Load Balancing for CFD Applications in Grid Computing Environment

Soon-Heum Ko*, Chongam Kim and Oh-Hyun Rho****
School of Mechanical and Aerospace Engineering
Seoul National University, Seoul, Korea, 151-742

Kum Won Cho***
KISTI Supercomputing Center, Daejeon 305-806, Republic of Korea

Abstract

The Grid is a communication service that collaborates dispersed high performance computers so that those can be shared and worked together. It enables the analysis of large-scale problem with the reduction of computation time by collaborating high performance computing resources in dispersed organizations. Thus, the present paper focuses on the efficient flow calculation using the Grid. To increase parallel efficiency, a simple load balance algorithm for the Grid computing is proposed and applied to various aerodynamic problems.

Key Word : Grid, Load Balancing

Introduction

According to Moore's law, computer speed doubles in every 18 months. In proportion to the development of computing resources, the problem size in CFD(Computational Fluid Dynamics) has been remarkably expanded. However, even now, a lot of problems require too enormous computing power to be analyzed using single parallel cluster. As an alternative proposal, the concept 'Grid' was planned and is on research now[1, 2].

It is obvious that the Grid enables a researcher to analyze large-scaled aerodynamic problems, for example, integral analysis of an airplane and detachment motion analysis of multi-stage launch vehicle. However, diverse communication speed among computing resources and various performances of processors can reduce parallel efficiency in the Grid.

Therefore, the present research focuses on the efficient flow calculation in the Grid. As an investigation of load balancing in the Grid, a simple load balance algorithm is proposed and applied to various problems. For the validation of proposed algorithm, the flowfield around a 3-D tangent-ogive cylinder is analyzed by collaborating multiple computing resources in Aerodynamic Simulations Laboratory, Seoul National University. And then, developed load balance algorithm is expanded to be applicable to CFD problems with domain decomposition techniques; e.g., multiblock technique and Chimera overset technique. As an application to multiblock problem, the flowfield around a 3-D boattail configuration is analyzed, and, a multi-element wing is analyzed as an application to Chimera overset mesh. From the results, proposed algorithm partitions the computational domain considering the performance of each processor and communication speeds among computing resources. And the results show the validity of proposed algorithm.

* Ph. D Candidate, Aerodynamic Simulation Laboratory, Seoul National University
** Professor, Seoul National University
E-mail : chongam@plaza.snu.ac.kr, Tel : 02-880-1915, Fax : 02-887-2662
*** Researcher, KISTI Supercomputing Center

GOVERNING EQUATIONS & NUMERICAL SCHEMES

The three-dimensional compressible thin-layer Navier-Stokes Equations are adopted as governing equations. As a spatial discretization, AUSMPW+(modified Advection Upstream Splitting Method Press-based Weight function)[3] has been applied and turbulent viscous components are evaluated using the k-ω SST(Shear-Stress Transport) turbulence model[4]. LU-SGS (Lower-Upper Symmetric Gauss-Seidel) scheme[5] is used as the implicit time integration method.

A single-block mesh with 71×91×81 mesh points is used for the analysis of a tangent-ogive cylinder, a boattail problem is analyzed by using multiblock mesh with mesh sizes of 81×25×81 and 81×25×161, and Chimera overset mesh for multi-element wing analysis has 249×37×81 mesh points as a main grid and 125×37×41 mesh points for a flap.

INVESTIGATION OF PARALLEL EFFICIENCY IN THE GRID

There have been many researches on the development of load balance algorithm for parallel computation. [6,7,8] However, any algorithm cannot be applied to all applications as load balancing is a NP-hard problem and the optimal parallelization depends on the characteristics of the problem itself. Thus, applicable references for efficient parallelization of CFD problems are deficient. [9,10]

Especially, in the Grid where dispersed computing resources collaborate, consideration of communication speeds among processors should be added for load balancing. Therefore, load balance algorithm for the Grid should satisfy the minimization of communication time among resources as well as optimal load distribution considering the capacity of each processor. And the algorithm should be simple enough for many researchers with little knowledge of computer science to implement easily.

