Image Cover Sheet

CLASSIFICATION	SYSTEM NUMBER
UNCLASSIFIED	505135

TITLE

DEVELOPMENT AND IMPLEMENTATION OF TASCFLOW3D FOR PARALLEL ARCHITECTURE

System Number:

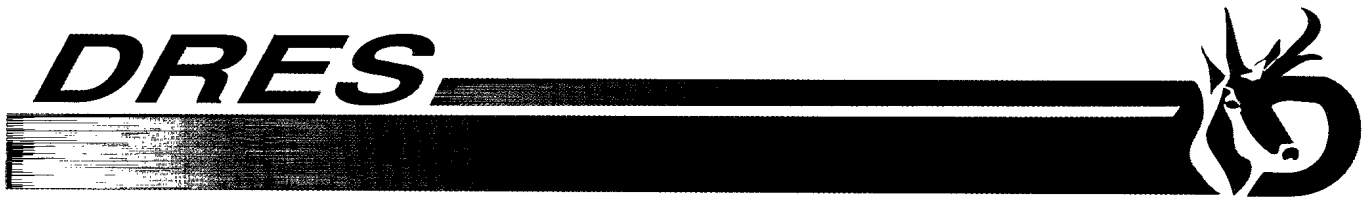
Patron Number:

Requester:

Notes:

DSIS Use only:

Deliver to:



DEFENCE RESEARCH ESTABLISHMENT SUFFIELD

CR 97-25

UNCLASSIFIED

Development and Implementation of TASCflow3D for Parallel Architecture *(Final Report)*

BY:

Advanced Scientific Computing Ltd.
Dr. Kevin J. Knill

SUBMITTED TO:
SCIENTIFIC AUTHORITY

D. Bergeron
Defence Research Establishment Suffield
Box 4000, Medicine Hat, Alberta, T1A 8K6

March 1997

CRAD

National Defence Défense nationale

Canada

Final Report

Development and Implementation of TASCflow3D for Parallel Architecture

Submitted by:
Advanced Scientific Computing Ltd.
554 Parkside Drive, Unit 4
Waterloo, Ontario, Canada
N2L 5Z4

Project Manager: Dr. Kevin J. Knill

Submitted to:
Defence Research Establishment Suffield
Administrative Authority: Mr. Noel Thomas
Scientific Authority: Dr. Denis Bergeron

Contract Number: W7702-6-R587/01/EDM

March, 1997

~~DRAFT~~

Final Report

Development and Implementation of TASCflow3D for Parallel Architecture

Submitted by:
Advanced Scientific Computing Ltd.
554 Parkside Drive, Unit 4
Waterloo, Ontario, Canada
N2L 5Z4

Project Manager: Dr. Kevin J. Knill

Submitted to:
Defence Research Establishment Suffield
Administrative Authority: Mr. Noel Thomas
Scientific Authority: Dr. Denis Bergeron

Contract Number: W7702-6-R587/01/EDM

March, 1997

~~DRAFT~~

Contents

1 Introduction

The Defence Research Establishment makes extensive use of the TASCflow computational fluids software (CFD) package from Advanced Scientific Computing to support research programs in Military Engineering and Bio-Chemical Hazard assessment. CFD can be computationally intensive when large integrated 3D problems are calculated. For this reason, DRES acquired a DEC 8400 multi-processor workstation with 10 CPU's and 4 GB of memory. This machine is intended to allow problems to be computed in parallel on several processors to speed up the overall performance of the computations.

DRES has contracted ASC to develop and apply TASCflow3D in a parallel environment under contract number W7702-6-R587/01/EDM. The parallel version of TASCflow was intended to be installed on the DEC 8400 machine. However, near the end of the contract, a request was made by the Scientific Authority to transfer the installation from the DEC 8400 machine at DRES to the DREV site. DREV utilizes a cluster of four Sun Ultra Sparc machines, linked together by a common network. Thus, this development has been adjusted to accommodate this installation change.

This report describes the parallelization algorithm and the manner in which it has been applied on a cluster of workstations. Expected performance on both multiprocessor workstations and a workstation cluster are postulated.

2 Design

Parallelization involves the concurrent execution of independent instructions within an algorithm. When applied to computational fluid dynamics, parallelization is intended to break a large integrated problem into smaller separate processes which can then be calculated using current computer chip technology. The connections between the independent processes are handled by a communication library which controls the transfer of data, recovering the single process execution stream. The optimum parallelization algorithm minimizes CPU time and memory while limiting the communication between processes.

