INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Load balancing on Clusters and GRIDS with mixed processor speeds

Stephen Wornom

N° ????

Janvier 2006

Thème NUM

ISSN 0249-6399

Load balancing on Clusters and GRIDS with mixed processor speeds

Stephen Wornom*

Thème NUM — Systèmes numériques
Projet Tropics

Rapport de recherche n° ???? — Janvier 2006 — 31 pages

Abstract: CFD simulations on Clusters and GRIDS having mixed processor speeds present several challenges to achieve efficient load balancing. If both the fast and slow processors are given the same amount of work, the faster processors will finish their computations first and wait for the slower processors to finish. To achieve load balancing more work must be given to the faster processors so that all the processors finish their computations at the same time (work is proportional to the processor mesh size). Another complication is that for current Clusters and GRIDS in the near future, the user does (will) not know in advance the mixture of fast and slow processors that will be assigned to their computation, thus the user cannot partition the mesh in advance of the CFD run! This difficulty is doubly complicated as the mesh partitioning step is usually performed on a desktop computer using one processor. For mesh partitioners executing on parallel computers, the complication arises in that the mesh partitioning code and the CFD code are separate MPI codes designed to be run independently of each other. As a result the two codes cannot simply be

* INRIA, 2004 Route des Lucioles, BP. 93, 06902 Sophia-Antipolis, France

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles, BP 93, 06902 Sophia-Antipolis Cedex (France)

run back-to-back as each code may be assigned different mixtures of fast and slow processors resulting in a partitioned mesh not optimal for the CFD run.

In this study, in order to overcome the problems related to computing with arbitrary mixtures of fast and slow processors, the mesh generator has been integrated into the CFD code. Thus optimal size partitions are automatically created for different mixtures of fast and slow processors. The efficiency of this approach is demonstrated for Clusters and GRIDs having both fast and slow processors.

Key-words: Computational fluid dynamics, GRID Computing, MecaGRID, GRID5000, F90, MPI, dynamic mesh creation and partitioning

Equilibrage de charge sur Clusters et GRILLES de calcul avec différents vitesse de processeurs

Résumé : On s'intéresse à l'équilibrage de charge des simulations numériques en Mécanique des Fluides Numérique sur Clusters et Grilles de calcul comportant des processeurs de vitesses différentes. Pour réaliser cet équilibrage, il est nécessaire d'attibuer entre deux synchronisations/communications plus de calculs aux processeurs plus rapides qu'aux processeurs les plus lents, de manière à ne pas perdre de temps en attentes. Une difficulté supplémentaire réside dans l'impossibilité en pratique de savoir quel nombre de chaque type de processeur sera attibué à la tâche au moment de son démarrage, information dépendant de la charge instantanée du système. L'utilisateur est dans ce cas dans l'impossibilité de préparer une partition adéquate avant le lancement du calcul. Or en général les logiciels disponibles pour les deux étapes, partitionnement, simulation proprement dite, sont distincts et à lancer sur le système séparément. Dans ce cas ils ne tournent pas sur le même assortiment de processeurs. Dans cette étude, générateur de maillage et partitionneurs sont íntégrés au solveur et seront donc lancés en une seule requète soumise au Cluster. Un partitionnement adapté au jeu de processeurs attibué peut donc être réalisé juste au moment du calcul. L'efficacité de cettte approche est démontrée pour différentes configurations de grilles de calcul et de Clusters fortement hétérogènes.

Mots-clés : Mécanique des fluides numérique, Grille de calcul, F90, MecaGRID, GRID5000

1 Introduction

CFD simulations on Clusters and GRIDS assigned processors with different speeds present several major challenges in order to achieve efficient load balancing. We use the term cluster to denote a computer comprised of a collection of homoegeneous processors and Cluster to refer to a computer comprised of several clusters at the same institution administered by the same system administrator. For example, the INRIA Cluster in Sophia Antipolis is comprised of three separate on-site clusters *nina*, *pf*, and *nef*. The term GRID refers to a Computational Grid composed of clusters belonging to different institutions, each having a different administator. An example is the MecaGRID[1] is comprised of clusters belonging to INRIA in Sophia Antipolis, the IUSTI[2] in Marseille and the CEMEF[3] in Sophia Antipolis. The GRID5000[4] is comprised of clusters at INRIA in Sophia Antipolis, Toulouse, Orsay, Rennes, Bordeaux, Grenoble, Lyon and Lille. The clusters forming a GRID are usually separated by large physical distances thus communication times between the different clusters of the GRID may be larger than those for Clusters.

The challenges that arise when simulations are assigned different types of processors relate to the different processor hardware characteristics (speeds, RAMs, and caches, ...etc). For example some processors may have 2 Ghz speed[5] with 1 GB of RAM and 512K cache while others have 1 Ghz with 1/2 GB of RAM with 256K cach with 256K cache. To achieve load balancing twice

[1] Province-Alpes-Côte d'Azur Project Funded by the French Ministry of Research
[2] Institut Universitaire des Systèmes Thermiques et Industriels
[3] l'Ecole des Mines de Paris à Sophia-Antipolis
[4] National ACI-GRID Project funded by the French Ministry of Research, CNRS and INRIA
[5] 2 Ghz is an approximation to the actual value of 2.2 Ghz.

as much work must be given to the faster processors so that both the fast and slow processors finish their computations in approximately the same time. If both the fast and slower processors are given the same amount of work, the faster processors will finish their computations first and wait for the slower processors to finish.