Thus, a load balancing algorithm is developed to satisfy the requisites mentioned above. To have optimal load distribution, all processors execute the same amount of calculations and check their own computation time. And the reverse of computation time is prescribed as the performance of each processor. Reduction of communication time is achieved by applying the 'Grouping Method'. Each processor communicates with all other processors and the processors with relatively faster link are considered as a group. And then, the whole computational domain is firstly partitioned to the group level and then partitioned again to each processor. It will increase the possibility that a processor is located near the processors with faster link, reducing the communication time in the inter-processor boundary. Finally, performance test is accomplished by using the same flow solver as the analysis solver, and communication time is checked by sending the residual computed from performance test. Therefore, the algorithm can be easily implemented to any CFD solvers simply by appending the grouping process and partitioning process to the solver.

To explain load balancing process easily, some assumptions are added. Firstly, all the clusters are located in dispersed organizations. It means that each cluster is considered as a single group as communication speed between resources in different parallel clusters is much slower than the communication speed between resources in the same cluster and the number of groups will be the same as the number of collaborating clusters. Secondly, a cluster is composed of many nodes that have the same capacity. It means that the performances of processors in the same group will be the same. Finally, a rectangular computational domain is allotted to each processor. Even though the complicated partitioning method as graph partitioning may increase parallel performance slightly, it requires severe efforts at generating partitioning routines and modifying boundary conditions routines. Thus, the whole computational domain is partitioned to be rectangular domains.

Procedure of the algorithm is as follows.

Algorithm : Load Balance Algorithm in the Grid

C	: Total Number of Clusters	t_M	: Computation Time of Processors in the M^{th} Cluster
M	: Cluster Number(M=1, ⋯ , C)	PF_{PN}	: Performance of Processor
N_M	: Number of Processors in Each Cluster	W_{PN}	: Amount of Work Allotted to Each Processor
TW_M	: Total Work Allotted to Each Cluster	I×J×K	: Size of Computational Domain
PN	: Processor Number (PN=1, ⋯ , N_1+⋯+N_C)	T_1, T_2	: Team Number

0. Supposition : PN from 1 to N_1 belong to 1st cluster and from N_1+1 to N_1+N_2 to 2nd cluster, and so on. Cluster number and group number are identical, meaning that the processors in the M^{th} cluster are M^{th} group.

1. Performance Test : Each processor executes several iterations of flow solver using the same computational domain(not applying boundary conditions) and check computation time, t_M. Then the performance of each processor is:

$$PF_{PN} = \frac{1}{t_M} \text{ , where } (N_1+\cdots+N_{M-1})+1 \le PN \le (N_1+\cdots+N_M) \text{ , } M=1,\cdots,C$$

And the capacity of M^{th} cluster, TW_M is defined as $TW_M = N_M \times PF_{PN}$, the multiplication of number of processors in that cluster and the performance of each processor in that cluster. And, the value of TW_M in this step is not the real amount of load to the cluster, but factors.

2. Communication Test : Each processor has no information about which cluster it belongs to. Thus, communication test is required to judge the cluster number the processor is located. To find out the cluster number, each processor communicates with all other processors as mentioned above.

From the result of communication test, grouping process is accomplished. And, each cluster is considered as a group in this case, based on the first assumption above.

3. Arrangement of Groups to Blocks(Optional) : This step is required when the problems with domain decomposition methods are analyzed. The explanation is carried out later on.

4. Partitioning of Computational Domain to Each Group : The whole computational domain is firstly partitioned to each group. Main concept is to split the domain into two zones by considering computing capacity of each team and repeat the splitting process until each group has its own domain. And the splitting process is as follows.

4.1 Domain Splitting into 2 Teams : Find the longest section among I, J and K(Suppose K is maximum.). Then the whole domain $I \times J \times K$ will be distributed into two parts by splitting K into K_1 and K_2, and each part will have the domain of $I \times J \times K_1$ and $I \times J \times K_2$.