Several key criteria have been identified in designing the parallelization algorithm for TASCflow:

- TASCflow needs to run parallel on both multiple processor, shared memory machines as well as on a cluster of workstations.
- The parallelization procedure must run without degradation in solver performance, robustness or efficiency.
- Algorithmic changes to the code should be minimized.
- The algorithm must operate efficiently on at least four processors.

- Communication routines should be encapsulated so as to allow alternative parallelization implementations with minimum effort (i.e. MPI and hardware specific).

These criteria helped to determine the approach taken in parallelizing TASCflow as described below.

3 Approach

A parallelization tool called PVM (Parallel Virtual Machine) is used in developing the parallel TASCflow. PVM was chosen because of its established history and demonstrated performance as well as support on a wide variety of machines and operating system platforms. There is a large network of users who have applied this tool providing opportunities to interact through user conferences and the internet. PVM is not the only parallelization tool and alternatives may be preferred in future. For this reason, the PVM routines have been encapsulated to allow alternative communication libraries to be implemented.

The parallelized feature is a designated capability of TASCflow Version 2.7. Operating from within the graphical user interface, when the user is ready to begin execution of the TASCflow solver, a solver monitor panel opens as shown in Fig. 1. This panel includes a graphical representation of convergence behaviour as well as a running log of the solution convergence (`TRO` file). At the top of the panel there is a section with widgets to control the TASCflow executable, including whether the execution will proceed in parallel mode.

If the user selects the parallel option, the Host Assignments panel, shown in Fig. 2 opens. From here, the user builds a Host list which identifies the CPU's available for the run. The available CPU's are contained in a database, established during the installation of the code. This database may contain independent workstations connected through a network, or a number of CPU's on a single multiprocessor workstation. Each user may establish their own list of hosts which is controlled by this panel. After finishing the host list, the user assigns each of the grid blocks to one of the available hosts. This may be done automatically for the user in which case the blocks will be distributed uniformly among the available hosts. The user may also selectively assign grid blocks to hosts. Diagnostic information is also available to show the fraction of grid assigned to each of the hosts.

It is important for the user to balance the load between each of the processors. To aid in this function, a grid manipulation tool is available to re-block the grid.

Once the CPU's are selected, execution may be initiated from the solver monitor panel, or directly from the command line.

3.1 The parallel processing algorithm

As shown in Fig. 3, parallel TASCflow has been structured on a master/slave process relationship. The master process controls the input/output, including start-up, file handling and output of information and result files. The master process also assesses convergence, handles all diagnostic checks and error handling. Once parallel execution is initiated, the master process starts, reads the grid, boundary conditions and restart information, spawns each of the slave processes and distributes all the information that each of the spawned process requires to that process. Processes are synchronized at interprocess communication points in the flow code. Information is required from other processes during a particular spawned processes' execution and the process will wait until it receives that information before proceeding with it's execution.

Each slave process performs several functions associated with the partition associated with that process:

1. Discretization: transport equations are discretized as though the partitions are independent. At boundaries to other partitions, information is passed between the processes.
2. Solver: the solver performs a sweep of the equations assembled in Step 1. If multigrid is used (default), restriction of the equations to a first coarse grid is performed.
3. Multigrid: the corrections associated with the restriction process are sent back to the master process. The slave process then receives back corrections from the master process which are then, in turn, prolongated onto the fine grid.

If multigrid is applied, the master process also has specific functions to perform at each time step:

1. Receive restriction corrections from each slave.
2. Execute the Algebraic multigrid solver to calculate corrections to the fine grid equations.
3. Once the multi-grid solver is converged, the corrections are passed back to each of the fine grid equations on the slave processes.

This technique ensures that the parallelization algorithm suffers no degradation of performance, robustness or accuracy compared to the current serial calculation. From the user perspective, there will be no difference between a serial and parallel execution with respect to their interaction with the master process.

There may be additional memory required when running the algorithm in parallel. Each slave process will run as if it were an independent problem and therefore, will require no additional memory. However, the master process will require up to 50 words per node. Most of this memory is required to start-up a problem and to run the multigrid solver.

