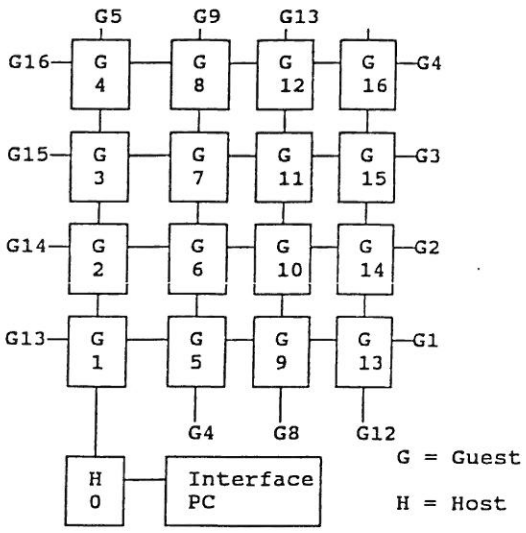


Figure 2. Layout of a Square Array of 16 Transputers showing link connections.

Routeing Strategy

Figure 2 illustrates the simple strategy chosen for routeing messages through a rectangular array of Transputers. Messages from the Host process pass along the lowest row of Transputers until they reach the column which includes their destination. Nearest neighbour communication occurs directly between adjacent Transputers. For a computational problem with a higher density of messages than a finite-volume scheme, a more subtle strategy would be required to avoid bottle neck problems in the bottom row of Transputers. Figure 2 also shows the connection of end of row Transputers to each other to provide a closed ring. This provides a short route for messages between the end of row Transputers. An equivalent fluid dynamic problem is where two boundaries are periodic with each other or the computational grid is closed ('O' Grid).

Each Transputer process is assigned an unique identity number. The Host process is set as Process 0. For a nearest neighbour type problem, where each transputer is exchanging information only with its direct neighbours, each link is coded as North, South, East or West to simplify message passing between neighbouring Transputers. The regular ordering of Transputers through the grid allows a simple algorithm to be used to determine which link a message needs to be sent on to reach a given destination.

Communications Overhead

The time each transputer spends dealing with communications rather than executing the numerical algorithm is an overhead. The Harness on each transputer has to process messages at high priority to allow

the overall calculation to proceed without delays. Similarly, the time a user process is idle waiting for an incoming message is an overhead and reduces the efficiency of the code. Good practice is to ensure that there is something for each GUEST process to carry out while awaiting incoming messages. This effectively 'hides' communication time and minimises the associated delays.

Deadlock

A potential problem when controlling concurrent communicating processes is deadlock. This occurs when every process is waiting to receive a message and no messages are sent so that the calculation can proceed. The provision of buffers and a routeing strategy which avoids bottle-necks minimises the risk of deadlock. However, even the best designed harness will fail if a user process generates messages which no process is waiting to receive. Eventually all buffers fill up with redundant messages and deadlock occurs.

Buffering

The amount of buffering required depends on the likely amount of communication and the effectiveness of the routeing strategy in avoiding bottle-necks. The number of buffers is restricted by the need to provide storage space for each additional buffer. A nearest neighbour problem gives a light communications load across a Transputer array, with messages only crossing between adjacent Transputers. A simple routeing strategy and a small number of buffers can adequately control communications.

Harness Structure

At the outermost level, the code mounted on each Transputer consists of two processes running in parallel as shown in Figure 3. These are the user's process (Host or Guest) and the Harness process.

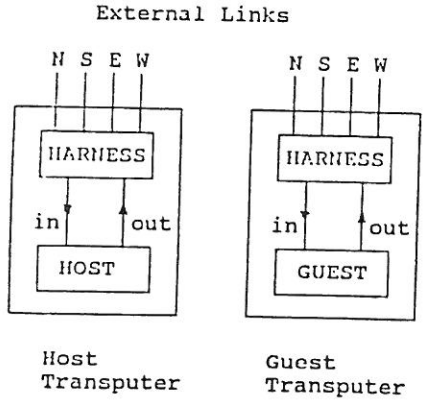


Figure 3. Schematic of Harness process controlling communication between GUEST and HOST processes and the Transputer external links.

The basic structure of the Harness is shown in Figure 4. Harness consists of a number of processes running in parallel. Data flows into the Harness process from any of the four external links connected to other Transputers. All four channels are multiplexed onto a single channel, reducing the need for a buffer for each individual link. The WhichBuffer process allocates incoming messages to the first available empty buffer. At present, for the finite-volume scheme there, are two input buffers.

