DEVELOPMENT OF A PARALLEL EXPLICIT CFD CODE FOR THREE-DIMENSIONAL FLOW SIMULATIONS OVER COMPLEX CONFIGURATIONS

Jose Nilton C. Gil, jniltongil@gmail.com
UBM – Centro Universitário de Barra Mansa, Rua Vereador Pinho de Carvalho, 267, Centro
27330-550, Barra Mansa, Rio de Janeiro, Brazil
Edson Basso, basso@iae.cta.br
CTA/IAE/ALA – Instituto de Aeronáutica e Espaço, Comando Geral de Tecnologia Aeroespacial
12228-903, São José dos Campos, São Paulo, Brazil
Nei Yoshihiro Soma, soma@ita.br
CTA/ITA/IEC – Instituto Tecnológico de Aeronautica, Comando Geral de Tecnologia Aeroespacial
12228-900, São José dos Campos, São Paulo, Brazil
João Luiz F. Azevedo, azevedo@iae.cta.br
CTA/IAE/ALA – Instituto de Aeronáutica e Espaço, Comando Geral de Tecnologia Aeroespacial
12228-903, São José dos Campos, São Paulo, Brazil

Abstract. *The need for numerical simulations of fluid flows has grown recently, both in industry and academic centers. Many CFD codes are developed for a large range of applications, including aerodynamic flows over aerospace configurations. The numerical treatment of these flows typically involves the definition of a computational domain that is adequately discretized through refined meshes. As the need of computational resources for more complex simulations is above the serial computer processing capability, the parallelization of such codes is of great importance. For the partitioning procedure of the numerical meshes and for the cluster node communication, the METIS and MPI (Message Passing Interface) libraries are used in a shared memory architecture. For the purpose of verifying the efficiency of the parallelization procedure, the CPU usage is monitored during the parallel execution of the code using the MPE libraries. Several tests of typical aerospace geometries are performed and the results showed the performance of the different aspects on the node communication procedures, allowing for an evaluation of the scalability of the present implementation.*

Keywords*: Parallelization, CFD, computer cluster, MPI*

1. INTRODUCTION

The necessary calculations for simulating fluid flow behavior on tridimensional surfaces normally involve techniques of computational fluid mechanics along with the adaptation of tridimensional meshes. These meshes are usually unstructured and need a significant number of nodes and volumes in cases of real applications. This causes the simulation to become unviable in most available computers, due to both the amount of time required for processing and the need for available memory. This paper describes the first adaptations of a program conceived to solve the problem using the method of finite volumes with Navier-Stokes equations. The program is used in a cluster through the partitioning of an unstructured mesh, where each node processes each piece of mesh and, by using messaging protocols, exchange information with other nodes. These nodes simultaneously process other mesh parts that border the latter so as to mold the obtained values and reproduce the complete solution to the problem. The approach presented in this work only sends and receives messages with data structured into vectors, not yet contemplating the refined mesh in the most critical regions. At the end of this paper, the results obtained with two cases that use different sized meshes are shown with the processing time consumed with the exchange of messages and its influence in the total time taken for calculating the solution.

2. EXISTING PROGRAM WITH SERIAL SOLUTION

The program called BRU3D developed in FORTRAN, which is the topic of a Master's dissertation authored by Scalabrin (2002), uses data files to retrieve data concerning the initial parameters of the system and the unstructured mesh, such as the number of nodes, number of volumes, type of contour surface, and number of nodes that compose the surfaces bordering the contour volumes. The program is structured into many procedures that deal with formatting and stating variables, recording files for TecPlot visualization (www.tecplot.com), refining mesh and volume, and calculating both the Sparlat and Allmaras' (1992) turbulence model and the scattered matrixes that are used in adapted routines published in Press (2002). The program basically operates by reading files with initial conditions, mesh data and connectivity tables, storing nodes, mesh volume and contour volumes in matrixes, calculating the residue based on a five-stage Runge-Kutta, and checking for conversion or digression. If the predetermined levels are not yet reached, it goes a step further in simulation, verifying the turbulence model and updating the solution applying the contour conditions. When reaching the number of predicted iterations or when calculating a smaller residue than the specified minimum, it records the solution, generates a visualization file for TecPlot, and records a file containing values ρ(mass), ρ*u, ρ*v and ρ*w (*momentum* vectors) and the energy value.

