Parallelization of ARC3D with Computer-Aided Tools

Haoqiang Jin, Michelle Hribar and Jerry Yan
MRJ Technology Solutions, NASA Ames Research Center
Moffett Field, CA 94035-1000

Abstract

A series of efforts have been devoted to investigating methods of porting and parallelizing applications quickly and efficiently for new architectures, such as the SGI Origin 2000 and Cray T3E. This report presents the parallelization of a CFD application, ARC3D, using the computer-aided tools, CAPTools. Steps of parallelizing this code and requirements of achieving better performance are discussed. The generated parallel version has achieved reasonably well performance, for example, having a speedup of 30 for 36 Cray T3E processors. However, this performance could not be obtained without modification of the original serial code. It is suggested that in many cases improving serial code and performing necessary code transformations are important parts for the automated parallelization process although user intervention in many of these parts are still necessary. Nevertheless, development and improvement of useful software tools, such as CAPTools, can help trim down many tedious parallelization details and improve the processing efficiency.

1 Introduction

The procurement of numerous high-performance computing systems at the NAS facility at NASA Ames Research Center over the past ten years has translated to repeated efforts of on adapting and porting both serial and parallel applications to the new systems. Several approaches have been pursued to facilitate the porting of parallel applications: rewriting the code by hand using message passing libraries, insertion of compiler directives to aid parallelizing compilers, and employing computer-aided tools.

The effectiveness of these approaches can be evaluated from several perspectives: ease of use, amount of work, portability, and efficiency. Programming by hand using message passing libraries usually achieves the best performance but is very time consuming, often prone to error. With data parallel languages and compiler directives, one can generate parallel code quickly, but often cannot attain good performance without extensive tuning. Automated parallel tools generally lack stability and have limitations in their ability to handle broad application areas. Even with these limitations, using compiler directives and computer-aided tools to parallelize the NAS Benchmarks [1] have shown promising results. The process of parallelization took a fraction of the effort that was needed for hand-coding and the performance of the generated codes in some cases is getting closer to the hand-coded versions [2].

To continue this effort, we extended our study to a moderate size computational fluid dynamics (CFD) application, ARC3D [3]. ARC3D solves Euler and Navier-Stokes equations in three dimensions using a single rectilinear grid. Unlike the NAS benchmarks, ARC3D contains a turbulent model and a more realistic boundary condition. The Beam-Warming algorithm is used to

approximately factorize an implicit scheme of finite difference equations, which is then solved alternatively in three directions. The implemented Alternating Direction Implicit (ADI) solver sweeps through each of the cardinal directions one at a time, with partial updating of the fields after each sweep.

ARC3D has been parallelized on the SGI Origin2000 system using compiler directives [4] (`http://science.nas.nasa.gov/Pubs/NASnews/97/07/article01.html`). Much of the effort in this work involved the optimization of cache performance and array locality on the distributed shared-memory system. Preliminary results from show promise for achieving very high sustained performance levels. We report here results of parallelizing ARC3D with the Computer-Aided Parallelization Toolkit (CAPTools) [5], a tool set that automates the generation of message-passing parallel programs with certain user intervention. We first briefly describe CAPTools and the computational environment, then discuss the steps taken in parallelization and optimization, and last present test results on the Origin2000 and the Cray T3E.

2 Computational Environment

The parallelization of ARC3D was done through CAPTools [5] — a software toolkit that automates the generation of message-passing parallel code — is developed at the University of Greenwich, United Kingdom (`http://www.gre.ac.uk/~captools`). CAPTools accepts FORTRAN-77 serial code as input, performs extensive dependence analysis, and uses domain decomposition to exploit parallelism. This approach is applicable to codes in computational aeroscience as well as many other types of science that use large spatial domains. CAPTools incorporates user knowledge in order to perform more accurate dependency analysis and, thus, produce more efficient parallel code. Lastly, CAPTools generates parallel codes that contain portable interface to message passing libraries, such as MPI and PVM, through a low-overhead layer (CAPLib). The user is able to perform more tuning later on and even to maintain the parallel code.

