MULTI-PLATFORM INTEGRATED DEVELOPMENT AID SYSTEM (MIDAS)

Young-Ki Lee, Holger Pfaender, and Taeyun P. Choi
Aerospace Systems Design Laboratory (ASDL)
Georgia Institute of Technology, Atlanta, GA 30313, USA

Keywords: *Distributed / Parallel / Grid Computing, JavaSpaces*

Abstract

The development of a Multi-platform Integrated Development Aid System (MIDAS), which aims to facilitate the task of implementing distributed simulations for aircraft design applications, is outlined. A series of benchmarking tests confirm the near linear scalability of computational performance attainable with MIDAS. The results of solving a simple computational fluid dynamics (CFD) problem in a distributed computing environment are included to showcase the benefits of employing distributed simulation for computationally demanding tasks.

1 Motivation

In recent years, multi-disciplinary aircraft design and simulation environments have required progressively greater computing capability. This is an unavoidable consequence of attempting to increase the quality of various disciplinary analyses by incorporating high-fidelity simulation tools into the early phase of conceptual design.

Unfortunately, such implementations are highly likely to extend the execution time of each computational run to a prohibitive level for large scale design problems. In order to prevent simulation from becoming the major bottleneck of the overall design cycle, one can either upgrade to hardware with enhanced capability or take advantage of the aggregate computing power of multiple machines. The latter approach, which is known as "distributed simulation technology,"[1] has been attempted in aerospace engineering through the employment of dedicated, high-performance computing clusters.[2] Since such costly computation infrastructures are often difficult to gain access to or financially infeasible, a more practical and cost-effective solution would be to create a distributed computing environment by interconnecting existing standard desktop computers. Nevertheless, an accessible framework, which can be easily used without an in-depth knowledge of parallel and network programming, to create such an ad hoc computing cluster for aircraft design applications has remained elusive.

Motivated by these limitations, the authors created the Multi-platform Integrated Development Aid System (MIDAS), which is aimed to be a user-friendly framework that enables the straightforward execution of a simulation task over a network of geographically scattered computers. This design goal is conceptually represented in Fig. 1. The abscissa of the figure represents the time a designer must invest in

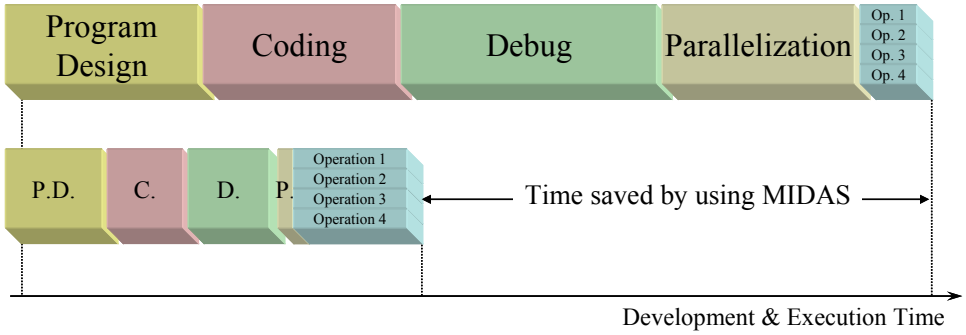

Fig. 1. Improvement in Program Development and Execution Process through MIDAS

order to learn, develop, and execute a program in a distributed computing environment. As it can be seen, even a novice of distributed simulation is projected to considerably shorten the program development and execution process when using MIDAS.

2 Design and Development of MIDAS

With recent advances in the computational power of standard desktops, the collective computing capability of a large scale network of such computers can rival that of a supercomputer. Initial research into the matter of distributed computing revealed that Java™ was designed as a network language from its conception. Not only does Java™ include the necessary network Input/Output (I/O) libraries, but also it guarantees platform independence and network security through the built-in implementation of Java virtual machine (JVM) [3]. Additionally, higher level network functionality such as web servers and database servers are either already included in the basic libraries or at the very least have a programming interface that can be utilized. Furthermore, Java™ includes a functionality called dynamic class loading that allows a JVM to load the code and data during the execution. A security model, which grants access to the dynamically loaded classes, ensures that it is not possible for users to inadvertently or maliciously access, delete, or send unauthorized data across the network. In light of these favorable features, Java™ was selected as the programming language for MIDAS.