3. CHANGES IN THE PARALLEL SOLUTION PROGRAM

In order to implement a parallel solution for the problem, we chose to use a pre-processing system for the unstructured mesh to be partitioned, which allows the simultaneous processing of each part in a different node of the cluster. We also opted for using the MPI protocol for exchanging messages between the nodes, also employing the MPE library for generating a log file capable of measuring the time consumed to send and receive messages and the actual processing time.

3.1 Mesh partitioning

For the mesh to be partitioned into parts that can be processed separately in each node, we developed a program that is able to generate new files with renumbered elements from the initial files fort.2, fort.3 and fort.5 and have the original name plus a numerical suffix that represents the processing node that will be used. The kmetis program was used in order to optimize unstructured mesh partitioning, generating equal-sized partitions, within a program that is responsible for pre-processing information. Therefore, if we break up the mesh into three parts, the fort.2 file, which is responsible for the node information, will be divided into three new files, named fort.2.0, fort.2.1 and fort.2.2. Similarly, the fort.3 and fort.5 files will also be partitioned, generating new files named according to the same criterion. Thus, the processor identified as 0 processes the files exhibiting the numerical suffix .0, the suffix .1 by the process identified as 1, and so forth. In addition to files fort.2, 3 and 5, other files are created containing intercepting surfaces between the mesh partitions so as to facilitate and optimize message exchanges between partitions, named common.0, common.1 and so forth, depending on the number of parts into which the mesh was divided.

3.2 MPI – Message Passing Interface

The MPI - Message Passing Interface (http://www.mcs.anl.gov/research/projects/mpi/ standard.html) is used by many unshared memory parallel systems. Although many commands are able to facilitate the implementation of programs that employ message exchanges, it is possible, with only six basic commands, to use the protocol in relatively complex programs (Gropp *et al*, 1998). For implementing programs using the MPI, the MPD daemon must be working in every node used for processing and the machines and processors available in each machine must be configured through the *mpd.hosts* file. If we observe the programs that use MPI, we will see that it is always necessary to start with the *MPI_INIT(ierr)* command and end with the *MPI_FINALIZE(ierr)* command. It is extremely important that the number of processes that were partitioned, the processing, and the identification of each process in execution are detected. In order to do so, the command *MPI_COMM_RANK(...)* returns the process identification within the set of processes involved. To evaluate the total number of processes, the *MPI_COMM_SIZE(...)* command is used. In addition to these commands, we must send and receive data among the processing nodes involved in commands MPI_SEND(...) and MPI_RECV(...). The basic syntax of these commands involves knowledge about the data being sent or received, data type, and the communicator that is being used.

3.3 Including MPI commands

The inclusion of commands for initializing, terminating, sending and receiving messages in this first version aimed to achieve acceptable results, whose values are very similar to the serial case. At this point, we are not yet worried with performance optimization due to processing distribution or memory use. Thus, the line *includes 'mpif.h'*, which is the necessary MPI library for executing the protocol in FORTRAN programs. The program that is modified for distributed functioning requires each processing node to have a set of data files that contain information about the mesh domain that will be processed in that node, in addition to common files that will be replicated to all the nodes containing parameters such as stop conditions, flow direction and speed, type of flow, Reynolds number, and time increment.

```
call MPI_INIT( ierr )
call MPI_COMM_RANK( MPI_COMM_WORLD, myid, ierr )
call MPI_COMM_SIZE( MPI_COMM_WORLD, numprocs, ierr)
```

Figure 1 – Basic commands for MPI initialization

Figure 1 shows the MPI commands allocated for initialization, the identification of process numbers stored in the variable myid, and the total number of processes stored in numprocs involved in the system. In the interboundary routine, after each iteration, the mesh boundary surface values are transferred to other mesh partitions until all surfaces are used.

```
call MPI_SEND(vetor_troca, ntamanho, MPI_DOUBLE_PRECISION,nother(k) , 0 ,
                MPI_COMM_WORLD,ierr)
call MPI_RECV(vetor_troca, ntamanho, MPI_DOUBLE_PRECISION, nother(k) , 0,
                MPI_COMM_WORLD,istatus,ierr)
```

Figure 2 – MPI commands responsible for sending and receiving surface data

Figure 2 shows the code lines responsible for sending and receiving surface data, stored in the variable *vector_exchange*, according to the hierarchy of process identification. That is, if the process rank is smaller, it first sends and then receives data.