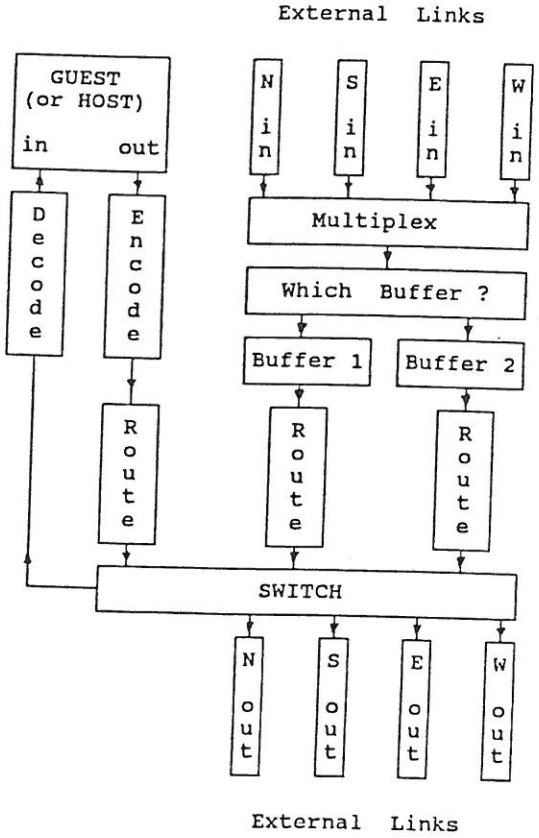


Figure 4. Schematic of Data Flow Through Harness process

Messages from the Guest (or Host) process are encoded into a standard form in the Encode process which also acts as an input buffer. A Route process is assigned to each input Buffer which determines the destination of a message. Based on the message destination, it decides which external link should be used to pass on the message or whether the message is for the resident Host (or Guest) process. Route sends the message to the Switch process which ensures that the message is passed out of the Harness either on one of the four external links or to the Decode process. The Decode process translates messages into a form readable by the Guest (or Host) process.

The structure of the harness ensures that only relevant messages are sent to the resident Guest (or Host) process on each transputer. Other messages are simply passed through the Harness, leaving by the link determined by the routeing strategy.

All harness processes run in parallel with each other. OCCAM2 currently allows processes to be assigned one of two priority levels (High or Low). The overall Harness process and all its component processes are set to High priority, so that incoming messages are dealt with immediately they arrive. This ensures that other Transputers further along the array will have to wait a minimum amount of time for messages.

Message Protocol

Different types of messages are allowed in OCCAM2. The content of messages is defined by protocols. The Harness process uses two simple message protocols:

1) User Protocol. This allows the Guest (or Host) process to send messages of: integer, Boolean, real, and string variables; and one or two-dimensional arrays of real numbers. Larger data structures are built up from a number of messages containing different parts of the structure. A component of the protocol defines the destination of a message as either to be sent to the Transputer connected to the North, South, East, or West links, or to the Host process. An individual Process number can also be stated and this is used when the Host process communicates to individual Guest processes.

2) Harness Protocol. This comprises a header of 25 bytes which defines the type of message, the identity number of the destination and source transputer. The remainder of the message is a variable length one-dimensional array of bytes. The use of this protocol simplifies the passage of messages through the Harness, each component process only having to deal with one type of message.

The Guest and Host processes use the User Protocol to send and receive messages. The protocol allows flexibility about the sorts of messages which can be sent between processes. As well as data the protocol also allows development messages (Debugging information) to be sent. The Encode and Decode processes use the protocol definitions to translate messages between the two protocols. The Harness Protocol is used for passing messages around the array of Transputers.

Measurement of Harness Performance

The communications between parallel processes is an overhead to the overall efficiency of a concurrent numerical algorithm. The more quickly messages can be passed between processes, the less delay there is in the calculation. OCCAM2 provides TIMER channels which allow measurements to be made of the number of transputer internal clock pulses a process uses during execution. Measurements of the time the Guest and Host processes spend communicating compared with how long they spend calculating were made. These timings were used to define an overall efficiency for a numerical algorithm running on a specific size of transputer array.

$$\cap = \frac{\text{Calculation Time}}{\text{Calculation Time + Communication Time}}$$

Code efficiency $\cap$ is defined as the ratio of actual calculation time to overall calculation time. The size of problem assigned to each Transputer influences the efficiency. Increasing the size of problem on each Transputer increases the code efficiency as proportionally less time is spent communicating. However, the calculation will take longer. Code efficiency is a useful parameter in determining the number of Transputers necessary to perform a calculation and the amount of local memory each transputer requires.

Comparing the overall calculation time for an identical problem run on different sizes of transputer arrays indicates the actual speed-up obtained.

$$S = \frac{T_1}{T_n}$$

Speed-up S is defined as the ratio of the time taken to perform the calculation on one transputer T_1 to the time taken on n Transputers T_n. A theoretical speed-up of n would occur if no communication took place between Transputers. For any algorithm which requires communication, an efficient Harness minimises the time spent communicating.