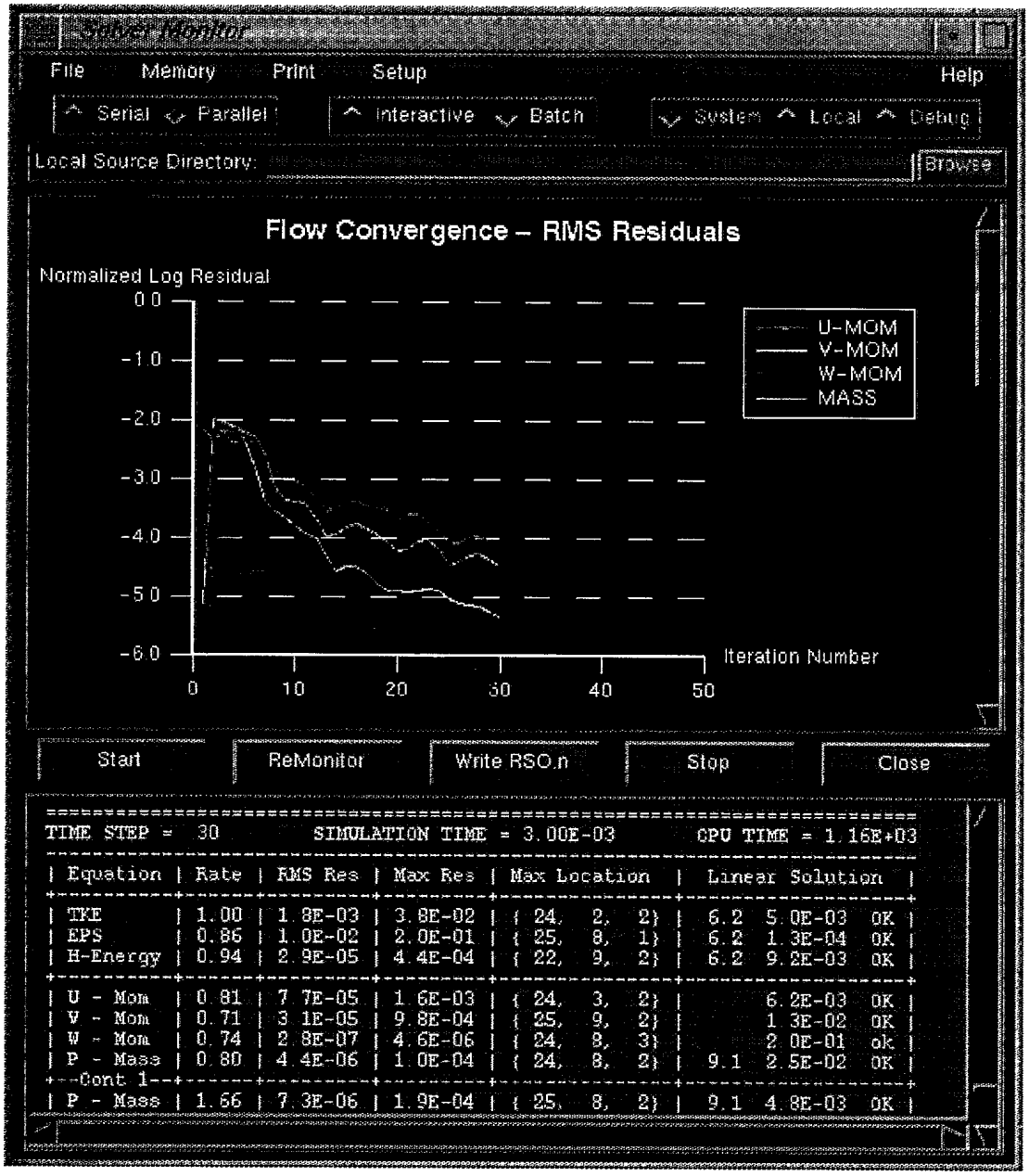


Figure 1: Solver monitor panel in TASCflow

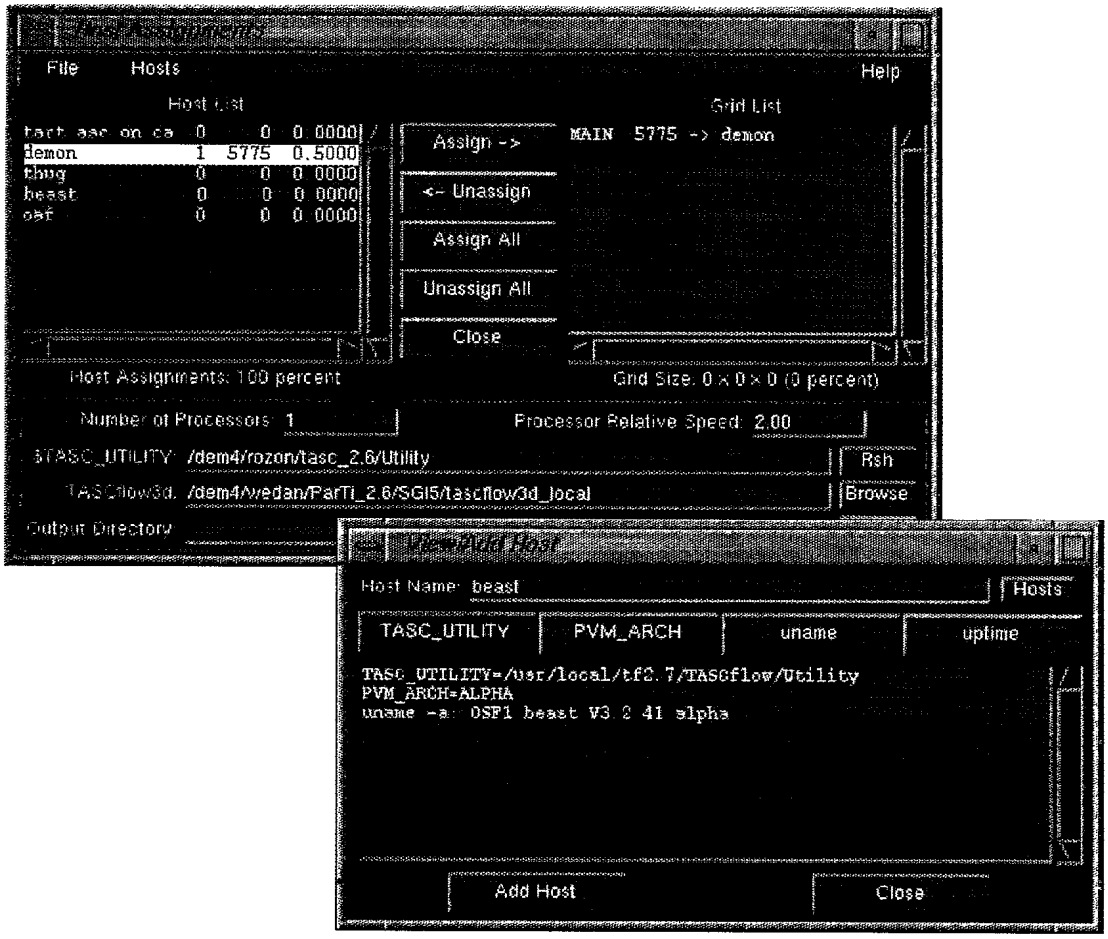


Figure 2: Host Assignments panel

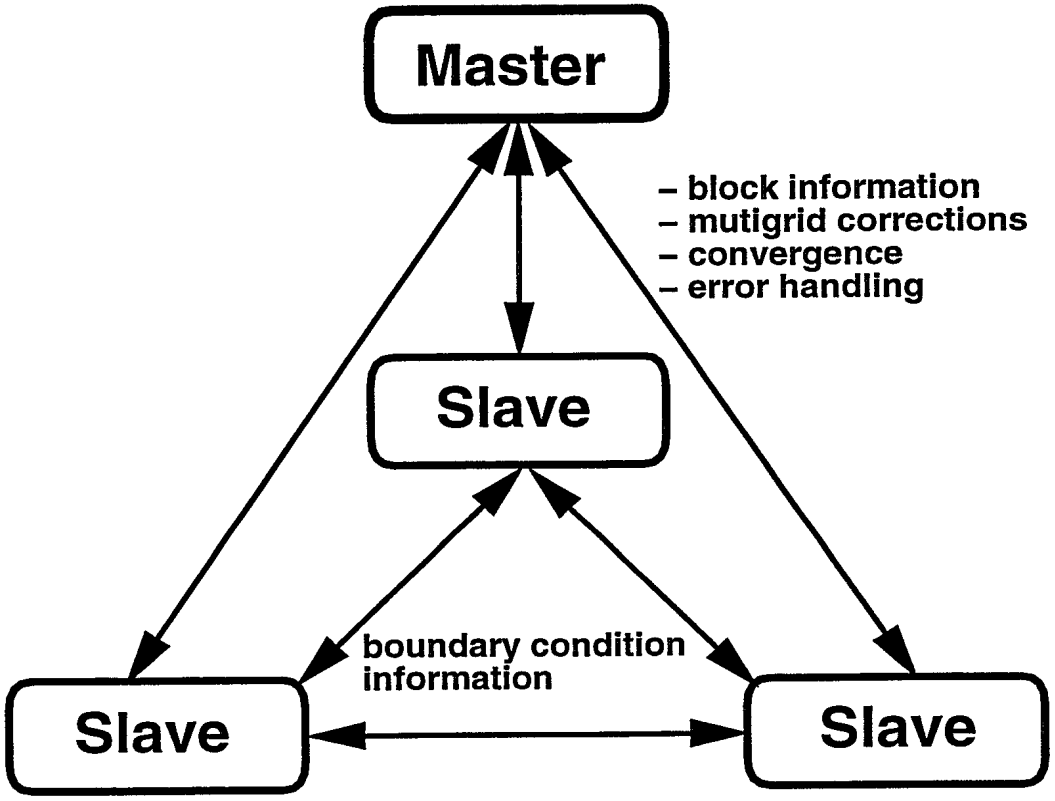


Figure 3: Master slave relationship in the parallel algorithm

3.2 Grid Partitioning tool

As has been discussed previously, when TASCflow3D executes in parallel mode on a given problem, each grid block in the problem domain is executed as a separate process, a slave process. In order to maximize the efficiency of your parallel computation for your parallel computer network, there is an optimal arrangement of grid blocks in terms of the number of blocks and size of each block. Ideally,

1. The number of grid blocks should equal the number of computer CPU's in the parallel network.
2. The number of nodes in each grid block should be proportional to the relative speed of the processor.

If the number of blocks is larger than the number of computers, then one computer will have to run more than one slave process, leading to inefficiencies. If the number of blocks is smaller than the number of computers, then one or more computers in the network cannot be used for the parallel computation.