The performance of generated parallel codes was tested on two platforms: an SGI Origin2000 system and a Cray T3E system. The Origin2000 installed at NAS (`http://www.nas.nasa.gov/`) is composed of 64 RISC-based central processing units (CPUs), with 16 gigabytes (GB) of globally addressable main memory, and 330 GB of available disk space. Each processor has two levels of separate data and instruction caches (32KB for L1 and 4MB for L2, two-way set associative). The system incorporates the Non-Uniform Memory Access (NUMA) architecture, a design that allows the construction of large systems with hundreds or even thousands of processors at a modest cost.

The Cray T3E at NASA/Goddard Space Flight Center (`http://www.gsfc.nasa.gov/`) consists of 512 processor elements (PEs), 64 gigabytes of globally addressable memory, and 480-gigabyte disk. Each PE is a 64-bit DEC Alpha microprocessor (300 MHz) with an 8 KB primary data cache and a 96 KB secondary cache that is three-way set associative.

3 Parallelization of ARC3D

3.1 Brief Description of ARC3D

To facilitate the discussion of parallelizing ARC3D, the basic structure of this code is first presented. As mentioned in Section 1, ARC3D uses an implicit scheme to solve Euler and Navier-Stokes equations in a three-dimensional (3D) rectilinear grid. The main component is an Alternating Direction Implicit (ADI) solver, which results from the approximate factorization of finite difference equations. The actual implementation of the ADI solver (subroutine `STEPF3D`) in the serial ARC3D is illustrated in Figure 1.

For each time step, the solver first sets up boundary conditions (`BC`), forms the explicit right-hand-side (`RHS`) with artificial dissipation terms (`FILTER3D`), and then sweeps through three directions (ξ–X, η–Y and ζ–Z) to update the 5-element fields, separately. Each sweeping consists of forming and solving a series of scalar penta-diagonal systems in a two-dimensional plane one at a time. Two-dimensional arrays are created from the 3D fields and are passed into the penta-diagonal solvers (`VPENTA3` for the first 3 elements and `VPENTA` for the 4 and 5th elements, both originally being written for vector machines) which perform Gaussian eliminations (forward and backward). The solutions are then copied back to the residual fields. Between sweepings there are routines (`TKINV`, `NPINV` and `TK`) to calculate eigenvector matrices and perform matrix-matrix multiplications. Finally the solution is updated for the current time step.

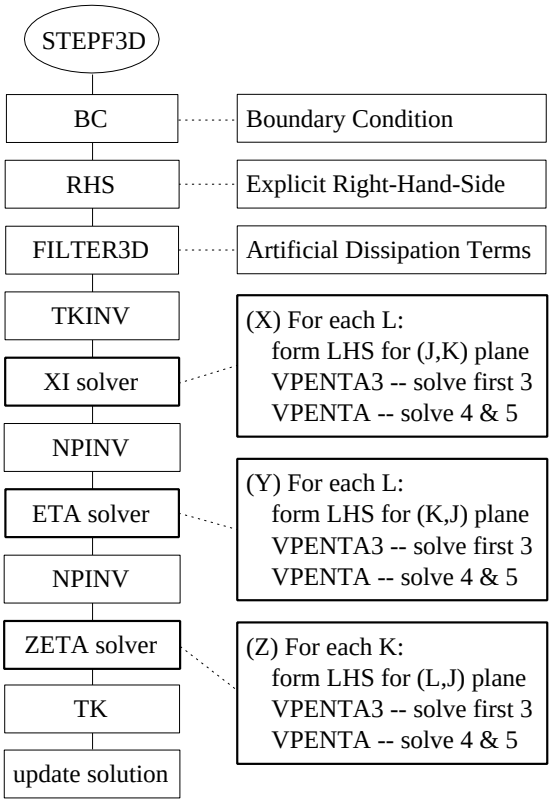

Figure 1: Block diagram of the ADI solver in the serial version of ARC3D. More descriptions on some of the blocks are given on the right side.

