The Scalability of Loop-Level Parallelism

by Daniel M. Pressel

ARL-TR-2557 August 2001

Army Research Laboratory
Aberdeen Proving Ground, MD 21005-5067

ARL-TR-2557 **August 2001**

The Scalability of Loop-Level Parallelism

Daniel M. Pressel
Computational and Information Sciences Directorate, ARL

Abstract

This report deals with the four main constraints on the scalability of programs parallelized using loop-level parallelism. They are as follows:

(1) The available parallelism in the algorithm.
(2) The availability and scalability of appropriate hardware (including the operating system and the compilers).
(3) Limitations in the design of the hardware.
(4) The cost of getting into and out of a parallel section of code.

This, in turn, will lead to two important discussions: (1) the theoretical limitations on the scalability of shared memory codes and (2) the role that the choice of hardware and usage policies play in determining the performance of a shared memory code.

These discussions will include examples from the author's own work in porting the implicit computational fluid dynamics code F3D from the Cray C90 to a variety of shared memory platforms.

Acknowledgments

The author thanks Marek Behr, formerly of the U.S. Army High Performance Computing Research Center (AHPCRC), for sharing his results and the many colleagues who worked on these research projects over the years and helped collect this data and prepare this report. The author would also like to thank the employees of Business Plus, especially Claudia Coleman and Maria Brady, who assisted in the preparation and editing of this report.

Special thanks to Tom Kendall, Denice Brown, and the entire systems staff at the ARL-MSRC for their support of the various projects for which these runs were originally done.

This work was made possible through a grant of computer time by the Department of Defense (DOD) High Performance Computing Modernization (HPCM) Program. Additionally, some of the results mentioned in this work came from projects that were funded as part of the Common High Performance Computing Software Support Initiative (CHSSI) administered by the DOD HPCM Program.

INTENTIONALLY LEFT BLANK.

Contents

INTENTIONALLY LEFT BLANK.

List of Figures

INTENTIONALLY LEFT BLANK.

List of Tables

INTENTIONALLY LEFT BLANK.

1. Introduction

OpenMP supports both task-level parallelism and loop-level parallelism. It appears that at least in the short term, many programs parallelized using OpenMP will use loop-level parallelism. This report addresses the scalability issues of loop-level parallelism, although portions of the discussion will be equally relevant to programs parallelized using OpenMP's task-level parallelism features. Most of this discussion is based on the author's work on systems from SGI, SUN, and Convex using their proprietary compiler directives for loop-level parallelism. This work has been recently supplemented with work done on a system from HP using KAI's guide program to run a program using OpenMP compiler directives. The following are the four main constraints on the scalability of programs parallelized using loop-level parallelism:

(1) the available parallelism in the algorithm,

(2) the availability and scalability of appropriate hardware (including the operating system and the compilers),

(3) limitations in the design of the hardware, and

(4) the cost of getting into and out of a parallel section of code.

Most work with parallel computing seems to have assumed that a program will have a nearly infinite level of parallelism. Historically, this assumption was required since achieving meaningful levels of performance on a parallel computer meant using hundreds, or even thousands, of processors. However, in considering the case of using loop-level parallelism to parallelize a loop for a three-dimensional (3-D) problem in **CFD**, the available level of parallelism will almost always be less than 1,000 and possibly less than 100. In this case, when using larger numbers of processors, the ideal speedup is no longer linear. Instead, the ideal speedup now resembles a stair step. The second constraint refers to the requirement for systems with enough processors. It also requires an appropriate level of support from the operating system and the compilers. The design of the hardware can be of particular importance. With the Cray T3D and the CRAFT programming model (Oberlin 99), it is important to make every effort to minimize the effective memory latency. This includes minimizing the cost of off node accesses for **NUMA** architectures. Furthermore, not only must an adequate level of memory bandwidth be ensured, but with as few points of contention as possible (e.g., bank conflicts, insufficient bus bandwidth for bus-based systems, or insufficient off-node bandwidth for NUMA- and **COMA**-based systems).

These lessons are now extremely important, especially when considering the possible production of software distributed shared memory implementations of OpenMP. It is not sufficient for a programming paradigm to be portable; it must also deliver acceptable and predictable levels of performance. Without these guarantees, OpenMP will be no more useful than **HPF**.