The decision of stopping the processing involves the residual values. In case of distributed processing, the evaluation of residual values obtained in each node is necessary. By comparing all residues, we make the decision considering the largest of all.

```
If (myid .ne. master) then call MPI_SEND(aux1,1,     MPI_DOUBLE_PRECISION, 0 , 0 ,
   MPI_COMM_WORLD,ierr)
else
   do n = 1,numprocs
     call MPI_RECV(aux_o,1,MPI_DOUBLE_PRECISION, n , 0 , MPI_COMM_WORLD,istatus,ierr)
     if(aux_o .gt. aux1) aux1 = aux_o
   enddo
…
```

Figure 3 – Logic used for evaluating maximum residue

Figure 3 shows that, in case the node is not the master node, it must send the residual value, calculated and stored in variable *aux1*, to the master node, which is responsible for receiving information from each processing node for the evaluation of maximum residue. In order to calculate the residue of each part of the processed mesh, the five-stage Runge-Kutta method is used, and in each stage, the boundary routine is invoked, which is responsible for updating contour volume information between mesh partitions and between the mesh and the external means. If the maximum residue is still out of the specified range in file *fort.1*, and still converges towards a solution, information is exchanged between boundary surfaces in each iteration and the new maximum residue starts being evaluated. The procedure repeats itself until the program stops and reaches the minimum required residue, diverges to values that are above the specified values, or reaches the number of parameterized iterations in file *fort.1.*

3.4 Generating log files

In order to evaluate the amount of time consumed to send and receive messages, the line *include mpef.h* was included. This automatically generates a file that is graphically visualized by the Java application jumpshot.jar, which is part of the MPICH2 package. Since the cluster does not possess a graphic interface, the log files generated in each processing node were transferred to a machine that executes an operational system with a graphical interface. Thus it is possible to visualize the log file that reports when the node was processing the meshes and when and how it executed data communication between nodes.

4. CASE STUDY

In order to verify the performance of the modified program, the consistency of the results, and the influence of communication time within the total processing time, two simple meshes were tested: a very small 13328-volume mesh and another medium-sized 360434-volume mesh.

For the case study, we used a computational cluster with 16 nodes with 2 processors AMD Opteron on each node using the operating system Gentoo Linux.

4.1 Volume Mesh of 13328

By counting the total processing time using a small mesh with 27200 nodes and 13328 volumes, tests were conducted using the program in the original (serial) and modified version, which varies from 2 to 7 processing nodes. Figure 4 shows the time consumed for 1000 iterations.

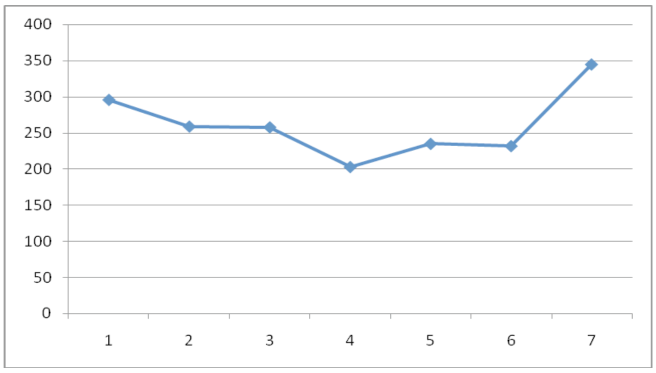


Figure 4 – Processing time due to the number of processors in use

In figure 4, the horizontal axis represents the number of nodes used and the vertical axis represents the time in seconds. The processing time obtained by the serial solution was larger than most cases of parallel processing. With two and three processing nodes, there was no significant time gain in relation to the serial solution. However, with four active nodes, the minimum amount of time was reached for this case. We observe that, from then on, the influence of communication time consumed with message exchanges compensated the time saved with processing significantly smaller mesh partitions, simultaneous in each node.