A true bench-mark of the performance of a numerical algorithm is a difficult quantity to assess. The performance of identical code can vary widely when implemented on different computers. For instance with limited memory machines, a trade-off has to be made between recalculating parameters when they are needed and storing them for the whole calculation. A larger problem can then be solved, but the calculation will take longer. On computers with shared access, the actual speed of solution will depend on the users priority and how many others are using the computer. No comparison has been made between the performance of the Euler Solver implemented on an array of Transputers and that of the same algorithm on other machines. However, the times used in calculating code efficiency and speed-up have been given in seconds based on the

T800's clock time of 64 μSec per tick. This allows the actual performance of the code to be assessed.

EULER FLOW SOLVER

The main part of this investigation has been the development of software tools to allow general C.F.D. problems to be solved on transputer arrays. To validate the Harness a two-dimensional explicit finite volume scheme developed by Ni[4] for solving the compressible Euler equations was implemented. A Fortran77 version of this method was available and this was translated into OCCAM2 to form the basis of the Guest process.

Numerical Method

The basic formulation consists of an solution of the unsteady Euler equations where **E**, **F**, **G** are four component vectors which correspond to the mass, cartesian x and y momentum, and total energy conservation equations. This can be written in compact vector form as:

$$\frac{\delta}{\delta t}\bar{U} + \frac{\delta}{\delta x}\bar{F} + \frac{\delta}{\delta y}\bar{G} = 0$$

where the state vector

$$\bar{U} = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho E \end{bmatrix}$$

and

$$\bar{F} = \begin{bmatrix} \rho u \\ \rho u^2+p \\ \rho uv \\ \rho (E+p/\rho)u \end{bmatrix} \qquad \bar{G} = \begin{bmatrix} \rho v \\ \rho uv \\ \rho v^2+p \\ \rho (E+p/\rho)v \end{bmatrix}$$

E is the total energy per unit mass

$$E = e + \tfrac{1}{2}(u^2 + v^2)$$

These are coupled with the auxiliary equations for a perfect gas:

$$p = (\gamma - 1) \rho e \quad \text{and} \quad e = C_v T$$

A steady-state solution is obtained by explicit time-marching from an initial condition and applying appropriate boundary conditions at the edge of the physical domain. Expressed in integral conservation form the equation becomes:

$$\frac{\delta}{\delta t}\iint \bar{U}\, dA + \oint (\bar{F}\, dy - \bar{G}\, dx) = 0$$

Ni's method uses a cell-vertex method where at each time-step the integral of fluxes through the cell edges are summed around the cell to give a first-order difference in state vector ΔU at the cell centre using the finite-volume approximation.

$$\Delta U_C = \left[\frac{F_1 + F_2}{2} \Delta y - \frac{F_3 + F_4}{2} \Delta y + \frac{G_1 + G_2}{2} \Delta x - \frac{G_3 + G_4}{2} \Delta x \right] \frac{\Delta t}{\Delta x \, \Delta y}$$

Where C refers to the centre of the finite-volume cell and 1,2,3,4 the four corners of the cell. The difference in fluxes at the cell centre is distributed to the four corners of the cell using Ni's distribution formula for each corner:

$$\delta\bar{U}_1 = \tfrac{1}{4}(\Delta\bar{U}_C - (\Delta t/\Delta x)\Delta\bar{F}_C - (\Delta t/\Delta y)\,\Delta\bar{G}_C)$$

$$\delta\bar{U}_2 = \tfrac{1}{4}(\Delta\bar{U}_C - (\Delta t/\Delta x)\Delta\bar{F}_C + (\Delta t/\Delta y)\,\Delta\bar{G}_C)$$

$$\delta\bar{U}_3 = \tfrac{1}{4}(\Delta\bar{U}_C + (\Delta t/\Delta x)\Delta\bar{F}_C + (\Delta t/\Delta y)\,\Delta\bar{G}_C)$$

$$\delta\bar{U}_4 = \tfrac{1}{4}(\Delta\bar{U}_C + (\Delta t/\Delta x)\Delta\bar{F}_C - (\Delta t/\Delta y)\,\Delta\bar{G}_C)$$

The signs of the second-order correction terms ΔF and ΔG depend on the position of the cell centre relative to the corner and has an effect similar to the use of upwind or downwind differencing.

$$U_1^{m+1} = U_1^{m} + (\delta U_1)_A + (\delta U_1)_B + (\delta U_1)_C + (\delta U_1)_D$$

The state vector U_1 at each vertex is updated by summing the corrections from the four neighbouring cells A,B,C,D.

