Research Institute for Advanced Computer Science
NASA Ames Research Center

Unstructured Adaptive Grid Computations on an Array of SMPs

Rupak Biswas
Ira Pramanick
Andrew Sohn
Horst D. Simon

RIACS Technical Report 96.13 July 1996

To appear in the *Proceedings of Parallel CFD'96*,
Capri, Italy, May 20–23, 1996

Unstructured Adaptive Grid Computations on an Array of SMPs

Rupak Biswas
Ira Pramanick
Andrew Sohn
Horst D. Simon

The Research Institute of Advanced Computer Science is operated by Universities Space Research Association, The American City Building, Suite 212, Columbia, MD 21044, (410) 730-2656

Work reported herein was supported by NASA via Contract NAS 2-13721 between NASA and the Universities Space Research Association (USRA). Work was performed at the Research Institute for Advanced Computer Science (RIACS), NASA Ames Research Center, Moffett Field, CA 94035-1000.

Unstructured Adaptive Grid Computations on an Array of SMPs

R. Biswas[a], I. Pramanick[b], A. Sohn[c], and H.D. Simon[d]

[a]Research Institute for Advanced Computer Science (RIACS),
MS T27A-1, NASA Ames Research Center, Moffett Field, CA 94035, U.S.A.

[b]Network Systems Division,
MS 8U-802, Silicon Graphics Inc., Mountain View, CA 94043, U.S.A.

[c]Department of Computer and Information Science,
New Jersey Institute of Technology, Newark, NJ 07102, U.S.A.

[d]National Energy Research Scientific Computing Center (NERSC),
MS 50A/5104, Lawrence Berkeley National Laboratory, Berkeley, CA 94720, U.S.A.

Dynamic load balancing is necessary for the parallel adaptive solution of unsteady problems in fluid dynamics, since their computational requirements change as the simulation progresses leading to load imbalance. JOVE is such a dynamic load-balancing framework. We study the performance of two different implementations of JOVE on the Silicon Graphics' POWER CHALLENGEarray. This parallel machine is an array of shared-memory symmetric multiprocessing (SMP) systems, an architecture that is becoming increasingly popular as the most useful model of scalable parallel computing. Parallel algorithms need to be designed to exploit the hybrid communication model offered by such an architecture, and in this paper, we study these issues as they relate to JOVE.

1. INTRODUCTION

The extremely large computational requirements for adaptive calculations on unstructured grids, both in terms of processing time and in-core memory, can be successfully addressed by the use of massive parallel processing. A parallel implementation for such a problem consists of partitioning the computational mesh, and mapping the resulting submeshes onto processors. However, the computational intensity of these unsteady CFD calculations is typically both time- and space-dependent. Therefore, a static partitioning/mapping leads to load imbalance, even to the extent of offsetting any advantages of parallel processing. As a result, a dynamic repartitioning and remapping of the workload among processors is necessary for parallel processing to be beneficial for such applications.

Effective dynamic load balancing depends on several issues, including a reliable measurement of computational load, minimization of runtime data movement, and minimization of inter-processor communication. Our dynamic load balancer, called JOVE [5], is intended to satisfy these requirements. It possesses three novel features. First, a dual graph representation of the computational mesh is used to keep the complexity and con-

nectivity constant during the course of an adaptive computation. Second, a new inertial spectral mesh partitioning method [4] is introduced that performs both faster and better than Recursive Spectral Bisection [3]. Finally, a new heuristic processor assignment procedure and an accurate metric for estimating the communication overhead are presented.

An array of SMPs is rapidly becoming the architecture of choice for scalable parallel computing, as evidenced by the announcement of such machines by several computer vendors. The POWER CHALLENGEarray from Silicon Graphics is one of the first examples of such an architecture, and combines the benefits of both shared- and distributed-memory machines. It offers the ease of programmability of shared-memory with the scalability of distributed-memory systems. It supports both these parallel processing paradigms at the extremes of a spectrum of a multitude of programming models. Not only do these mixed models utilize the underlying architecture maximally, they present challenges in parallel processing that are absent in the two standard paradigms. In this study, we analyze JOVE with respect to the various programming models available on the POWER CHALLENGEarray, focusing on a "flat" message-passing model and an array of SMP-centric "hybrid" model where parallel tasks are aware of local (on the same SMP) versus remote (on a different SMP) threads and communicate with one another accordingly. Results indicate that the hybrid approach to parallel programming on such architectures is indeed an attractive one, hiding some of the costs associated with traditional message passing.