During the η (Y) and ζ (Z) sweepings the copying of data between 2D and 3D arrays (as implied by "forming LHS for (K, J) and (L, J) planes" in Figure 1) involves array transposition which is "unfriendly" to cache-based RISC systems. The profile information of different blocks will be further discussed in Section 4 when we compare performance.

3.2 The First Parallel Version

Using CAPTools to parallelize ARC3D was relatively straightforward. The serial version was first read into CAPTools. A full dependence analysis was performed and took about fifteen minutes on an SGI R5000 workstation. Two parallel codes were generated: one with 1-dimensional partitioning and the other with 2-dimensional partitioning. The "*Copy Routine*" function in CAPTools was used to duplicate routines `VPENTA3` and `VPENTA` for the Z-solver (both 1-D and 2-D partitions) and the Y-solver (2-D partition) in `STEPF3D`. The duplication is necessary since the data passed into these two routines will have different data distributions for different solvers after the data partition is done. One version of each routine cannot handle the two different data distributions.

The L (or Z) dimension of the solution data array was partitioned for the 1-D case and both L and K dimensions (or Z and Y) were partitioned for the 2-D version. The final generation of parallel codes had the following options selected: minimum slabs of 2, Gather/Scatter for communication, reduced memory. The last option is necessary for the effective use of memory on large number of processors.

The Automated Instrumentation Monitoring System (AIMS) [6] (`http://science.nas.nasa.gov/Software/AIMS/`) was used to collect and visualize trace information from the execution. AIMS automatically instruments either MPI or PVM message passing programs. Execution of the instrumented codes will produce event traces which can be displayed by the visualizer, VK, later on to pinpoint potential performance bottlenecks in parallel programs.

The trace data gathered from one ADI iteration (on 4 processors on an Origin2000) in the 1-D and 2-D partitioned versions of ARC3D are presented in Figure 2 as space-time diagrams. The color bar (different shadings in gray scale) in the plot represents execution of different subroutines, the white space between color bars indicates idle time due to waiting for a message or a barrier, and the lines drawn across processors are messages.

In the 1-D version, there are a few messages exchanged in the beginning of the ADI solver (`BC` and `FILTER3D`), no messages were used in solving for the X and Y directions. However, many relatively small messages were generated from the pipelining calculations for planes along the Z direction. Many idle-time gaps, as indicated by white spaces, are caused by frequently filling and emptying small pipelines of Gaussian elimination for each plane. Because of the large amount of idle time, the total time spent in the Z-solver was even more than the sum of those in the first two directions. A slight load imbalance was also observed in this version.

The 2-D partition shows a similar behavior although the pipelines are shorter and appear along both Y and Z directions. However, the total execution time is shorter than the 1-D partition, primarily due to reduced idle time in the solver. The routine `BC` for boundary conditions performed worse than in the 1-D version because `TRIB` (a routine inside `BC`) is called with partitioned data in the 2-D version but with unpartitioned data in the 1-D version.