The goal of this study is to achieve efficient simulations using the MPI CFD software, *AERO3D*, developed through a collaboration between the University of Colorado at Boulder, the University of Montpellier, and INRIA Sophia Antipolis - see Fezoui et al [4], Farhat and Lanteri [3], and Lanteri [6]. The *AERO3D* software is intensively used, in particular by the universities of Montpellier, Pisa and Pau in several CFD research studies (see for example Camarri et al [1], El Omari et al [2], and Koobus and Farhat [5]).

The parallelization strategy in *AERO3D* uses mesh partitioning to subdivide a global mesh into smaller partitions[6]; each partition being computed by a different processor, for example 32 partitions solved using 32 processors. The message-passing programming model of Fezoui et al [4], Farhat and Lanteri [3], and Lanteri[6] using the Message Passing Interface (MPI) communication library ensures software portability from one parallel system to another. The mesh partitioning algorithms and the generation of the corresponding communication data structures are computed in a PREPROCESSING step.

Mesh partitioners are often non-parallel codes executed on desktop computers. For the *AERO3D* code, the preferred mesh partitioners are METIS[7] and

[6]Sometimes called subdomains or blocks

[7]http://www-users.cs.umn.edu/~karypis/metis/

TopDomDec[8], both non-parallel codes. Other mesh partitioners that execute on parallel machines using MPI will be discuss later. The usual procedure is to run the mesh partitioner first (PREPROCESSING) to create the partitions to be used by the CFD run. This is followed by the execution of the CFD software.

In general, when running on a Cluster (or future GRIDS), the user will not know in advance the mixture of processors that will be assigned to the computation. This is currently true for the INRIA *nina-pf* cluster which has both 2 Ghz and 1 Ghz processors; the user requests the total number or processors to be used and the job manager assigns the processors according to availability. Thus, if a user requests 32 processors on the INRIA Cluster[9], 32-*nina* and 0-*pf*, 24-*nina* and 8-*pf*, 8-*nina* and 24-*pf*, 0-*nina* and 32-*pf* ... etc are various possible allocations the user could receive. Note that for runs on the INRIA Cluster, all the requested processors must be available before any processor is assigned to the job. This is very efficient and the INRIA Cluster is an excellent example of how the future generation of GRIDs will function.

As of the date of this report, this is not true neither for the MecaGRID or the GRID5000[10] where the user specifies which clusters of the GRID are to be used and the number of processors on each cluster. In contrast to the INRIA Cluster, both the MecaGRID and the GRID5000 assign processors on the individual clusters as soon as they become available. Thus if the user requests 128 processors and 120 are immediately available, the 120 are assigned to the simulation and will start execution. However, the job will wait at the first

[8] University of Colorado at Boulder

[9] The INRIA Cluster as of Janurary 13, 2006 has 64 processors, 32-*nina* and 32-*pf*

[10] Both first generation GRIDS

MPI barrier until the remaining eight processors are assigned and reach the same barrier. This may be hours or days away effectively blocking the 120 processors that remain assigned but inactive from being used by other users. This is extremely inefficient as the cluster(s) on which the processors were requested may be totally saturated while other clusters on the GRID have immediate processors available. Future GRIDS will function like the INRIA *nina-pf* Cluster, that is, the user requests the total number of processors and the GRID job manager will assign the processors according to availability on all the clusters of the GRID. Achieving this goal is one of political decision rather than technology.

What are the consequences of not knowing, in advance, the mixture of fast and slow processors that will be assigned to a computation? The major consequence is that the user cannot partition a mesh as a PREPROCESSING step.

If we assume that the mesh partitioner is also an MPI code, this difficulty is doubly complicated as the mesh partitioner code and the CFD code are separate MPI codes designed to be run independently of each other.

In this study, to overcome the difficulties related to simulations involving fast and slow processors, the mesh generator has been integrated into the CFD software. This approach assures that the mesh partitions will be optimal for different combinations of fast and slow processors assigned to the CFD execution.

2 Dynamic memory allocation

In order to achieve load balancing on Clusters and GRIDS with mixed processor speeds, two conditions must be met. First, the CFD software must have dynamic memory allocation. To achieve this, an F90 version of the *AERO3D* software was created - see Wornom [8], the original code was programmed in F77. As a consequence the size of the F77 executable is determined at compile time based on the mesh data of the largest partition and the same executable is used on all the processors. This leads to situations where the RAM for slower processors is too small to run the executable. F90 permits dynamic memory allocation thus the executable size for each processor is proportional to the local processor mesh size.

The second condition is dynamic mesh partitioning. This is achieved by integrating the mesh generator into the CFD software.

