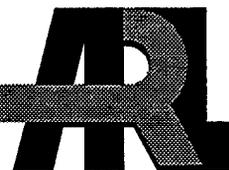


ARMY RESEARCH LABORATORY



Results From the Porting of the Computational Fluid Dynamics Code F3D to the Convex Exemplar (SPP-1000 and SPP-1600)

by Daniel M. Pressel

ARL-TR-1923

March 1999

Approved for public release; distribution is unlimited.

DTIC QUALITY INSPECTED 2

19990413050

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

Army Research Laboratory

Aberdeen Proving Ground, MD 21005-5067

ARL-TR-1923

March 1999

Results From the Porting of the Computational Fluid Dynamics Code F3D to the Convex Exemplar (SPP-1000 and SPP-1600)

Daniel M. Pressel

Corporate Information and Computing Center, ARL

Abstract

This report discusses the continuing efforts to port the F3D computational fluid dynamics code to RISC-based SMPs. Originally, this program was optimized for Cray vector supercomputers such as the Cray C90. Previous attempts to run this code on SGI Power Challenges, Convex Exemplars, as well as systems from SUN and Digital Equipment demonstrated a level of performance that was so low as to be utterly useless (in many cases it became necessary to kill the job before the first time step had completed). After making a concerted effort to port the program to an SGI Power Challenge (R8000 processor), acceptable levels of performance were finally achieved (Pressel 1997). Using this version of the code as the starting point, an effort was made to produce a program that ran efficiently on both systems from SGI and Convex. Unfortunately, a number of limitations with the Convex Exemplar were discovered that limited the success of this effort.

Acknowledgments

The author wishes to express his gratitude to Sally Parker of the Space and Naval Warfare Systems Command (SPAWAR) Systems Center, San Diego, CA (who came into work on numerous weekends and holidays to reboot the system after it had crashed), and Sharon L. Shaw of Hewlett-Packard (HP)/Convex for their assistance in providing the resources necessary to carry out this project. Additionally, he wishes to thank Karen Heavey, Jubaraj Sahu, Ph.D., James Collins, Ph.D. (formerly of the U.S. Army Research Laboratory [ARL]), Walter Sturek, Ph.D., and Charles Nietubicz for their contributions to the success of this project.

This work was made possible by the grant of computer time by the Department of Defense (DOD) High Performance Computing Modernization Program. Additionally, it was funded as part of the Common High Performance Computing Software Support Initiative (CHSSI) administered by the DOD High Performance Computing Modernization Program.

The author also wishes to thank Chuck Kennedy and the people of the Survivability/Lethality Analysis Directorate (SLAD), ARL, for giving him access to their Silicon Graphics Inc. (SGI) Challenge computer. He also wishes to thank SGI for loaning ARL the additional hardware required to perform runs out to 36 processors on this system and 18 processors on the Power Challenge located at the ARL Major Shared Resource Center (MSRC).

INTENTIONALLY LEFT BLANK.

Table of Contents

	<u>Page</u>
Acknowledgments	iii
List of Figures	vii
List of Tables	ix
1. Introduction	1
2. The Architecture of the SGI Power Challenge	3
3. The Architecture of the Convex Exemplar	4
4. Performance Issues	6
4.1 Serial Performance	7
4.2 Parallel Performance Using a Single Hypernode	14
4.3 Parallel Performance Using Multiple Hypernodes.....	15
5. Results	20
6. Future Plans	31
7. Conclusions	36
8. References	39
Abbreviations	41
Glossary	43
Distribution List	45
Report Documentation Page	49

INTENTIONALLY LEFT BLANK.

List of Figures

<u>Figure</u>		<u>Page</u>
1.	The Simple Approach to Processing a Plane of Data Using Two Loops That Sweep Through the Plane in Orthogonal Directions.....	8
2.	Using a Common Outer Loop When Processing a Plan of Data Using Two Loops That Sweep Through the Plane in Orthogonal Directions.....	9
3.	Using Blocking to Improve Performance When Processing a Plane of Data Using Two Loops That Sweep Through the Plane in Orthogonal Directions.....	10
4.	An Alternative Approach to Using Blocking to Improve Performance When Processing a Plane of Data Using Two Loops That Sweep Through the Plane in Orthogonal Directions.....	11
5.	An Example of Multiple Levels of Blocking	13
6.	Performance Results for the 1-Million Grid-Point Test Case	21
7.	Performance Results for the 3-Million Grid-Point Test Case	22
8.	A Performance Comparison Between Using Optimatation Techniques That Are Well Suited and Appropriate for Use on Any RISC-Based SMP, and Using Additional Techniques That Are Specifically Designed to Overcome the Limitations of the Convex Exemplar	23
9.	A Comparison of the Predicted and Measured Levels of Performance for SGI Challenge and Power Challenge Systems (1-Million Grid-Point Test Case).....	32
10.	A Comparison of the Predicted and Measured Levels of Performance for One, Two, and Four Hypernode Subcomplexes on a Convex SPP 1600 (1-Million Grid-Point Test Case).....	33
11.	A Comparison of the Predicted and Measured Levels of Performance for SGI Challenge and Power Challenge Systems (3-Million Grid-Point Test Case).....	34
12.	A Comparison of the Predicted and Measured Levels of Performance for One, Two, and Four Hypernode Subcomplexes on a Convex SPP 1600 (3-Million Grid-Point Test Case).....	35

INTENTIONALLY LEFT BLANK.

List of Tables

<u>Table</u>	<u>Page</u>
1. Performance Results.....	26
2. Predicted Speedup for a Loop With 15 Units of Parallelism	31

INTENTIONALLY LEFT BLANK.

1. Introduction

This project was begun as a result of an effort to run a large memory (1.5 GB) computationally intensive job on a Silicon Graphics Inc. (SGI) Power Challenge. The code that was selected for this effort was an implicit 3-D CFD solver known as F3D (Sahu and Steger 1990). It was already known that, for the specified problem, this code required a bit under 9 CPU* min to run on a Cray C90 (when using one processor) (a bit over 10 CPU min if run as an out of core solver on the same hardware, when using the solid-state disk to hold the data not currently resident in main memory).† Attempts to run the in-core solver version of this code on an SGI Power Challenge (75 MHz, and again using just one processor) required over 5 hr of CPU time. Obviously, this was not an acceptable situation. While there were many theories as to what was causing this problem, the correct answer was that codes that are well-tuned for Cray vector processors are rarely, if ever, well-tuned to run on RISC-based systems (especially in regard to taking full advantage of **cache**). There were some who expressed concerns that this incompatibility either could not be overcome or would require the use of a different algorithm. Our experience indicated that extensive *implementation-level* tuning could overcome most, if not all, of the performance problems. This was an important conclusion since it allowed us to use the new systems without changing the algorithm (Pressel 1997).

Once a reasonable level of serial performance had been achieved, there were discussions on how the program should be parallelized. This was far from being a trivial consideration since traditionally implicit CFD codes are considered to be inherently difficult to parallelize. The two most commonly used techniques are:

- (1) Switch to an explicit algorithm, which is easy to parallelize.

* Note: All items in bold type are defined in the Glossary.

† The test case only involved computing 10 time steps. Production runs frequently involve processing hundreds or even thousands of time steps and, generally, take many hours to finish.

- (2) Use domain decomposition (frequently, this will require significant changes to the algorithm if one is to avoid serious degradation to the convergence properties of the algorithm).

A thorough analysis of the problem indicated the possibility that a third solution might exist. This solution was based on two concepts:

- (1) The original version of F3D is highly vectorizable.
- (2) Vectorization is a form of parallelism that works at the loop level.

Therefore, in theory, it should be possible to parallelize F3D (and probably most vectorizable programs) using other forms of parallelism that work at the loop level. Traditionally, this has not been considered to be the most useful of observations since:

- (1) Loop-level parallelism does not in general support the use of large numbers of processors. Additionally, on traditional **MPPs**, one needed hundreds, if not thousands, of processors to achieve *supercomputer* levels of performance.
- (2) On many architectures, it is difficult to show parallel speedup when using this approach. Furthermore, even when such speedup is demonstrated, the absolute level of performance tends to be so low as to make the effort useless.

However, technology keeps changing and an analysis of the SGI Power Challenge indicated that it was an example of a class of computers that should be ideally suited for use with loop-level parallelism. This class of computers is known as RISC-based cache-coherent shared memory **SMPs**. The Convex Exemplar is marketed as being another member of this class of computers.

While there are a number of architectural differences between the two machines, it was hoped that this approach to optimizing and parallelizing F3D could be made to work efficiently on both the

SGI Power Challenge and on the Convex Exemplar. This report will discuss how this effect was approached, what the results were, and the reason for those results.

2. The Architecture of the SGI Power Challenge

Before one can discuss the results of this effort, it is helpful to have a short introduction to the architectures involved. The SGI Power Challenge can be thought of as the prototypical RISC-based cache-coherent shared memory SMP. As such, it will be discussed first. From the hardware perspective, it has the following properties:

- (1) Powerful RISC-based processors. In this case, R8000 processors rated at 300 MFLOPS each.
- (2) This version of the Power Challenge had up to 18 processors and up to 16 GB of memory, although one could not configure a system with both 18 processors and 16 GB of memory.
- (3) Each processor had a large off-chip cache (4 MB).
- (4) Hardware protocols maintain the coherency between all of the caches and main memory. This is a key requirement if shared memory is to be implemented entirely in hardware.
- (5) Shared memory is entirely implemented in hardware. All of the boards are connected by a common system bus that implements a snoopy bus protocol for cache coherency. This bus gives a *uniform memory access time*. The speed of the bus is a key limiting factor in determining how many processors can run in a system at one time without running out of memory/bus bandwidth.

From the OS perspective, SMP means that the combination of the OS and the hardware have been designed in such a way as to allow any processor to safely execute any portion of the kernel of the OS. This is key to avoiding points of contention that would otherwise limit system performance.

3. The Architecture of the Convex Exemplar

The Convex Exemplar is also based on powerful RISC processors (HP-PA 7100 for the SPP-1000 and HP-PA 7200 for the SPP-1600). Each processor has a large off-chip data cache (1 MB) and a separate large off-chip instruction cache (1 MB). The designers of this system realized that a common system bus connecting all of the boards in a system could severely constrain the performance and scalability of their system. Therefore, they chose to use a more complicated design. Some of the key properties of this design are:

- (1) Processors are grouped into Hypernodes, with each Hypernode containing 8 processors. The SPP-1000 supported 128 processors, but with the SPP-1600, the stated maximum system size was reduced to 64 processors.
- (2) At the OS level, Hypernodes are configured into Subcomplexes, with a Subcomplex containing at least one processor from each of the Hypernodes that make up the Subcomplex.
- (3) Each Hypernode has its own pool of memory (normally either 1 or 2 GB of memory).
- (4) Peripherals are directly connected to a Hypernode. Processors on other Hypernodes have degraded access to nonlocal peripherals. This makes it desirable to give each Subcomplex its own local scratch partition.
- (5) At boot time, the memory on the system is configured for three distinct classes of usage. This process makes the memory system noticeably less efficient in its use of resources than

that of more traditional designs such as the SGI Power Challenge. The three usage classes are:

- Hypernode local memory (possibly shared among multiple Subcomplexes). This memory can only be accessed by processors on the same Hypernode. Access to this memory is much faster than to global memory, but it is of limited value to shared memory jobs running across Hypernodes.
 - A portion of the local memory from each Hypernode that makes up a Subcomplex may be used to create global memory for that Subcomplex. Global memory is cache coherent, but, in general, it is noticeably slower to access. There is also a significant limit on the size of global memory, so that, in general, there will be less than 2 GB of global memory for a single Subcomplex.
 - A portion of the local memory from each Hypernode could also be used to make up what Convex refers to as the CTI cache. This is a Hypernode level (as opposed to the more commonly used processor level) **DRAM** (as opposed to **SRAM**) cache. It is used to accommodate the larger latencies associated with accessing global memory. Unfortunately, given certain design constraints, the larger the size of the CTI cache, the smaller the maximum size of global memory (even if there is local memory to spare).
- (6) The large off-chip data cache uses a cache line size of 32 bytes, which is significantly shorter than that used by SGI (512 bytes with 128 byte sectors for the R8000 and 128 bytes for the R4400 and R10000 processors).
- (7) The **TLB** on the Hewlett-Packard (HP) processors is smaller than that used on the R8000 processor. Additionally, the Convex Exemplar uses a smaller page size (4 KB vs. 16 KB). The net result is that while the R8000 TLB is large enough to map 4 MB of memory with room to spare, the HP TLBs can only map a little under 0.5 MB of memory. When dealing

with large data sets, this makes it more likely that the processors on the Convex Exemplar will **thrash their TLBs**.