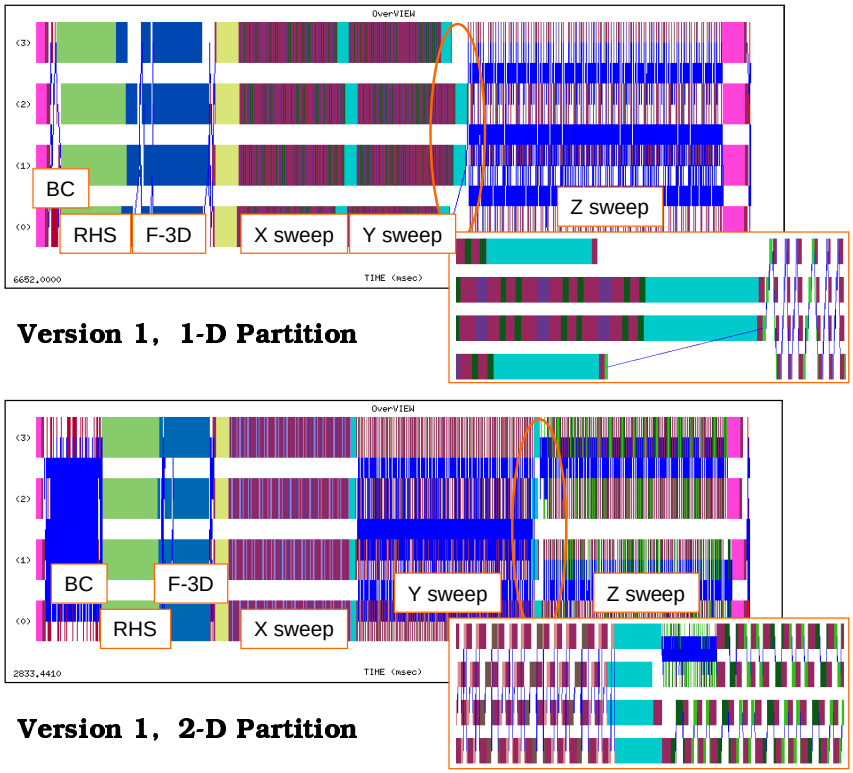

Figure 2: Space-time diagrams for the 1-D (upper panel) and the 2-D (lower panel) partitioned versions of ARC3D. The time window is for one ADI iteration. Enlarged areas show more details of the small pipelines in solving for different directions.

3.3 The Second Parallel Version

As shown in Figure 2, due to large number of small pipelines in the Y- and Z-solvers, the performance of the first parallel version was not expected to be good. This will become very clear from the performance data as discussed in Section 4. The bottleneck is due to the algorithm used in the original serial version. In order to improve the performance and let CAPTools fully explore the pipelining parallelism, it is necessary to modify the serial code.

The point where changes to the serial version were made is shown in Figure 3. The forming and solving of penta-diagonal systems for 2-D planes (see Figure 1) were combined to 3-D grid points for all three directions. In each direction of sweeping, we first form the Left-Hand-Side (LHS) matrix elements for all grid points in 3-D, then call the "penta" solver to perform the Gaussian elimination. This modification involved some loop regroupings and eliminated many repeated calls to the "penta" routines. The fundamental algorithm (the ADI solver) was not altered in the code. Because of the requirement of storing intermediate results for 3-D grid points (instead of 2-D in the original version), the new version uses about twice as much memory as the original code. There are now different "penta" routines used for different directions.

The revised version of ARC3D (Version 2) was parallelized through CAPTools in a similar way as discussed in the previous section. Again we produced parallel versions for both 1-D and 2-D partitions. The space-time diagrams for the new versions are illustrated in Figure 4. The main observation is that all the small pipelines shown in Version 1 are now grouped together in the new version and those small idle spaces are mostly eliminated. The change for Version 2 increased the granularity of the pipeline algorithm, which CAPTools can automatically explore. The pipeline setup and finish can now be done for a set of 2-D planes together instead of one plane at a time, as shown in the expanded display of Figure 4. Although there still exist white spaces due to the pipelining filling and emptying, the overall length of the pipelines in one solver is much smaller, thus less amount of time for communication. The performance of these versions will be compared in the next section.

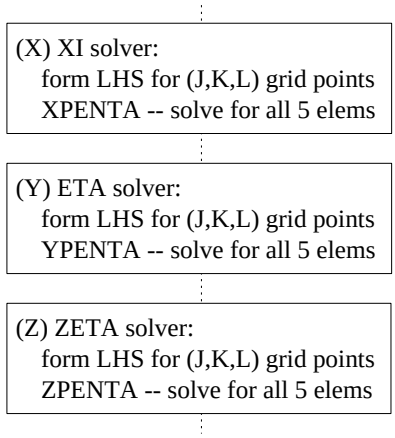

Figure 3: Revised version of the three solvers for exploring better pipelining parallelism in the Gaussian elimination stage.