3 Mesh generators and mesh partitioners

3.1 Standard procedure

In the standard procedure there are three independent software involved in a CFD simulation, a mesh generator, a mesh partitioner, and the CFD software.

The first step is to create or generate a mesh for the simulation. There are several commercial software available. An example is GSH3D developed at INRIA in collaboration with SIMULOG[11]. In this study we prefer free open source software MeshCanale developed at the University of Pisa. Mesh gener-

[11] available at INRIA

ators may be non-parallel codes or parallel codes. GSH3D[12] and MeshCanale are both non parallel codes executed on desktop workstations.

The second step is to partition the mesh created by the mesh generator. If none exists, the user must write an interface so that the mesh file created by the mesh generator can be read by the mesh partitioner. For the mesh partitioners used with *AERO3D*, the mesh file is named *fluid.sinus* - see Appendix A for details. This interface already existed MeshCanale and was written for GSH3D by the author. The user specifies the number of partitions and the mesh partitioner partitions the mesh specified in the *fluid.sinus* file.

Next, if none exists, the *AERO3D* user must write an interface so that the partitioned data created by the mesh partitioner is written in a format readable by the *AERO3D* software. These files are named *flu-00001, flu-00002*, where 1 and 2 refer to partitions 1 and 2, ..., and the *flu.glob* file that contains the mesh global parameters - see Appendix B for additional details. Writing this interface is not a trivial step and is a major reason why the PARmetis[13] mesh partitioner is not presently used at INRIA to partition meshes for the *AERO3D* software. The other reason is that the interface to read the *fluid.sinus* file has not been written. At the present time, the partitions created with the mesh partitioners developed at the CEMEF[14], TopDomDec[15], METIS[16], and MeshCanaleMP (see section 3.2) are directly readable by the *AERO3D* software. The CEMEF, PARmetis and MeshCanaleMP are mesh partitioning codes that

[12] The 1999 sequential version of GSH3D is installed at INRIA Sophia Antipolis.

[13] http://www-users.cs.umn.edu/~karypis/metis/

[14] Centre de mise en forme des matériaux de l'Ecole des Mines de Paris-Sophia Antipolis

[15] University of Colorado at Boulder

[16] *AERO3D* interface written by Professor Farhat, University of Stanford, Palo Alto, CA

execute on parallel computers. The third step is the execution of the CFD software that reads the partition data created by the mesh partitioner.

In summary, the standard procedure consists of executing three independent codes, the mesh generator, the mesh partitioner andthe CFD software.

3.2 Integrated parallel procedure

The meshes used in the MecaGRID study of Wornom [9] were generated using the F77 non-parallel mesh generator MeshCanale run on a workstation. For this reason and the availability of the source code, MeshCanale was selected as the first mesh generator to be integrated into the *AERO3D* software.

We started by creating an F90 version of the MeshCanale code in order to have dynamic memory allocation. Next the adaptation to running in parallel using MPI was accomplished. The major effort was in writing subroutines to identify the different processors assigned to the simulation and a load balancing algorithm to define the partitions.

In MeshCanaleMP the mesh is generated simultaneously for all partitions. Therefore the notion of partitioning a global mesh created by a mesh generation code is no longer valid. Generating the mesh directly on each partition has several advantages over the standard approach[17], in particular for large meshes ($>$ 2 million vertices). Executing the mesh partitioner separately requires additional memory to store the global mesh file (*fluid.sinus*). This additional memory is in addition to the memory needed to create the partitions. By integrating the mesh generator into the CFD software, this additional memory

[17] Generating the global mesh with a grid generator and partitioning the global mesh using a mesh partitioning code

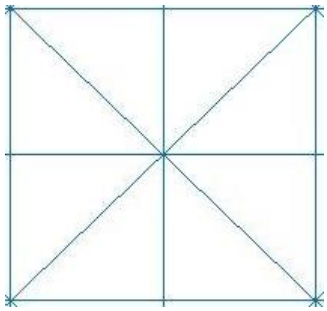

Figure 1: MeshCanale English-Flag design

is avoided (there is no global mesh file to read). The memory saved permits much larger mesh partitions to be created than would be otherwise possible. This effectively allows us to create partitions for very large meshes not possible when using the standard approach. The MeshCanaleMP software is available, thus the details of the F90 MPI implementation of the mesh generator (mesh partitioning) will not be discussed.

4 Mesh generator - MeshCanale

MeshCanale was developed at the University of Pisa by Dr. Camarri and is composed of three parts. First, MeshCanale creates a structured hexahedra mesh and then divides each hexahedra into six tetrahedra using the English-Flag design shown in Figure 1 for a plane. The third part writes the *fluid.sinus* file containing the number of vertices, tetrahedra, and external faces, the Cartesian coordinates, the tetrahedra connectivity, and the connectivity for the external faces and the type of boundary conditions to be applied.