4. Performance Issues

There are three main aspects to reviewing the performance of any code on the Convex Exemplar:

(1) Single-processor performance. This includes such topics as:

- The theoretical peak performance of a single processor.
- What percentage of peak performance one gets when using a single processor with local memory on a dedicated Hypernode and why.
- What percentage of peak performance one is likely to see when using the Convex Exemplar as a *throughput*-oriented machine running principally serial jobs.

(2) Parallel-processor performance using a single Hypernode. This is primarily interested in issues such as the supported programming paradigms and the level of scalability that they can deliver. It will, however, also cover issues relating to the limitations of using a single Hypernode.

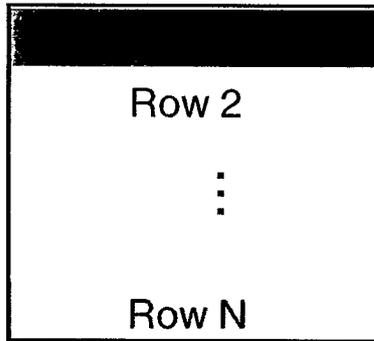
(3) Parallel-processor performance using processors on multiple Hypernodes. In other words, How well does Convex's concept of how to build an SMP compare to those used by SGI in building the Power Challenge?

The next three sections will, in turn, consider each of these areas.

4.1 Serial Performance. It was expected that, since both the Convex Exemplar and the SGI Power Challenge are based on RISC microprocessors using large caches as part of a memory hierarchy, it should be possible to produce a single code that exhibits a high level of performance on both machines. However, it was also assumed that there might be some key differences between the machines that would have to be taken into account. The key differences and how they were addressed were:

- (1) The Convex Exemplar uses a smaller cache than does the SGI Power Challenge. The original port to the Power Challenge would have shown poor performance for sufficiently large problem sizes had our Power Challenges had enough memory in them. With the smaller cache size on the Convex Exemplar, this point was reached much sooner. The solution to this problem was to block code in a wider range of subroutines. Unfortunately, not all subroutines were easy to block in two directions. In the one case, where the subroutine could only be blocked into strips rather than squares, it was necessary to parameterize the strip width based on the cache size. Presently, the cache size is a hard-coded parameter (see Figures 1-4 for details).
- (2) There are fine differences in the cache structure between the two machines (e.g., cache line size). These differences tend to make the effective cost of a cache miss to be larger on the Convex Exemplar (substantially larger when using global memory). As a result, it became more important to reduce the number of cache misses to the largest extent possible. In some cases this meant removing optimizations that added cache misses, while only producing a marginal speedup on the Power Challenge. It was felt that, with the trend toward increasing processor speeds, this would be a clear win on the next generation of machines from all vendors. Even so, it should be noted that, when running this code, the Convex Exemplar spends more time on cache misses than does the Power Challenge.
- (3) Different machines have different TLB sizes. Additionally, many architectures use two or more levels of cache. In an attempt to support the widest range of processor designs as possible, the decision was made to use multiple levels of blocking wherever possible (e.g., in

Loop 1



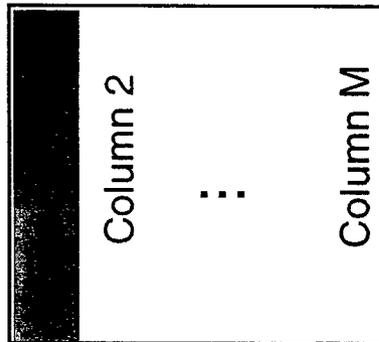
PROBLEM: Regardless of whether one uses C or Fortran 77, one of the two loops will have a high cache miss rate.

POSSIBLE SOLUTION: Perform matrix transposes.

OBJECTION: Matrix transposes add a lot of overhead while doing no useful work. In some cases, they will actually be slower than doing nothing at all.

∞

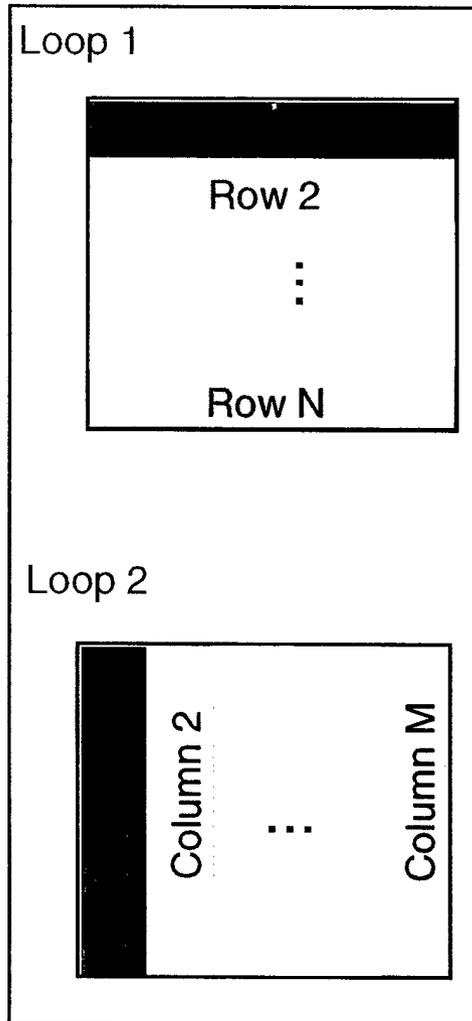
Loop 2



PREFERRED SOLUTION: See Figure 2.

Figure 1. The Simple Approach to Processing a Plane of Data Using Two Loops That Sweep Through the Plane in Orthogonal Directions.

Common Outer Loop



6

ASSUMPTIONS:

- (1) There are multiple planes of data being processed by each loop.
- (2) There was an outer loop for each loop that was not shown in Figure 2 that steps through the planes of data.
- (3) The outer loops for Loop 1 and Loop 2 could be combined into a common outer loop.
- (4) A plane of data fits in cache.

OBJECTION:

The fourth assumption can fail badly for large problem sizes or small caches.

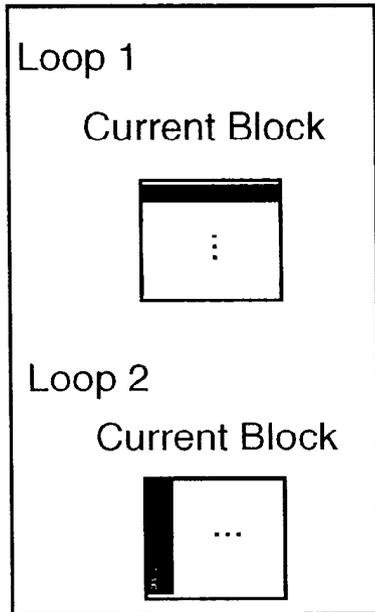
PREFERRED SOLUTIONS:

See Figures 3 and 4.

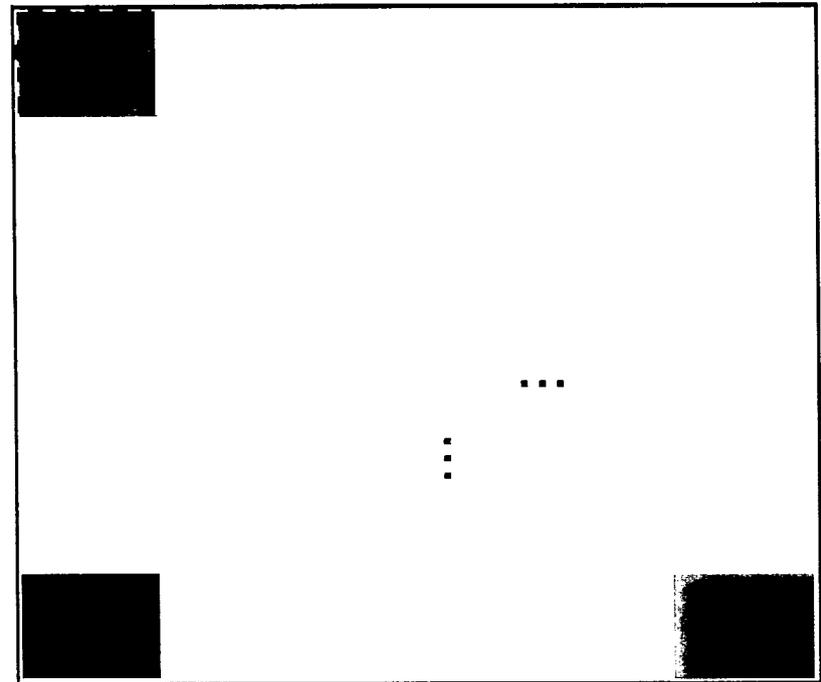
Figure 2. Using a Common Outer Loop When Processing a Plane of Data Using Two Loops That Sweep Through the Plane in Orthogonal Directions.

Common Outer Loop

Loop By Block Number



Blocked Access of a Plane of Data



10

ASSUMPTIONS:

- (1) It is possible to block in both directions.
- (2) It is practical to use block sizes that are small enough to fit in the outermost level of cache.

OBJECTIONS:

- (1) Assumption 1 may be wrong.
- (2) Assumption 2 may be suboptimal.