Possibly the most interesting constraint is the final constraint, the cost of getting into and out of a parallel section of code. Many systems seem to have a lower bound for this number of around 2,000 cycles when using 10 or more processors. The upper bound for this number is virtually unlimited and can easily exceed 1-million cycles. The reason for this disparity is that the lower bound is driven by hardware considerations, while the upper bound is strongly affected by usage policies (e.g., the time sharing of processors). A proposed solution is to have the system reduce the number of threads assigned to a job if the system becomes **overloaded**. However, this can result in very unpredictable run times. In addition, when using larger numbers of processors, this can move the job from the high side of a stair step to the low side of the stair step. Consequently, a significant amount of computer time for a production run can be wasted.

The fourth constraint, the cost of getting into and out of a parallel section of code, also becomes relevant in that it is easy to see the desirability of parallelizing outer loops (middle loops can sometimes be used, but are not as desirable). Additionally, maximizing the amount of work in the parallelized loop may require merging loops. However, even after all of these transformations have been made, some loops, especially those in boundary condition routines, may not have enough work to justify the overhead of parallelization. This implies that Amdahl's Law may be a significant limitation when using larger numbers of processors. The traditional solution to this problem is to discuss scaled speedup. However, the available parallelism will not be proportional to the problem size unless a loop nest is parallelized in all directions. This violates one of the premises of scaled speedup. Therefore, even for fairly large problem sizes, Amdahl's Law cannot be ignored. Of course, when dealing principally with large problem sizes, it might be possible to justify parallelizing and optimizing a larger subset of subroutines. However, for many projects this does not appear to be a good assumption.

2. Available Parallelism

Most efforts to parallelize programs have assumed that the available parallelism is nearly infinite. Therefore, they have stressed concepts such as load balancing, optimizing the communications pattern for a grid undergoing domain decomposition, and even some such generic sounding concepts as Amdahl's Law

and the need to optimize the ratio between computation and communication. However, when parallelizing loops, there is the very real possibility that one or more of the expensive loops will have a dependency in at least one direction. If this is the case, then the available parallelism will no longer be strictly proportional to the problem size. As a result, the ideal speedup will no longer be linear. Instead, it will resemble a stair step. Table 1 and Figure 1 demonstrate this effect for a 3-D problem with dependencies in two out of three directions, where there are only 15 iterations in the loop being parallelized.

Table 1. 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

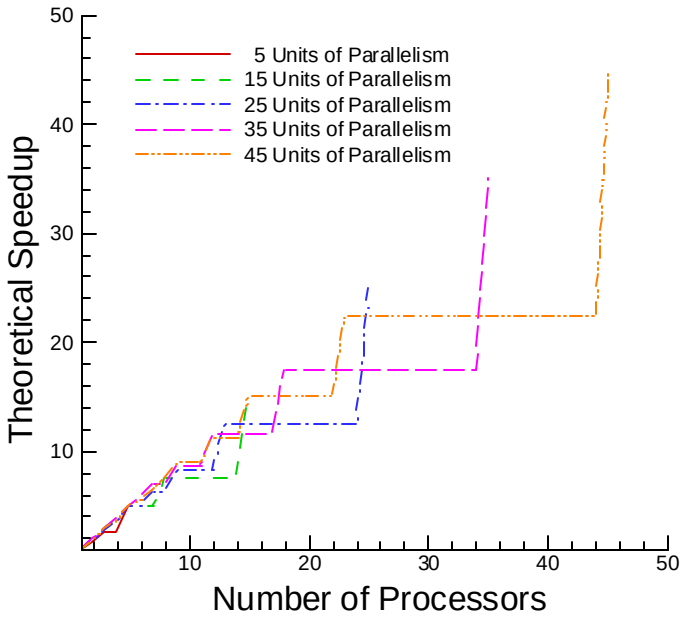

Figure 1. Predicted speedup for loops with various levels of parallelism.

For larger problem sizes and for two-dimensional problems, it is reasonable to assume that the available parallelism will be greater than 15. However, when scaling to 100 processors, it is likely that this stair-stepping effect will become significant. For example, if the loop being parallelized has 200 iterations, there will be stair steps at 50, 67, and 100 processors (a maximum of 4, 3, and 2 units of work per processor, respectively).

3. The Availability and Scalability of Appropriate Systems

While it may have some educational value (e.g., helping one to get a Ph.D.), creating parallel programs and programming techniques without regard to our ability to efficiently run those programs/use these techniques will not help in getting the job done. In terms of the hardware, this means several things:

(1) In theory, peak speed of the hardware must be great enough to meet the performance requirements of a job.