And, combine all groups into two teams. For example, when 3 different groups exist (in this case, it means that 3 clusters are used as the composition of cluster is identical to the composition of group), there can be $\frac{2^3-2}{2}$ cases. And each team will have the capacity, T_1 and T_2, as shown below :

Team 1	Team 2
$T_1 = TW_1$	$T_2 = TW_2+TW_3$
$T_1 = TW_2$	$T_2 = TW_3+TW_1$
$T_1 = TW_3$	$T_2 = TW_1+TW_2$

In general, there exist $\frac{2^C-2}{2}$ different cases.

The optimal load balancing takes place when the load ratio of two teams, $\frac{T_1}{T_2}$ is equal to the ratio of splitted domain size, $\frac{K_1}{K_2}$. However, it is impossible as K_1 and K_2 are integer values and they have the constraint that $K_1+K_2=K$. Thus, a test parameter is introduced to find an optimal case :

$$Test=|K\times\frac{T_1}{T_1+T_2}-[K\times\frac{T_1}{T_1+T_2}]|\times|K\times\frac{T_2}{T_1+T_2}-[K\times\frac{T_2}{T_1+T_2}]|$$

where [] means Gaussian operator and || means absolute value.

The optimized grouping is when '*Test*' is minimized. And, at that time, K_1 and K_2 are

$$K_1 = round(K\times\frac{T_1}{T_1+T_2}),\ K_2 = round(K\times\frac{T_2}{T_1+T_2})$$

where *round* means round function.

And, group 1 and group 2 have total work of $I\times J\times K_1$ and $I\times J\times K_2$, respectively.

If 2 different clusters are used, $K_1 = round(K\times\frac{TW_1}{TW_1+TW_2})$ and $K_2 = round(K\times\frac{TW_2}{TW_1+TW_2})$. And, real work loaded in each cluster is $TW_1 = I\times J\times K_1$ and $TW_2 = I\times J\times K_2$.

4.2 Distribution of Work to Each Cluster System : Processes in Step 4.1 are repeated in every team until all cluster systems have their own total work.

5. Assignment of Work to Each Processor : Work allotted to each cluster is divided into all processors in the same cluster system. It is achieved simply by applying the method in step 4 to the processor level. Then, work assigned to each processor is calculated as follows:

$$W_{PN} = \frac{TW_M}{N_M} \text{ where } N_1+\cdots+N_{M-1}+1\le PN\le N_1+\cdots+N_M,\ M=1,\cdots,C$$

6. Flow Analysis : Flow solver starts.

Now, the procedure in step 3 is to be described. The procedure is very similar to step 4 and only redistribution of groups is required.

3.1 Allocation of Groups into Each Block : The same process as Step 4 is carried out to allocate computing resources to each block firstly. In this process, the ratio of T_1 and T_2 should be alike to the ratio of S_1 and $S_2 + \cdots + S_n$, where S_b means the size of b^{th} block. That is,

$$Test=|(S_2+\cdots+S_n)\times T_1-S_1\times T_2|$$

should be minimized.

3.2 Division of A Group into 2 Groups : To optimize *Test* value, some processors in one group are assigned to other block. Migrated processors are considered as a new group and they are withdrawn from former one. Therefore, total number of groups increase by one and the capacities of groups, TW_{former} and TW_{new} are computed again considering redistributed number of processors, N_{former} and N_{new}.

These processes are iterative until all blocks are endowed with their own computing resources.

Even though it is assumed that each cluster constitutes a group, that assumption is introduced only for convenient explanation of the algorithm. As for the first assumption, there is no problem when many clusters in one organization collaborate with the clusters in different organizations. The only standard for the algorithm to classify the resources into groups is the relative differences of communication speeds. Therefore, if multiple clusters in one organization are used, the algorithm will judge that clusters as one group. And, the violation of the second assumption doesn't make

any trouble, either. Each processor checks its performance in the first step. Thus, the capacity of each group is easily obtained by the summation of performances of processors in that group.

Finally, developed algorithm has the following characteristics.

1. Optimal Balancing of Loads : Each processor fulfills the performance test and the allocation of load is accomplished based on its own performance. Especially, performance test is carried out using the flow solver of the user. Therefore, the result of load balancing is optimized for that solver.

2. Reduction of Communication Time : By grouping the resources with faster link, the algorithm increased the possibility to reduce the communication time in the inter-processor boundary. However, it is not optimized result to minimize communication time. Change of arrangement of resources and modification of partitioning technique may improve communication efficiency.