PREFERRED SOLUTIONS:

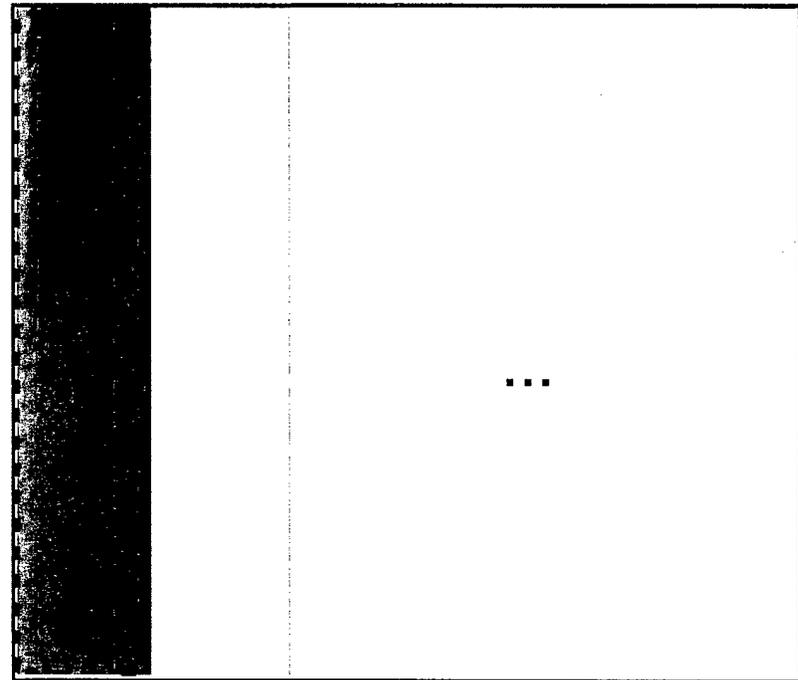
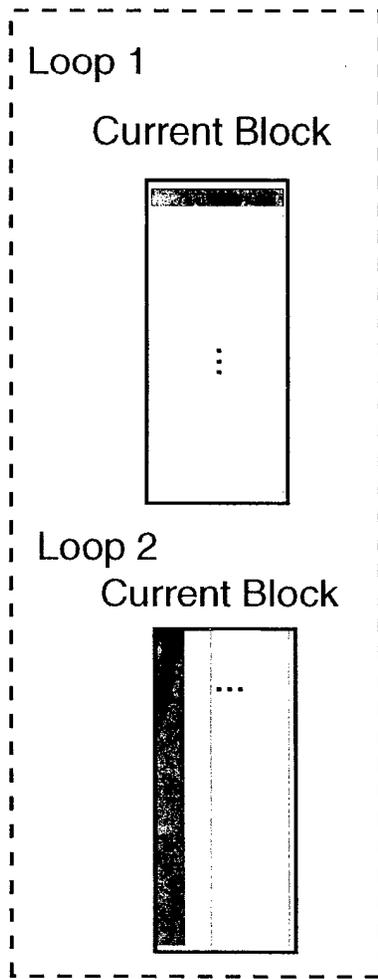
- (1) See Figure 4.
- (2) See Figure 5.

Figure 3. Using Blocking to Improve Performance When Processing a Plane of Data Using Two Loops That Sweep Through the Plane in Orthogonal Directions.

Common Outer Loop

Blocked Access of a Plane of Data

Loop By Block Number



REQUIREMENTS:

The block width takes into account the number of rows in a plane, the amount of memory needed per grid point, and the size of the cache.

OBJECTIONS:

Generally less efficient than the fully blocked solution in Figure 3.

Figure 4. An Alternative Approach to Using Blocking to Improve Performance When Processing a Plane of Data Using Two Loops That Sweep Through the Plane in Orthogonal Directions.

matrix transpose routines). This allows one to use a relatively small-block size for the innermost level of blocking, while still supporting larger block sizes for those machines that can take advantage of them. However, this created another problem. Small block sizes can imply a high level of loop overhead, and extra care had to be taken to minimize this effect (see Figure 5 for details).

- (4) The HP-PA 7100 and 7200 processors used by Convex rely upon a multiply-add instruction for their peak level of performance. The MIPS R8000 processor used by SGI also relies upon such an instruction. However, there were significant differences in how these two instructions worked. As such, the instruction on the HP processor favored vector-type code, although such code is a poor match for architectures using cache. On the other hand, the instruction used on the MIPS R8000-favored code that was well tuned for RISC processors and cache in general. As a result, the SGI compilers had no problems making good use of their multiply-add instruction, while the Convex and HP compilers were rarely able to make efficient use of their instruction (in fact, for most subroutines, the best performance was achieved by telling the compiler not to use that instruction). Apparently, this was a sufficiently common problem that the newer HP-PA 8000 processors have a second multiply-add instruction that is very similar to that found in the MIPS R8000. Only for one subroutine (BTRI) was it possible to make good use of the multiply-add instruction supported by the HP microprocessors, and only then by using the HP compiler rather than the Convex compiler.
- (5) Even though the BTRI subroutine exhibits a vanishingly small cache miss rate on both machines, it is still the single most expensive subroutine on both machines. As such, it is important to have the most efficient implementation of this routine as possible. In this vein, an exquisitely tuned implementation was produced for the Power Challenge that bares a striking resemblance to assembly code, while maintaining the portability of **Fortran**. It turned out that a number of assumptions were made in producing this routine that were not optimal for the Convex Exemplar, as well as some of the other machines we were looking at. Therefore, the decision was made to produce a new version of this routine that would

Outer Loop By Outer Block Number

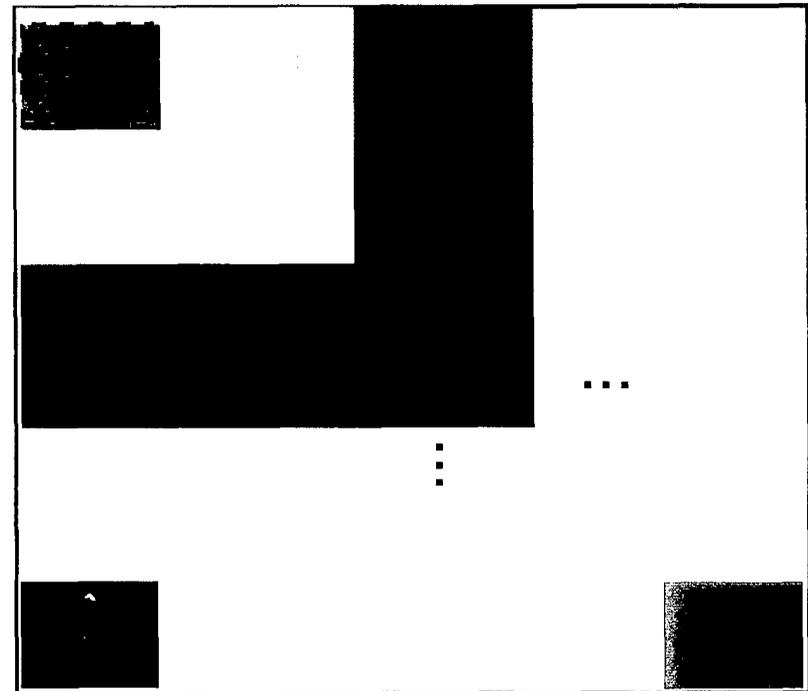
Middle Loop(s) By Middle Block Number(s)

Inner Loop By Inner Block Number

Current Block



Blocked Access of a Plane of Data



13

USE:

When done well, multiple levels of blocking can produce highly efficient code that works well with a wide range of cache and TLB sizes and designs.

LIMITATIONS:

It can be very difficult to use this technique with any but the most simple loops. In F3D, its use has been primarily restricted to matrix transpose operations.

Figure 5. An Example of Multiple Levels of Blocking.

have a smaller foot print in cache (in terms of the size of the scratch arrays) and require fewer scratch variables be *locked* into registers for optimal levels of performance. The net result is a version of the subroutine that significantly outperforms the earlier version of the code on the Convex Exemplar, runs almost as fast on the R8000 Power Challenge, runs faster on R10000-based machines from SGI, and is more compatible with the use of larger numbers of processors on the Convex Exemplar and the SGI Origin 2000.

While there were clearly architectural features that made the performance of this code when using one processor on a Convex Exemplar to be slightly lower than desired, the level of performance was more than high enough to justify continuing on to the next stage in this effort. The main disappointment at this stage was that no way could be found to run jobs with an address space in excess of 2 GB. This prevented us from running jobs with more than about 3 million grid points. Even when we parallelized the code across multiple Hypernodes, this problem was never solved, and, in fact, it became worse due to the more restrictive limits on the amount of global memory a system could have.

4.2 Parallel Performance Using a Single Hypernode. The parallelization effort within a single Hypernode went quite smoothly. It was quite simple to translate the SGI-compiler directives into Convex-compiler directives. Additionally, both sets of compiler directives can coexist in the same code, which greatly simplifies the job of maintenance. The main issues that arose at this point were:

- (1) Hypernodes only have eight processors. Given the lower peak speed of the processors on the Convex Exemplar (relative to the SGI Power Challenge), and the somewhat lower percentage of peak that this program achieves on the Convex Exemplar, being limited to a maximum of eight processors was frequently undesirable.
- (2) There were a number of issues that developed with some of the optimization features of the Convex Fortran compiler being buggy. Since the Convex compiler considers parallelization to be an optimization level, it was not possible to simply back down to a lower level of

optimization. Instead, it was necessary to identify which optimizations were buggy and then use suboptions to disable those features. In contrast, the SGI approach, which considers parallelization and optimization to be separate features, made it much easier to deal with compiler limitations.

- (3) It was found to be less convenient to control a number of important system parameters (e.g., maximum stack size) on the Convex Exemplar than on other commonly used machines.
- (4) When running on a dedicated Subcomplex, it was found that one could obtain a modest improvement in performance by having the processors *spin forever* when sitting at spin locks. However, if the Subcomplex were shared with other users, this could result in a serious drop in performance if the Subcomplex became even slightly overloaded (i.e., more than eight processes running on a single Hypernode). Similar issues exist on the SGI Power Challenge, but the unusual architecture of the Convex Exemplar seemed to substantially increase the frequency with which overloading occurred.

In summary, the effort to use multiple processors on a single Hypernode was quite successful. However, the limited performance that one saw when using a single Hypernode gave a strong impetus to the desire to parallelize across Hypernodes.

4.3 Parallel Performance Using Multiple Hypernodes. The parallelization effort when using multiple Hypernodes was a more complicated undertaking. Four issues had to be addressed:

- (1) How should one go about using multiple Hypernodes?
- (2) Once one has decided how to use multiple Hypernodes, one has to actually make the approach work. At times the documentation was not always clear on this point, resulting in a number of false starts.

(3) Is there any tuning that can/should be done at the system level that can improve performance?

(4) Finally, are there any additional types of tuning that are unique to this environment?

These issues will now be considered in greater detail:

(1) There are three main ways in which a program can be parallelized across multiple Hypernodes:

- Parallelize the program using message passing code, and only message passing code. This method might, in fact, have produced better results than the method that was selected (at least for programs that lend themselves to message passing). However, it would have meant producing a very different version of the code, and that violated the intent of this exercise.
- Parallelize the program entirely using compiler directives. This was an obvious continuation of the work that was already under way. Additionally, since the Convex Exemplar had been selected based on its support for the *shared memory* programming paradigm, this appeared to be a fair test of the machine.
- There are those who advocate using multiple levels of parallelism within a single program. If this approach had been selected, then one would probably use compiler directives within the individual Hypernodes, with message passing code between Hypernodes. This method was rejected for two main reasons. It seemed to violate the intent of the exercise, and it represented a major commitment of resources.

(2) Having decided to treat the Convex Exemplar as if it was a traditional shared memory architecture (e.g., *uniform memory access*), it was necessary to figure out how to do this. By using the system-level command **MPA**, one can cause the threads to be spread over as

many processors in a single Subcomplex as one desires. However, this does not mean that the memory is also spread over the Subcomplexes. By default, all of the memory is local to the Hypernodes, and this will cause the program to bomb. One can use MPA to make some or all of the memory global, but this raises questions such as:

- Which arrays should be global?

- How does one differentiate between the different arrays?

- Is there enough global memory in the system?

In general, one can make thread local data structures reside in Hypernode local memory for performance reasons and place all of the remaining data structures in global memory. This can be easily implemented using MPA. Additional tests were run placing certain arrays in local memory (this usually worked best for relatively invariant data, in which case, the entire array was replicated on each Hypernode). While this did have a measurable impact on performance, in general, the impact was too small to justify the effort, or the waste of memory.

A major problem was that even though the system being used had four Hypernodes with a total of 8 GB of memory, system limitations prevent one from configuring the system with more than 2 GB of global memory (the actual limit is somewhat less than that and is dependent on a number of factors that are beyond the scope of this report). However, this made it difficult to run some jobs that could easily be run when using either a single Hypernode on the Convex Exemplar or on the SGI Power Challenge. In some cases, it was possible to shoehorn a job in by placing certain arrays in local memory, but this was clearly an undesirable kludge.