Time Step and Numerical Smoothing

To converge to a steady-state solution local time-stepping is used. The time step for each finite-volume cell is set to the maximum that will keep the numerical scheme stable. The formula given by Ni[4] is used.

$$\Delta t \le \min \left[\frac{\Delta x}{|u| + a}, \frac{\Delta y}{|v| + a} \right]$$

Where a is the local speed of sound for each cell.

It is necessary to provide numerical smoothing to capture shock features and prevent oscillatory growth of errors. The method used is that given by Ni where smoothing terms are introduced into the distribution formula as follows:

$$\delta U_1 = \tfrac{1}{4}(\Delta U_C - (\Delta t/\Delta x)\Delta F_C - (\Delta t/\Delta y)\,\Delta G_C + \mu(\bar{U} - U^1)$$

Where $\bar{U} = \tfrac{1}{4}(U_1 + U_2 + U_3 + U_4)$

and $\mu = \sigma ((\Delta t/\Delta x) + (\Delta t/\Delta y))$. The artificial damping factor σ is in the range 0 to 0.1.

Initial and Boundary Conditions

The cell-vertex scheme ensures that the state vector values **U** are calculated at the boundaries of the physical domain. As an initial condition the flow is assumed to be at the freestream value. The finite volume approximation to the integral equation expresses the difference in state vector **U** at a cell-centre in terms of fluxes across the four edges of the cell. Six different types of boundary condition have been implemented for cell edges which are on the boundary of the domain of interest. Mapping sub-regions of the physical domain onto arrays of Transputers is thereby simplified. The join between two sub-regions on different Transputers is classed as a boundary to the sub-region and is specified during the initial problem set up. The boundary conditions are: Internal, where two edges join on the same transputer; Normal, a boundary between sub-regions across which fluid can flow; Inflow and Outflow, for flow in and out of the domain which is nearly normal to the freestream direction, updated using the method of characteristics to determine the domain of influence; Solid, is treated as a reflective boundary with flow kept tangential to the surface (Hall[6]); Free, for boundaries at a distance where changes to the far-field can be assumed negligible

Convergence

The calculation is assumed to have converged to a steady-state when the maximum change in the x-momentum (ρu) component of the state vector **U** is everywhere less than $1.0\ e^{-5}$.

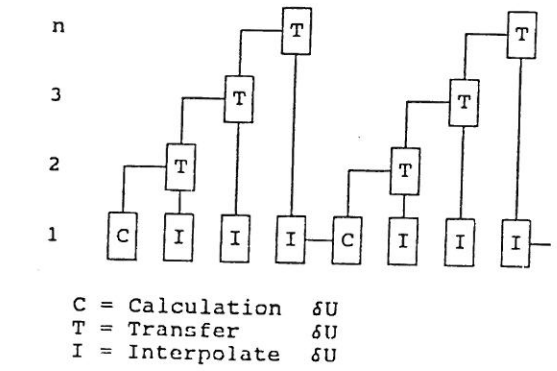


Figure 5. Saw-tooth Cycle for Ni's Multiple-Grid scheme

Multiple-Grid

To improve the rate of convergence of the flow solver the Multiple-Grid scheme of Ni[4,5] was implemented. The coarsest level of updating information is restricted to the area of grid mapped onto each transputer and the boundary information is treated in the same manner as for the transfer of information on the finest level of calculation. Only minimal modification to the code was required to fully implement the Mutiple-Grid scheme. Figure 5. shows the saw-tooth cycle for transferring cell-vertex corrections δU to coarser and coarser grids. After each transfer operation, the corrections were interpolated back to the finest grid and the state vector **U** updated.

Parallel Implementation of Euler Solver

The implementation of Ni's method for solving the compressible Euler equations is straightforward. The method used is based on geometric parallelism with a Host process controlling a number of Guest processes. Each Guest process is assigned a sub-region of the physical domain containing the same number of finite-volume cells. Figure 6 illustrates the overall scheme, with the left and right halves of the figure indicating the role of the Host and Guest processes.