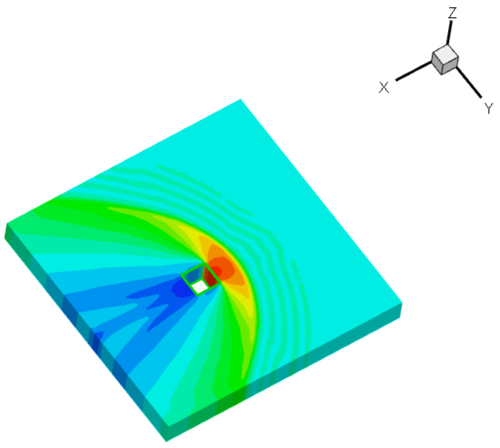
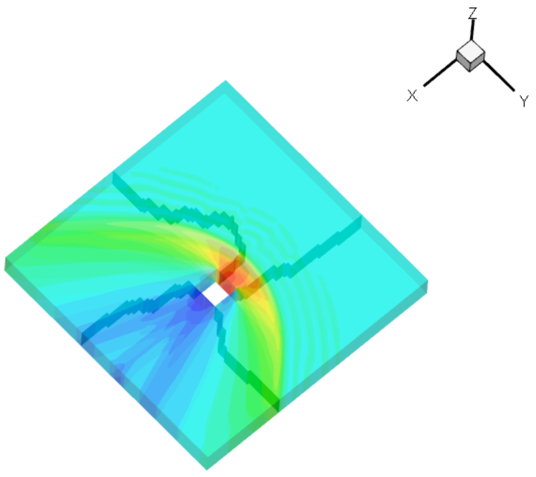


Figure 5 – Result of simulation using the serial program

Figure 5 shows the result obtained after 1000 iterations using the serial version of the program. One must observe that the figure shows the behavior of the flow applied the x axis direction, and its effect when it comes across a square bar in the middle.

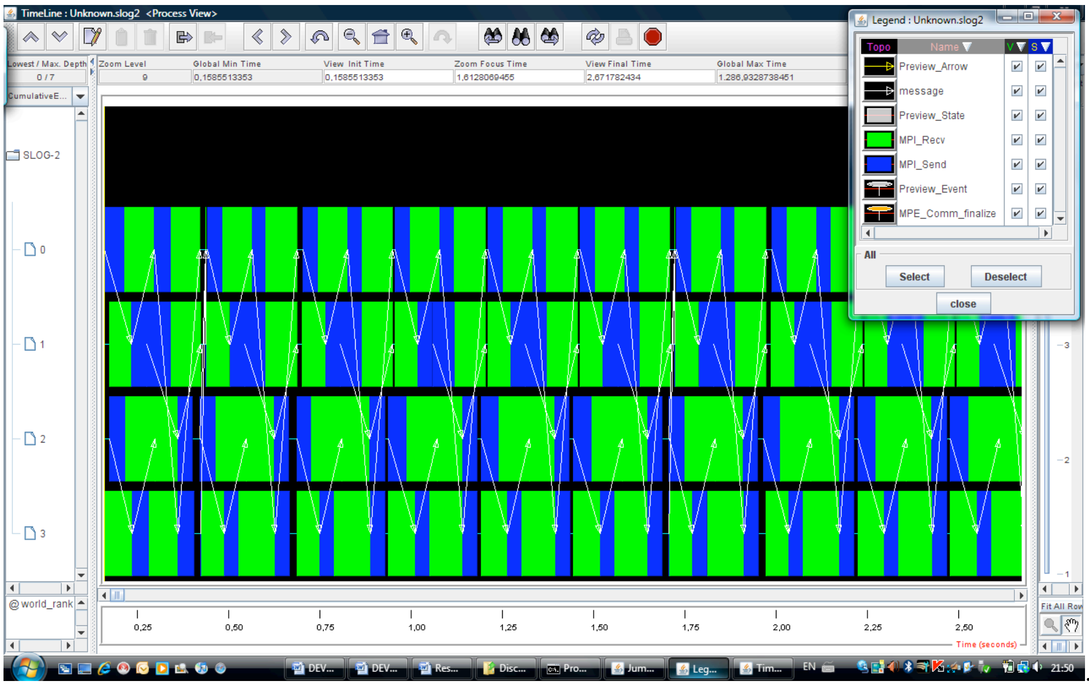


...