- (3) A major problem with placing so many arrays in global memory is that the average memory latency for accessing global memory is significantly larger than it is for accessing local

memory. In an attempt to minimize this effect, the Convex Exemplar is equipped with the ability to use a portion of each Hypernode's memory as an extremely large cache. This is the CTI cache that was mentioned earlier, and it can be 64, 128, 256, or 512 MB per Hypernode in size. With help from Sally Parker and Sharon Shaw, the effect of using CTI caches with 64, 128, and 256 MB of memory was measured (the effort to use 512 MB of memory ran into problems and was abandoned). For this program, the 256-MB size produced the best results, allowing the program to run about 22% faster.

- (4) As previously mentioned, one can attempt to maximize the number of arrays that are kept in local memory. Unfortunately, this will frequently mean duplicating entire arrays on all of the Hypernodes, and that can tie up a lot of memory. Even worse, the process of initializing those arrays will substantially increase the startup time. This is partly due to significant contention that this can create in the memory system. Additionally, there is the problem that if the Subcomplex has N Hypernodes in it, the amount of work being performed at startup is now N times greater. In some cases, tradeoffs were made between computation and memory accesses. Given the increased cost of the memory accesses, it became clear that in most cases it was better to do the computation. It also was clear that as processors became faster, this would be the correct optimization on an ever increasing number of systems. Similarly, this analysis indicated that efforts should be made to minimize the number of cache misses associated with writing to arrays residing in global memory (e.g., minimize the use of matrix transpose operations in favor of other methods that can minimize the number of cache and TLB misses).

Considering the rather sizable performance hit one takes by using global memory, an interesting question is, Why was the CTI cache not more effective at reducing the penalty for using global memory? A related question is, Why were the other efforts to minimize/eliminate these penalties not more successful? There are several answers to these questions:

- (1) Access patterns can change from one loop to the next (e.g., one loop may be parallelized in the J direction, while the next one might be parallelized in the K direction). While it is

sometimes possible to minimize this effect, it cannot be eliminated. There are two consequences to this effect:

- Data from relatively invariant arrays might be stored in the CTI cache of more than one Hypernode. This *wastes* a valuable resource.
 - Cache misses associated with writes are always more expensive than cache misses associated with reads. Since certain arrays are updated by almost every loop in the program, one can expect to have an unavoidably large number of write misses associated with those arrays. In some cache cases, this effect can be so large as to call into question the benefit of the CTI. Unfortunately, in general, there is no practical manner in which these arrays can be stored in local memory.
- (2) Attempts to minimize the number of accesses to global memory have costs associated with them. This may be the cost of doing additional calculations. Or this may be the cost of copying the data to local memory in all of the Hypernodes. Regardless of the nature of the costs, what is important is that one is not replacing a cost X with a cost 0 . Rather, one is replacing a cost X with a cost Y , where Y is less than X .
- (3) Manually copying arrays to local memory in each Hypernode has a limited amount of scalability associated with it (one will get at best a parallel speedup of M , where in this case M is the smallest number of processors being used by this job in any of the Hypernodes). This makes this process inherently expensive, but it is of some value for arrays that are largely invariant. However, for arrays that change every time step, using this approach becomes unacceptably expensive.
- (4) Some of the techniques used to reduce the number of matrix transpose operations also had the effect of creating loops where all of the processors would be walking through a set of pages in memory at the same time and in the same order. This created additional contention

in the memory system, which limited the potential for parallel speedup in those loops. Overall, this optimization was still a win, but once again it was not a zero cost win.

From this analysis, one can see that system designs that use very large DRAM caches (either in the form of a CTI cache or COMA) are likely to exhibit significant performance problems that cannot be entirely eliminated.

5. Results

There are several commonly used metrics that one could use as a measure of how successful this project has been. While it is generally not practical to apply every possible metric to a project, the following discussion should give a fair assessment of this project.

One major aspect of this project was to gauge how successful Convex was in extending the *shared memory* programming paradigm. In this respect, it is less important as to whether or not Convex was successful than it is to have confidence in the conclusions. As this report indicates, it was assumed that some additional modifications and tuning would be required. As Figures 6 and 7 indicate, the efforts involving local memory were relatively successful. Therefore, it is safe to assume that most of the limitations in performance have to do with how Convex extended the *shared memory* paradigm and were not a failure on our part to effectively tune the code for their machine. Additionally, as Figure 8 indicates, the efforts to tune the code for parallelization across Hypernodes (e.g., replicating certain arrays in the local memory on each Hypernode) clearly demonstrated the performance problems associated with the use of global memory. Unfortunately, it also demonstrated that shared memory solutions to the problem were of limited value and were generally not acceptable for use on a production code in a production environment.

From this, it can be concluded that Convex's extension to the *shared memory* paradigm was not very successful. Does this mean that the Exemplar is a bad machine? Not necessarily. It might be a good choice for message passing code. It might also be a reasonable choice for a throughput-

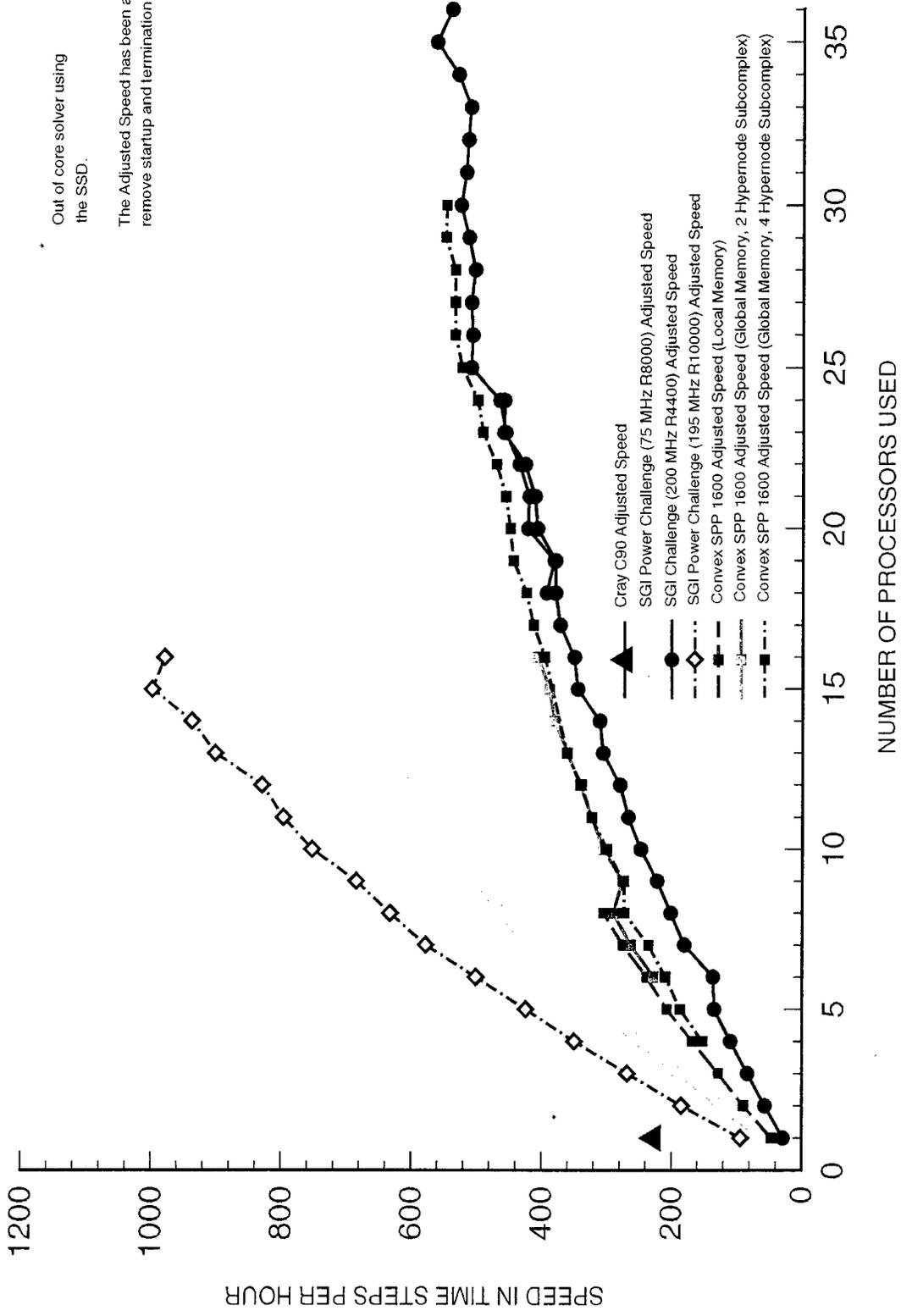


Figure 6. Performance Results for the 1-Million Grid-Point Test Case.

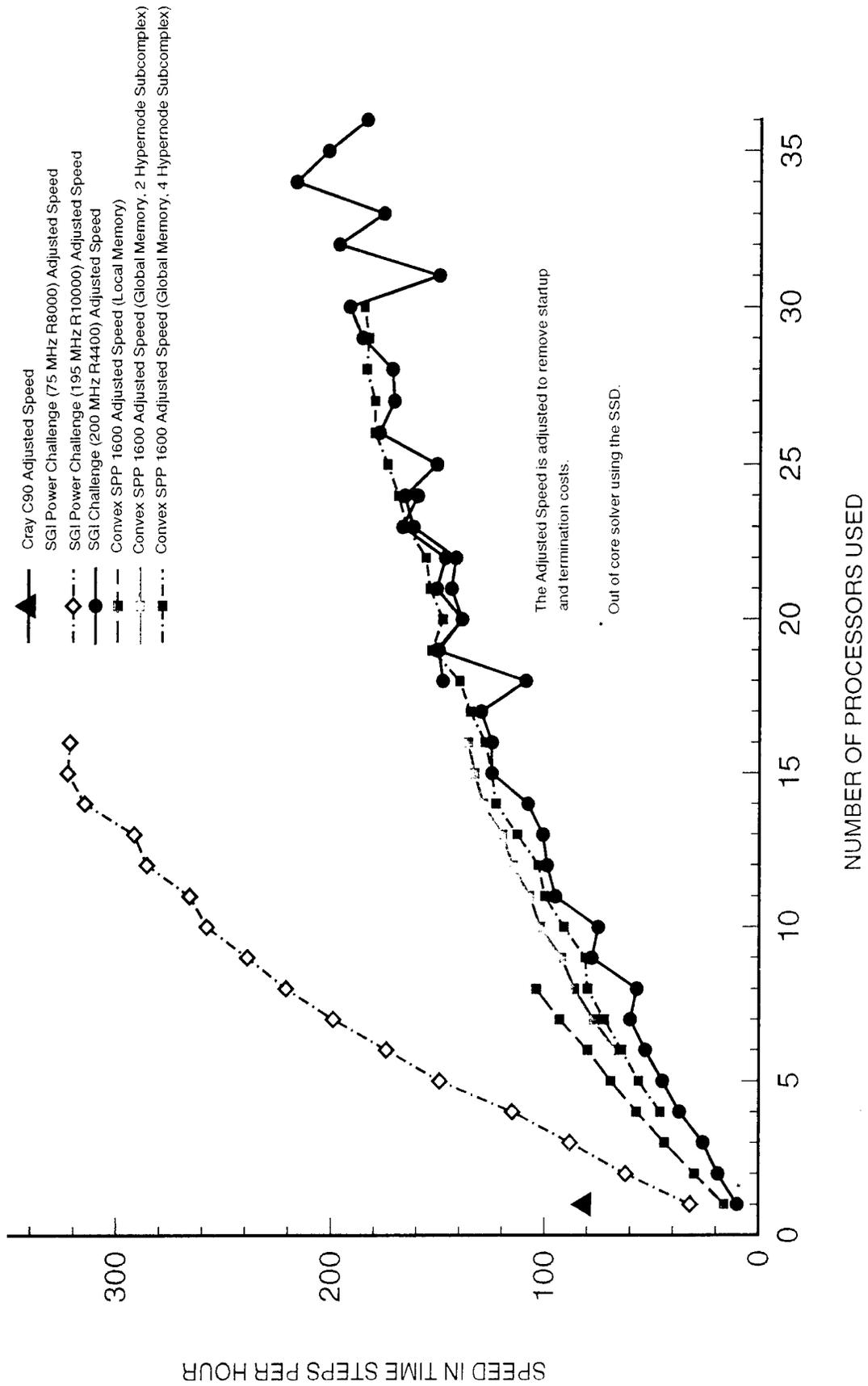


Figure 7. Performance Results for the 3-Million Grid-Point Test Case.