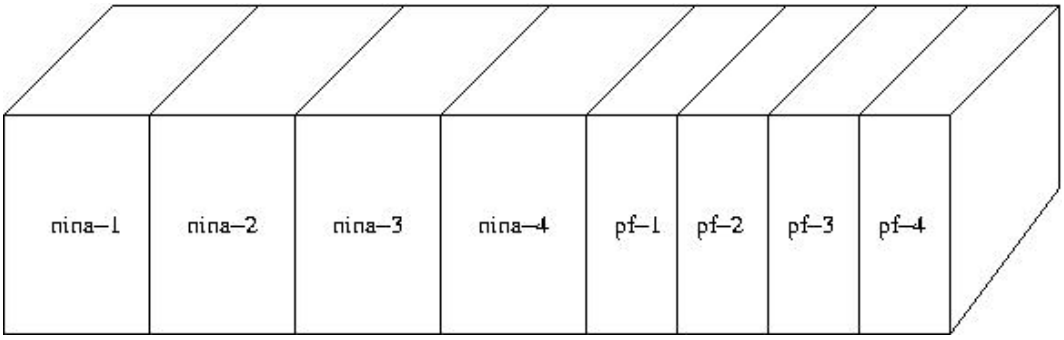

Figure 2: Partitioning by slicing

4.1 Partitioning options

The MeshCanaleMP code was developed to study blast waves interacting with a high density fluid of spherical shape in a rectangular channel - see Figure 3 (from Wornom et al [7]) shows the solution after the blast wave has passed through the bubble. The blast wave is moving from the lower left to the upper right of the figure. For this example a simple rectangular mesh with Imax, Jmax, and Kmax equal to 51, 51, 97 (51x51x97) was used.

Two obvious partitioning options exists: 1) Slicing and 2) Dicing. Figure 2 illustrates slicing using 8 processors, 4-*nina* and 4-*pf*; the 4-*nina* partitions are two times larger than the 4-*pf* partitions reflecting the faster speed of the *nina* processors. Message passing occurs between the partition interfaces. Note the only messages passed between the fast cluster (*nina*) and the slow cluster (*pf*) occur between processor 4 of *nina* and processor 1 of *pf*.

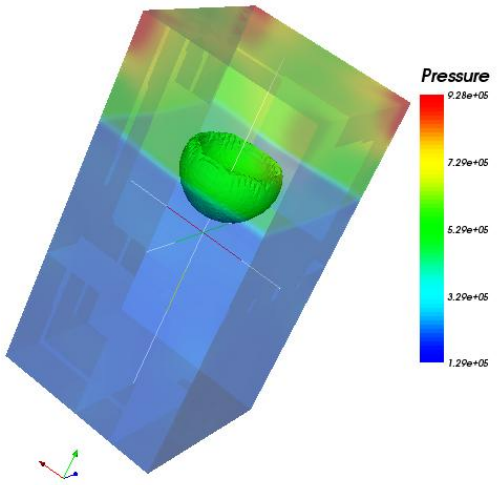

Figure 3: Blast wave pressure contours after 720 time steps

Using the slicing option, one can easily show that the minimum message passing occurs if the mesh is sliced in the counter direction opposite to the

$$min(Imax \times Jmax,\ Imax \times Kmax,\ Jmax \times Kmax). \tag{1}$$

For this example the mesh is a simple rectangular mesh with Imax, Jmax, and Kmax equal to 51, 51, 97 (51x51x97). Therefore the mesh was sliced in the k-direction.

The partitions created by slicing should be optimal for VPN (Virtual Private Network) GRIDS like the MecaGRID. For VPN GRIDS the processors have private IPAs and only one processor is used to exchange information between processors on different clusters of the GRID using tunneling between

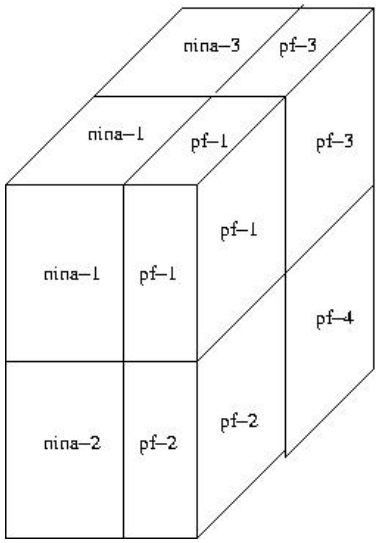

Figure 4: Partitioning by dicing

the frontend machines of the clusters comprising the GRID. Using the slicing method, only one *nina* processor exchanges information with one *pf* processor. If many processors are trying to pass data at the same time as will occur if 100 or more processors are involved, the VPN will experience greater transfer slowdowns when non-slicing partitioning methods are used. For GRIDS with public IPAs, such as GRID5000, the partitioning method will be less important.

A alternative to slicing would be "dicing" where the mesh is divided into blocks. Figure 4 illustrates dicing with 8 processors, 4-*nina* and 4-*pf*. As can be seen, the message passing between the different processors is more complicated than that shown in Figure 2 where the slicing option is used. With 8 processors one can see that each *nina* CPU exchanges messages with a *pf* pro-

cessor in addition to its *nina* neighbors. Thus there are four messages passed between the *nina* cluster and the *pf* cluster, likewise for the *pf* processors. The complexity increases as the total number of processors increases.