Further background research led to the discovery of an enabling Remote Method Invocation (RMI) technology called JavaSpaces™. It is an implementation of a persistent "object" storage called a "space."[4] This space provides a logically shared memory where data can be stored, accessed, and updated in real-time without requiring a physically shared memory. Therefore, the application of JavaSpaces™ provides a basis for MIDAS to create a flexible, scalable, and reliable master-and-worker type distributed computing environment across a local area network (LAN), as shown in Fig. 2.

Shown here are the workers, generic computation engines, which can search for, receive and run any tasks placed in the space by the master. Each task is implemented as an "entry" into the space [4]. "Entry" is the implementation of a generic object that the JavaSpaces™ Application Programming Interface (API) uses to store it in the space. In this context, each task is an object that contains the data as well as the necessary reference executable code to complete the computational job. The workers wait for entries into the space that match the template of a task. Once a task is found, it is removed from the space and its local copy is created on the worker machine. Subsequently, the worker dynamically loads the remote code into its native JVM and executes it. After the job is completed, the worker writes the result back to the space as another instantiation of an entry. Each result only contains a number of data objects storing the result values. Although only one master machine was used for this study, the JavaSpaces™ technology allows the interaction between mul-

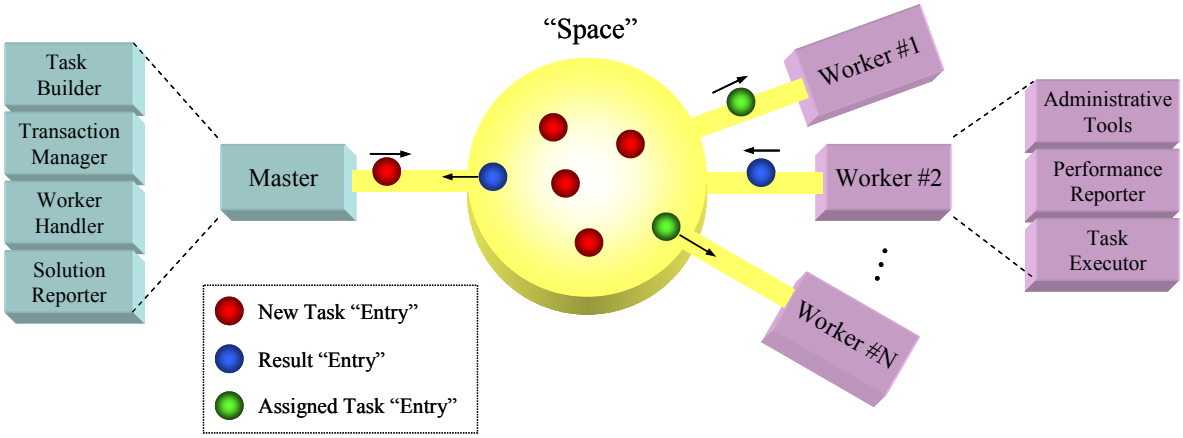

Fig. 2. System Level Architecture of MIDAS

tiple master computers and multiple worker machines, which forms a basis for the expansion of computing power.

The implementation of such a distributed simulation scheme in a space is inherently self-load balancing, because idle workers would continue to take the tasks and return the results until the space is empty of new tasks. Therefore, there is no need for a special assignment mechanism that actively distributes tasks among workers.

The master side is responsible for creating new tasks and collecting the results. This part of the infrastructure consists of modules that relate to user interactions, resource and worker management, and task construction. As long as all machines on the network have the compatible versions of JVM, such an infrastructure is able to distribute the tasks amongst a collection of heterogeneous, multi-platform workers. Functions which automatically add or delete the workers and dynamically adjust the priority of the tasks are also built into the master. The latter feature is especially necessary in order to preserve enough resources for the local users of the worker computers.