2. OVERVIEW OF JOVE

We have developed a software environment that uses global dynamic load balancing for unsteady adaptive CFD calculations [5]. Figure 1 gives an overview of our approach to load balancing. The system consists of three modules: the load balancer JOVE, a CFD flow solver, and the 3D_TAG mesh adaptor [1]. Details of the CFD solver are given in [6]; all that is needed here is that the solver produces error estimates that are then used by 3D_TAG to refine and/or coarsen the mesh. Note that both the adaption and solver phases are simulated for the results reported in this paper.

The first step of JOVE is `Pre_eval(new)` which rapidly determines if the new mesh warrants further action in terms of repartitioning and processor reassignment. If repartitioning is recommended, `Partition(new)` divides the new mesh into subgrids. A new inertial spectral bisection algorithm [4] is used to rapidly update a partition from one grid to the next. The `Evaluate(old,new)` step consists of assigning partitions to processors such that the communication overhead for data migration is minimized. It calculates two numbers: the computational gain `comp` that would be achieved by having a balanced partitioning, and the communication cost `comm` of moving all the data to correctly map partitions to processors. If `comp` is larger than `comm`, it is advantageous to use the new partitioning. In that case, the CFD simulation is interrupted while all the necessary data is redistributed based on the new processor assignments. The CFD calculation is then restarted on the new partitions. Otherwise, the new partitioning is discarded and JOVE waits for the next adapted mesh.

2.1. Dual graph representation

The dual graph representation of the initial computational mesh is one of the key features of this work (cf. Fig. 2). The most significant advantage of using the dual graph is

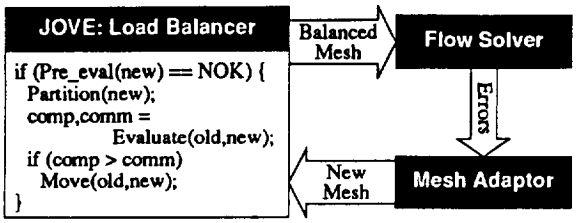

Figure 1. Our dynamic load balancing framework.

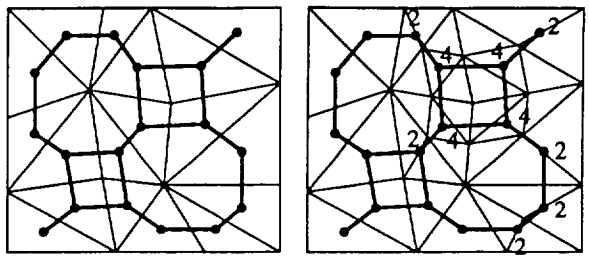

Figure 2. Initial and adapted meshes with the dual graph and computational weights.

that its complexity and connectivity remains unchanged during an adaptive computation. The partitioning and load-balancing times therefore depend only on the initial problem size and the number of processors used.

Parallel implementations of CFD solvers usually require a partitioning of the mesh such that each element belongs to an unique partition. The tetrahedral elements of a three-dimensional mesh are the vertices of the dual graph. An edge exists between two dual graph vertices if the corresponding elements share a face. A graph partitioning of the dual thus yields an assignment of elements to processors. Each dual graph vertex has two weights: w_{comp}, that models the computational workload for the corresponding element, and w_{comm}, that models the communication cost of moving the element from one processor to another. New grids obtained by adaption are translated to the two weights for every element in the initial mesh (cf. Fig. 2).

2.2. Inertial spectral mesh partitioning

If `Pre_eval(new)` determines that the dual graph with the new w_{comp} is not load balanced, JOVE invokes the mesh partitioning procedure. Several partitioning algorithms are available for unstructured grids; however, a new procedure that combines the high quality of spectral methods [3] with an efficient update strategy is used. This new algorithm [4] is based on the center of inertia of the vertices of the dual graph and utilizes information from the initial spectral partitioning. It is thus capable of rapidly updating a partition from one grid to the next. The algorithm in Fig. 3 briefly explains the method.