Regarding the balancing of the number of nodes on each processor, consider the situation where you have two computers, one twice as fast as the other. Ideally you want the grid blocks arranged so that there are a total of two grid blocks, (one for each computer), with one grid block twice as big as the other (the faster computer runs the bigger block). In this way, the two parallel processes proceed through their respective portions of the domain at the same rate, and finish the same task in approximately the same wall clock time. This *load balancing* is critical for overall efficiency of the parallel computation, since there are a number of times in the computation when all slave processes must synchronize. If one processor is overloaded relative to a second processor, the second processor will have to idle and wait for the overloaded processor to catch up to a give synchronization point. This idle time translates into longer wall clock time for the computation and hence inefficiency.

The problem is addressed by use of a grid partitioning tool. This tool essentially transforms an existing multi-block grid, consisting of any number and size of grid blocks, into a specified number of grid blocks, each of a specific portion of nodes per grid block. The grid partitioning tool is intended as a pre-processing tool for the grid (GRD) file. It does not convert any other file or dataset during the process of changing the grid block topology (e.g. does not convert boundary condition files or solution files). The partitioner should be thought of as a grid file converter, and used as such (i.e. used once the desired grid has been completed, but before applying block-off zones and boundary conditions).

The partitioner asks a series of questions to determine how you would like the grid partitioned, including how many grid blocks and what the relative performance (hence sizes) should be assumed for each grid block. The process of re-partitioning the grid into new grid blocks is then automatic. The resulting new grid topology will have the desired number of grid blocks, independent of how many grid blocks the original grid system had. In addition,

the number of nodes in each grid block will be proportional to the relative speeds assigned for each processor that is anticipated to be used for each grid block.