Figure 6 – Result of simulation with a four-node parallel program

Figure 6 shows the mesh divided into 4 parts. Slightly different colorations are observed in figure 6 in relation to figure 5 due to the illuminating effect, which is necessary for enhancing the boundaries of mesh partitions.

In order to better assess the influence of the communication between processing nodes in the results, libraries were included for generating logs related to the sending and receiving of data.


TimeLine : Unknown.slog2 <Process View>

Lowest / Max. Depth	Zoom Level	Global Min Time	View Init Time	Zoom Focus Time	View Final Time	Global Max Time
0 / 7	9	0,1585513353	0,1585513353	1,6128069455	2,671782434	1.286,9328738451

....

Figure 7 – Depiction of the communication between processors obtained through MPE

Figure 7 shows the sending and receiving of data during processing distributed into 4 nodes. It is possible to observe the disparity of time spent with communication compared to the time spent in processing, both which obviously occur between communication blocks. The vertical axis shows nodes identified by a ID value and the horizontal axis shows time in seconds. Arrows indicate communications and its directions, as they occur between processors.

From the label screen, overlapping the top right-hand corner of the figure, we can observe the moments when data is sent and received between the nodes, as well as to which node the data is sent or received. In order to expressively observe the influence of time spent, it was necessary to use a mesh with a more significant size.

4.2 Volume Mesh of 360434

When migrating the tests for a 360434-volume mesh, about ten times larger than the previous one, we observed that the influence of communication in the total processing time was very large. This caused the time using 2 nodes to be minimum compared with the time observed with 3,4,5,6 or 7 nodes. In this case, there was no execution of the serial program because it was impossible to execute with the available equipment.

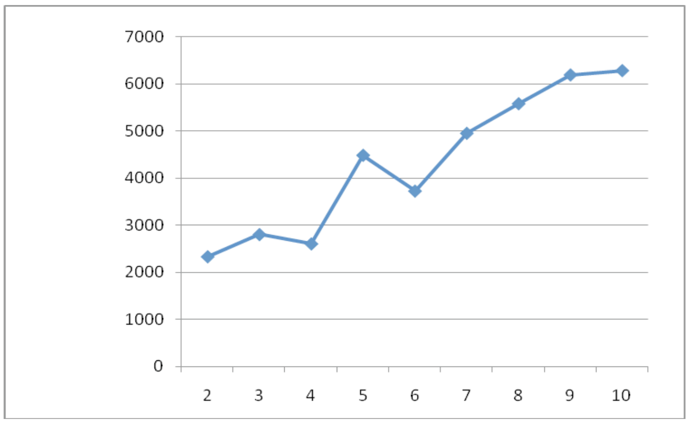


Figure 8 – Processing time according to the number of processors

Figure 8 shows the total time in seconds across the vertical axis and the number of nodes used for processing across the horizontal axis. The discrepancies observed in nodes 3 and 5 may be attributed to other programs that were consuming resources from the cluster during the tests, but there is a clear tendency to increase processing time due to the number of nodes used.