3 Distributed Computing Environment

After the initial development of MIDAS was complete, we were able to create and experiment with clusters that consisted of various desktop computers. The computational efficiency of our home-grown distributed computing environment was measured through a series of benchmarking tests.

3.1 Implementation of MIDAS

The initial implementation of MIDAS involved the creation of generalized workers that would simply exist as a Java Archive (JAR) containing the necessary libraries and policy files. After the generic worker connects to the space, as illustrated in Fig. 3, it writes the status information about itself to the space and waits for any available tasks.

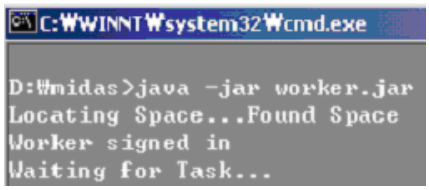

Fig. 3. Screen Shot of MIDAS Generic Worker

In the beginning, the implementation of the master was also done as a simple executable that would obtain information about the connected workers, write the tasks to the space, and then wait until all results were received from the space. For convenience, a generalized graphical user interface (GUI) version of the master was created to let the user easily control the system status (Fig. 4), submit tasks (Fig. 5), and avoid the inconvenience of having to change the entire underlying code every time tasks are to be distributed.

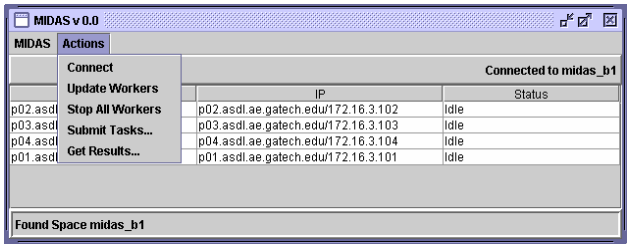

Fig. 4. MIDAS Master Status Screen

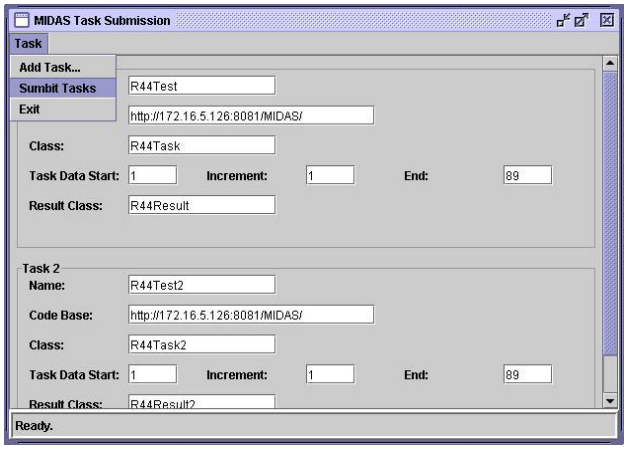

Fig. 5. MIDAS Task Submission Menu

The current implementation of the space, transaction manager, and other network services are those provided by Sun's libraries, and the system is developed using the IncaX IDE Community Edition [5]. For practical reasons, currently the network services all run on the

master, but they are not required to. Each of the network services can be run on separate computers. Nevertheless, this should only be truly necessary under a very heavy computational load. Alternatively, a designated computer can be used as the dedicated provider of the LAN. This has the advantage that all necessary network services will be available continuously. In the future, it is envisioned to incorporate the execution and management of these network services into the GUI to provide an easy-to-use network service manager that can be run independently of the master, but without the use of proprietary management products.

At the time of this writing, it is required to manually write and compile the tasks by using the programming interface as shown in Fig. 6. Nevertheless, once a task is implemented, it can be submitted to the master and the GUI of MIDAS is used for the task's execution and the retrieval of results. As a future enhancement to the GUI, it is envisioned to create an element that contains the programming interface information and allows the creation of a code in a simple text editor. Subsequently, the written code can be automatically compiled through the GUI on the master machine and become immediately available for task execution.