5 Heterogeneous partitioning

Consider the blast wave problem with a rectangular global mesh of 51x51x97 (Imax, Jmax, Kmax) with Imin = Jmin = Kmin = 1. Application of equation 1 shows that the minimum message passing occurs when the slicing method is applied in the k-direction.

The mesh is generated (partitioned) in two steps. First, we determine how many tetrahedra will be computed on the fast cluster and how many on the slow cluster. Suppose we have two clusters with different processors speeds, one with 2.2 Ghz and the other with 0.999 Ghz. For illustration purposes we take the ratio of the fast processors to the slow processors equal to 2. For the heterogeneous partitioning we want the fast cluster to compute twice as many tetrahedra as the slower cluster. To accomplish this we give 2/3 of the total tetrahedra to the fast cluster and 1/3 to the slower cluster. The general expressions to be used when defining the hexagonal input (i,j,k values for each partition) to the MeshCanaleMP code are given by

$$\Delta k^{fast} = \frac{Kmax - 1}{1 + \frac{N^{slow}}{2 \times N^{fast}}} \tag{2}$$

$$\Delta k^{slow} = (Kmax - 1) - \Delta k^{fast} \tag{3}$$

Equations 2-3 returns an integer value only for equal number of fast and processors or only *nina* or *pf* processors are used - see Table 1 which shows an example for 32 total processors for Kmax = 97 with different mixtures of fast and slow processors.

Mixture		Theoretical		Code	
N^{fast}	N^{slow}	Δk^{fast}	Δk^{fast}	Δk^{fast}	Δk^{fast}
32	0	96.000	0.000	96	0
28	4	74.667	21.333	75	21
24	8	57.600	38.400	58	38
20	12	43.636	52.364	44	52
16	16	32.000	64.000	32	64
12	20	22.154	73.846	23	73
8	24	13.714	82.286	14	82
4	28	6.400	89.600	7	89
0	32	0.000	96.000	0	96

Table 1: Computation of intervals

When the number of fast processors is not an integer, and extra k-plane is given to the faster cluster (column Code).

The next step is defining the size of the partitions. These are given by the relation

$$\delta k^{fast} = \Delta k^{fast}/N^{fast} \tag{4}$$

$$\delta k^{slow} = \Delta k^{slow}/N^{slow} \tag{5}$$

For most mixtures, δk^{fast} and δk^{slow} will not be integers and adjustments are made similar to that for Δk^{fast} and Δk^{slow}.

5.1 Executing MeshCanaleMP separately of *AERO3D*

In this section we discuss features related to running MeshCanaleMP as a stand alone program. MeshCanaleMP reads its data from the file *CanaleMP.data*. Table 2 shows the differents entries in the file *CanaleMP.data*

read (myunit,*) LargeMesh
read (myunit,*) nWRTsinus, nWRTflu
read (myunit,*) Imax,Jmax,Kmax
read (myunit,*) xmin,xmax
read (myunit,*) ymin,ymax
read (myunit,*) zmin,zmax
read (myunit,*) bcImax, bcImin
read (myunit,*) bcJmax, bcJmin
read (myunit,*) bcKmax, bcKmin

Table 2: *CanaleMP.data* file

The LargeMesh flag was added for cases where very large meshes are used (> 2 million vertices). The LargeMesh flag is independent of the MeshCanaleMP option - see section 5.3.

The flags *nWRTflu* and *nWRTsinus* control writing of the *flu-00001, flu-00002, ..., flu.glob* files and the *fluid.sinus* file. The flags are set to 1 to activate.

The number of mesh points in i, j, k are given by Imax, Jmax, and Kmax. The max/min values of x are given by xmax/xmin, similar for the y and z

Mesh	CPUs	vertices	fluid.sinus	flu-xxxx	Total	WRTsinus	WRTflu
51x51x97	16	252K	30 MB	3 MB	3.7 sec	1.4 sec	2 sec
101x101x193	48	2M	233 MB	9 MB	28.8 sec	10 sec	19 sec
201x201x395	48	16M	1860 MB	69 MB	233 sec	90 sec	142 sec
401x401x729	48	117M	(1.49 GB)	-	45 sec	-	-
401x401x729	64	117M	(1.49 GB)	4 MB	7 sec	-	-

Table 3: MeshCanaleMP examples

coordinates. The boundary condition types are set by the bcImax, bcImin, bcJmax, bcJmin, bcKmax, and bcKmin.

Table 3 gives examples of different runs using MeshCanaleMP for meshes up to 117 million vertices and 64 processors on the INRIA *nef* cluster[18]. The first four runs show the total time, the time to write the *fluid.sinus* file[19], and and the partition files *flu-00001, flu-00002*, ... etc and the *flu.glob* file. Note that the majority of the time is in writing these files. For a mesh with 16 million mesh vertices the *fluid.sinus* file is 1.68 GB and the 48-processor *flu-xxxxx* files 69 MB, these files were written unformatted. The formatted files are approximately four times larger. It is obvious that for large simultations one must avoid writing these files.

[18]The *nef* cluster has 64 processors at 2.2 Ghz speed and was used for the development of the MeshCanaleMP code.