(2) Since it is rare to get close to 100% of peak performance, the peak performance must be great enough that even after the serial and parallel performance are discounted to appropriate degrees, a reasonable expectation of success exists.

(3) The choice of hardware must be well matched to the parallelization technique. In particular, if the technique relies on shared memory, then a production effort should probably be based on the use of hardware shared memory.

(4) To meet the needs of a job, the usage policies must support using a sufficiently large percentage of a system's resources by a single job.

(5) The operating system must be sufficiently scalable to properly support the size of the system, not just the size of the job. For shared memory systems running UNIX, a major rewrite of the operating system (in particular, the way locks are used to protect critical sections/data structures) was required.

(6) The compilers must support the paradigm being used, and this support needs to be efficient.

Traditionally, all of these areas have been a problem when it comes to shared memory programming. There were two types of shared memory systems—those with a small number of powerful, but expensive processors, and those with a moderate number of weak, but cheap processors. On systems with powerful processors, it was difficult to get permission to own multiple processors for the life of the run. On the other hand, for a long time, the systems with weak

processors were too weak to be of value for high performance computing (HPC). Therefore, until recently, there was really no appropriate platform for running an HPC job using the shared memory paradigm. Arguably, the SGI Challenge/Power Challenge product line was one of the first systems to support this paradigm in a meaningful way.

4. Hardware Limitations

While the previous section discussed some of the more obvious limitations in the design of the hardware, some of the subtler issues include the following:

(1) There needs to be sufficient memory bandwidth. Unfortunately, there is a lot of disagreement as to exactly what this means. However, the presence of a shared bus design (e.g., the design of the SGI Challenge/Power Challenge product line) can be a significant factor in determining a system's scalability.

(2) Similarly, large cache/TLB miss latencies can be hard to tolerate.

(3) For many designs, the effective cache miss latency is of particular interest. This is best expressed in terms of the number of peak floating point operations per cache miss. This value takes into account the following factors:

   (a) the minimum cost of a cache miss,

   (b) additional costs due to insufficient memory bandwidth and/or an insufficient number of memory banks resulting in contention,

   (c) additional costs associated with an off-node access in a NUMA design,

   (d) the cost of bottlenecks associated with going off node in any design with the concept of a node,

   (e) the cost of bottlenecks that can arise in designs with the concept of a node when there is contention for a single node's memory banks (e.g., all of the processors are accessing a single page of memory on a system that maps an entire page of memory to a single node's memory banks),

   (f) any costs associated with COMA-style **DRAM** caches, data replication, and similar attempts to avoid high latency operations, and

   (g) the tradeoffs that various microprocessor design teams have made in terms of clock speed vs. the number of floating point operations per cycle.

The author's experience on the KSR1 and the Convex Exemplar SPP-1600 clearly demonstrates the importance of the concept of the effective cache latency. On both of these systems, the actual cost of going off node was too large to be easily tolerated by a shared memory program. As a result, even though the SGI Challenge/Power Challenge had the obvious limitation of a shared memory bus, it could easily out perform these systems (Pressel 1999). In some cases, even well-designed systems, such as the SGI Origin 2000 and the SUN HPC 10000 have exhibited problems with contention.

These results come from systems that were designed to be used as ashared memory platform. When considering the obstacles to running software-distributed shared memory (with little or no hardware support), the effective cache miss latency will clearly fair even worse. This can significantly affect the tuning strategies needed for software-distributed shared memory environments, and may even effect the appropriateness of that type of platform for many algorithms (Jiang and Singh 1999). Two examples of this include Oberlin's speech that was mentioned in the introduction (Oberlin 1999) and the results listed in Table 2 and Figures 2-a and 2-b.

5. Parallelization Costs

There is concern about the ratio between the costs of computation and communication with message-passing codes. However, with shared memory applications, there is no explicit communication. Instead, the cost of getting in and out of a parallel section of code is of concern. Obviously, it is desirable to have as much work as possible per section of code, and this is generally achieved using the following techniques:

(1) parallelizing outer loops to the greatest extent possible,

(2) never parallelizing inner loops,

(3) combining loops under a common outer loop,

(4) leaving some loops unparallelized, and

(5) avoiding usage policies that will significantly increase the overhead associated with a parallel section.

The fourth technique may seem strange, but the following explanation might provide clarity.