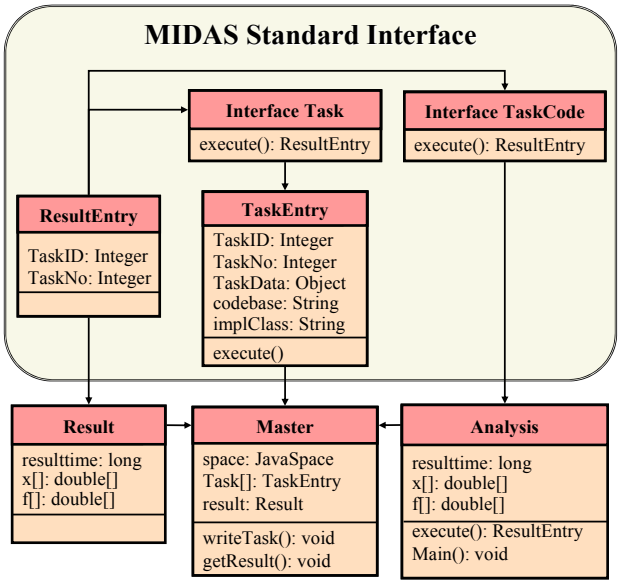

Fig. 6. MIDAS Class Hierarchy

3.2 Benchmarking Tests

Generally, the computational performance of a distributed computing environment is measured by the speed-up, which is given as the ratio of the execution time on a single processor to that of n number of processors, i.e., $SU = T(1) / T(n)$. The ideal case is when the speed-up is identical to the number of machines n. In practice, the speed-up will be less due to the prescribed communication bandwidth ("the amount of data that can be sent from one computer to another through a particular connection in a certain amount of time"[6]) and latencies ("the delay in transmitting a message from one computer to another"[1]). Our goal was to measure this relationship between the reduction in execution time and the number of employed machines.

For this purpose, SciMark 2.0, which is a benchmarking tool for Java™ platforms developed by the National Institute of Standards and Technology [7], was used to measure computational speed in millions of floating point operations per second (Mflops). For this experiment, up to four computers of identical specifications were used to create four separate distributed computing environments. Table 1 lists the average performance each environment spent to finish executing all five algorithms. As expected, the results confirm that the speed-up is linearly proportional to the number of CPUs.

Table 1. Linear Speed-Up vs. Number of CPUs

# of CPUs	1	2	3	4
Mflops (Speed-up)	90.7 (1)	177.5 (1.96)	259.3 (2.86)	338.0 (3.73)

3.3. Efficiency Measurements of Clusters

The purpose of this experiment was to test the effectiveness of MIDAS, in creating an efficient multi-platform computing environment. Eight such clusters of machines were created, as listed in Table 2 from the desktop computers listed in Table A1 of the Appendix.

In order to equitably measure the performance of different machines, a normalized performance index (PI) was created. It is defined in such a way that a baseline PI of 1 indicates the computational capability of a single Pentium IV

Table 2. Clusters of Heterogeneous Machines

Cluster ID	Loads (λ)	# of CPUs	B_i	P_i	$G4_i$	G5	ΣPI
Θ_1	36	6	✓	✓✓✓	✓	✓	9.5
Θ_2	72						
Φ_1	36	8	✓✓✓	✓✓	✓✓	✓	11.2
Φ_2	72						
Ψ_1	36	10	✓✓✓✓	✓✓✓	✓✓	✓	13.8
Ψ_2	72						
Π_1	36	18	✓✓✓✓	✓✓✓✓✓✓ ✓✓✓✓✓	✓✓	✓	27.0
Π_2	72						

Linux machine (B_i's in Table A1). Furthermore, one computational load (1 λ) is defined as a task that takes 96.4 seconds (1 τ) to complete on the baseline machine.

Consequently, it was possible to compare the computational efficiencies of the eight clusters, as depicted in Fig. 7. The ideal τ for each cluster to finish computing a load was obtained by dividing the given λ number by its respective aggregate *PI* value. The solid line in Fig. 7 indicates this ideal case that can only happen if all tasks were evenly distributed to each machine within a cluster with zero latencies. Each cluster's actual τ under different loads was then compared against their ideal τ. The gaps in the horizontal direction of Fig. 7 between the ideal line and the marked points represent the inescapable losses that occur from different machines completing the same unit-load task at different rates.