[19]*fluid.sinus* file is input to the mesh partitioner. It is also needed for graphics.

5.2 Executing MeshCanaleMP from within *AERO3D*

The basic parameters and control flags for the *AERO3D* software are found in the *flu.data* file. These include the number of time steps, whether the run uses the explicit or implicit algorithm, whether the time scheme is 1st or 2nd order, ... etc.

To execute the MeshCanaleMP mesh partitioner within the *AERO3D* execution, four additional flags have been added; these flags are given for seven different examples in Table 4 (1=activate).

Run	LargeMesh	nMeshCanaleMP	nWRTflu	nWRTsinus
1	0	0	0	0
2	0	0	0	1
3	0	1	1	1
4	0	1	0	1
5	0	1	0	0
6	1	1	0	0
7	1	0	0	0

Table 4: Flags related to accessing MeshCanaleMP within *AERO3D* software

After the processors have been allocated for the *AERO3D* run, the MeshCanaleMP program is executed, if the parameter $nMeshCanaleMP = 1$ to create the mesh and the partitions.

If nMeshCanaleMP = 0, the MeshCanaleMP is not used and the files with the data for each partition must already exist. Run 1 corresponds to the case

where the user has existing partition data files[20], *flu-00001, flu-00002*, ... etc and the *flu.glob* file. This is the standard procedure for the *AERO3D* code.

Several subroutines that have been written for the MeshCanaleMP are useful for runs where the MeshCanaleMP is not used. For example, the graphical program needs the *fluid.sinus* file containing the global mesh data in addition to the solution files. Normally the *fluid.sinus* is the input to the mesh partitioner and is available. Should it not be available, as sometimes occurs, the *AERO3D* code will write the *fluid.sinus* file if nWRTsinus = 1. Run 2 asks *AERO3D* to write the *fluid.sinus* file for the existing *flu-00001, flu-00002*, ... etc and the *flu.glob* file used.

Run 3 runs the MeshCanaleMP from within the *AERO3D* code and writes the *flu-00001, flu-00002*, ... etc and the *flu.glob* file and the *fluid.sinus* file.

Runs 4-5 are for very large meshes ($>$ 2 million vertices). For large meshes writing the *flu-00001, flu-00002*, ... etc and the *fluid.sinus* file should be avoided as these files can be extremely disk space consuming[21]. Since the *flu-00001, flu-00002*, ... etc files are created doing the run there is no real reason to save them. Run 5 does not save the *fluid.sinus* thus assumes that the graphical files have been created within the *AERO3D* run.

5.3 *LargeMesh* = 1

For meshes on the order of 16 million vertices and larger, the *fluid.sinus* and the files containing the solutions at different time steps are too large to be

[20] See APPENDIX B for description

[21] The sysops is very appreciative when these extremely large files are not written

viewed with ParaView[22] on a workstation with 4 GB of RAM. The solution files are named *solf.000100.data* where 000100 is the solution at the 100th time step.

When *LargeMesh* = 1, the *fluid.sinus* and the solution file *solf.000100.data* are written separately for each processor. The files are written as *fluid.sinus-00001, fluid.sinus-00002*, .. etc and *solf-00001.000100.data, solf-00002.000100.data*, ... etc. These smaller files are much smaller files than if all the processor solutions were written on a single file. As a consequence, they can be viewed using ParaView run sequentually or the MPI version of ParaView[23].

Runs 6 and 7 show examples where the LargeMesh option is used with and without the MeshCanaleMP code.

6 Subroutines added to the *AERO3D* software

During this study six useful subroutines were added to the *AERO3D* code.

6.1 Subroutines used by MeshCanaleMP

The following subroutines have been added to the *AERO3D* software for the MeshCanaleMP option.

`CanaleMP.f`

`GetHostNames.f`

`MeshCanaleMP_F90.f`

[22] http://www.kitware.com
[23] Work in progress.

6.2 Useful subroutines with/without the MeshCanaleMP option

`WRTflu.f`

`WRTsinus.f`

`WRTsinusPart.f`

`WRTsolPart.f`

7 Results

In this section, the benefit of using heterogeneous partitioning on Clusters and GRIDS with mixed fast and slow processors is illustrated. The *nina-pf* Cluster, the MecaGRID, and the GRID5000 were chosen for these tests. Table 5 shows the processor speeds for the different clusters used.

	cluster(Location)		Processor speed (Ghz)	
Cluster/GRID	Fast cluster	Slow cluster	Fast cluster	Slow cluster
INRIA Cluster	*nina* (Sophia)	*pf* (Sophia)	2.2	1.0
MecaGRID	*iusti* (Marseilles)	*pf* (Sophia)	2.0	1.0
GRID5000	*sophia* (Sophia)	*icluster2* (Grenoble)	2.2	0.9

Table 5: Processor speed for tests

The test case computed flow in a nozzle. Two meshes were studied; the first mesh contains 252K vertices with 1.44 million tetrahedra and the second 502K vertices with 2.88 million tetrahedra. The efficiency of using homogeneous and heterogeneous partitions was studied.