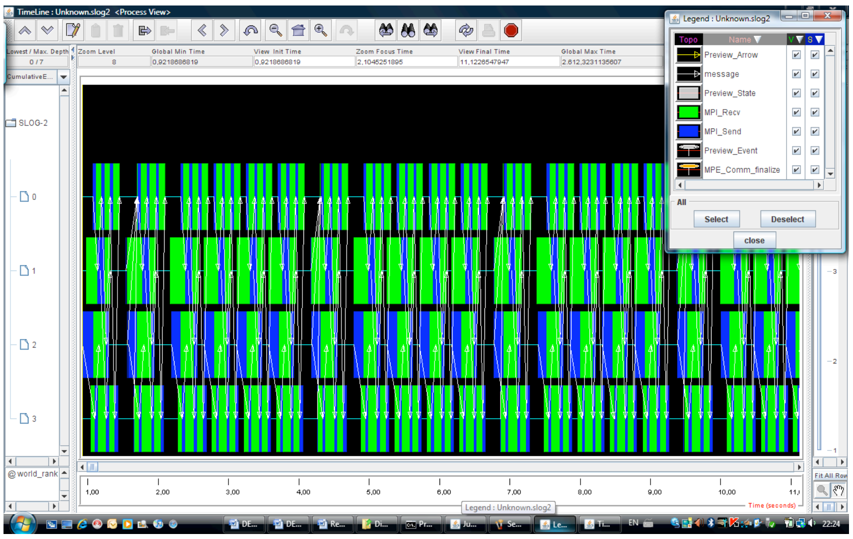


Figure 9 – Communication flow between nodes using four-part mesh partitioning

Figure 9 shows the communication between four processing nodes and, once again, it is obvious that the total time spent with communication is substantially greater than the time spent with processing. It is worth noting that there are cycles composed of a quick communication between nodes, followed by five communication blocks that repeat themselves until the end of the processing. This can be explained by the exchange of messages needed for evaluating the largest residue found followed by the process involving the five-stage Runge-Kutta method, where the interboundary routine is invoked between and at the end of each stage.

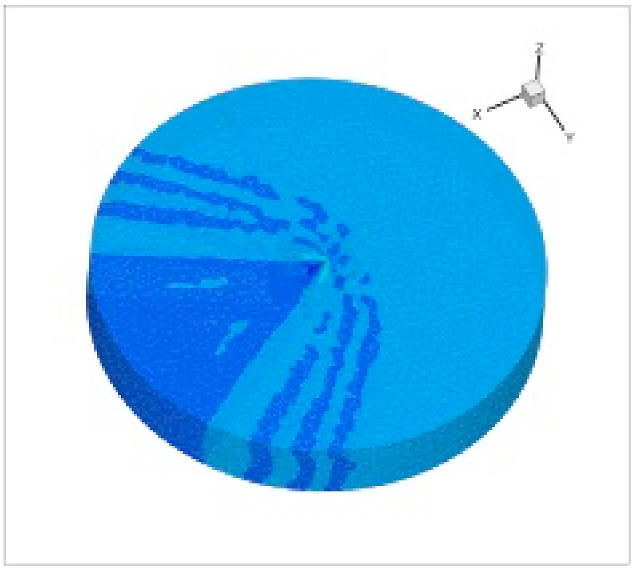
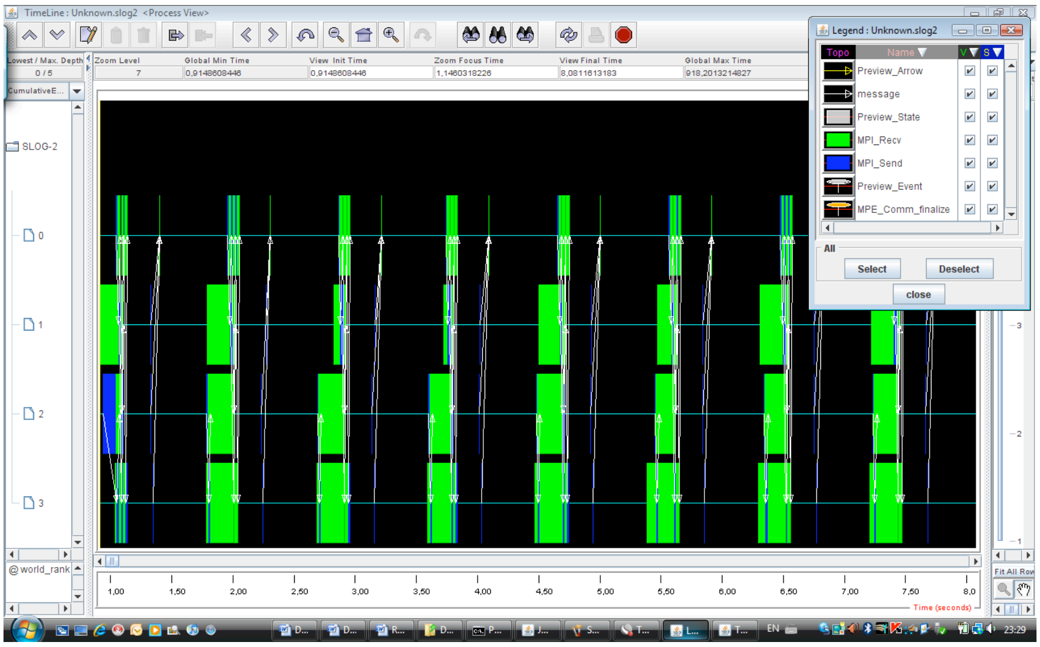


Figure 10 – Result of the simulation after 1000 iterations

Figure 10 represents the mesh after 1000 iterations, processed evenly in four nodes of the cluster. For the processing time to be more competitive, we opted for decreasing the communication between nodes, invoking the interboundary routine communication only at the end of the five-stage Runge-Kutta method and not between each stage.