On many shared memory systems, the cost of the overhead associated with parallelization is at least 2,000 cycles when using 10 or more processors. Under unfavorable circumstances, it can easily exceed 1-million cycles. To keep the cost of the overhead down to no more than 1% of the total **CPU** time, the parallel

Table 2. The performance of various versions of the F3D code when run on modern scalable systems (1-million grid point test case).[a]

System	Peak Processor Speed (MFLOPS)	No. of Processors Used	Version	Speed	
				(time steps/hr)	MFLOPS
SGI R10K O2K	390	8	Compiler Directives	793	1.04E3
SGI R12K O2K	600		**SHMEM**	382	4.99E2
SGI R10K O2K	390	32	Compiler Directives	2138	2.79E3
SGI R12K O2K	600		SHMEM	989	1.29E3
	600		Compiler Directives	2877	3.76E3
SGI R10K O2K	390	48	Compiler Directives	2725	3.56E3
SGI R12K O2K	600		SHMEM	1083	1.42E3
	600		Compiler Directives	3545	4.63E3
SGI R10K O2K	390	64	Compiler Directives	2601	3.40E3
SGI R12K O2K	600		SHMEM	1050	1.37E3
	600		Compiler Directives	3694	4.83E3
SGI R10K O2K	390	88	Compiler Directives	3619	4.73E3
SGI R12K O2K	600		SHMEM	1320	1.73E3
	600		Compiler Directives	5087	6.65E3
Cray T3E-1200	1200	8	SHMEM	349	4.56E2
		32		1062	1.39E3
		48		1431	1.87E3
		64		1705	2.23E3
		88		2443	3.19E3
		128		2948	3.85E3
IBM SP 160 (MHz)	640	8	MPI	199	2.60E2
		32		342	4.47E2
		48		420	5.49E2
		64		423	5.52E2
		88		396	5.18E2
Sun HPC 10000	800	8	Compiler Directives	999	1.31E3
		32		2619	3.64E3
		48		3093	4.04E3
		56		3391	4.43E3
		64		2819	3.68E3
HP V-Class	1760	8	Compiler Directives	1632	2.13E3
		14		2392	3.13E3

[a] For additional details, see Behr et al. (2000).

section must process between 2,000,000 to an excess of 10-billion cycles worth of work (in terms of serial time). If there are 1-million grid points, and this section of code is processing all of those points, then it is easy to satisfy the lower bound, although there will likely be trouble satisfying the upper bound. However, for boundary condition routines, the amount of work per section is likely to be two orders of magnitude less (in this case), making it difficult to satisfy even the lower bound. Therefore, it is frequently desirable to leave some of the boundary condition routines unparallelized.

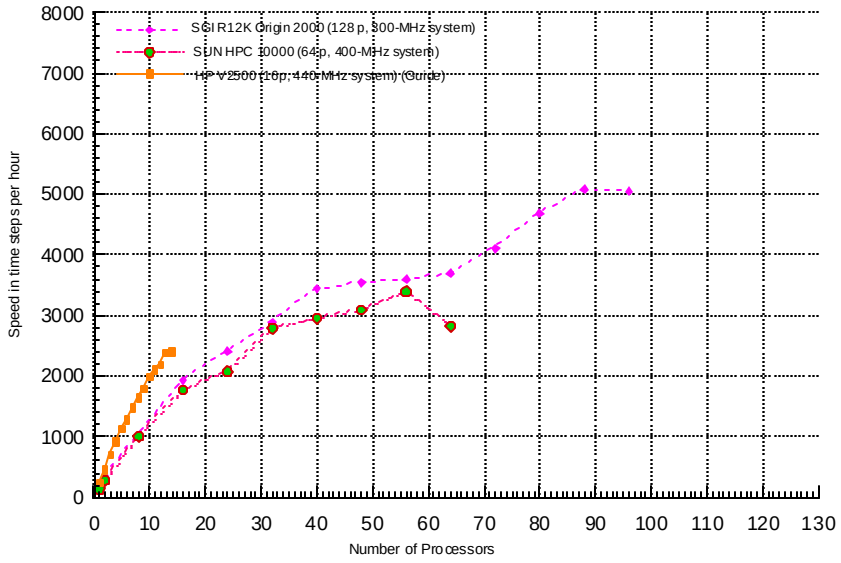

Figure 2-a. The performance of the shared memory version of the F3D code when run on a modern scalable SMP (1-million grid point test case).*