Cluster/GRID	Mesh size	CPUs	wall time		speedup
			homogeneous	heterogeneous	
INRIA Cluster	252K	8-*nina* 8-*pf*	738 sec	477 sec	1.55
MecaGRID	252K	8-*iusti* 8-*pf*	776 sec	588 sec	1.32
GRID5000	252K	8-*sophia* 8-*icluster2*	-	-	-
INRIA Cluster	502K	16-*nina* 16-*pf*	843 sec	553 sec	1.52
INRIA Cluster	502K	24-*nina* 8-*pf*	738 sec	529 sec	1.39
GRID5000	502K	8-*sophia* 8-*icluster2*	-	-	-

Table 6: 252K vertices: homogeneous vs heterogenous partitioning for 16 processors

For the homogeneous mesh partitioning all the partitions were of equal size. For the heterogeneous mesh partitioning the mesh was partitioned so that the fast-cluster partitions were twice as large as the slow-cluster partitions.

Table 6 shows the wall times for a simulation of 150 time steps for different size meshes and different mixtures of fast and slow processors for the different mesh sizes. From these tables, an important reduction in simulation time is noted using the heterogeneous partitioning for the case where there are equal numbers of fast and slow processors; a speedup of 1.55 for the 252K vertices mesh and 1.52 for the 502 vertices mesh (the theoretical speedup for these cases is 1.5). For the case where 24 fast and 8 slow processors were used, the reduction in simulation time was 1.39 (the theoretical value is 1.25)[24]. These reductions in simulation times indicate that good load balancing is achieved when heterogeneous partitions are used. Figure 5 shows the computational times for 16 processors to compute the fluxes in *AERO3D* for both the homo-

[24]The theoretical value of 1.25 was based on a speed ratio of 2 whereas the hardware speed ratio is approximately 2.2. This may explain why the real speedup was greater than the quoted theoretical value

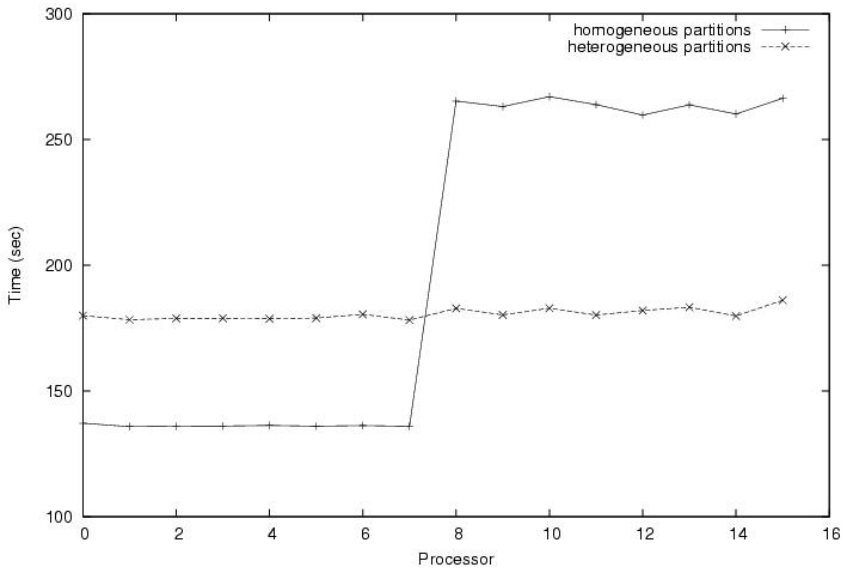

Figure 5: Optimal vs nonoptimal partitions

geneous and heterogeneous partitions. Note the approximaly equivalent times for the heterogeneous partitions.

8 Conclusions

In this study, in order to overcome the problems related to computing with arbitrary mixtures of fast and slow processors occuring on clusters and GRIDS with different speed processors, the mesh generator and the mesh partitioner have been integrated into the CFD code. Thus optimal load balancing partitions are automatically created for different mixtures of fast and slow processors.

Executing MeshCanaleMP from within the *AERO3D* code assures that the mesh partitions will always be optimal for any mixture of fast and slow processors assigned for the *AERO3D* simulation.

A test case on the INRIA *nina-pf* cluster produced a speedup of 1.45 relative to the same run using the homogeneous partitioning which compares well with the theoretical speedup of 1.5, indicating good load balancing when heterogeneous partitions are used.

9 Acknowledgements

The MecaGRID research was made possible by the ACI-GRID 2002 Project of the French Ministry of Research[25]. The GRID5000[26] research is part of a research effort developping a large scale nation wide infrastructure for Grid research. This project is an initiative of the French ACI Grid Incentive and is funded by the French Ministry of Research, CNRS and INRIA. The author would like to thank Hervé Guillard and Alain Dervieux for their support of this work.

References

[1] S. Camarri, M.V. Salvetti, B. Koobus, and A. Dervieux. Large-eddy simulation of a bluff-body flow on unstructured grids. *Int. J. Num. Meth. Fluids*, 40:1431–1460, 2002.