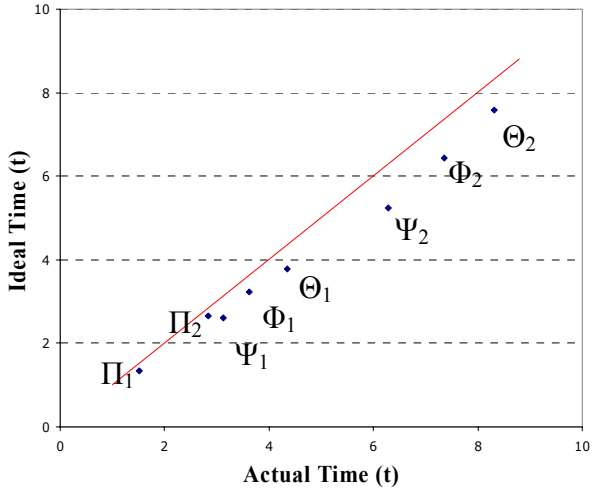

Fig. 7. Computational Efficiencies of Various Clusters

4 Application of MIDAS

The idea of using computational fluid dynamics (CFD) in a distributed computing environment is usually hindered by the complexities of the implementation. In addition, the highly interdependent nature of the problem requires high-bandwidth links between CPUs. This brings in an additional disadvantage of limiting the attainable speed-up in massive environments that are required for complex problems.

Nevertheless, if an aircraft designer's goal is to solve a number of relatively simple CFD cases for the purpose of creating, for example, a meta-model to be used during conceptual design, then the use of a low-bandwidth distributed system would be sufficient. A design problem of such limited complexity is that of predicting temperature exposures on a sharp-edged supersonic airfoil.

4.1 Problem Setup

Various sharp-edged airfoils have been used as control surfaces on supersonic planes. Sharp-edged airfoils produce an attached oblique shock at the leading edge, expansion waves at the convergent aft end, and finally another oblique shock attached to the trailing edge. A CFD problem that utilizes Roe's Flux Difference Splitting (FDS) [8] with a Total Variation Diminishing (TVD) Minmod Limiter [9] in generalized coordinates [10] to perform a 2D-Explicit Upwind Navier-Stokes solution for a structured grid was chosen as an application for the current version of MIDAS. The creation of a grid with good orthogonality that still allows good boundary layer resolution is a key factor in obtaining good results. Fortunately, the

fixed geometry allowed a single grid to be sufficient for all cases. The grid was created manually with the use of an elliptic Poisson equation with control terms [11], and it is shown in Fig. 8.

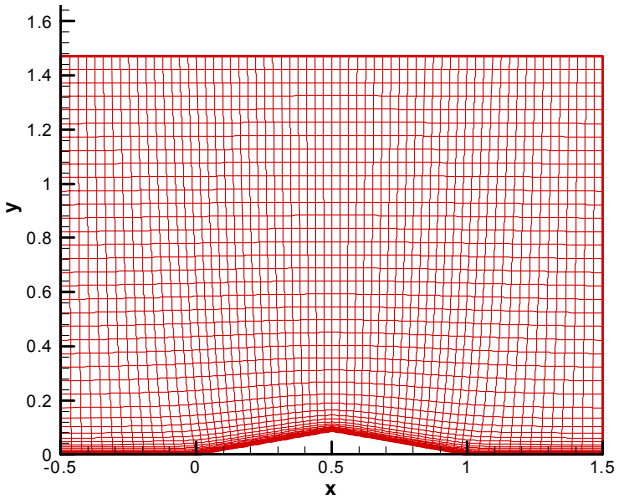

Fig. 8. Grid to Solve CFD Problem of Supersonic Airfoil

A number of cases with varying Mach Numbers and altitude were run to create a database of surface temperatures and temperature gradients for the given supersonic airfoil. By imposing an adiabatic boundary condition, we examined the worst-case scenario that is actually advantageous for designs where active cooling is not desirable because of added complexity.