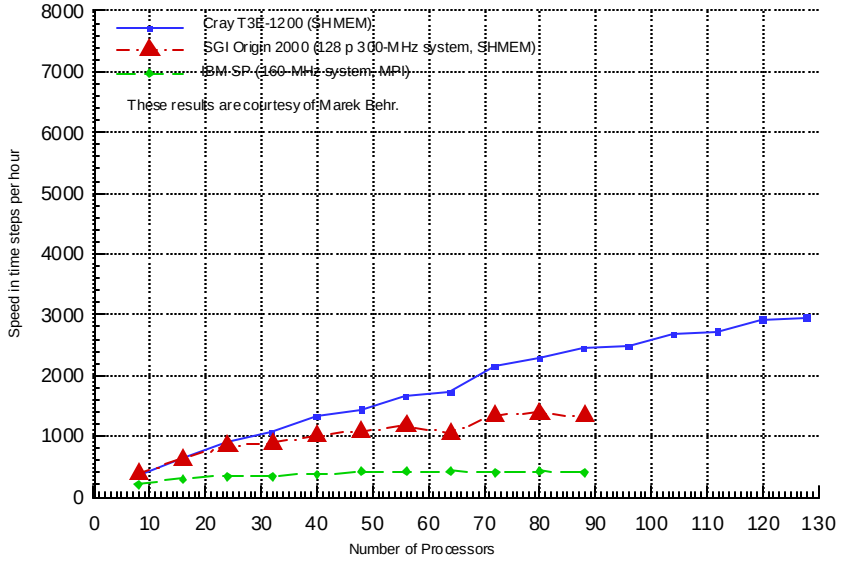

Figure 2-b. The performance of the distributed memory version of the F3D code when run on a modern scalable SMP / MPPs (1-million grid point test case). *

* The speeds have been adjusted to remove startup and termination costs.

Furthermore, as the processors get faster, the lower bound will almost certainly continue to get larger.

If the cost of getting into and out of a parallel section of code is not well constrained, then it is very difficult to show parallel speedup. As Tables 3 and 4 and Figures 3 and 4 show, this can be a serious problem when a system becomes overloaded. Some people have suggested that the solution to this is to have the operating system automatically decrease the number of threads (and by inference the number of processors) assigned to a job. However, the following are three objections to this approach:

Table 3. The effect on performance and the consumption of CPU time from running a parallel job on an overloaded HP V-Class.

No. of Processors Used	Wall Clock Time (s)	User CPU Time (s)	System CPU Time (s)
1	3524	3244	8
2	1698	3301	72
3	1203	3303	186
4	1974	3625	2302
5	1871	3630	2696
6	2554	3837	4955
7	3166	4051	7089
8	2915	3915	7223

Note: The job was run for 200 time steps.

(1) Even if it is assumed that the ideal speedup is linear speedup, the result can be an undesirable variability in the run time. It can be difficult for the user to identify the cause of this variability, resulting in the hardware and/or the paradigm getting a bad reputation.

(2) The proposed solution would take processors away from shared memory jobs, but not from message-passing jobs. Again, this would result in the paradigm getting a bad reputation, while inadvertently rewarding the owner of the message-passing job.

(3) For jobs where the ideal speedup is a stair step, reducing the number of processors by just one or two can result in a significant decrease in performance. Unfortunately, the decrease in performance is not the only issue. Since the run time has significantly increased with presumably only a modest decrease in the number of processors being used, the total

Table 4. The effect on performance and the consumption of CPU time from running a parallel job on an overloaded SGI Origin 2000.

No. of Processors Used	Wall Clock Time (s)	User CPU Time (s)	System CPU Time (s)
1	503	390	5
5	225	512	7
10	256	729	9
15	360	935	11
20	1322	2263	36
25	2119	3423	138
30	3691	4414	188

[a] The job was run for 40 time steps.