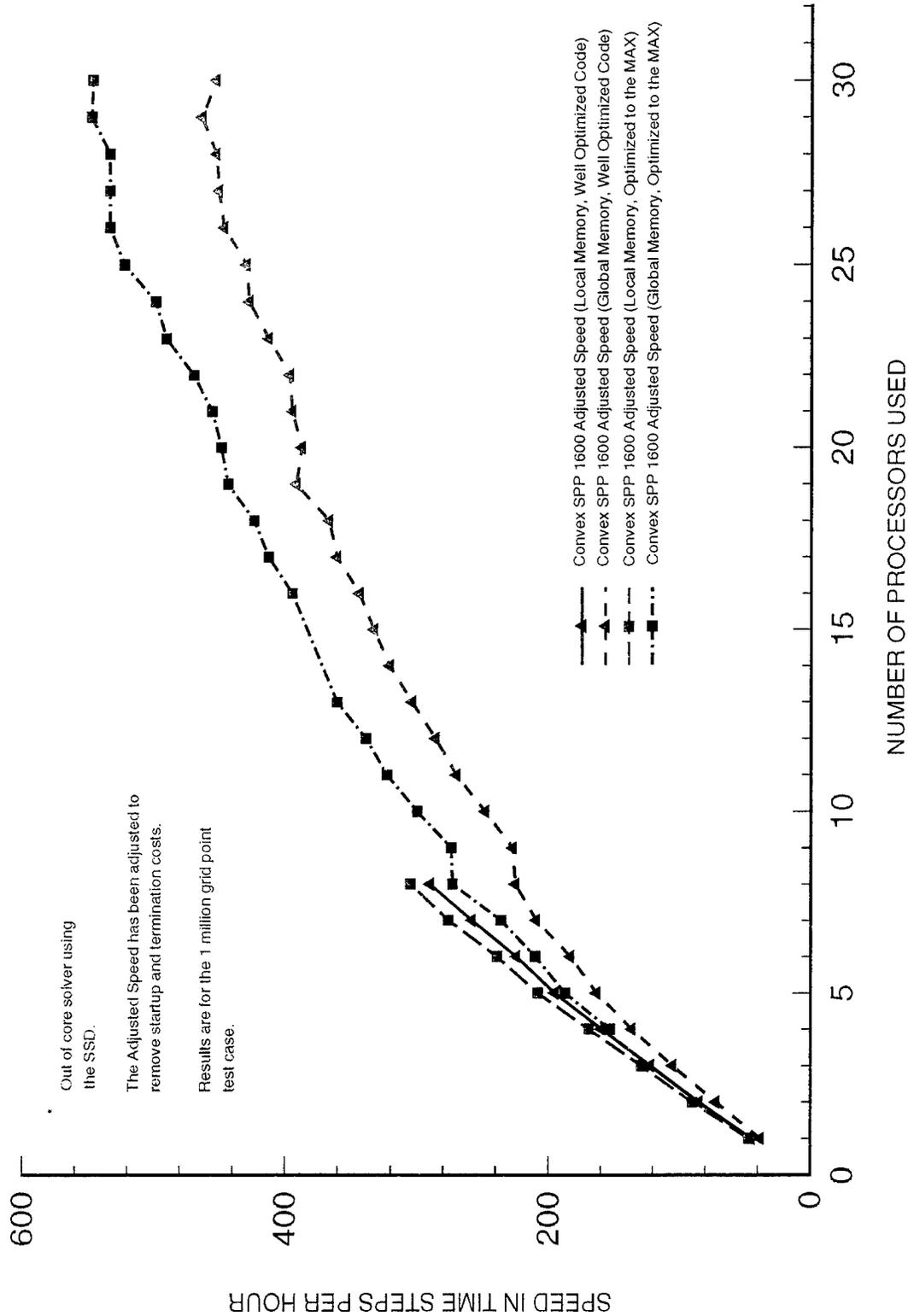


Figure 8. A Performance Comparison Between Using Optimization Techniques That Are Well Suited and Appropriate for Use on Any RISC-Based SMP, and Using Additional Techniques That Are Specifically Designed to Overcome the Limitations of the Convex Exemplar.

oriented environment where jobs would not be parallelized across Hypernodes. It just means that Convex failed to meet one of their more important design goals. This is important since there are other choices that one can make for throughput and message passing environments that might not require the same design tradeoffs that Convex made in designing this system.

Another key point was establishing that it was possible to write well-tuned code that would be portably efficient across *shared memory* environments from multiple vendors. So long as one only looks at the results for local memory in Figures 6 and 7, it would appear as though this goal was achieved. That it was far less successful when it came to runs involving global memory should be considered to be a limitation of the hardware, rather than a flaw in the basic concept.

This research was conducted as part of a **CHSSI** project. As such, one goal should certainly be to deliver efficient usable software. Therefore, a key question (after determining that the answers were correct—one of many services that Karen Heavey supplied as part of this project) is how well did the software perform relative to our expectations? Before one can answer that question, one needs to determine what those expectations were. Many people have argued that loop-level parallelism will not produce speedups in excess of a factor of 4–16 (depending on who one talks to and which system they are talking about). In some cases, this conclusion is based on some supposedly fundamental limitation of using loop-level parallelism. In other cases, this conclusion was based on the limitations of using a bus-based design (a major reason why Convex went with a more scalable design). Therefore, it would have been nice to see better than a factor of 4–16 speedup when using this code on multiple processors.

Another metric is to look at how well the code performed on the Exemplar vs. various bus-based systems from SGI. In theory, there are two reasons why the Exemplar should have outperformed the SGI boxes. First, with a theoretical peak speed for 30 processors of 7.2 **GFLOPS**, the machine was 12–200% faster than the SGI systems it is being compared to. Secondly, its performance is not hurt by the negative impact of using a bus-based design.

One final metric in this area is how does the performance of the code on the SGI and Convex machines stack up against the performance of the vector code running on one processor of a C90. Obviously, if the code does not run at least as fast on the new machines as on the C90, the value of this effort will be in doubt. Ideally, the new machines should either outperform the C90 run and/or be significantly more cost effective to use. For this part of the metric, we assume an allocated cost of about \$2 million for one processor of the C90. For the SGI systems, we assume that each of the three systems we looked at could be had for under \$1 million (the exact amount is, of course, configuration dependent). For the Convex Exemplar, the allocated cost for a single Hypernode is again assumed to be under \$1 million, while a four Hypernode system would cost between \$1 million and \$2 million.*

If one looks at Figures 6 and 7 and Table 1, several things become clear:

- (1) All of the platforms (including considering a single Hypernode of a Convex Exemplar using local memory to be a platform) were able to equal or exceed the performance of a single processor of a C90. In all cases, this was achieved both in terms of performance and the price-performance ratio that can be equally important in a throughput-oriented production environment (a common use of Cray vector computers).
- (2) The Convex Exemplar was in no way the fastest of the platforms. In fact, it was only slightly faster than the SGI Challenge, which it should have comfortably beaten by a factor of two.
- (3) All three of the SGI systems do show some performance degradation that is in large part due to the limited bus and memory bandwidth of these systems. However, this effect was found

* SGI and Convex supplied the cost data for their systems, while the allocated cost for one processor of a C90 is based on data found on the National Aeronautics and Space Administration (NASA) Ames web server.

Table 1. Performance Results

Machine Type	Number of Processors	Problem Size (Millions of Grid Points)	Performance (TS/Hr)	Price Performance (TS/Hr/Million Dollars)	Speed Relative to One Processor	
					Same Machine	Cray C90
Cray C90	1	1	227	117	1.0	1.0
	1	3	81	44	1.0	1.0
SGI Challenge	35	1	562	657	19.4	2.5
	35	3	202	242	20.2	2.5
SGI Power Challenge	18	1	754	1,170	11.1	3.3
	18	3	275	363	12.0	3.4
SGI R10K Power	15	1	999	1,970	10.6	4.4
	15	3	323	508	10.1	4.0
Convex SPP-1600 Local Memory	8	1	305	730	6.5	1.3
	8	3	104	242	6.5	1.3
Convex SPP-1600 Global Memory	30	1	547	372	11.6	2.4
	30	3	185	126	11.6	2.3

Note: Speedups relative to a single processor were computed based on single-processor runs using local memory.

to be manageable and in no way represents an impenetrable wall that makes it impossible to make effective use of more than a handful of processors.

- (4) Similarly, at this point, there seems to be no inherent limits to the use of loop-level parallelism with up to 36 processors, and possibly more. The biggest single limit at this point is the requirement/desirability of working in an efficient shared memory environment, and that current designs do not support unlimited numbers of processors. It is likely that this limit is to some extent inherent with this class of machines, although the Convex Exemplar is clear evidence that the vendors are attempting to extend the range of supported system sizes.

In order to better judge how well the code is performing on each of the machines, one needs to have a very good idea as to how well the code would perform on an ideal machine with a similar configuration. While different groups may approach this problem in different ways, the approach that was used in this case was based on the following assumptions and principles:

- (1) The version of F3D that has been used on Cray vector platforms at NASA Ames and the U.S. Army Research Laboratory (ARL) for many years is acceptably efficient on those platforms (Sahu and Steger 1990; Sahu 1990).
- (2) The actual performance of a code on other platforms is highly dependent on a number of factors. One of the more important factors is how well the tuning of the code has been optimized for the platform being used. While in theory this tuning can be performed either manually or by a compiler, it is our belief that a combination of those techniques will produce the best results. Therefore, this model will assume that the code has been aggressively tuned by hand and compiled with a state-of-the-art optimizing compiler (with an appropriate selection of optimization options turned on).
- (3) When considering the serial performance of the code, it is difficult to predict how well the code should perform on any given machine. However, since the vendors frequently market

their systems on the basis of *peak performance*, this report will assume that the predicted serial performance will be proportional to the peak speed of the processors. Furthermore, the measured speed of the vector optimized code on a Cray C90 will be used as the basis for predicting the speed of the code on other machines. This does not mean that this relationship will always hold. For most machines, there are well-known classes of codes that perform poorly on a given machine. Sometimes, this has to do with memory access patterns, while in other cases, it may be an indication that the code is not vectorizable (Kaufmann and Smarr 1993). Even so, given the marketing policies of most major computer companies, this seems like a reasonable place to start. The following are two final notes as to how this will be done:

- There can be significant variations in startup and termination costs from one system to the next. Additionally, in general, the sections of code responsible for these costs either cannot be parallelized or show limited speedup when they are parallelized (e.g., due to interaction with the OS and/or **I/O** devices). If one were to obtain fully converged solutions involving runs lasting for over 1,000 time steps, these costs might not be very important. However, it is impractical to perform such measurements for all of the runs that are required when carrying out this type of study. Instead, shorter runs were used with the run times adjusted in an effort to remove the startup and termination costs.
- When comparing the performance of a code running on multiple machines, the actual number of operations performed per second is of secondary importance. What is really important is how long does the run take to complete. Since this project made every effort not to change either the algorithm or its behavior, the run time should be proportional to the time required to perform a single time step (after making the adjustments mentioned in the previous paragraph). Therefore, all measurements of speed will be made in *time steps per hour*. One key advantage to this strategy is that different architectures may benefit from different optimization strategies. Furthermore, since it is reasonable to assume that these strategies will result in different numbers of operations being performed

(sometimes by more than a factor of two), this policy will allow us to concentrate on the only performance metric that is important to the user:

Time to completion.