4.2 Results

As expected, the results of the simulation show an attached oblique shock at the leading edge, followed by an expansion, which is followed by another shock. As shown in Fig. 9, the strength of the shock varies greatly with Mach number, which is evident in how rapidly the shock dissipates away from the airfoil. At Mach 1.4 and an altitude of 18 *km*, the leading edge shock is fairly weak and does not propagate very far before losing a substantial amount of strength. The result for Mach 2.8, however, is much stronger and therefore propagates much farther out.

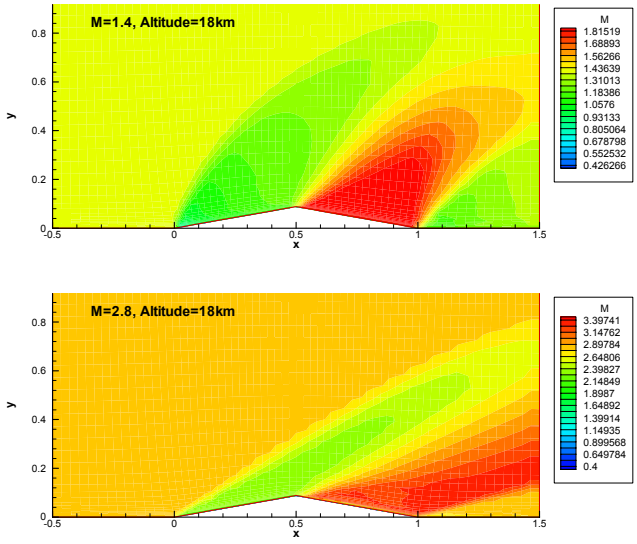

Fig. 9. Contours of Shock and Expansion Waves

As a result, the leading edge temperatures show a strong influence of the Mach number as well, as shown in Fig. 10. The free stream temperature boundary conditions were varied according to the 1976 standard atmosphere model. In contrast, the temperature variation with altitude shows only minimal variation especially since the altitude range covers the Tropopause, an isothermal layer of the atmosphere.

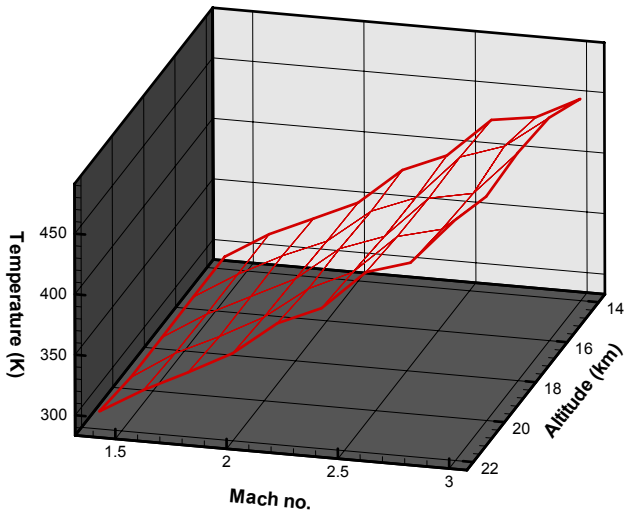

Fig. 10. Surface Plot of Aerodynamic Heating

In order to obtain the results depicted in the above figures, a total of 45 simulation cases were required to be executed. Overall, this is a computationally demanding task that requires approximately 40 minutes for a single case to be completed on one of the baseline machines. Nevertheless, using all 18 computers listed in Table A1, the computations were completed in roughly half an hour. This is an impressive speed-up, which shows a very good scalability of the distributed computing environment created by MIDAS.

5 Conclusion

The nearly linear scalability in computational performance demonstrated by MIDAS and its successful application to solving a two-dimensional CFD problem indicate that there is a great deal of potential for our framework to be employed for larger aircraft design problems. Future work with later versions of MIDAS will seek to provide a more functional and easy-to-use GUI and apply distributed simulation to large scale multidisciplinary optimization and analysis problems. The authors are planning to develop MIDAS into a facilitator framework for engineers who do not have much background in programming so that they can still carry out any type of generic object-oriented modeling and distributed simulation tasks.