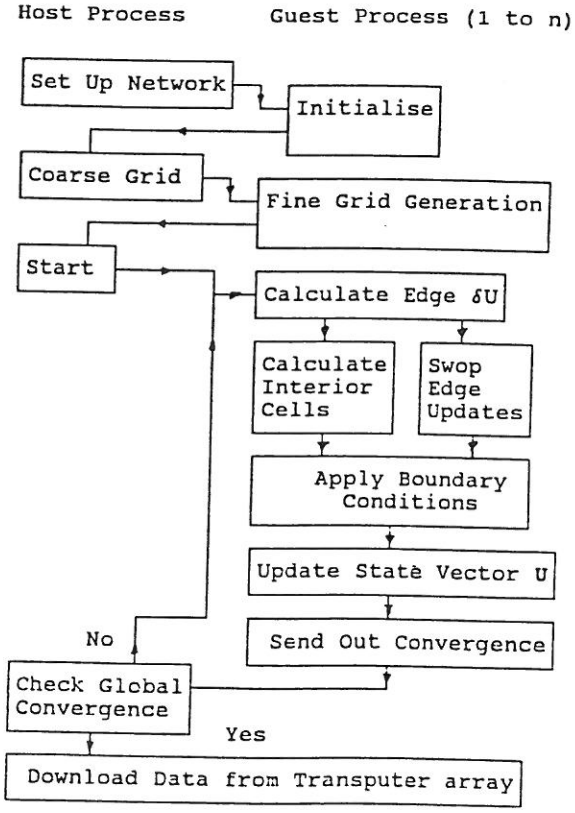


Figure 6. Schematic of Host and Guest processes for Euler Solver.

Initialise and Set Up Network

Each Guest process is told the dimensions of the Transputer size and its individual identity number. This information is used to initialise the Routeing process and for determining the identity number of a Guest Transputers four neighbours Transputers.

Grid Generation

The definition of the solution domain for an explicit finite volume scheme requires a grid mesh mapped over the physical space. The generation and quality of this grid are important parameters in the efficiency and accuracy of the numerical scheme. The requirement for the same physical problem to be solved across different sizes of Transputer arrays meant that the pre-processing of the solution grid was most easily carried out by making it an integral part of the Euler Solver.

A simple input file format was adopted. For each geometric problem, four external edges are defined, each edge corresponds to a single type of boundary. As a first approximation the required number of interior nodes are produced by direct interpolation between the four edges. The initial grid is refined using the elliptic grid generation method described by Steger and Sorenson[8]. A Successive Over Relaxation scheme is used to converge the grid. The edge values of the grid are held fixed.

The actual generation of the grid is divided into two steps. On the host process a coarse grid is elliptically refined. This grid is then subdivided into a number of regions equal to the number of Transputers available. The four edges of each of these sub-regions along with the corresponding boundary types are sent to the Guest processes. On each Guest the process is repeated with the required mesh refined using the elliptic grid generator. The coordinates of the edge values on adjacent Transputers are identical and are held fixed.

The format of the grid generation process allows a physical problem to be automatically matched to the number of Transputers available. More complex geometries can be solved by breaking the overall physical domain into several segments each mapped onto a number of arrays within the Transputer array.

Numerical Method

On each Guest process the first operation at each time-step is the calculation of δU updates for all the edge cells. Next the interior cell updates to the state vector U are calculated. In parallel with this for any edges in common with other Transputers δU values for the edge are exchanged. The communication of edge updates in parallel with the calculation of the state vector updates effectively hides the communication time.

The exchanging of information across a sub-region boundary takes place in two stages. First North-South edge information is exchanged and then East-West information. This ensures that all the information for corner nodes is sent to all four possible Transputers upon which the corner node is defined.

The number of messages each Transputer needs to send and receive at each time step is proportional to the number of Normal sub-region boundaries with other Transputers. For each edge on the Transputer which is a Normal boundary, two two-dimensional arrays containing δU updates are sent.

After message communication and calculating the interior updates, the boundary node state vectors are updated using the assigned boundary conditions. Finally, the actual interior state vectors are updated.

Global Convergence

For a distributed problem the determination of convergence requires that the maximum convergence on all Transputers is known and that when all the individual maxima are below the convergence threshold the calculation is stopped. This is achieved by all the Guest Transputers sending convergence information back to the Host process. The Host then confirms that the solution has converged or allows the calculation to proceed. The global convergence test causes a delay in execution while the Transputers await the instruction to proceed or terminate. The delay is dependent on the number of Transputers and the average number of Transputers each message has to pass through to arrive at the Host. The larger a Transputer array the greater the delay associated with a global convergence test. A method to reduce this delay is to carry out a convergence check after a regular number of timesteps. Effectively this convergence check synchronises all the Guest processes at regular intervals.

Data Retrieval

If required the steady state solution obtained on each Transputer can be downloaded onto the PC terminal for subsequent processing. For large sizes of Transputer arrays the amount of data storage required is at least equal to the total memory capacity of the Transputer array.

TEST CASE RESULTS

Results are shown for two test geometries representative of both internal and external two-dimensional flow. Figure 7 shows the overall grid used to solve a 64 by 64 cell problem flow over a circular-arc bump placed in a channel with a height equal to the chord of the bump. The maximum height of the bump is 10% of its length.

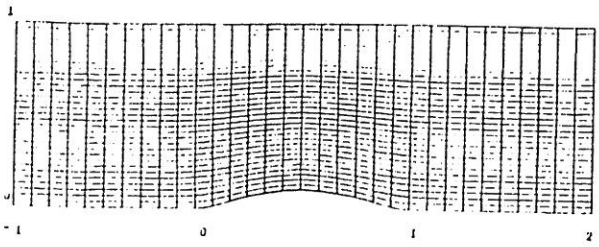


Figure 7. Grid for 10% thick Circular-Arc Bump in Channel.