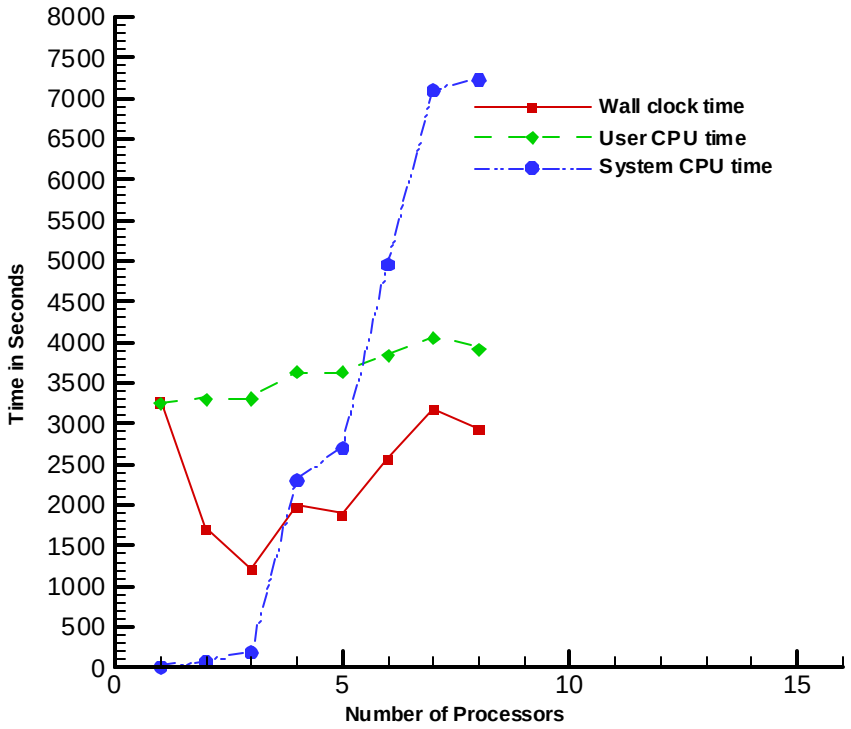

Figure 3. The effect on performance and the consumption of CPU time from the running of a parallel job on an overloaded HP V-Class.

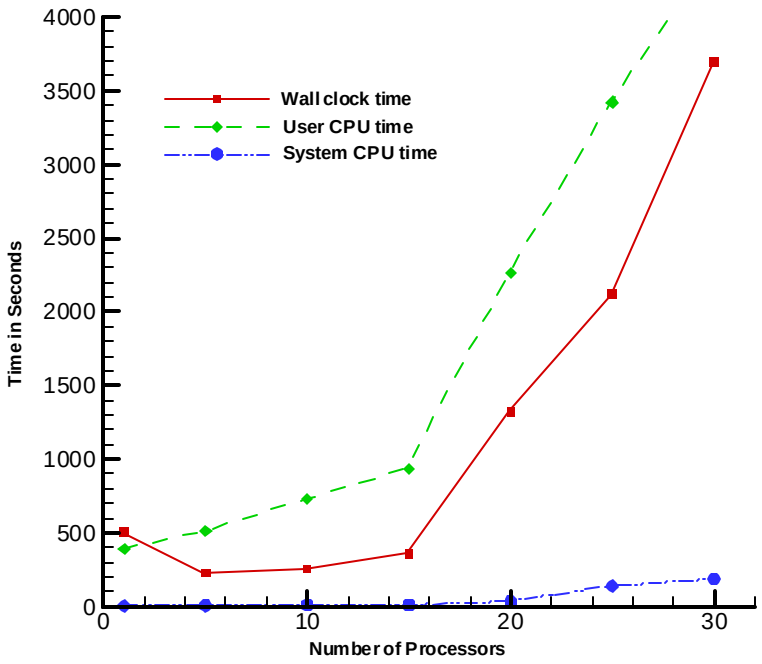

Figure 4. The effect on performance and the consumption of CPU time from the running of a parallel job on an overloaded Origin 2000.

amount of CPU time has increased. At many sites, the users (or their allocations) are charged on the basis of CPU time, and in some cases, on the basis of memory usage (**MB** hours).

The solution to this problem that was adopted at the U.S. Army Research Laboratory has been to implement a queuing system that actively manages the load factor. This is done by quiescing some of the jobs (primarily background jobs that are not charged to a user's allocation). This allows the use of background jobs to keep the system busy while waiting for a foreground job to be runable and/or to finish a nonparallelized section of code (e.g., input/output). Another consequence of the need to leave some portions of the code unparallelized is Amdahl's Law. Even if the cost of these sections is limited to 1% of the serial CPU time, the effect of Amdahl's Law will be noticeable when using more than about 30 processors. Furthermore, when using 100 or more processors, the effect is likely to be dominant. For a well-parallelized job using loop-level parallelism, there is likely to be a sweet spot between 32 and 64 processors. For most jobs, the incremental benefit of using additional processors may not justify the cost.