(4) When considering parallel performance, one might hope for linear speedup, but there are three main problems with this hope:

- For fixed-sized problems, one may run out of available parallelism. If this happens, then the best one can hope for is linear speedup up until that point, with no additional speedup past that point.
- Even when one does not run out of available parallelism, there are a number of reasons why, for fixed-sized problems, one is unlikely to see linear speedup for very large numbers of processors (e.g., Amdahl's law, communication and synchronization costs) (Almasi and Gottlieb 1994).
- Linear speedup simply predicts that the relationship between speed and the number of processors used is of the form:

$$\text{Speed} = A + B * N,$$

where A and B are constants and N is the number of processors being used. Frequently, A is assumed to zero, although when fitting a curve to actual data, this need not be the case. The real problem arises when one rewrites this equation as:

$$\text{Speed} = A + C * D * N,$$

where $B = C * D$ and C is the speed of the code when using a single processor. In this case, if A is nonzero, D will not equal 1.0. This implies that even though the speedup is linear, doubling the number of processors need not double the speed of the system.

Similarly, if $A = 0.0$ and $D = 1.0$, but C is a small fraction of the percentage of peak performance for a single processor, one may see linear speedup and still fail to meet the overall performance goals of the project.

- (5) A common solution to the issues in paragraph 4 on the previous page is to talk about something called scaled speedup (Gustafson 1988). The main problem with scaled speedup is that it assumes that the available parallelism is proportional to the problem size. While for many of the algorithms commonly used on large parallel computers, this assumption is correct, for many classes of algorithms, this assumption is dead wrong. Unfortunately, for this code, the available parallelism is roughly proportional to the cube root of the problem size. This means that in order to use 64 processors, the problem size would need to be 262,144 times the size of the problem that one would normally solve using just one processor. Clearly, under these conditions, scaled speedup is not a useful metric.
- (6) At this point, it will be assumed that the correct scaling function is a stairstep, with the exact shape of the curve depending on the grid dimensions (see Table 2 for an example of how this curve naturally arises when using loop-level parallelism). For sufficiently large problem sizes (or small numbers of processors), this curve will closely approximate linear speedup (and with the assumptions for serial performance, this is a meaningful result). It will also be assumed that secondary effects such as Amdahl's law and communication and synchronization costs can be ignored, since most shared memory systems have only a few processors. Hardware effects such as limited memory bandwidth will also be ignored, since the extent to which measurements deviate from the predicted levels of performance will be a good metric of how well the system is living up to expectations.

Table 2. Predicted Speedup for a Loop With 15 Units of Parallelism

Number of Processors	Maximum Units of Parallelism Assigned to a Single Processor	Predicted Speedup
1	15	1.000
2	8	1.875
3	5	3.000
4	4	3.750
5-7	3	5.000
8-14	2	7.500
15	1	15.000

Figures 9-12 show the predicted and measured results for a variety of SGI and Convex systems. If one looks at the results for the SGI Challenge system, one will see excellent agreement between the measured and predicted levels of performance. This is strong evidence that the methodology used has merit. If one considers the SGI Power Challenge and SGI R10K Power Challenge, one does see some deviations from the predicted performance when using larger numbers of processors. Overall, however, the agreement between predicted performance and measured performance is still pretty good. The same can also be said about the Convex Exemplar when using up to eight processors with local memory on a single Hypernode. However, results when using more than eight processors on a Convex Exemplar are very disappointing.

6. Future Plans

At the present time, several additional efforts relating to this code are underway. Some of these are:

- (1) Marek Behr, Ph.D., of the U.S. Army High Performance Computing and Research Center (AHPCRC) has been porting the same code to the Cray T3D and other traditional distributed memory RISC-based **MIMD** MPPs. The author has been collaborating with him to share some of his serial optimizations with this effort.

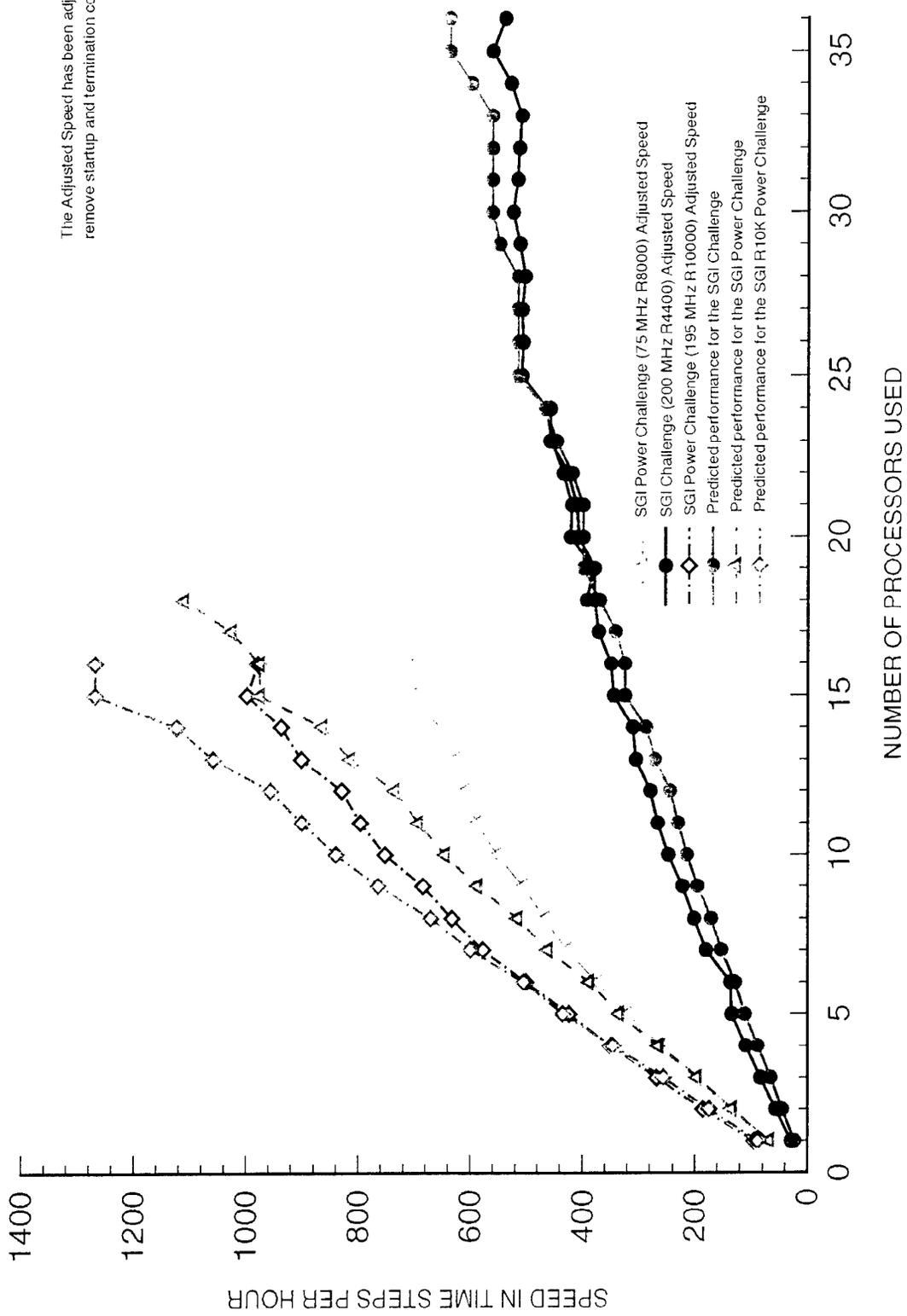


Figure 9. A Comparison of the Predicted and Measured Levels of Performance for SGI Challenge and Power Challenge Systems (1-Million Grid-Point Test Case).

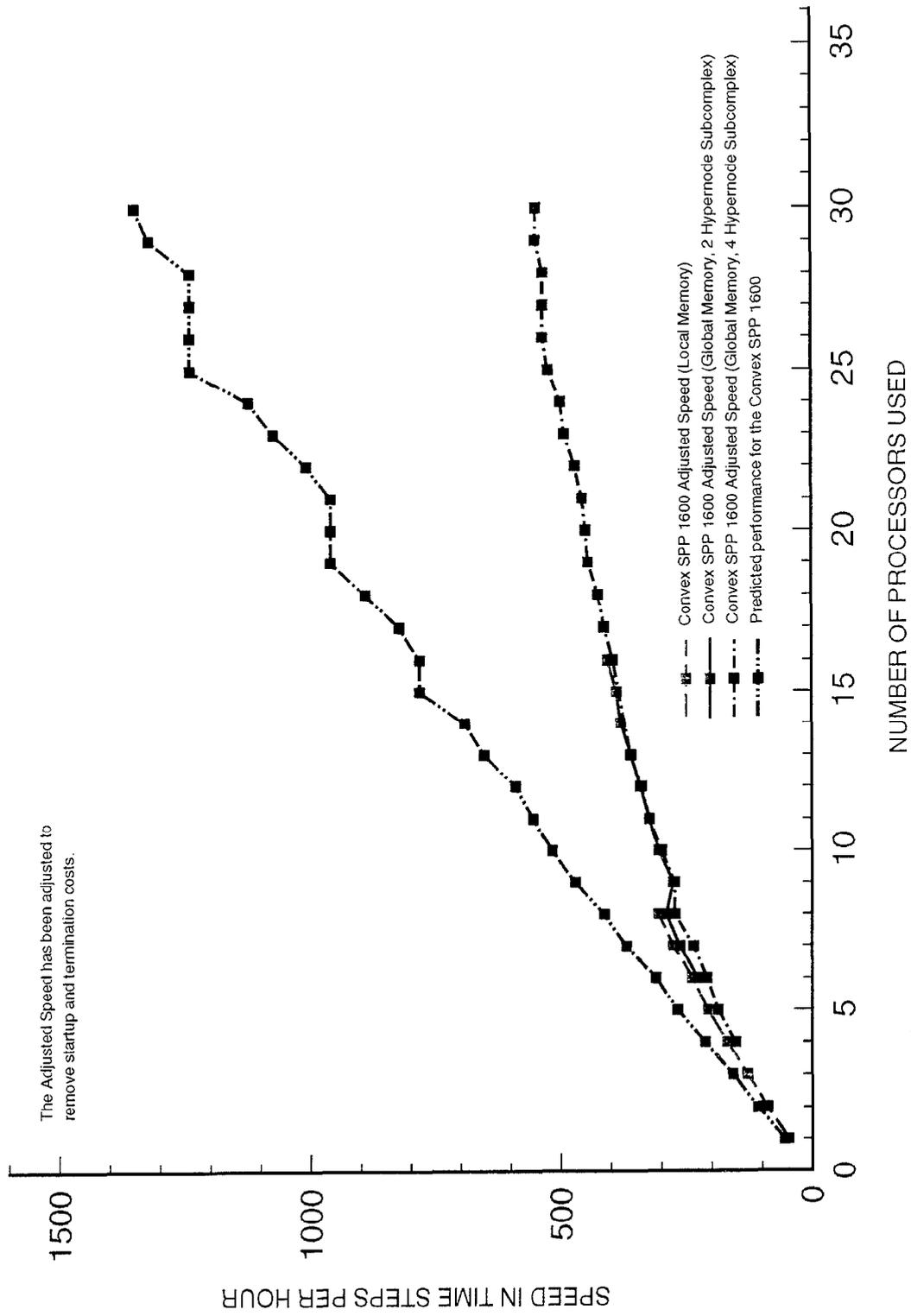


Figure 10. A Comparison of the Predicted and Measured Levels of Performance for One, Two, and Four Hypernode Subcomplexes on a Convex SPP 1600 (1-Million Grid-Point Test Case).