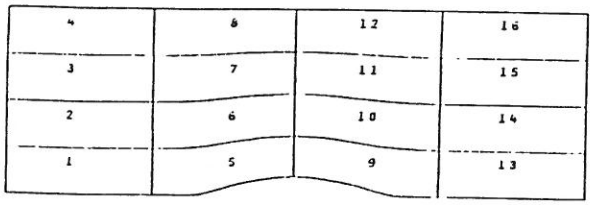
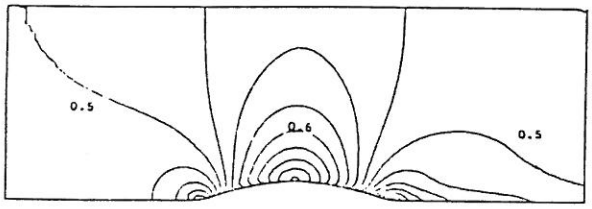


Figure 8. Sub-Division of Channel Grid Onto 16 Transputer array.

Figure 8 shows the sub-division of the grid corresponding to a 16 Transputer array. The numbering of the sub-divisions corresponds to the layout of Transputers in Figure 2.

Results for three different inflow Mach Numbers corresponding to those shown in Ni[4] are given. Figure 9 is for a subsonic inflow with Mach number equal to 0.5 and shows an isomach contour plot and the velocity on the upper and lower edges of the channel.



Iso-Mach Contour Plot

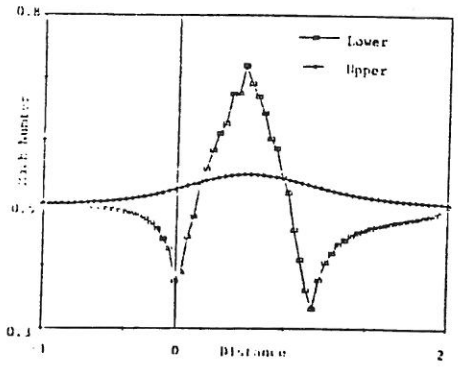
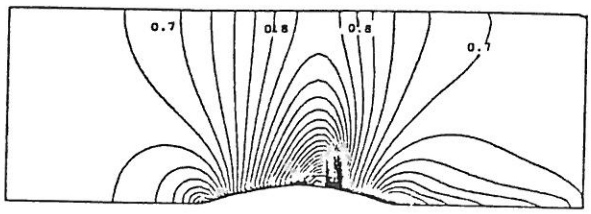
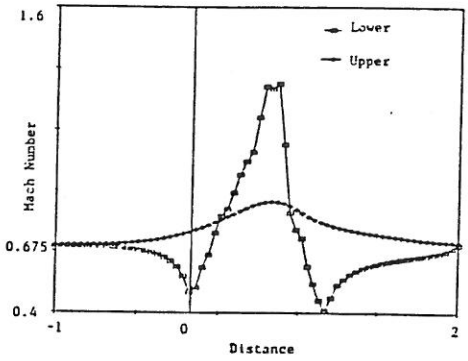


Figure 9. Subsonic Flow over 10% Bump in Channel at Mach Number of 0.5.

A transonic Mach number of 0.675 was used to generate Figure 10.



Iso-Mach Contour Plot



Velocity on Upper and Lower Surface

Figure 10. Transonic Flow over 10% Bump in Channel at Mach Number of 0.675

A NACA0012 aerofoil section was also tested. The grid generated for the NACA 0012 aerofoil section is shown in Figure 11 for a 64 by 64 grid.

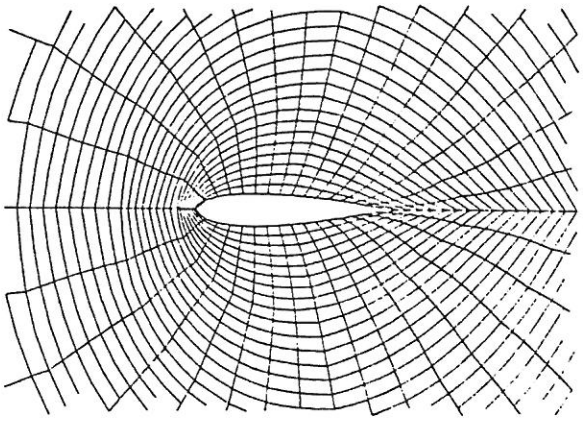


Figure 11 Grid for NACA 0012 aerofoil with mesh size of 64 by 64.