TimeLine : Unknown.slog2 <Process View>

Lowest / Max. Depth	Zoom Level	Global Min Time	View Init Time	Zoom Focus Time	View Final Time	Global Max Time
0 / 5	7	0,9148608446	0,9148608446	1,1460318226	8,0811613183	918,2013214827

Legend : Unknown.slog2

Topo	Name	V	S
	Preview_Arrow	✓	✓
	message	✓	✓
	Preview_State	✓	✓
🟩	MPI_Recv	✓	✓
🟦	MPI_Send	✓	✓
	Preview_Event	✓	✓
	MPE_Comm_finalize	✓	✓

Figure 11- Communication flow between nodes without communication between Runge-Kutta stages

Figure 11 shows the program's performance when communication was reduced during the residue calculation using Runge-Kutta's five stages. The figure reveals the moments when there is information exchange for evaluating the maximum residue followed by a significant processing of information for the calculation via the Runge-Kutta model and, after its conclusion, the communication between nodes for information exchange within the border surfaces, both between mesh partitions and between the mesh and the wing profile located at the center. With this modification, the case with 1,000 iterations using 4 processing nodes changed from 2,621.679 seconds to 914.156 seconds. The fact that there is less information exchange between nodes during the calculation of the residue causes the number of iterations necessary for converging a single minimum residue value to be higher.

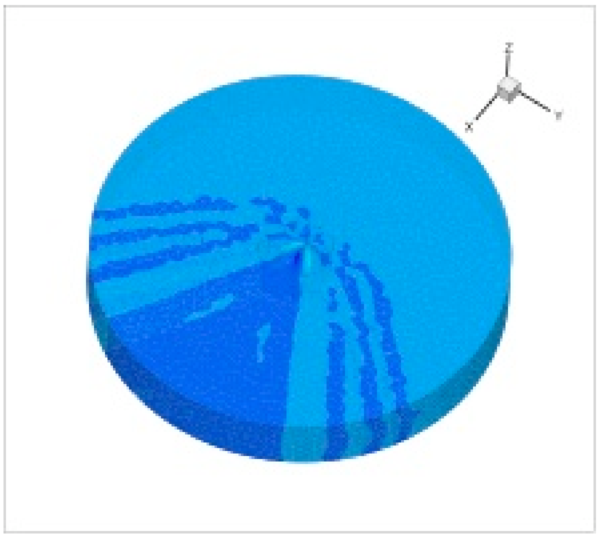


Figure 12 – Result of Simulation without using message exchanges between stages of the Runge-Kutta method

In tests using 2 to 10 processing nodes, time always decreased when one more node was added and the maximum residue without communication between Runge-Kutta stages were larger. The average difference was 37% in observed cases.

Figure 12 shows the processing result after 1000 iterations, without data exchange between the five stages of the Runge-Kutta model. The result profile is identical to the result found when communication occurred through the boundary routine between each stage of rk5s. In order to validate the thesis that the final values tend to be identical, we applied tests that established an identical maximum residual value and evaluated the number of iterations and the total time needed to reach the pre-established residual values. These tests concluded that, on average, 20% more iterations are needed to achieve the pre-established results of the residue. Meanwhile, the total time required for the same number of iterations wavered between 20% of the total time in the worst case and 10% of the total time.