2.3. Processor reassignment and cost computation

The objective of `Evaluate(old,new)` is to map new partitions to processors such that the redistribution cost is minimized. A similarity matrix S is computed that indicates how the communication weights of the new partitions are distributed over the old partitions. The entry S_{ij} is the sum of the w_{comm} of all the dual graph vertices that have moved from old partition i to new partition j. A new partition j with the largest value of S_{ij} is called the dominant partition for old partition i. The data redistribution cost can be minimized by remapping the processor assigned to an old partition to its corresponding dominant partition. The dominant partitions have to be rearranged so that each new partition is dominant for exactly one old partition. However, this rearrangement constitutes an optimization problem with a polynomial-time solution. Instead, we obtain a suboptimal solution in linear time using the algorithm in Fig. 4. Note that the inner loop is executed only for those partitions i that have more than one dominant weight ($ndp[i] > 1$).

```
for (i=0; i<log(npart); i++) {          // npart = #partitions
  Find an inertial vector of the unpartitioned vertices
  Construct an inertial matrix using the inertial vector
  Symmetrize the inertial matrix
  Find the eigenvectors of the inertial matrix
  Project vertex coordinates on eigenvector 0
  Sort projected coordinates
  Divide the unpartitioned vertices into 2 sets
}
```

Figure 3. Pseudocode for inertial spectral mesh partitioning.

```
for (i=0; i<npart; i++)                 // npart = #partitions
  for (j=1; j<ndp[i]; j++) {        // ndp[i] = #dom weights
    Find min dominant weight S[l][i] from new partition i
    Find max non-dominant weight S[l][k] from
                        old partition l so that ndp[k] < 1
    Mark S[l][i] non-dominant
    Mark S[l][k] dominant
    Set ndp[k] to 1
  }
```

Figure 4. Pseudocode for processor reassignment.

The computational gain due to repartitioning is proportional to the decrease in the load imbalance achieved by running the adapted mesh on the new partitions rather than on the old partitions. If it requires T_{iter} secs to run one iteration of the flow solver on one element of the original mesh, and if it is expected that the next mesh adaption will occur after N_{adapt} solver iterations, the total computational gain for the new partitioning is $T_{\text{iter}} N_{\text{adapt}} (W_{\text{max}}^{\text{old}} - W_{\text{max}}^{\text{new}})$, where $W_{\text{max}}^{\text{old}}$ and $W_{\text{max}}^{\text{new}}$ are the sum of the w_{comp} on the most heavily-loaded processor for the old and new partitionings, respectively.

The redistribution cost is calculated from the similarity matrix S obtained after processor reassignment. Two machine-dependent parameters, the remote-memory latency time T_{lat} and the message setup time T_{setup}, are used to calculate the actual cost. If the CFD solver and mesh adaptor require M words of storage per element, and if C and N are the total number of elements and sets of elements to be moved, respectively, the total communication overhead for mapping new partitions to processors is $CMT_{\text{lat}} + NT_{\text{setup}}$.

3. ARRAYS OF SMPs

Clusters of SMPs, connected by a high-speed network, are becoming the architecture of choice for most new parallel computers. Supercomputer manufacturers such as Silicon Graphics, DEC, and Convex already market such machines, and other vendors have announced plans of moving from totally distributed machines to such a hybrid architecture. Arrays of SMPs combine the efficiency, flexibility, and ease of programmability of shared-memory multiprocessing with the scalability of message-passing architectures. This leads to a better computation-to-communication ratio since messages are exchanged between SMPs for larger blocks of parallel computation. Each node of such a parallel architecture consists of tens of processors, and is thus capable of supporting programs requiring medium levels of parallelism. At the same time, it provides a larger effective memory capacity than an equivalent distributed-memory machine. This is particularly useful for applications with large computational, memory, and I/O requirements that cannot be accommodated on individual workstation-class machines that form the building blocks of traditional distributed-memory computers. On the other hand, applications requiring higher levels of parallelism and/or having larger memory requirements (than can be supported within one such node) can be restructured to span multiple SMPs of the array.

The Silicon Graphics POWER CHALLENGEarray consists of up to eight POWER CHALLENGE systems, referred to as POWERnodes, connected by a HiPPI interconnect. It offers a two-level communication hierarchy, where processors within a POWERnode

communicate via a fast shared-bus interconnect, and processors across POWERnodes communicate via a high-bandwidth HiPPI interconnect. The inter-POWERnode message-passing overhead can be amortized over the computation, since parallel tasks within a POWERnode can use global shared memory to communicate shared data.

Applications that span multiple parallel POWERnodes need to be reengineered to take advantage of the hybrid communication model offered by such an architecture. Existing parallel algorithms designed for shared-memory or shared-nothing distributed-memory computers will typically need to be modified by varying extents to exploit this hybrid communication model. There are several approaches to modifying these algorithms, and the method of choice for any particular application will ultimately depend on the computation and communication characteristics of the algorithm used. Additionally, the programming models offered on such a system may also exploit the hybrid model as is the case with the implementation of SGI's MPI and PVM libraries.