4 Performance

The maximum theoretical speed up due to parallelization is linearly proportional to the number of processors. Assuming no degradation due to inter-process communication, the speed-up factor would be equal to the number of processors. Because the multigrid is conducted on the master process, the actual maximum speed-up for n processors may be expressed in equation form as:

$$S_u = \frac{1}{f_s f_{sc} + \frac{f_s(1-f_{sc})+(1-f_s)}{N_{\rm pr}}} \tag{1}$$

where f_s is the fraction of execution time in the solver, f_{sc} is the fraction of solver time spent in the coarse multigrid and $N_{\rm pr}$ is the number of processors. The first term in the denominator represents the fraction of time spent by the Master process and the second term represents the fraction of time spent by slave process.

The efficiency is given by:

$$\eta = \frac{S_u}{N_{\rm pr}} = \frac{1}{f_s f_{sc} N_{\rm pr} + f_s(1-f_{sc}) + (1-f_s)} \tag{2}$$

The speed up, S_u as a function of the number of processors is presented in Fig. 4 for a variety of cases. The speed up estimated is that which would be obtained if there is no communication overhead whatsoever and the processes are evenly loaded. In other words, the flow code will perform below the curve defined by Eqn. 1.

To determine the potential performance increase which can be achieved through parallelization, Eqn. 1, can be considered in terms of known solver performance. The fraction of time spent in the solver, f_s, is approximately 30-50% for a wide variety of problems. The fraction of solver time spent in the coarse multigrid, f_{sc}, is 40%. Substitution of $f_s = 0.3$ and $f_{sc} = 0.4$ into Eqn. 1 results in the lower curve of Fig. 4. Speed-up increases from 1.8 for two processors to 2.8 for four processors. Because of the serial performance of the coarse multigrid solver, this curve begins to flatten out, meaning little increase in performance for greater than four processors. As shown in Fig. 5, the efficiency is 70% for four processors, increasing to 90% when two processors are utilized.

The limitation in parallelization performance is due almost entirely to the serial multigrid. Assuming that the first level of multigrid could be conducted on the slave processors, the fraction of solver time spent in the coarse multigrid would decrease. This would be approximately equivalent to decreasing f_{sc} from 0.4 to 0.1. As shown in Fig. 4 as the middle

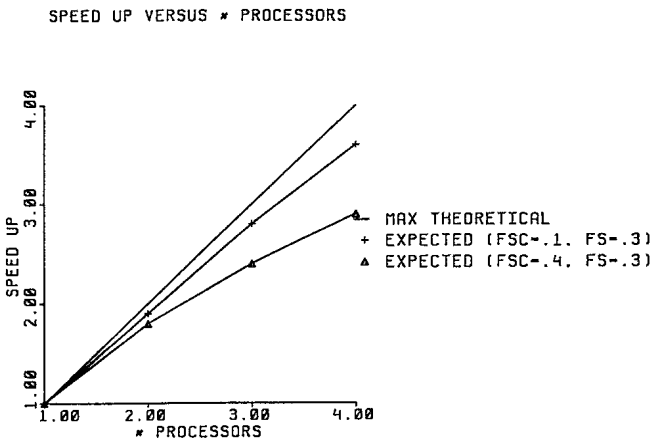


Figure 4: Parallelization Speed Up

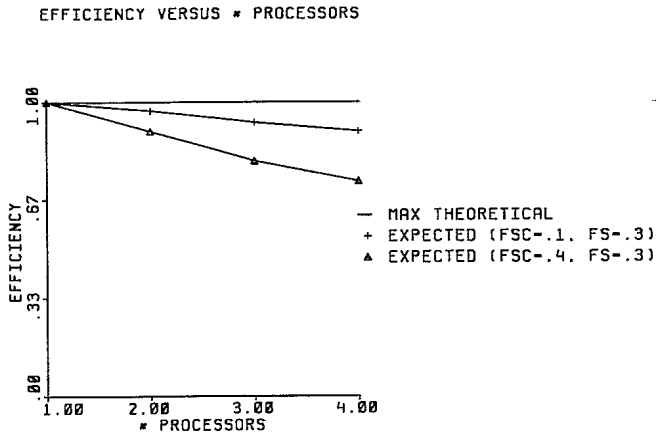


Figure 5: Parallelization Efficiency

curve, the theoretical speed-up would be 3.6 on four processors representing an efficiency of 90%. Thus, the need to parallelize the multigrid solver is apparent should it be necessary to operate in parallel on more than four processors.