[25] http://www.recherche.gouv.fr/recherche/aci/grid.htm
[26] http://www.grid5000.org/

[2] K. El Omari, E. Schall, B. Koobus, and A. Dervieux. Turbulence modeling challenge in airship CFD studies. *Proceedings of the eigth Zaragoza-Pau conference of applied mathematics and statistics.*, Jaca, 15-17 sept. 2003.

[3] C. Farhat and S. Lanteri. Simulation of Compressible Flows on a Variety of MPPs: Computational Algorithms for Unstructured Dynamic Meshes and Performance Results. *Comp. Meth. Appl. Mech. and Eng.*, 119:35–60, 1994.

[4] L. Fezoui, F. Loriot, M. Loriot, and J. Regere. A 2D Finite Volume /Finite Element Euler Solver on M.I.M.D. parallel machine. *Proceedings of the High Performance Computing II Conference , M. Duran and F. El Dabaghi Eds.*, Montpellier, 1991.

[5] B. Koobus and C. Farhat. A Variational Multiscale Method for the Large Eddy Simulation of Compressible Turbulent Flows on Unstructured Meshes - Application to Vortex Shedding. *Comp. Meth. Appl. Mech. and Eng.*, 193:1367–1383, 2004.

[6] S. Lanteri. Parallel Solutions of Three-dimensional Compressible Flows. Technical Report RR-2594, INRIA - Sophia Antipolis, June 1995.

[7] S. Wornom, B. Koobus, H. Guillard, A. Murrone, and A. Dervieux. Seven-equation, two-phase flow three-dimensional calculations using a mixed-element-volume method. Technical Report RR-5560, INRIA - Sophia Antipolis, April 2005.

[8] Stephen Wornom. Optimizing a CFD fortran code for GRID Computing. Technical Report RT-303, INRIA - Sophia Antipolis, April 2005.

[9] Stephen Wornom. Mecagrid: Rapport d'avancement pour la période du 01/02/03 au 30/07/03. Technical report, INRIA Sophia Antipolis, July 30, 2003.

APPENDIX A

fluid.sinus Formats

Table 7 shows various formats for the *fluid.sinus* mesh file where *ns*, *nt*, *nfac* are the number of vertices, tetrahedra, and external faces, *coor*, the Cartesian coordinates, *nu* the tetrahedra connectivity, *logfac* the boundary conditions to be applied at the external faces, and *nsfac*, the connectivity for the triangular external faces.

```
write(myunit,*) ns, nt, nfac
write(myunit,*) (coor(1,i), coor(2,i), coor(3,i), i=1,ns)
write(myunit,*) (nu(1,i), nu(2,i), nu(3,i), nu(4,i), i=1,nt)
write(myunit,*) (logfac(i), i=1,nfac)
write(myunit,*) (nsfac(1,i), nsfac(2,i), nsfac(3,i), i=1,nfac)

write(myunit) ns, nt, nfac
write(myunit) (coor(1,i), coor(2,i), coor(3,i), i=1,ns)
write(myunit) (nu(1,i), nu(2,i), nu(3,i), nu(4,i), i=1,nt)
write(myunit) (logfac(i), i=1,nfac)
write(myunit) (nsfac(1,i), nsfac(2,i), nsfac(3,i), i=1,nfac)
```

Table 7: *fluid.sinus* formats: Top) formatted Bottom) unformatted

APPENDIX B

Description of the *flu-0000n* files.

In the flu-0000n we find successively:

ipd = subdomain number (involved by this file flu-xxxxx)
ns, nt, nfac = local number of nodes, tetrahedra and boundary facets
(local means associated with the considered subdomain)
nghd = number of neighboring subdomains

For i=1,nghd we read:

ishd(i) = identification (number) of the ith neighboring subdomain
insghd(ishd(i)) = number of nodes located on the common interface between the considered subdomain and its ith neighboring subdomain
For ii=1,insghd(ishd(i)) we read:
isghd(ii,ishd(i)) = local number of these common nodes
EndFor ii

EndFor i

For is=1,ns we read:
coor(1,is), coor(2,is), coor(3,is) = x-,y- and z-coordinate of node is

EndFor is

For jt=1,nt we read:
nu(1,jt), nu(2,jt), nu(3,jt), nu(4,jt) = local number of the 4 vertices
of tetrahedron jt
EndFor jt

For ifac=1,nfac we read
logfac(ifac) = boundary identification for each boundary facet ifac
nsfac(1,ifac), nsfac(2,ifac), nsfac(3,ifac) = local number of the 3 vertices
of boundary facet ifac
EndFor ifac

For is=1,ns
irefd(is) = 1 if node "is" is an internal node for the considered subdomain,
0 else
(by internal node, we mean a node which does not belong to the common
interfaces shared with neighboring subdomains)
EndFor is

For is=1,ns
igrefd(is) = global number (i.e. number in the global mesh) of node is
EndFor is

REMARK = all the previous variables are integer except coor(,) which is real.

Unité de recherche INRIA Sophia Antipolis

2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes

4, rue Jacques Monod - 91893 ORSAY Cedex (France)

Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique

615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)

Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)

Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)

Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)

Éditeur

INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)

http://www.inria.fr

ISSN 0249-6399