The POWER CHALLENGEarray supports a hierarchy of parallel programming models that can be useful for the entire spectrum ranging from fine-grained to coarse-grained parallelism. It provides both shared-memory and message-passing programming models within a single POWERnode as well as across the array itself. The MPI and PVM message-passing libraries provided on the POWER CHALLENGEarray use shared memory to communicate between the message-passing tasks, if these tasks are on the same POWERnode. The MIPSPro Fortran and C compilers on each POWERnode provide automatic and user-assisted parallelization using threads within a POWERnode. Details on the various parallel paradigms available on the POWER CHALLENGEarray are given in [2].

4. FLAT vs. HYBRID JOVE

For the problem at hand, we studied two different approaches to parallelism on the POWER CHALLENGEarray. The first approach is the "flat" JOVE model which uses MPI for inter-task communication. As mentioned in the last section, SGI's MPI implementation automatically uses shared memory for inter-task communication within a POWERnode, resorting to message passing only for communication between tasks on distinct POWERnodes. Of course, the application code does not differentiate between local and remote tasks, hence the name "flat."

The second approach is a hybrid one, where explicit shared memory is used within a POWERnode and message passing is used between POWERnodes. This approach consists of exactly one MPI task per POWERnode that spawns off a number of parallel threads on its POWERnode using the MIPSPro compiler-assisted pragma directives. The parallel threads (including the original MPI task) divide the adaption and computation (flow solver) phases among themselves, with the MPI task taking care of inter-task communication, partitioning, and load balancing. This scheme is represented in Fig. 5.

The hybrid approach thus has a fewer number of MPI tasks than the flat model. This not only reduces the number of messages being exchanged, but also increases the size of the messages. These factors result in a reduction in the latency and bandwidth costs associated with message-passing systems, leading to a decrease in the overall communication costs. Also, the decrease in the number of MPI tasks reduces the cost of the partitioning

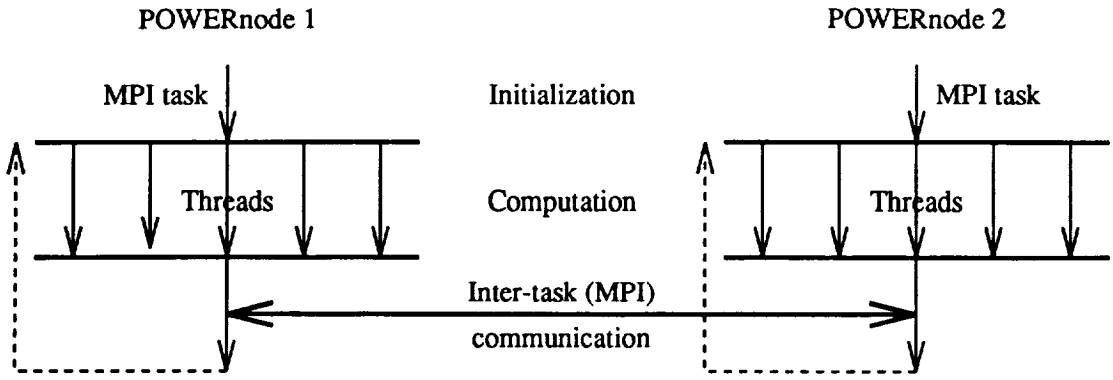

Figure 5. Hybrid MPI model on the SGI POWER CHALLENGEarray.

phase of JOVE. The cost of creating and maintaining the threads is additional in the hybrid approach, but as the results indicate, this is more than compensated by the decrease in the communication costs.

5. RESULTS

The experiments in this study consisted of running the sequential JOVE algorithm, a parallel implementation with static partitioning (referred to as NOLOAD), and the flat and hybrid versions of JOVE on a POWER CHALLENGEarray. Two three-dimensional unstructured meshes were used: BRICK, consisting of 2500 tetrahedral elements and 4600 edges, and STRUT, consisting of 14,504 tetrahedral elements and 57,387 edges. The number of adaptions was set at 20, 35, and 50 for BRICK, and 20 and 35 for STRUT. Both the adaption and flow solver phases were simulated.