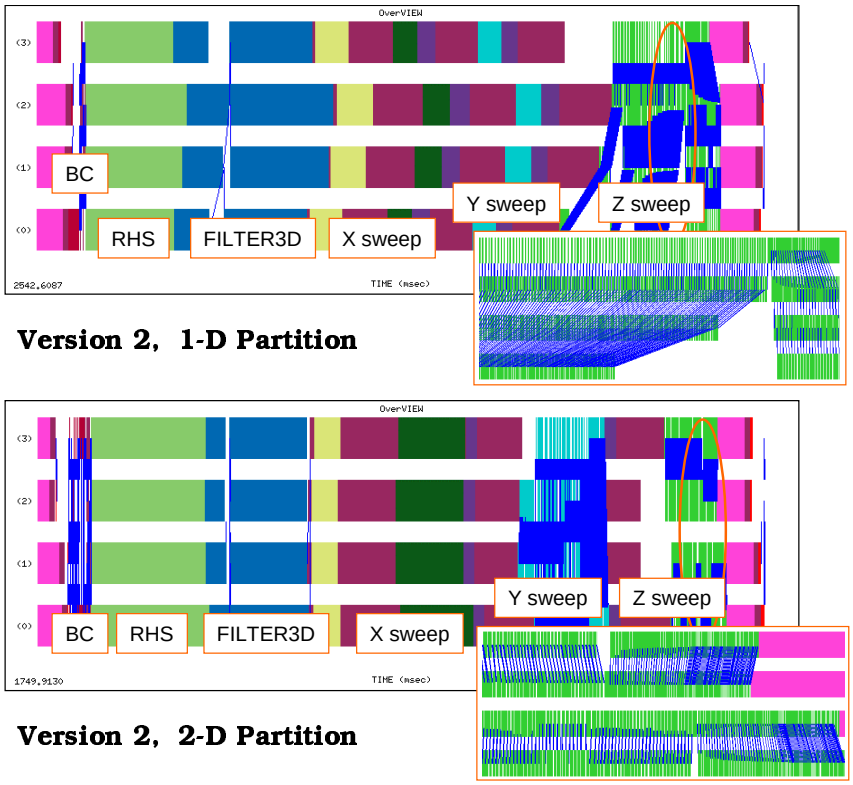

Figure 4: Space-time diagrams for the 1-D (upper panel) and the 2-D (lower panel) partitioned versions of the revised ARC3D. The time window is for one ADI iteration. Enlarged areas show more details of the pipelines in the Z direction. The pipelines due to forward eliminations are grouped together, so are those from backward substitutions.

4 Performance Comparison

4.1 Single-Node Performance

The two parallel versions of ARC3D with 2-D partition (as discussed in Section 3) were tested on a single CPU of the Origin2000. The '`pcsamp`' profiling tool on the Origin2000 was used to obtain statistics of individual routines. The results are summarized in Table 1. Overall, the single-node performance of Version 2 is about a factor of two better than Version 1, mainly from the improvements in `STEPF3D` and different `PENTA` routines. The improvements in Version 2 come from two areas.

1. In `STEPF3D` of Version 1, data were first copied to two-dimensional working arrays, the `VPENTA` solver was then called before the solution was copied back. For the sweeps on the second and third dimensions, the copying of 2-D arrays also involved array transposition, which was cache unfriendly. The copying process apparently exhibited sizable overhead, as indicated by a change of time from 33 seconds to 10 seconds in `STEPF3D` after the copy was eliminated in Version 2.

2. Many repeated calls to `VPENTA` and `VPENTA3` within loops in Version 1 were combined into one of each call to `XPENTA`, `YPENTA`, and `ZPENTA` in Version 2. This regroupment reduced the routine calling overhead.

Table 1: Performance data generated from the `pcsamp` profiling on a single CPU of Origin2000 for two different versions of ARC3D.