3. Simplicity of the Algorithm : Both performance test and communication test is accomplished by using existing flow solver. Therefore, only grouping routine and partitioning routine are to be added to the existing flow solver.

RESULTS

Validation

As a validation problem of the proposed load balance algorithm, a tangent-ogive cylinder is analyzed. Mesh is shown in Fig. 1 and parallel computations are carried out using computers in Aerodynamic Simulations Laboratory, Seoul National University. The result of flow analysis is shown in Fig. 2, showing the validity of the flow solver. Performances of collaborating resources are shown in table 1.

For the collaboration of resources in WAN(Wide Area Network), Globus[11] is installed to each resource and, parallel processing is executed using MPICH-G2[12], the Grid-enabled version of MPICH.

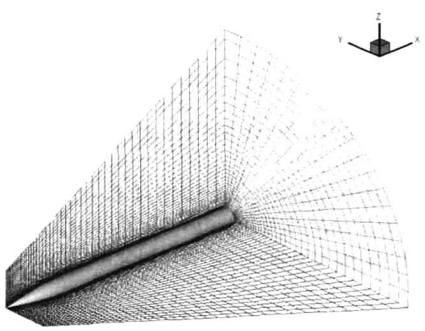


Fig. 1. Single-block Mesh around a Tangent-ogive Cylinder

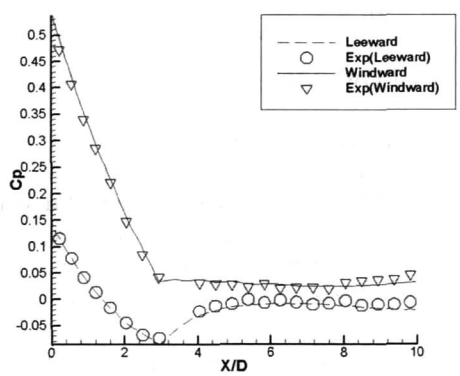


Fig. 2. Cp of a Tangent-ogive Cylinder ($Ma=1.98$, $Re=0.39*10^6$, $AoA=10^\circ$)

Table 1. Computing Resources

	CPU(s)	RAM
SPM cluster(4 Nodes / 8 Procs)	P-III, 933 MHz	512 MB
fastcfd2 PC(2 Procs)	P-IV, 1.4 GHz	1 GB
mana PC(1 Proc)	P-III, 866 MHz	256 MB
fastcfd4 PC(1 Proc)	P-III, 500 MHz	512 MB

To show the validity of proposed algorithm, elapsed time of flow solver with load balancing is compared with the result without load balancing. In imbalanced case, the whole domain is explicitly partitioned with the same sized and partitioned domain is allotted to each processor. To see the performance of the algorithm, 3 different computations are performed. Firstly, as a reference case, flow solver iterates with no modification. Secondly, to check communication time of inter-processor boundaries, boundary communication routine is erased from the flow solver and computation is performed. Finally, to show the validity of load balancing, convergence check routine is deleted and time for pure calculation is checked. To compare the results, all computations do 1000 iterations. 12 processors are collaborated and the result of mesh allocation to the processors is shown in table 2. As collaborating resources are located in the LAN(Local Area Network), machine-to-machine communication is the slowest link and, each machine is considered as a group after the grouping process. Even though SPM is a single cluster with 4 nodes, all nodes in the cluster are considered as different groups as node-to-node communication speed is far slower than processor-to-processor communication speed inside the same node.