The POWER CHALLENGEarray used for our experiments consisted of four 90 MHz R8K POWER CHALLENGE machines, with the number of processors on each varying from 12 to 16. The sequential code was run on one of the processors of the array, while the parallel codes were run on 1, 2, or 4 POWERnodes, each with 2, 4, or 8 processors.

The NOLOAD strategy was run on each of these configurations for direct comparisons with the flat JOVE model. For both NOLOAD and flat JOVE, the number of MPI tasks was equal to the number of processors used in each configuration. For the hybrid JOVE model, the number of POWERnodes determined the number of MPI tasks, and the number of processors on each POWERnode determined the number of threads that the MPI task running on that POWERnode would spawn.

The graphs in Figs. 6 to 9 show the results obtained from four different perspectives. The advantage of load balancing over a static partitioning technique is illustrated in Fig. 6. This graph plots the ratio of the times required by NOLOAD versus flat JOVE as the number of processors is increased from 2 to 32. These results demonstrate the significant advantage of load balancing for the class of applications studied here. JOVE consistently outperforms NOLOAD, the improvement being bigger for the larger STRUT mesh.

Figure 7 shows the speedup of flat JOVE as the number of processors is increased from 2 to 32. The speedup varies from 1.69 to 1.98 on 2 processors, from 3.36 to 3.90 on 4 processors, from 5.88 to 7.86 on 8 processors, from 12.50 to 15.29 on 16 processors, and

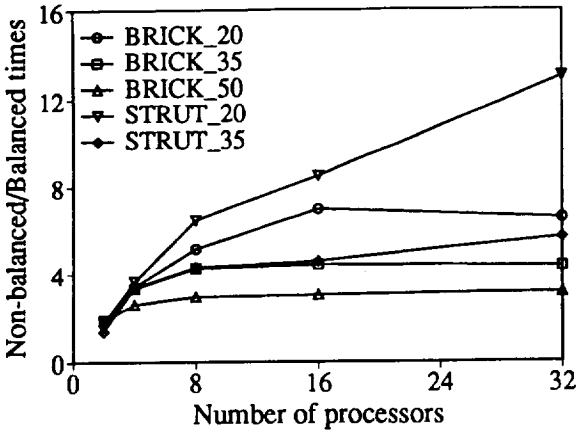

Figure 6. Comparison of total execution times with and without JOVE.

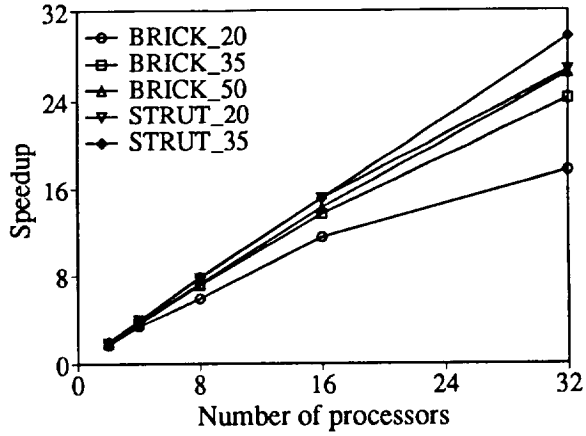

Figure 7. Parallel speedup of flat JOVE.

from 22.64 to 31.20 on 32 processors. These results demonstrate that even in absolute terms, JOVE yields significant speedup performance. Furthermore, the speedup curves for the different number of adaption steps are almost linear, indicating scalability. For a given mesh and a fixed number of processors, the speedup increases as the number of adaptions is increased. This is because the computational workload per processor increases with the number of adaptions, resulting in a higher speedup value. Finally, the speedup values are larger for STRUT than for BRICK. This indicates that for large realistic meshes, load balancing will be even more beneficial.

Figure 8 shows how the total execution time of JOVE is distributed among its various phases for 50 adaptions on the BRICK mesh. Observe that the partitioning and evaluation costs are negligible compared to the communication and computation costs. Note that the computation cost consists of both the adaption and flow solver times. The communication cost is more than three orders of magnitude less than the computation cost when two processors are used, but is only an order less when 32 processors are used. Under these conditions, the hybrid JOVE model should perform significantly better than the flat model since the former reduces communication costs.