The corresponding sub-division of the grid onto sixteen Transputers is given in Figure 12.

The resultant grid is an 'O' grid with two sides of the Transputer array joined on a line downstream from the trailing-edge. Iso-mach contours are shown for a test case given by Jameson[8] with a freestream Mach Number of 0.5 and angle of attack of 3^{o} in Figure 13.

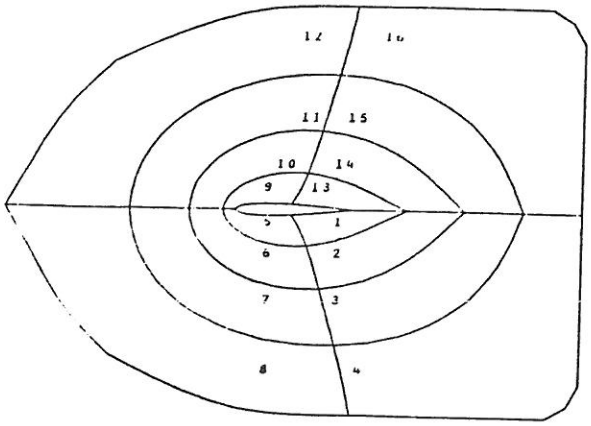


Figure 12. Sub-Division of Physical Domain around a NACA 0012 aerofoil for an Array of 16 Transputers

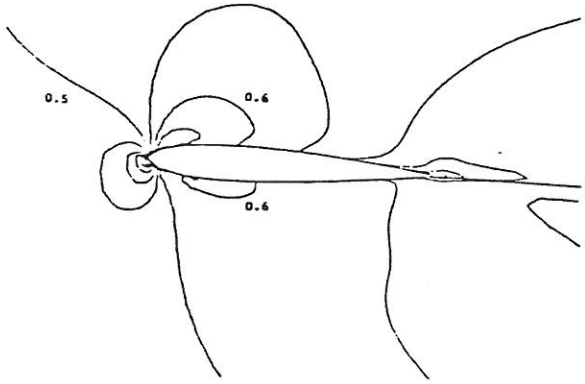


Figure 13 Iso-Mach Contours around a NACA 0012 section at incidence of 3^{o} and at 0.5 Mach No.

CODE PERFORMANCE

Results are presented for the performance of the Euler Solver and Harness on square arrays of up to 16 Transputers. The 128 Meganode machine is still not fully operational. A standard test case for transonic flow at a Mach number of 0.675 over the 10% thick circular-arc bump was used for all the performance measurements. No Multiple-Grid acceleration was used. The data presented is based on the average time to perform one iteration with the same number of finite volume cells on all Transputers in the array.

In addition to the average Iteration time the average amount of time each Transputer spent communicating convergence information to the Host Transputer was measured. The nearest neighbour communication takes place in parallel with the calculation and cannot be measured. The nearest neighbour communication time on each Guest is equal to the difference in time taken to carry out an identical calculation with and without communication across sub-region boundaries.

Figure 14 shows the Speed-Up S obtained when an identical problem was run on different numbers of Transputers. It can be seen that as the overall problem size is increased (more finite-volume cells per Transputer) a greater Speed-Up was obtained. With 16 Transputers a Speed-Up of 13.4 was obtained for a 2304 cell problem compared to 9.7 for a 256 cell problem.

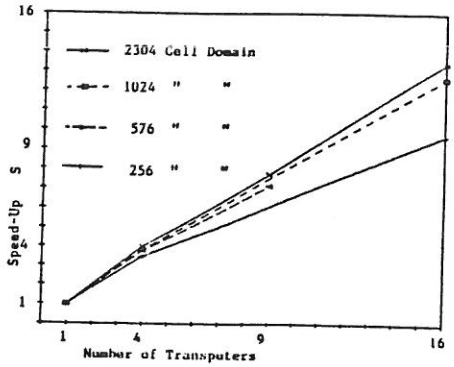


Figure 14 The Speed-Up S of Euler Solver against Number of Transputers for a given overall number of finite volume cells

In Figure 15 the Code Efficiency η is plotted for different numbers of Transputers and amounts of finite volume cells per Transputer. As expected, the smaller the number of cells allocated to an individual Transputer the lower the Code efficiency as the Transputer spends a proportionally greater amount of time communicating. For only one Transputer the Code efficiency is close to a 100% as the only communication which occurs is the passing of convergence information to the Host. As the Transputer array size increases, the code efficiency drops as the Transputers have to communicate nearest neighbour information. It is interesting to note that after 9 Transputers there is no appreciable decrease in Code Efficiency with 16 Transputers. The amount of time to carry out a global convergence check is small compared to that for nearest neighbour exchange. Therefore, the individual Code efficiency of a Transputer array will be determined by the slowest Guest process. In the case of 4 Transputers, all 4 have to exchange information across two edges, however, for 9 and 16 Transputer arrays the slowest processes are the interior Transputers which have to exchange information across all four edges. The Code-Efficiency and speed-up will be determined by the time-delay associated with communication across all four edges of a Guest process. This explains why there is no appreciable decrease in Code Efficiency between 9 and 16 Transputers. This allows predictions for the Speed-Up of larger Transputer arrays to be made as the individual Code efficiency is independent of the Transputer array size.