Table 2. Allocation of Load to Processors

Initial Group	PE (Processing Element)	Size	Ratio
GRP01 (SPM Node01)	PE00	27*48*31=40176	1.69
	PE01	27*48*31=40176	1.69
GRP02 (SPM Node02)	PE02	23*42*42=40572	1.70
	PE06	24*42*42=42336	1.78
GRP03 (SPM Node03)	PE03	35*48*24=40320	1.69
	PE04	35*48*25=42000	1.76
GRP04 (SPM Node04)	PE05	35*48*24=40320	1.69
	PE08	35*48*25=42000	1.76
GRP05 (fastcfd2)	PE07	35*42*38=55860	2.34
	PE09	35*42*38=55860	2.34
GRP06 (mana)	PE10	23*42*42=40572	1.70
GRP07 (fastcfd4)	PE11	16*48*31=23808	1

The validity of proposed algorithm is confirmed from the results shown in table 3. In table 3, computation times for 3 different analyses are presented and, the reductions of calculation time and communication time are obtained from these data. Firstly, communication time in inter-processor boundary is calculated from the difference of analysis 1 and 2. From the results, communication time in proposed algorithm is about 14.1% more than the result without load balancing. It is unavoidable that present algorithm partitions the whole mesh unequally and some processors with better performances will have more inter-processor boundary area.

Elapsed time for convergence check is calculated from the difference of second and third analysis. And the results of 2 tests are nearly the same. It is natural in the sense that the amount of communication in convergence check part is directly proportional to the number of processors.

Now, the results of pure calculation time show the validity of proposed algorithm. The algorithm is ideal if the reduction ratio of calculated time between 2 test cases is proportional to the reduction rate of domain size allotted to the lowest-performanced processor. As the performance of PE11 is

lowest of all, PE11 requires the longest calculation time in the analysis without load balancing. And elapsed time is about 1319 seconds. However, in the present algorithm, elapsed time for the calculation is only about 802 seconds. As the ratio of domain size to the worst processor(PE11) is 56.7% and the ratio of calculation time is about 60.8%(=802/1319), the algorithm is appropriate.

Table 3. Variation of Computation Time

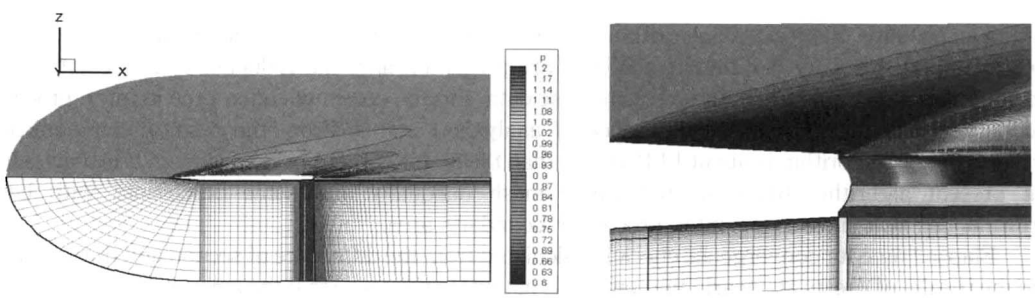
Test Case / Analysis	With Load Balancing(①)	Without Load Balancing(②)
Full Analysis (①)	1438.620	1883.366
No Boundary Comm. (②)	904.344	1415.335
Pure Calculation (Worst) (③)	801.787	1318.729

Analysis in the LAN

As an application problem of proposed load balance algorithm, a boattail configuration with a propulsive jet is analyzed.[13] A multiblock mesh is generated for the analysis of the present problem. The first block is generated along the body of the configuration, while the second block represents downstream of the body. Sizes of blocks are 81×25×81 and 81×25×161, respectively. And, generated computational mesh and pressure contour are presented in Fig. 3. 1000 iterations are performed and elapsed times of present algorithm and uniform distribution are compared to show the validity in the problem with domain decomposition method. The result of mesh distribution to the processors is shown in table 4.

Table 5 shows computation times for 3 analyses as explained in previous chapter. Firstly, communication time in inter-processor boundary is increased a little. From the results, communication time in proposed algorithm is about 11.9% more than the result without load balancing. And, elapsed time for convergence check shows nearly the same result between 2 tests. Finally, the result of calculation time is decreased a lot as expected. In the present composition of computing resources, PE02 shows the lowest performance. Therefore, PE02 requires the longest calculation time in imbalanced test case, about 1317 seconds. In the present algorithm, time for the calculation of PE02 (located in the 2nd block) is only about 774 seconds. As the load to PE02 is 58.5% of the second case and the ratio of computation time is about 58.7%(=774/1317), it is guaranteed that the present algorithm balances the load.