6. Conclusions

Shared memory programs can be successfully scaled past the commonly quoted limitation of 4–16 processors. However, there are a number of constraints that limit the ultimate scalability of these jobs. Of particular interest is the effective cost of a cache miss. This value is sufficiently important to generally preclude using software-distributed shared memory in production HPC jobs.

The effect of usage policies on performance was also discussed. Overloading a system can waste a significant amount of CPU time. Furthermore, at production sites, the preferred solution to this problem is not to reduce the number of processors assigned to a shared memory job. Instead, the preferred solution is to actively manage the load (including the quiescening of background jobs) in an attempt to avoid overloading the system.

7. References

Behr, M., D. M. Pressel, and W. B. Sturek, Sr. "Comments on CFD Code Performance on Scalable Architectures." *Computer Methods in Applied Mechanics and Engineering*, published by Elsevier Science Ltd., 2000.

Jiang D., and J. P. Singh. "Does Application Performance Scale on Modern Cache-Coherent Multiprocessors: A Case Study of a 128-processor SGI Origin 2000." The Proceedings of ISCA'99, The 26th International Symposium on Computer Architecture, published by IEEE Computer Society, Los Alamitos, CA, May 1999.

Oberlin, S. Keynote speech given at ISCA'99. The 26th International Symposium on Computer Architecture, May 1999.

Pressel, D. M. "Results From the Porting of the Computational Fluid Dynamics Code F3D to the Convex Emplar (SPP-1000 AND SPP-1600)." ARL-TR-1923, U.S. Army Research Laboratory, Aberdeen Proving Ground, MD, 1999.

INTENTIONALLY LEFT BLANK.

Glossary

CFD	Computational fluid dynamics
COMA	Cache only memory architecture
CPU	Central processing unit
DRAM	Dynamic random access memory
HPF	High performance Fortran
MB	Million bytes
NUMA	Nonuniform memory access
OVERLOADED	The load factor exceeds the number of processors in the system, resulting in the time sharing of processors.
TLB	Translation lookaside buffer

INTENTIONALLY LEFT BLANK.