The performance of flat and hybrid JOVE are compared in Fig. 9. Results show that the hybrid model outperforms the flat model in almost all the cases, giving upto a 28% better speedup on 32 processors. The speedup ratio increases with the number of processors; this is expected as the savings in the communication costs for hybrid JOVE increases as the number of processors is increased. The advantage of hybrid JOVE is the largest for 20 adaptions on BRICK, but decreases as the number of adaptions is increased. Also, hybrid JOVE does better in general than flat JOVE for BRICK than for STRUT. These results are however more a consequence of the fact that the absolute performance of flat JOVE for STRUT is better for a larger number of adaptions, being fairly close to the theoretical maximum (15.11 on 16 processors). In such cases, hybrid JOVE still outperforms flat JOVE, but is limited by the theoretical maximum. When the adaption and the flow solver phases are actually incorporated into JOVE (as opposed to being simulated as in this study), the speedup of flat JOVE may not be so close to the ideal. In such cases, we expect hybrid JOVE to consistently show significant advantage over flat JOVE.

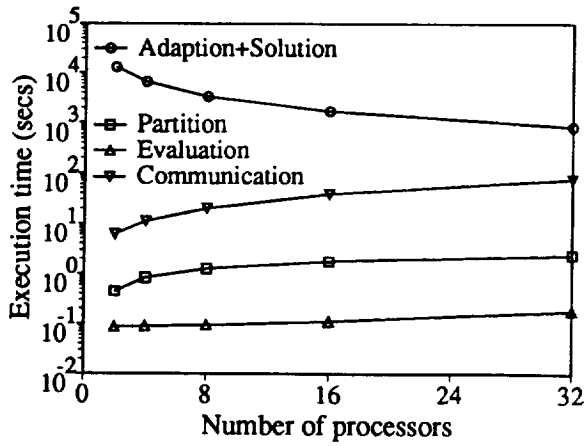

Figure 8. Anatomy of total execution times for the BRICK mesh with 50 adaptions.

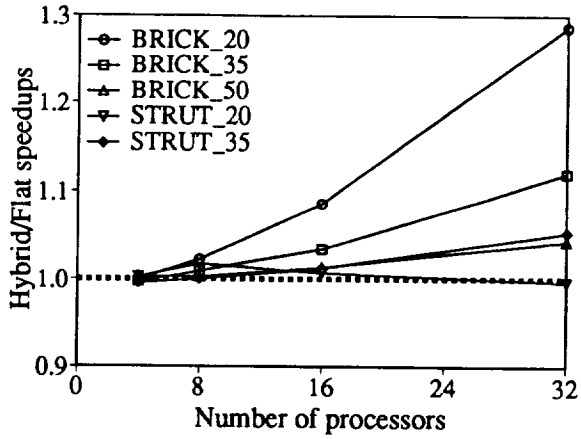

Figure 9. Comparison of total execution times of flat and hybrid JOVE.

6. SUMMARY

Dynamic load balancing is necessary for parallel adaptive methods to solve unsteady CFD problems on unstructured grids. We have presented such a dynamic load balancing framework called JOVE, in this paper. Results on a four-POWERnode POWER CHALLENGEarray demonstrated that load balancing gives significant performance improvements over no load balancing for such adaptive computations. The parallel speedup of JOVE, implemented using MPI on the POWER CHALLENGEarray, was significant, being as high as 31 for 32 processors. An implementation of JOVE that exploits "an array of SMPs" architecture was also studied; this hybrid JOVE outperformed flat JOVE by upto 28% on the meshes and adaption models tested. With large, realistic meshes and actual flow-solver and adaption phases incorporated into JOVE, hybrid JOVE can be expected to yield significant advantage over flat JOVE, especially as the number of processors is increased, thus demonstrating the scalability of an array of SMPs architecture.

Future work needs to study the performance of flat and hybrid JOVE on an array of R10K POWER CHALLENGEs, which are particularly suited for integer-based codes such as JOVE. Also, further experimentation with the actual flow solver and mesh adaption codes incorporated into JOVE, and studies with larger, realistic production meshes are needed to further add to the evidence of the advantages of dynamic load balancing and of the advantages of the array of SMPs architecture.

REFERENCES

1. R. Biswas and R.C. Strawn, Appl. Numer. Math. 13 (1994) 437.
2. POWER CHALLENGEarray Technical Report, Silicon Graphics, 1995.
3. H.D. Simon, Comp. Sys. in Engrg. 2 (1991) 135.
4. H.D. Simon, Irregular'96 (1996) to appear.
5. A. Sohn, R. Biswas, and H.D. Simon, 8th ACM SPAA (1996) to appear.
6. R.C. Strawn and T.J. Barth, J. AHS 38 (1993) 61.