Version 1 without loop regrouping			**Version 2** with loop regrouping		
time(%)	**cum. time(%)**	**procedure**	**time(%)**	**cum. time(%)**	**procedure**
33.05s(41.4)	33.05s(41.4)	`STEPF3D`	10.44s(24.5)	10.44s(24.5)	`STEPF3D`
10.48s(13.1)	43.53s(54.5)	`RHS`	8.64s(20.3)	19.09s(44.9)	`RHS`
7.06s(8.8)	50.59s(63.4)	`FILTER3D`	6.40s(15.0)	25.48s(59.9)	`FILTER3D`
3.60s(4.5)	54.19s(67.9)	`VPENTA_X`	3.68s(8.7)	29.16s(68.6)	`XPENTA`
3.37s(4.2)	57.56s(72.1)	`VPENTA3_X`			
3.41s(4.3)	60.97s(76.4)	`VPENTA_Z`	2.90s(6.8)	32.06s(75.4)	`ZPENTA`
3.37s(4.2)	64.34s(80.7)	`VPENTA3_Z`			
3.65s(4.6)	67.99s(85.2)	`VPENTA_Y`	2.65s(6.2)	34.71s(81.6)	`YPENTA`
3.36s(4.2)	71.35s(89.4)	`VPENTA3_Y`			
2.04s(2.6)	73.39s(91.9)	`TKINV`	1.83s(4.3)	36.54s(85.9)	`TKINV`
1.87s(2.3)	75.26s(94.3)	`TK`	1.76s(4.1)	38.30s(90.1)	`TK`
1.62s(2.0)	76.88s(96.3)	`NPINV`	1.52s(3.6)	39.82s(93.6)	`NPINV`
0.42s(0.5)	77.31s(96.8)	`BC`	0.37s(0.9)	40.19s(94.5)	`BC`
2.54s(3.2)	79.85s(100.0)	others	2.34s(5.5)	42.53s(100.0)	others

When we started testing the parallel codes generated in the first pass, we noticed that on a single processor the 1-D partitioned code with a $64 \times 64 \times 64$ grid size ran about 20% slower than the 2-D partition. Normally we would expect opposite since the 2-D partition contains more overhead

due to much more checks on masked statements and communication calls. After inspecting the generated codes we realized that many multi-dimensional data arrays in the 1-D partition had a size of 64 for the first and second dimensions, while the 2-D partition had 64 only for the first dimension of these arrays (CAPTools padded the partitioned dimensions automatically when the reduced-memory option was used). The size 64 could cause many cache conflicts in loop operations since the size of cache lines is usually a multiple of 64 (e.g. 64 for the L1 data cache on the Origin2000). For loops in the third dimension the cache-line address of many array elements would be the same if both the first and second dimensions have a size of 64, as for the 1-D partition. The padding in the second dimension for the 2-D partition would avoid some of these cache collisions, resulting in a better performance.

In order to verify this observation, we took the modified serial version (Version 2) of ARC3D and compared the cache performance with and without padding in the array size. In the unpadded version arrays are declared as (64,64,64) while in the padded version the size of each dimension is simply increased to (65,65,65). The `perfex` profiling tool on the Origin2000 was used to gather information on performance counters. The result is compiled in Table 2. As we can see from the table, the impact of array padding to the cache performance is tremendous. The improvement ranges from a factor of 2 to 7 for many cases and with a factor of 4.4 for the overall performance.

Table 2: Performance data generated from the `perfex` command on a single CPU of Origin2000 for the new serial version of ARC3D with and without padding in array dimensions.

	Version 2a without padding	**Version 2b** with padding	**Improving factor**
Cycles	180.9s	41.2s	4.39
L1 cache misses	41.0(22.7%)	11.4(27.7%)	3.59
L2 cache misses	101.6(56.1%)	13.8(33.5%)	7.36
TLB misses	5.7(3.2%)	6.4(15.4%)	(0.90)
Qwords rewritten from L1	13.8(7.6%)	4.3(10.5%)	3.18
Qwords rewritten from L2	30.4(16.8%)	5.0(12.1%)	6.09
L1 Cache Line Reuse	1.96	9.58	4.89
L2 Cache Line Reuse	2.38	5.93	2.49
L1 Cache Hit Rate	0.66	0.91	0.25
L2 Cache Hit Rate	0.70	0.86	0.16
MFLOPS (per process)	16.68	73.84	4.43

Two common ways to optimize for cache involve rearranging cache dimensions for reuse and padding common block and arrays to reduce cache conflict. The first technique rearranges array dimensions to maximize the loop invariant references in the fastest-running, leftmost dimensions. The second technique of padding array dimensions was used in the above example. Compilers usually can perform common block padding automatically (via the '`-O3`' option). However, it did not improve performance as much for the current test case where multi-dimensional arrays are in common blocks.