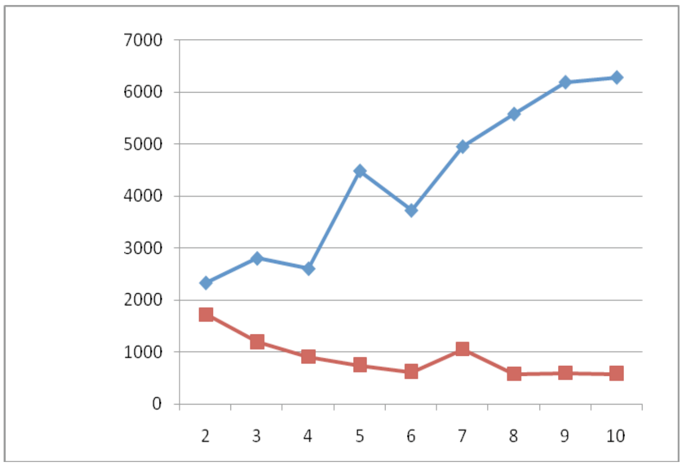


Figure 13 – Comparison between the total processing time with communication and without communication in the Runge-Kutta model

Figure 13 represents the time obtained with two versions of the program. The horizontal axis shows time in seconds for 1000 iterations using the 360434-volume mesh and the vertical axis shows the number of processing nodes in use. The upper curve shows the performance of the solution with message exchanging between Runge-Kutta stages, while the lower part reflects the time spent by the version that only exchanges information once after processing the five stages.

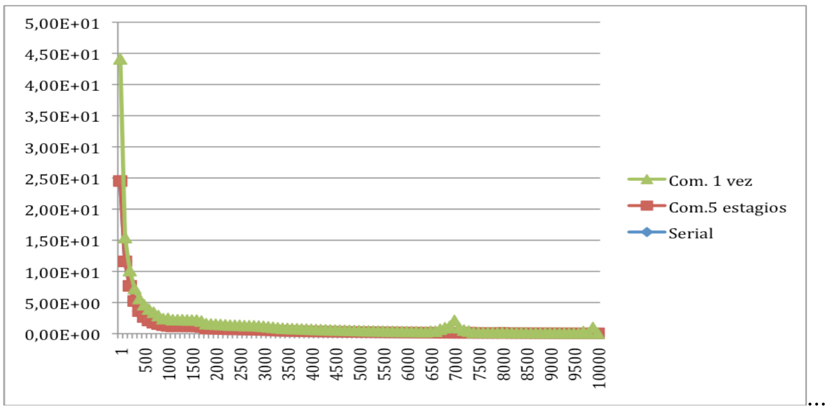


Figure 14 – Comparing the convergence curves within approaches

Figure 14 shows the convergence curve between the serial solution, a parallel solution with communication between stages, and another that only communicates once, in each iteration, during the execution of the five Runge-Kutta stages. The vertical axis contains the maximum residue observed in each iteration and the horizontal axis contains the number of iterations.

It should be highlighted that the number of iterations employed, which is 1000 for the 13328-volume mesh, is still far from the expected solution with the minimum specified residual value and consequent solution convergence.

5. CONCLUSION

We can therefore sustain that the initial results for processing using message-sending protocols for CFD problems with medium-sized tridimensional meshes are highly satisfactory. This leads us to consider extending the solution of processing distribution to mesh refinement routines, in hope of rendering the processing of real simulation problems viable by spending substantially less processing time than the initially spent in the beginning of the research.

Since the undergone tests have not yet contemplated the convergence of residual values into acceptable values, that is, a value with an order of magnitude 0.00001, we will provide new tests with various-sized meshes, seeking to establish and validate the above-mentioned distributed processing in all value spectrums, which are research topics of problems involving correlated situations.

6. REFERENCES

Scalabrin, Leonardo Costa, *Numerical Simulation of Three-Dimensional Flows Over Aerospace Configurations*, M.S. Dissertation, Aeronautics Engineering Department, Instituto Tecnologico de Aeronautica, Sao Jose dos Campos, Sao Paulo, Brazil, 2002.

Press, W.H., Flannery, B.P.,Teukolsky, S.A.,Vetterling, W.T., *Numerical Recipes in Fortran 77- The art of the scientific computing*, Cambridge University Press, 2nd. ed, 1992.

Gropp, W. et. al., *MPI - The Complete Reference. The MPI Extensions*, 2nd ed. MIT, 1998.

Spalart, P.R. e Allmaras S.R., *A One-Equation Turbulence Model for Aerodymics Flows*, Proceedings of the 30th AIAA Aerospace Science Meeting and Exhibit, Reno, NV, 1992.

7. RESPONSIBILITY NOTICE

The authors are the only responsible for the printed material included in this paper.