However, load imbalances between processors in different blocks are shown in the result. For example, the load to PE00 is about 7% more than PE10 even though they have the same performance and, time consumption of processors in the 1st block is 6.7% more than elapsed time in the 2nd block. It is because a processor should be positioned inside a single block in the present algorithm. Therefore,



a) The Whole Flowfield b) Near Base Region

Fig. 3. Pressure Contour and Multiblock Mesh of a Boattail Problem ($Ma = 2.0$, $Ma_{exit} = 2.5$, $P_{exit} = 3.0*P_{\infty}$)

load imbalance between blocks takes place. This phenomenon will be insignificant as the number of processors increases, while it will cause trouble if the number of blocks is relatively excessive.

Table 4. Allocation of Load to Processors

Block	Initial Group	PE	Size	Ratio
1	GRP01 (SPM Node01)	PE00	40*24*40=38400	1.71
		PE01	40*24*40=38400	1.71
	GRP04 (SPM Node03)	PE05	40*24*40=38400	1.71
		PE06	40*24*40=38400	1.71
2	GRP03 (SPM Node02)	PE03	40*24*38=36480	1.62
		PE04	40*24*38=36480	1.62
	GRP06 (SPM Node04)	PE09	36*24*42=36288	1.62
		PE10	44*24*34=35904	1.60
	GRP04 (fastcfd2)	PE07	40*24*54=51840	2.31
		PE08	40*24*54=51840	2.31
	GRP02 (fastcfd4)	PE02	36*24*26=22464	1
	GRP07 (mana)	PE11	44*24*34=35904	1.68

Table 5. Variation of Computation Time

Analysis \ Test Case	With Load Balancing (①)	Without Load Balancing (②)
Full Analysis (①)	1286.3	1741.4
No Boundary Comm. (②)	921.3	1415.2
Pure Calculation (Worst) (③)	(Block 1) 825.6 (Block 2) 773.6	(PE02) 1317.0

As the second application, flow around multi-element airfoil is analyzed using Chimera overset mesh. NLR 7301 airfoil is stretched to spanwise direction with taper ratio and wingtip is smoothened. Surface mesh with streamlines is shown in Fig. 4. Main grid is four times larger than sub-grid in mesh size and, 10 processors are collaborated to compare present result with the analysis without load balancing. Result of domain partitioning is shown in table 6.

The results of comparison between two cases are shown in table 7. Firstly, communication time in the boundary is decreased about 3.8% in proposed algorithm. It seems the result is contradictory to the result presented in the analysis of multiblock. However, it is feasible in the sense that communication for domain connectivity takes place as well as inter-processor boundary communication in Chimera grid technique. It is clear that some processors with better performances have more inter-processor boundary area and they require more communication time. But, communication cost for domain connectivity solution is basically load-imbalanced as donor cells and fringe cells are concentrated to some specific positions and sometimes better communication cost can be gained when superior processors contain less donor and fringe cells.

Time for convergence check in present algorithm is increased about 2 times. It is by the reason of synchronization problem. Computation time per iteration can be varied even though same calculations

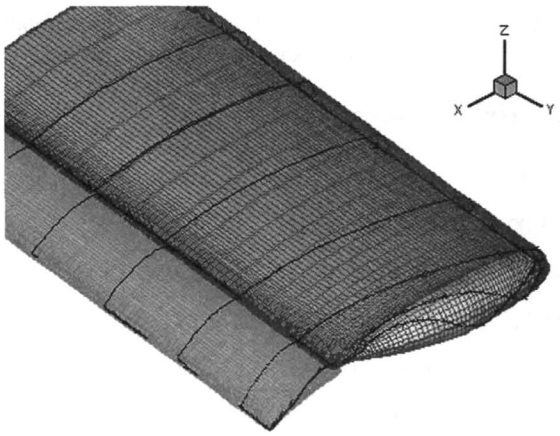


Fig. 4. Surface Mesh with Streamlines (Ma=0.185, Re=2.51*10^6)

are performed in the processor and, it can cause this kind of result. However, increase of computation time based on synchronization problem is a very small portion of total computation time, not more than 2%.