There is still room to improve the single-node performance. For example, in Table 2, Version 2b still has a 34% L2 cache miss rate and a 28% L1 cache miss rate. This could be improved by

more aggressive optimizations, such as loop splitting, loop fusing, grouping statements, etc. This is beyond the scope of the current investigation. The study by Taft [4] reported that the optimized version achieved close to 100 MFLOPS (million floating point operations per second) on a single processor in comparison to 73.8 MFLOPS in our case.

4.2 Parallel Performance

The scalability of the four parallel versions (Versions 1 and 2 with 1-D and 2-D partitions) of ARC3D was tested on both the Origin2000 and the Cray T3E. Two types of `CAPLib` were used for the test: an MPI implementation for Origin2000 and a SHMEM implementation for T3E. On the T3E, the SHMEM `CAPLib` is 10 to 20% faster than the corresponding MPI version. The problem size used in all tests is $64 \times 64 \times 64$. Both parallel versions produced from the two passes described in Section 3 were tested on the two architectures. In order to examine the effect of cache performance, arrays were padded in Version 2 but not in Version 1.

The execution time and speedup factor for different runs on number of processors from 1 to 36 are summarized in Figure 5. The green (dashed) curves are results from the first parallel version (both 1-D and 2-D partitions) of ARC3D, while the blue (solid) curves are results from the second improved version. To compare the overhead introduced by CAPTools, the execution times of the revised serial version on a single processor are also plotted in Figure 5 as a filled diamond for the Origin2000 and a filled down-triangle for the T3E. The opened diamond and down-triangle are from the unpadded serial version. The MFLOP value was obtained from the hardware counter on the Origin2000 and MFLOPS was calculated from the wall-clock time.

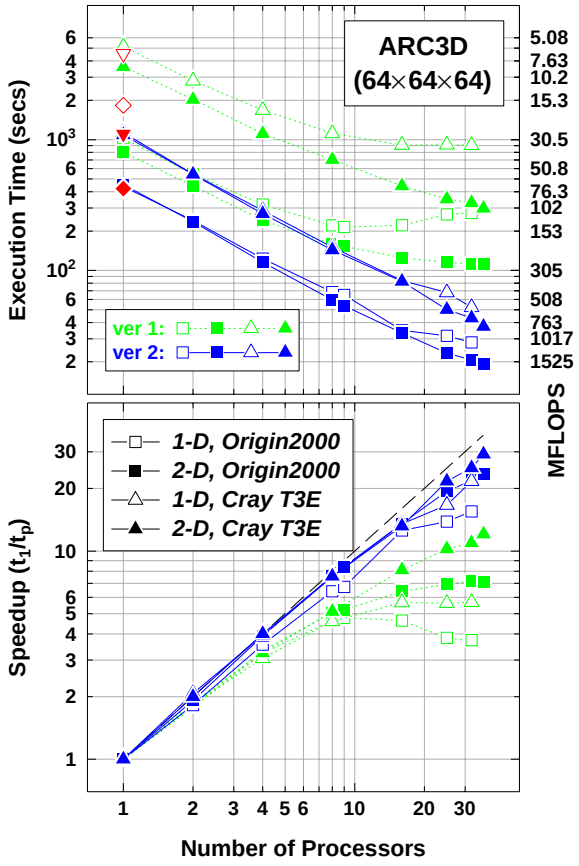

Figure 5: Execution time (upper part) and speedup factor (lower part) for various parallel versions of ARC3D on the Origin2000 and the Cray T3E.

Several conclusions can be made from Figure 5.