Acknowledgment

The authors gratefully appreciate the effort and guidance of MIDAS team leader, Mr. Jung-Ho Lewe, in preparing this paper. We would also like to thank our faculty advisor, Dr. Dimitri Mavris, for his encouragement and support.

References

[1] Fujimoto R. *Parallel and distributed simulation systems*, John Wiley & Sons, Inc., New York, 2000.

[2] Kim S, Lee C and Kim J. Large-Scale Structural Analysis by Parallel Multifrontal Solver Through Internet-Based Personal Computers. *AIAA Journal*, Vol. 40, No. 2, pp 359-367, 2002.

[3] Eck D. *Introduction to programming using Java Version 4.0*. URL:http://math.hws.edu/javanotes [cited 28 May 2004].

[4] Freeman E, Hupfer S and Arnold K. *JavaSpaces principles, patterns, and practice*. Addison-Wesley, Reading, 1999.

[5] IncaX. URL:http://www.incax.com [cited 28 May 2004].

[6] Congress Online Project. URL: http://www.congressonlineproject.org/glossary.html/ [cited 19 May 2004].

[7] National Institute of Technology. URL: http://math.nist.gov/scimark2/ [cited 9 June 2004].

[8] Roe P L. Approximate riemann solvers, parameter vectors and difference schemes. *J. Comput. phys.*, Vol. 43, pp 357-372, 1981.

[9] Chakravarthy S R and Osher S. A new class of high accuracy TVD schemes for hyperbolic conservation laws. *23rd AIAA Aerospace Sciences Meeting and Exhibit*, Reno, Nevada, AIAA Paper 85-0363, 1985.

[10] Kontinos D A and McRae D S. Rotated upwind strategies for solution of the euler equations. *32nd AIAA Aerospace Sciences Meeting and Exhibit*, Reno, Nevada, AIAA Paper 94-0079, 1994.

[11] Thompson, et al. *Numerical grid generation*. Elsevier, New York, 1985.

Appendix

As the *PI* values listed in Table A1 in the following page indicates, a machine's hardware specification alone is not an adequate measure of computing performance. The *PI* value of a desktop computer is influenced by factors such as the type of OS, JVM version, and number of user-installed software, in addition to the CPU, clock speed and amount of RAM. This explains why, a Pentium III machine (P6) surpasses a Pentium IV machine (P7) in performance and the personal computers with the same hardware specification (P2, P8, and P9) have different *PI* values.

Table A1. Hardware and JVM Specifications of Machines Used

Machine ID	CPU	Clock Speed	RAM	OS	JVM Version	*PI*
B_1, B_2, B_3, B_4	Intel Pentium IV	1.6GHz	1GB	Linux	1.4.2_04	1.0
P1	Intel Pentium III	866MHz	256MB	Windows 2000	1.4.0	1.1
P2	Intel Pentium IV	2.0GHz	512MB	Windows 2000	1.4.2	1.6
P3	Intel Pentium IV	3.2GHz	1GB	Linux	1.4.2_04	2.3
P4	Intel Pentium III	700MHz	128MB	Windows 2000	1.4.0	1.1
P5	Intel Pentium III	700MHz	224MB	Windows 2000	1.4.2_03	1.1
P6	Intel Pentium III	866MHz	256MB	Windows 2000	1.4.0	1.4
P7	Intel Pentium IV	1.5GHz	256MB	Windows 2000	1.4.0	1.3
P8	Intel Pentium IV	2.0GHz	512MB	Windows 2000	1.4.1_01	1.7
P9	Intel Pentium IV	2.0GHz	512MB	Windows 2000	1.4.0	1.9
P10, P11	Intel Pentium IV	2.26GHz	512MB	Windows 2000	1.4.0_01	2.4
$G4_1$, $G4_2$	Power PC G4	866MHz	1GB	Mac OS	1.4.2_03	1.3
G5	Power PC G5	1.6GHz	512MB	Mac OS	1.4.2_03	2.3