Finally, calculation time is about 79.6% comparing to imbalanced case. As the ratio of load assigned to the lowest-performanced processor(PE09) between 2 cases is 61.3%, pure calculation time of the present analysis should be about 60% to 65% comparing to the second case. However, reduction of computation time is not as good as expected and it is presumed that load imbalance is present in domain connectivity calculation. Totally, proposed algorithm gained 17.5% increase of parallel efficiency comparing to the result without load balance. However, in the analysis of Chimera overset mesh, the efficiency of present algorithm is not as good as expected. Mainly it is caused by the absence of optimized parallel algorithm for Chimera overset mesh and a propound investigation on parallel algorithm for Chimera mesh is keenly required.

Table 6. Allocation of Load to Processors

Grid System	Initial Group	PE	Size	Ratio
Main Grid (wing)	GRP01 (SPM Node01)	PE00	62*36*40=89280	1.63
		PE01	62*36*40=89280	1.63
	GRP02 (SPM Node02)	PE02	62*36*40=89280	1.63
		PE03	62*36*40=89280	1.63
	GRP04 (fastcfd2)	PE05	75*37*46=126900	2.31
	GRP05 (SPM Node04)	PE06	75*36*33=89100	1.62
	GRP06 (mana)	PE08	49*36*49=86436	1.58
	GRP07 (fastcfd4)	PE09	49*36*31=54864	1
Sub-grid (flap)	GRP03 (SPM Node03)	PE04	62*36*40=89280	1.63
		PE07	62*36*40=89280	1.63

Table 7. Variation of Computation Time

Analysis \ Test Case	With Load Balancing (①)	Without Load Balancing (②)
Full Analysis (①)	52068	63120
No Boundary Comm. (②)	47189	58050
Pure Calculation (Worst) (③)	45621	57304

Analysis in the WAN

Now, the algorithm is applied to the collaboration of resources in the Grid. A multiblock problem of boattail configuration is analyzed collaborating three different clusters in K* Grid. Totally 12 processors are collaborated and 14431 iterations are executed until flow solver converges. Capacities of resources are presented in table 8.

Table 8. Computing Resources in the Grid

	CPU(s)	RAM / Node	Number of Nodes
Venus(KISTI)	P-IV, 2.0 GHz	512 MB	64
Jupiter(KISTI)	P-IV, 1.7 GHz	1 GB	16
Hush1(Postech)	P-III, 733 MHz	128 MB	24

Elapsed time until convergence is seen in Fig. 5. The result presented in Fig. 5 mainly shows 2 features of the Grid. Firstly, the sharp decrease of communication speed between processors in the WAN reduces parallel efficiency a lot. It is evident when 5^{th}(Collaboration of 6 processors in Venus, 4 in Jupiter and 2 in Hush1) and 6^{th}(8 in Venus, 4 in Jupiter collaborating) cases are compared, where the former collaborates all resources in the WAN, while the latter uses the resources in the same LAN. As the performance of collaborating resources in the 6^{th} case is better, it is natural that computation time of the 6^{th} case is less than that of 5^{th} case. However, increase of elapsed time from 6^{th} to 5^{th} case is excessively large, nearly 30%. It means that the collaboration should be performed when the use of network is scarce, e.g., nighttime, unless collaborating organizations are connected with exclusive lines.

And, from the comparison of 1^{st} and 2^{nd} case, it is assumed that addition of computing resources from other organization can increase computation time. Performance of resources is much better in the 2^{nd}(Collaboration of Jupiter and Hush 1) case than the 1^{st}(Sole Use of Hush1) case. However, decrease of computation time is slight. It is because collaborating resources in the 2^{nd} case are connected using WAN and, sometimes increase of the whole resources can increase elapsed time. Therefore, 'Killing Mechanism' , the judgment whether to add some resources to the current composition of resources, should be imported to the present algorithm.