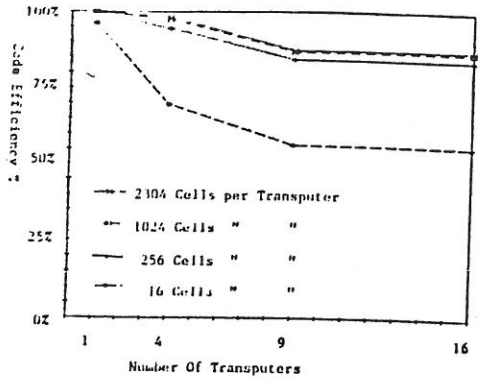


Figure 15. Code Efficiency ∩ against Number of Transputers for different numbers of Finite Volume cells per Transputer

A performance chart for a given number of finite-volume cells and the physical time to carry out a thousand time-steps on a specific number of Transputers is shown in Figure 16. An estimated plot for a 121 Transputer array is included. This is based on individual Transputers in a 16 and 121 Transputer array having the same Code Efficiency. For each size of Transputer array there is a linear relationship between the number of cells and the time to carry out a thousand iterations. Increasing the Transputer array size increases the speed of solution but each individual Transputer's code efficiency decreases. The minimum number of cells assigned to an individual Transputer is 16 so for a given array size there will be a minimum problem size. Correspondingly, the amount of memory assigned to a Transputer determines the maximum number of cells which can be allocated to a Transputer. For a given computational load per cell the total number of finite-volume cells and the amount of memory assigned to each individual Transputer will determine the optimum array size.

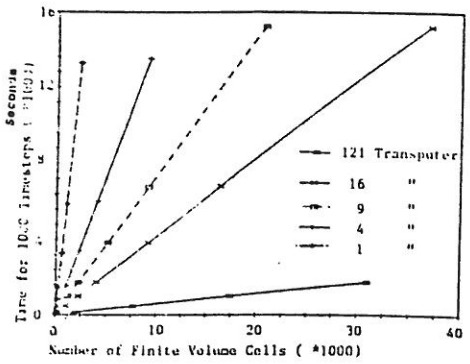


Figure 16 Performance Chart of Time to carry out 1000 iterations against the overall number of finite volume cells for different sizes of Transputer arrays

CONCLUSIONS

It was found that the implementation of a representative explicit numerical fluid dynamic algorithm onto an array of Transputers was a straightforward exercise.

A Communications Harness was written to carry out the necessary inter-Transputer communication and this greatly eased the development of the numerical algorithm. The development of the Harness also allows other C.F.D. algorithms to be easily implemented at a future date.

The explicit two-dimensional Euler Solver provided a good method of testing the performance of an array of Transputers. It was found that for Transputer array sizes greater than four the individual Code efficiency ∩ was independent of the Transputer array size. This implies that the time to converge should be inversely proportional to the number of Transputers which is as expected for geometric parallelism.

REFERENCES

(1) Jong, J-M., "A Comparison of Parallel Implementations of the Flux corrected Transport Algorithm", Master of Science Thesis, Utah State University, 1989.

(2) Inmos Ltd., "Transputer Development System", Prentice-Hall. 1988

(3) Hoare, C.A.R. ed., "OCCAM 2 Reference Manual", INMOS Ltd., Prentice-Hall, 1988.

(4) Ni, R.H., "A Multiple-Grid scheme for Solving the Euler Equations", A.I.A.A. Journal, VOL 20., No.11, Nov 1982.

(5) Ni, R.H., "Multigrid Convergence Acceleration Techniques for Explicit Euler Solvers and Applications to Navier-Stokes Calculations.", Von Karman Institute lecture notes 1986.

(6) Hall, M.G., "Cell-Vertex Multigrid Schemes For Solution of the Euler Equations", Royal Aircraft Establishment TM Aero 2029, H.M.S.O. 1985.

(7) Steger, J. and Sorenson, R. "Automatic Mesh-Point clustering Near a Boundary in Grid Generation with Elliptic Partial Differential equations", J. Computational Physics, Vol 33, pp 405-410 (1979).

(8).Jameson, A., "Solution of the Euler Equations for Two Dimensional Transonic Flow by a Multigrid Method", MAE Report No.1613, Princeton University, June 1983.