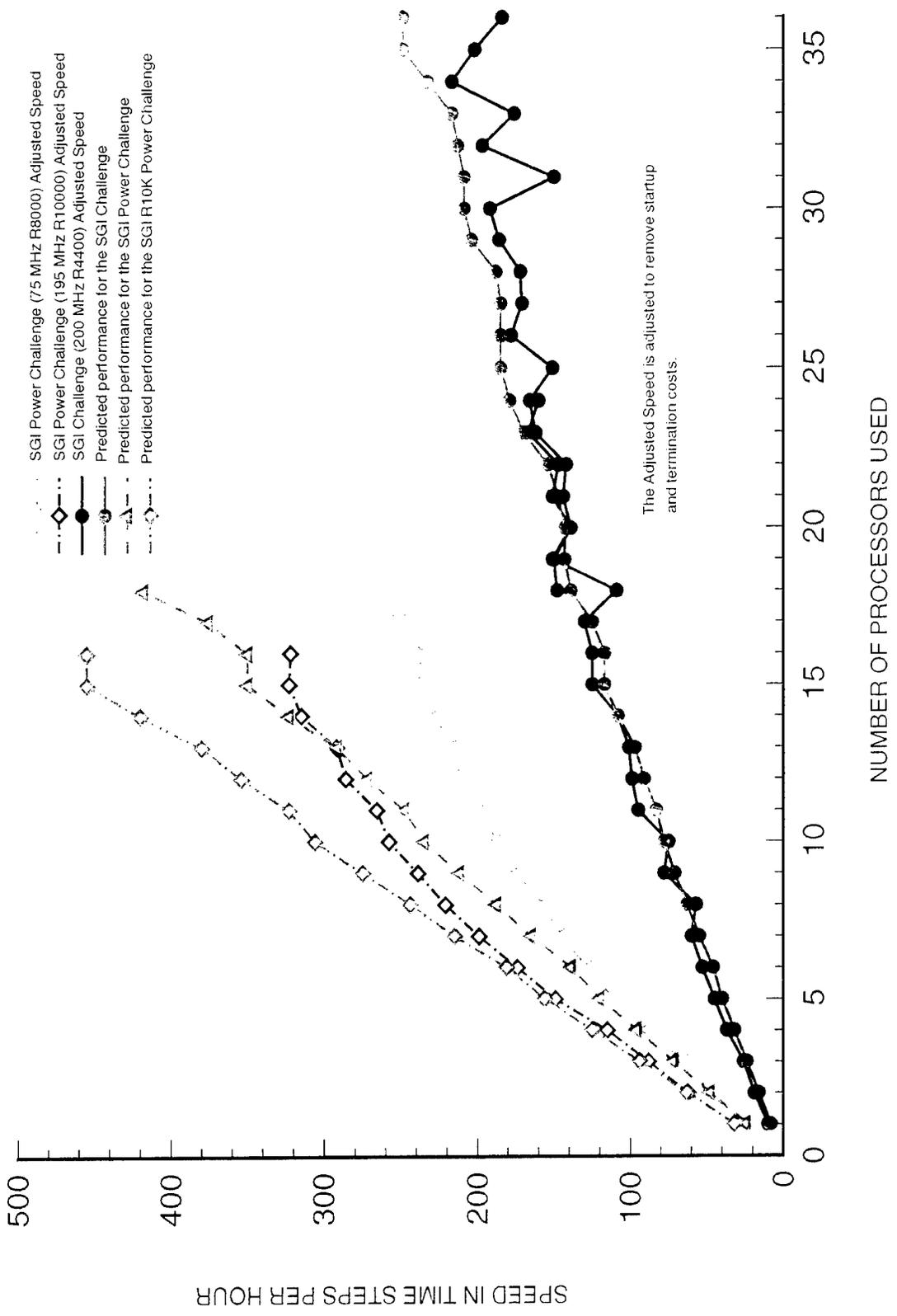


Figure 11. A Comparison of the Predicted and Measured Levels of Performance for SGI Challenge and Power Challenge Systems (3 Million Grid-Point Test Case).

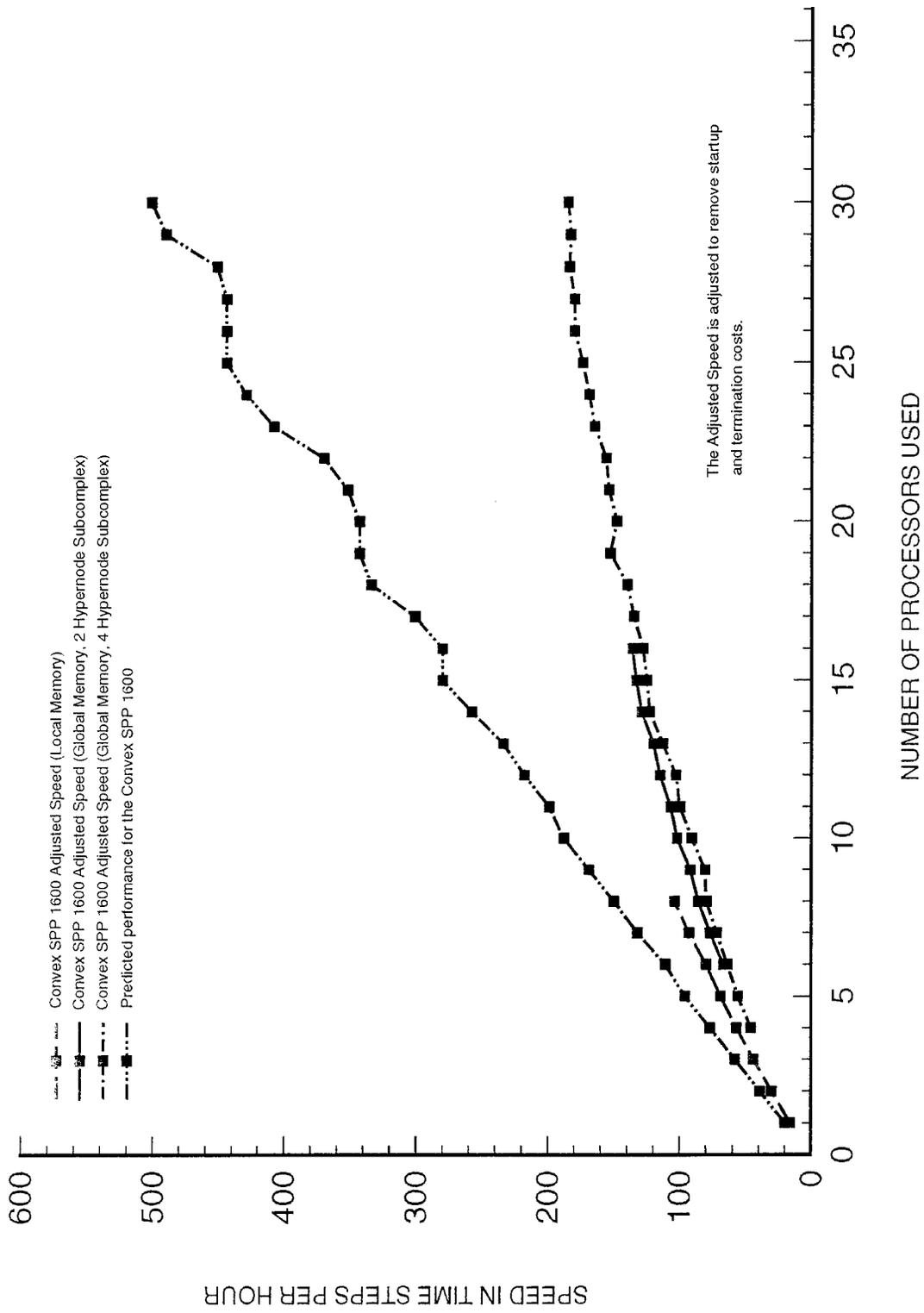


Figure 12. A Comparison of the Predicted and Measured Levels of Performance for One, Two, and Four Hypernode Subcomplexes on a Convex SPP 1600 (3-Million Grid-Point Test Case).

- (2) James Collins, Ph.D., ARL, is currently adding several additional modules to this code, including an implementation of the **CHIMERA** code.
- (3) James Collins, and Jubaraj Sahu, Ph.D., ARL, and others are beginning work on a formal plan to revalidate the tuned code.
- (4) The author has begun initial investigations to identify and, if necessary, correct any problems that might occur when the problem is scaled to even larger sizes. Unfortunately, most of the currently delivered systems either lack sufficient memory and/or sufficient address space to handle problems that are significantly larger than those used in this study.
- (5) Some of the routines currently in the F3D code were neither optimized (other than to the extent necessary to maintain their validity) nor parallelized. While many of these routines are so fast that there was no need to tune them, a significant percentage of the untuned routines are computationally intensive, but were not used in processing the benchmark case. Eventually, it would be desirable if some effort was made to either improve the speed of these routines or to delete them from the standard distribution.

7. Conclusions

It is now clear that some—probably many, but possibly not all—computationally intensive codes can be tuned so that a meaningful range of problem sizes can be run with an acceptable level of performance on the current generation of RISC-based shared memory SMPs. Having said that, it is also clear that transitioning code to these machines will be far from the plug-and-play process many potential users are hoping for. When performing this work, it is highly desirable to have the same code work on more than one vendor's product line. This effort shows that, in general, it is possible to achieve such a goal. However, it has also shown that since the design of some machines is better than the design of other machines, the results will not, in general, be uniformly good. This is not a new conclusion, nor is it unique to shared memory platforms running programs parallelized

using loop-level parallelism. However, given the limited number of vendors currently making these systems for the high performance computer market, this conclusion is more important than might otherwise be the case.

INTENTIONALLY LEFT BLANK.

8. References

- Almasi, G. S., and A. Gottlieb. *Highly Parallel Computing*. Redwood City, CA: The Benjamin/Cummings Publishing Company, Inc., second edition, 1994.
- Gustafson, J. L. "Reevaluating Amdahl's Law." *Communications of the ACM*, vol. 31, no. 5, pp. 532-533, May 1988.
- Kaufmann, W. J. III, and L. L. Smarr. *Supercomputing and the Transformation of Science*. New York, NY: Scientific American Library, 1993.
- Pressel, D. M. "Early Results From the Porting of the Computational Fluid Dynamics Code, F3D, to the Silicon Graphics Power Challenge." ARL-TR-1562, U.S. Army Research Laboratory, Aberdeen Proving Ground, MD, December 1997.
- Sahu, J. "Numerical Computations of Transonic Critical Aerodynamic Behavior." *American Institute for Astronautics and Aeronautics Journal*, vol. 28, no. 5, pp. 807-816, May 1990 (also see BRL-TR-2962, December 1988).
- Sahu, J., and J. L. Steger. "Numerical Simulation of Transonic Flows." *International Journal for Numerical Methods in Fluids*, vol. 10, no. 8, pp. 855-873, 1990.