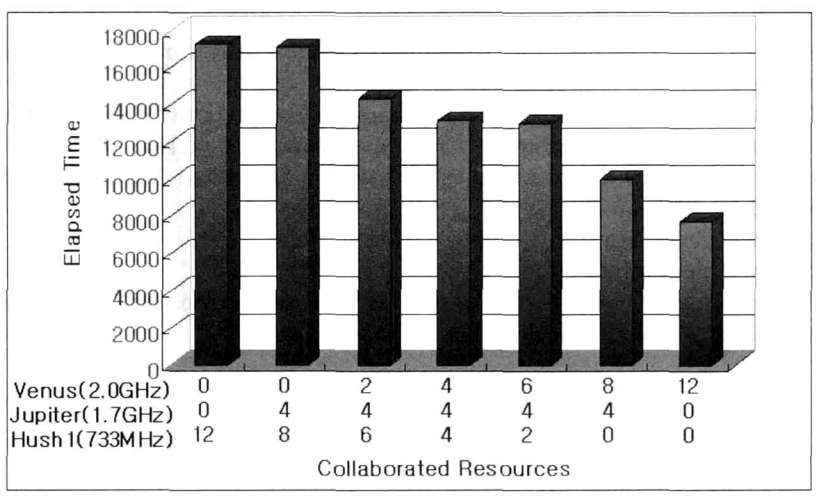


Fig. 5. Elapsed Time of Flow Solver

Conclusions

A load balance algorithm for CFD applications in Grid computing is proposed and applied to various CFD problems. The algorithm considers both various performances of processors and different communication speeds among resources. And the algorithm automatically partitions the whole domain to each processor. Developed algorithm is validated by analyzing a tangent-ogive cylinder in the LAN and applied to the problems with domain decomposition methods. From the results, proposed algorithm shows a good load balance and guarantees an appropriate increase of parallel efficiency.

Acknowledgement

The present work was supported by KISTI(Korea Institute of Science and Technology Information). Authors are grateful for their supports in many respects.

References

1. W. E. Johnston, D. Gannon and B. Nitzberg, 2001, *Information Power Grid Implementation Plan: Research, Development, and Testbeds for High Performance, Widely Distributed, Collaborative, Computing and Information Systems Supporting Science and Engineering* , NASA Ames Research Center, Version 2.0.

2. M. M. Resch, 2000, "Metacomputing in High Performance Computing Center," *IEEE 0-7659-0771-9/00*, pp. 165-172.

3. Kyu Hong Kim, Chongam Kim and Oh Hyun Rho, 1998, "Accurate Computations of Hypersonic Flows Using AUSMPW+ Scheme and Shock-aligned-grid Technique," *AIAA Paper 98-2442.*

4. F. R. Menter, 1994, "Two-Equation Eddy-Viscosity Turbulence Models for Engineering Applications," *AIAA Journal,* Vol. 32, No. 8, pp. 1598-1605.

5. Yoon, S., and Jameson, A., 1988, "Lower-Upper SymmetricGauss-Seidel Method for the Euler and Navier-Stokes Equations," *AIAA Journal*, Vol. 26, No. 9, pp. 1025-1026.

6. C. Hui and S. Chanson, 1996, *Theoretical Analysis of the Heterogeneous Dynamic Load Balancing Problem Using a Hydro-Dynamic Approach*, Technical Report HKUST-CS96-01.

7. M. H. Willebeek-LeMair and A. P. Reeves, 1993, "Strategies for Dynamic Load Balancing on Highly Parallel Computers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, pp.979-993.

8. G. Cybenko, 1989, "Dynamic Load Balancing for Distributed Memory Multiprocessors," *Journal of Parallel and Distributed Computing*, vol. 7, pp.279-301.

9. Y. P. Chien, et. al, 1995, "Load-balancing for Parallel Computation of Fluid Dynamics Problems," *Computer Methods in Applied Mechanics and Engineering*, 120, pp.119-130.

10. J. Rantakokko, 2000, "Partitioning Strategies for Structured Multiblock Grids," *Parallel Computing*, Vol.26, pp.1661-1680.

11. http://www.globus.org.

12. http://www.niu.edu/mpi.

13. G. S. Deiwert, 1984, "Supersonic Axisymmetric Flow over Boattails Containing a Centered Propulsive Jet," *AIAA Journal*, Vol.22-10, pp. 1358-1365.