1. The effect due to cache performance on a single processor in Version 1 progresses to a large number of processors.

2. Version 1 does not scale well beyond 8 processors. The 2-D partition performs much better than the 1-D partition for large number of processors. At 32 processors, the 2-D partition is about 3 times faster than the 1-D partition on both machines.

3. Version 2 on one processor improves by a factor of 2 on the Origin2000 and 4 on the T3E in comparison with the first version. The more dramatic change of performance on the T3E may be due to the smaller cache size in the machine (see Section 2) as it translates into even worse performance for the first version.

4. The new version scales much better for both 1-D and 2-D partitions and is from 5 to 16 times better than Version 1 on 32 processors. The new 2-D partition has achieved a speedup of 30 for 36 processors on the T3E. The best execution time on the 32 Origin2000 processors corresponds to a performance of 1500 MFLOPS, which is about half of what was achieved by Jim Taft [4]. Programs ran slower (by a factor of two) on the T3E compared to on the Origin2000, but scaled better on the T3E.

5. Very little overhead from CAPTools was observed. Both the 1-D and 2-D partitions of Version 2 have similar single-node execution time which is very close to the result from the serial version.

6. The pipeline algorithm scales very well up to 36 processors in the test case, although this may not be true for even large number of processors. However, with increase of the problem size, good results are still expected, especially for the 2-D partition.

5 Summary

The parallelization of ARC3D with CAPTools was relatively straightforward. The generated parallel version achieved reasonably well performance, for example, having a speedup of 30 for 36 T3E processors. However, this performance could not be obtained without modification of the original serial code, in particular regrouping loops in the serial code to increase the granularity of the pipeline algorithm. Performance tools (such as AIMS) are very important in helping pinpoint performance bottlenecks.

The source-code transformation to the serial version as discussed in the report was performed by hand at this point. However, this type of transformation can be implemented into tools like CAPTools to help reduce tedious work. In many cases, improving serial code and performing necessary code transformations are important parts for the automated parallelization process. User intervention in many of these parts is still necessary. Nevertheless, useful software tools, such as CAPTools, can help trim down many tedious parallelization details and improve the processing efficiency. CAPTools is still envolving and will, hopefuly, be able to perform more complex source code transformations in a later version.

The current work provides some details of what are involved in parallelizing a CFD application. The experience will be useful for parallelization of even larger applications with computer-aided tools. The future work will include study of more complex applications and any techniques needed to help automated parallelization.

The authors wish to thank E. Evan, S. Johnson, and P. Leggett for valuable discussion on parallelizing ARC3D and their support on CAPTools, and M. Frumkin and A. Waheed for their valuable comments on the report.

References

[1] D. Bailey, T. Harris, W. Saphir, R. Van der Wijngaart, A. Woo, and M. Yarrow, "The NAS Parallel Benchmarks 2.0," *RNR-95-020*, NASA Ames Research Center, 1995.

[2] M. Frumkin, M. Hribar, H. Jin, A. Waheed, and J. Yan, "A Comparison of Automatic Parallelization Tools/Compilers on the SGI Origin2000 using the NAS Benchmarks," 1998.

[3] T.H. Pulliam, "Solution Methods In Computational Fluid Dynamics," *Notes for the von Kármán Institute For Fluid Dynamics Lecture Series*, Rhode-St-Genese, Belgium, 1986.

[4] J. Taft, "Initial SGI Origin2000 Tests Show Promise for CFD Codes," *NAS News*, July-August, page 1, 1997.

[5] M. Cross, C.S. Ierotheou, S.P. Johnson, P. Legget, and E. Rvans, "Software Tools for Automating the Parallelisation of FORTRAN Computational Mechanics Codes," *Parallel and Distributed Processing for Computational Mechanics*, 1997.

[6] J.C. Yan, S.R. Sarukkai, and P. Mehra, "Performance Measurement, Visualization and Modeling of Parallel and Distributed Programs using the AIMS Toolkit," *Software Practice & Experience*, Vol. 25, No. 4, pages 429-461, 1995.