1 PROGRAM DIRECTOR
C HENRY
1010 N GLEBE RD STE 510
ARLINGTON VA 22201

1 DPTY PROGRAM DIRECTOR
L DAVIS
1010 N. GLEBE RD STE 510
ARLINGTON VA 22201

1 DISTRIBUTED CENTERS
PROJECT OFFICER
V THOMAS
1010 N GLEBE RD STE 510
ARLINGTON VA 22201

1 HPC CTRS PROJECT MNGR
J BAIRD
1010 N GLEBE RD STE 510
ARLINGTON VA 22201

1 CHSSI PROJECT MNGR
L PERKINS
1010 N GLEBE RD STE 510
ARLINGTON VA 22201

1 RICE UNIVERSITY
MECHANICAL ENGRNG &
MATERIALS SCIENCE
M BEHR MS 321
6100 MAIN ST
HOUSTON TX 77005

1 J OSBURN CODE 5594
4555 OVERLOOK RD
BLDG A49 RM 15
WASHINGTON DC 20375-5340

1 NAVAL RSCH LAB
J BORIS CODE 6400
4555 OVERLOOK AVE SW
WASHINGTON DC 20375-5344

1 WL FIMC
B STRANG
BLDG 450
2645 FIFTH ST STE 7
WPAFB OH 45433-7913

1 NAVAL RSCH LAB
R RAMAMURTI CODE 6410
WASHINGTON DC 20375-5344

1 ARMY AEROFLIGHT
DYNAMICS DIRECTORATE
R MEAKIN M S 258 1
MOFFETT FIELD CA 94035-1000

1 NAVAL RSCH LAB
HEAD OCEAN DYNAMICS
& PREDICTION BRANCH
J W MCCAFFREY JR CODE 7320
STENNIS SPACE CENTER MS
39529

1 US AIR FORCE WRIGHT LAB
WL FIM
J J S SHANG
2645 FIFTH ST STE 6
WPAFB OH 45433-7912

1 US AIR FORCE PHILIPS LAB
OLAC PL RKFE
CAPT S G WIERSCHKE
10 E SATURN BLVD
EDWARDS AFB CA 93524-7680

1 NAVAL RSCH LAB
DR D
PAPACONSTANTOPOULOS
CODE 6390
WASHINGTON DC 20375-5000

1 AIR FORCE RSCH LAB DEHE
R PETERKIN
3550 ABERDEEN AVE SE
KIRTLAND AFB NM 87117-5776

1 NAVAL RSCH LAB
RSCH OCEANOGRAPHER
CNMOC
G HEBURN
BLDG 1020 RM 178
STENNIS SPACE CENTER MS
39529

1 AIR FORCE RSCH LAB
INFORMATION DIRECTORATE
R W LINDERMAN
26 ELECTRONIC PKWY
ROME NY 13441-4514

1 SPAWARSYSCEN D4402
R A WASILAUSKY
BLDG 33 RM 0071A

NO. OF COPIES	ORGANIZATION
	53560 HULL ST SAN DIEGO CA 92152-5001
1	USAE WATERWAYS EXPERIMENT STATION CEWES HV C J P HOLLAND 3909 HALLS FERRY RD VICKSBURG MS 39180-6199
1	US ARMY CECOM RSCH DEVELOPMENT & ENGRNG CTR AMSEL RD C2 B S PERLMAN FT MONMOUTH NJ 07703
1	SPACE & NAVAL WARFARE SYSTEMS CTR K BROMLEY CODE D7305 BLDG 606 RM 325 53140 SYSTEMS ST SAN DIEGO CA 92152-5001
1	DIRECTOR DEPARTMENT OF ASTRONOMY P WOODWARD 356 PHYSICS BLDG 116 CHURCH ST SE MINNEAPOLIS MN 55455
1	RICE UNIVERSITY MECHANICAL ENGRNG & MATERIALS SCIENCE T TEZDUYAR MS 321 6100 MAIN ST HOUSTON TX 77005
1	ARMY HIGH PERFORMANCE COMPUTING RSCH CTR B BRYAN 1200 WASHINGTON AVE S MINNEAPOLIS MN 55415
1	ARMY HIGH PERFORMANCE COMPUTING RSCH CTR G V CANDLER 1200 WASHINGTON AVE S MINNEAPOLIS MN 55415

NO. OF COPIES	ORGANIZATION
1	NAVAL CMD CNTRL & OCEAN SURVEILLANCE CTR L PARNELL NCCOSC RDTE DIV D3603 49590 LASSING RD SAN DIEGO CA 92152-6148
1	UNIVERSITY OF TENNESSEE COMPUTER SCIENCE DEPT S MOORE 1122 VOLUNTEER BLVD STE 203 KNOXVILLE TN 37996-3450
	ABERDEEN PROVING GROUND
31	DIR USARL AMSRL CI N RADHAKRISHNAN AMSRL CI H C NIETUBICZ AMSRL CI H W STUREK AMSRL CI HC D PRESSEL D HISLEY R NAMBURU R VALISETTY D SHIRES R MOHAN M HURLEY P CHUNG J CLARKE C ZOLTANI A MARK AMSRL CI HS D BROWN R PRABHAKARAN T PRESSLEY T KENDALL P MATTHEWS K SMITH AMSRL WM BF H EDGE AMSRL WT PB J SAHU K HEAVY P WEINACHT

INTENTIONALLY LEFT BLANK.

INTENTIONALLY LEFT BLANK.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	August 2001	Final, October 1999–June 2000

4. TITLE AND SUBTITLE
The Scalability of Loop-Level Parallelism

5. FUNDING NUMBERS
665803.731

6. AUTHOR(S)
Daniel M. Pressel

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
U.S. Army Research Laboratory
ATTN: AMSRL-HI-HC
Aberdeen Proving Ground, MD 21005-5067

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TR-2557

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 deals with the four main constraints on the scalability of programs parallelized using loop-level parallelism. They are as follows:

(1) The available parallelism in the algorithm.
(2) The availability and scalability of appropriate hardware (including the operating system and the compilers).
(3) Limitations in the design of the hardware.
(4) The cost of getting into and out of a parallel section of code.

This, in turn, will lead to two important discussions: (1) the theoretical limitations on the scalability of shared memory codes and (2) the role that the choice of hardware and usage policies play in determining the performance of a shared memory code.

These discussions will include examples from the author's own work in porting the implicit computational fluid dynamics code F3D from the Cray C90 to a variety of shared memory platforms.

14. SUBJECT TERMS	15. NUMBER OF PAGES
OpenMP, loop-level parallelism, supercomputer, high performance computing, parallel programming	24
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	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-2557 (Pressel) Date of Report August 2001

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)