INTENTIONALLY LEFT BLANK.

Abbreviations

AHPCRC	U.S. Army High Performance Computing and Research Center
ARL	U.S. Army Research Laboratory
CHSSI	Common High Performance Computing Software Support Initiative
DOD	Department of Defense
HP	Hewlett-Packard
MSRC	Major Shared Resource Center
NASA	National Aeronautics and Space Administration
NWSC	Naval Warfare System Center
SGI	Silicon Graphics Inc.
SLAD	Survivability/Lethality Analysis Directorate

INTENTIONALLY LEFT BLANK.

Glossary

Cache	a high-speed memory used to temporarily store data that has recently been accessed, or is likely to be accessed in the near future
CFD	computational fluid dynamics
CHIMERA	a method for handling overlapping zones
CHSSI	Common High Performance Computing Software Support Initiative
COMA	cache only memory architecture
CPU	central processing unit
DRAM	dynamic random access memory
Fortran	the most commonly used scientific programming language
GFLOPS	giga floating-point operation per second
I/O	input/output
MFLOPS	mega floating-point operation per second
MIMD	multiple instruction/multiple data
MPA	a system utility on convex systems
MPP	massively parallel processor
OS	operating system
RISC	reduced instruction set computer
SMP	symmetric multiprocessor
SRAM	static random access memory
thrashing a TLB	to have a high rate of TLB misses, which in the limit can approach (or even in some rare instances exceed) one miss per memory access
TLB	translation lookaside buffer

INTENTIONALLY LEFT BLANK.

<u>NO. OF COPIES</u>	<u>ORGANIZATION</u>
2	DEFENSE TECHNICAL INFORMATION CENTER DTIC DDA 8725 JOHN J KINGMAN RD STE 0944 FT BELVOIR VA 22060-6218
1	HQDA DAMO FDQ DENNIS SCHMIDT 400 ARMY PENTAGON WASHINGTON DC 20310-0460
1	OSD OUSD(A&T)/ODDDR&E(R) R J TREW THE PENTAGON WASHINGTON DC 20301-7100
1	DPTY CG FOR RDE US ARMY MATERIEL CMD AMCRD MG CALDWELL 5001 EISENHOWER AVE ALEXANDRIA VA 22333-0001
1	INST FOR ADVNCD TCHNLGY THE UNIV OF TEXAS AT AUSTIN PO BOX 20797 AUSTIN TX 78720-2797
1	DARPA B KASPAR 3701 N FAIRFAX DR ARLINGTON VA 22203-1714
1	NAVAL SURFACE WARFARE CTR CODE B07 J PENNELLA 17320 DAHLGREN RD BLDG 1470 RM 1101 DAHLGREN VA 22448-5100
1	US MILITARY ACADEMY MATH SCI CTR OF EXCELLENCE DEPT OF MATHEMATICAL SCI MAJ M D PHILLIPS THAYER HALL WEST POINT NY 10996-1786

<u>NO. OF COPIES</u>	<u>ORGANIZATION</u>
1	DIRECTOR US ARMY RESEARCH LAB AMSRL D R W WHALIN 2800 POWDER MILL RD ADELPHI MD 20783-1145
1	DIRECTOR US ARMY RESEARCH LAB AMSRL DD J J ROCCHIO 2800 POWDER MILL RD ADELPHI MD 20783-1145
1	DIRECTOR US ARMY RESEARCH LAB AMSRL CS AS (RECORDS MGMT) 2800 POWDER MILL RD ADELPHI MD 20783-1145
3	DIRECTOR US ARMY RESEARCH LAB AMSRL CI LL 2800 POWDER MILL RD ADELPHI MD 20783-1145
	<u>ABERDEEN PROVING GROUND</u>
4	DIR USARL AMSRL CI LP (305)

<u>NO. OF COPIES</u>	<u>ORGANIZATION</u>
1	PM CHSSI J GROSH SUITE 650 1110 N GLEVE ROAD ARLINGTON VA 22201
1	ARMY HIGH PERFORMANCE COMPUTING RSRCH CTR M BEHR SUITE 101 1100 WASHINGTON AVE SOUTH MINNEAPOLIS MN 55415
1	COMMANDER CODE C2892 C HOUSH 1 ADMINISTRATION CIRCLE CHINA LAKE CA 93555
1	WL FIMC S SCHERR BLDG 450 2645 FIFTH ST SUITE 7 WPAFB OH 45433-7913
1	NSWC CODE B44 A B WARDLAW SILVER SPRING MD 20903-5640
1	NAVAL RSRCH LAB CODE 6400 J BORIS 4555 OVERLOOK AVE SW WASHINGTON DC 20375-5344
1	WL FIMC B STRANG BLDG 450 2645 FIFTH ST SUITE 7 WPAFB OH 45433-7913
1	NAVAL RSRCH LAB CODE 6410 R RAMAMURTI WASHINGTON DC 20375-5344
1	ARMY AEROFLIGHT DYNAMICS DIR R MEAKIN MS 258 1 MOFFETT FIELD CA 94035-1000

<u>NO. OF COPIES</u>	<u>ORGANIZATION</u>
1	NAVAL RSRCH LAB CODE 7320 J W MCCAFFREY JR STENNIS SPACE CENTER MS 39529
1	US AIR FORCE WRIGHT LAB WL FIM J J S SHANG 2645 FIFTH STREET STE 6 WPAFB OH 45433-7912
1	USAF PHILIPS LAB OLAC PL RKFE CPT S G WIERSCHKE 10 EAST SATURN BLVD EDWARDS AFB CA 93524-7680
1	USAE WATERWAYS EXPERIMENT STATION CEWES HV C J P HOLLAND 3909 HALLS FERRY ROAD VICKSBURG MS 39180-6199
1	US ARMY RSRCH LAB AMSRL PS E B S PERLMAN FT MONMOUTH NJ 07703
1	NCCOSC RDT&E DIV NRAD CODE 404 R A WASILAUSKY 53570 SILVERGATE AVE SAN DIEGO CA 92152-5180
1	US AIR FORCE ROME LAB RL OCTS R W LINDERMAN GRIFFISS AFB NY 13441-5700
1	NCCOSC RDTE DV NRAD CODE 7601T K BROMLEY 5180 SILVERGATE AVE SAN DIEGO CA 92152-5180
1	DIRECTOR DEPT OF ASTRONOMY PROF P WOODWARD 356 PHYSICS BLDG 116 CHURCH STREET SE MINNEAPOLIS MN 55455
1	DIRECTOR ARMY HIGH PERFORMANCE COMPUTING RSRCH CTR T TEZDUYAR 1200 WASHINGTON AVE SOUTH MINNEAPOLIS MN 55415

NO. OF
COPIES ORGANIZATION

- 1 DIRECTOR
ARMY HIGH PERFORMANCE
COMPUTING RSRCH CTR
B BRYAN
1200 WASHINGTON AVE
SOUTH MINNEAPOLIS MN 55415
- 1 DIRECTOR
ARMY HIGH PERFORMANCE
COMPUTING RSRCH CTR
G V CANDLER
1200 WASHINGTON AVE
SOUTH MINNEAPOLIS MN 55415
- 1 NAVAL CMND CONTROL AND
OCEAN SURVEILLANCE CTR
L PARNELL
NCCOSC RDTE DIV D3603
49590 LASSING ROAD
SAN DIEGO CA 92152-6148
- 1 SPAWARSSYSCEN D4123
S PARKER
RM 331
53140 SYSTEMS ST
SAN DIEGO CA 92152-7560

ABERDEEN PROVING GROUND

- 15 DIR USARL
AMSRL CI C NIETUBICZ
AMSRL CI HA W STUREK
AMSRL CI HC
D PRESSEL
D HISLEY
C ZOLTANI
J GROSH
A PRESSLEY
T KENDALL
P DYKSTRA
AMSRL SC S A MARK
AMSRL WM B
H EDGE
J SAHU
K HEAVEY
P WEINACHT
AMSRL WM TC K KIMSEY

INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			<i>Form Approved OMB No. 0704-0188</i>	
<small>Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.</small>				
1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE March 1999	3. REPORT TYPE AND DATES COVERED Final, Jan 96 - Dec 96		
4. TITLE AND SUBTITLE Results From the Porting of the Computational Fluid Dynamics Code F3D to the Convex Exemplar (SPP-1000 and SPP-1600)		5. FUNDING NUMBERS 611102H4800		
6. AUTHOR(S) Daniel M. Pressel				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) U.S. Army Research Laboratory ATTN: AMSRL-CI-HC Aberdeen Proving Ground, MD 21005-5067		8. PERFORMING ORGANIZATION REPORT NUMBER ARL-TR-1923		
9. SPONSORING/MONITORING AGENCY NAMES(S) AND ADDRESS(ES)		10. SPONSORING/MONITORING AGENCY REPORT NUMBER		
11. SUPPLEMENTARY NOTES				
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.		12b. DISTRIBUTION CODE		
13. ABSTRACT (Maximum 200 words) This report discusses the continuing efforts to port the F3D computational fluid dynamics code to RISC-based SMPs. Originally, this program was optimized for Cray vector supercomputers such as the Cray C90. Previous attempts to run this code on SGI Power Challenges, and Convex Exemplars, as well as systems from SUN and Digital Equipment, demonstrated a level of performance that was so low as to be utterly useless (in many cases, it became necessary to kill the job before the first time step had completed). After making a concerted effort to port the program to an SGI Power Challenge (R8000 processor), acceptable levels of performance were finally achieved (Pressel 1997). Using this version of the code as the starting point, an effort was made to produce a program that ran efficiently on both systems from SGI and Convex. Unfortunately, a number of limitations with the Convex Exemplar were discovered that limited the success of this effort.				
14. SUBJECT TERMS computational fluid dynamics, supercomputing, high-performance computing		15. NUMBER OF PAGES 50		16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL	

INTENTIONALLY LEFT BLANK.

USER EVALUATION SHEET/CHANGE OF ADDRESS

This Laboratory undertakes a continuing effort to improve the quality of the reports it publishes. Your comments/answers to the items/questions below will aid us in our efforts.

- 1. ARL Report Number/Author ARL-TR-1923 (Pressel) Date of Report March 1999
- 2. Date Report Received _____
- 3. Does this report satisfy a need? (Comment on purpose, related project, or other area of interest for which the report will be used.) _____

- 4. Specifically, how is the report being used? (Information source, design data, procedure, source of ideas, etc.) _____

- 5. Has the information in this report led to any quantitative savings as far as man-hours or dollars saved, operating costs avoided, or efficiencies achieved, etc? If so, please elaborate. _____

- 6. General Comments. What do you think should be changed to improve future reports? (Indicate changes to organization, technical content, format, etc.) _____

CURRENT ADDRESS	_____
	Organization

	Name E-mail Name

	Street or P.O. Box No.

	City, State, Zip Code

7. If indicating a Change of Address or Address Correction, please provide the Current or Correct address above and the Old or Incorrect address below.

OLD ADDRESS	_____
	Organization

	Name

	Street or P.O. Box No.

	City, State, Zip Code

(Remove this sheet, fold as indicated, tape closed, and mail.)
(DO NOT STAPLE)

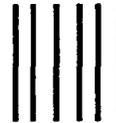
DEPARTMENT OF THE ARMY

OFFICIAL BUSINESS

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO 0001, APG, MD

POSTAGE WILL BE PAID BY ADDRESSEE

DIRECTOR
US ARMY RESEARCH LABORATORY
ATTN AMSRL CI HC
ABERDEEN PROVING GROUND MD 21005-5067



NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