The clock time for the run scales inversely with the number of processors. The master process requires approximately $1/5$ *th* of the CPU time so that the performance enhancement is theoretically limited to 5 processors. In practice, the actual limit will be fewer processors and will depend on load balance and communication. On a multiple processor machine, communication should be reasonably fast and would probably not exceed 5-20% of the total CPU time. Thus, we would consider that the upper limit on processors would be approximately 4 with an 80-85% utilization of the slave CPU power for computations.

On a network cluster of workstations, the communication will become more significant. Again, assuming that the load on each CPU is evenly balanced, the parallelization performance is estimated to be limited to 3 processors with a 70% CPU utilization efficiency.

4.1 Duct Test Case Performance

A turbulent duct case was setup and solved using the serial and parallel versions of TASCflow on the 10 processor DEC 8400 at DRES. Table 1 presents a summary of the 4 block grid that was used for the simulation. A parallel process is spawned for each of the grid blocks in the domain and a process is started for the Master. Therefore, there was a total of five TASCflow processes running for the parallel simulation of the duct flow.

Partition	Process	(ID:JD:KD)	Nodes	Node Fraction
MAIN	1	(25:15:15)	5625	.25
_B	2	(25:15:15)	5625	.25
_C	3	(25:15:15)	5625	.25
_D	4	(25:15:15)	5625	.25

Table 1: DUCT: Testcase Partition Node Summary

The work required for each spawned process is a directly proportional to the number of nodes on each grid block or partition. The maximum theoretical speed-up, due to parallelization, can be estimated from Fig. 4 for four processors as 2.9. The actual speed-up will be less than this due to communication and network overhead.

Table 2 presents the timing summary of the 4 partition or grid block duct test case. The wall clock speed-up experienced on the multi-processor machine at DRES was approximately 2.2 times the serial simulation time. This compares well with the theoretical maximum of 2.9 times.

Partition	Process	Wall Clock (s)	CPU (s)	Speed Up	
				Wall	CPU
Serial		431	427		
Master	0	193	74.9	2.2	2.5
MAIN	1	180	94.7		
_B	2	180	97.9		
_C	3	180	97.9		
_D	4	180	95.1		

Table 2: DUCT: Multigrid Run Time Summary

4.2 Cone Cylinder Flare Test Case Performance

A Mach 4 cone cylinder flare test case was setup and solved using the serial and parallel versions of TASCflow on cluster of three SGI R5000 computers. Table 3 presents a summary of the 3 block grid that was used for the simulation.

Partition	Process	(ID:JD:KD)	Nodes	Node Fraction
MAIN	1	(101:31:2)	6262	.33
GRD_1	2	(101:31:2)	6262	.33
GRD_2	3	(101:31:2)	6262	.33

Table 3: Cone Cylinder Flare: Testcase Partition Node Summary

The maximum theoretical speed-up, due to parallelization, can be estimated from Fig. 4 for three processors as 2.4. The actual speed-up will be less than this due to communication and network overhead.

Table 4 presents the timing summary of the 3 partition grid block cone-cylinder-flare test case. The CPU speed-up experienced on the SGI cluster at ASC was approximately 1.88 times the serial simulation time. This compares well with the theoretical maximum of 2.4 times.

5 Conclusions

TASCflow Version 2.7 has been parallelized using the PVM tool. This allows TASCflow to run on a multiple processor machine and on a cluster of workstations. A separate process is initiated for each subgrid. The subgrids can be automatically generated by using a grid partitioning tool.

Substantial speed-up over serial operation is obtained when 2-4 processors are utilized. A

Partition	Process	CPU (s)	Speed Up CPU
Serial		2471.0	
Master	0	232.6	1.88
MAIN	1	1081.2	
GRD_1	2	1038.1	
GRD_2	3	1149.1	

Table 4: Cone Cylinder Flare: Multigrid Run Time Summary

benchmark low speed test case, run on four processors of the DEC 8400 machine at DRES indicated a speed-up of 2.2 over serial wall clock time. A benchmark high speed flow test case run on a cluster of three SGI workstations at ASC demonstrated a speed-up of 1.88 over the serial run.

Given the current parallelization algorithm, the number of processors which can be efficiently utilized is limited to four. This limitation is attributable to the serial operation of the multigrid algorithm. Further work would focus on parallelizing the multigrid algorithm to increase the number of processors which can be effectively utilized.

#505135