LEGIBILITY NOTICE

A major purpose of the Technical Information Center is to provide the broadest dissemination possible of information contained in DOE's Research and Development Reports to business, industry, the academic community, and federal, state and local governments.

Although a small portion of this report is not reproducible, it is being made available to expedite the availability of information on the research discussed herein.

LA-UR88-1836

CONF-8805181--2

Received by OSTI

AUG 0 4 1988

Los Alamos National Laboratory is operated by the University of California for the United States Department of Energy under contract W-7405-ENG-36

LA-UR--88-1836

DE88 014456

TITLE: PARALLEL PROCESSING A REAL CODE-A CASE HISTORY

AUTHOR(S): David A. Mandell
Harold E. Trease

SUBMITTED TO: Los Alamos National Laboratory
Workshop on Instrumentation for
Future Parallel Systems

DISCLAIMER

MASTER

Los Alamos National Laboratory
Los Alamos, New Mexico 87545

FORM NO 836 R4
ST NO 2629 5/81

PARALLEL PROCESSING A REAL CODE - A CASE HISTORY

David Mandell and Harold Trease
Computational Physics Group
Los Alamos National Laboratory

ABSTRACT

A three-dimensional, time-dependent Free-Lagrange hydrodynamics code has been multitasked and autotasked on a Cray X-MP/416. The multitasking was done by using the Los Alamos Multitasking Control Library, which is a superset of the Cray multitasking library. Autotasking is done by using constructs which are only comment cards if the source code is not run through a preprocessor. The 3-D algorithm has presented a number of problems that simpler algorithms, such as 1-D hydrodynamics, did not exhibit. Problems in converting the serial code, originally written for a Cray 1, to a multitasking code are discussed. Autotasking of a rewritten version of the code is discussed. Timing results for subroutines and hot spots in the serial code are presented and suggestions for additional tools and debugging aids are given. Theoretical speedup results obtained from Amdahl's law and actual speedup results obtained on a dedicated machine are presented. Suggestions for designing large parallel codes are given.

1. INTRODUCTION

Large three-dimensional, time-dependent hydrodynamics codes take an excessive amount of time to execute even on the largest supercomputers currently available, such as a Cray X-MP/416. In order to reduce the execution time to practical values, it is desirable to multitask codes so that all of the processors can be used. A desirable goal is to provide users with overnight turn-around.

The POLLY code was originally written for a Cray 1[1] and in multitasking it for a Cray X-MP numerous problems were encountered that did not occur when smaller codes were multitasked. These problems are discussed

below. The code was multitasked using the Los Alamos Multitasking Control Library[2]. In order to speedup the serial POLLY code it was decided to rewrite this code and the new code, called X3D, is a substantially faster serial code. X3D was autotasked using the Los Alamos Autotasking method developed by Bobrowicz[3].

By examining Amdahl's Law[4,5],

$$S = 1 / (1 - P + P / NP),$$

where S is the speedup (best serial wall clock time / parallel wall clock time), P is the fraction of serial code that is parallel, and NP is the number of physical processors.

It becomes clear that in order to achieve high efficiencies, the serial portion of a code must be reduced to an extremely small fraction. In a real code this presents a significant challenge. The first step in doing this is to examine the serial code in order to determine where the time is being spent. The Los Alamos utility TALLY[6] was used to do this and the timing results are presented.

In Section 2 the multitasking done for the POLLY code is described in the hope that the problems encountered will enable other code developers to avoid similar problems. Section 3 presents the autotasking method used and describes the resulting autotasked version of X3D. Four processor TALLY results, showing where processors were waiting are presented. Results for the multitasked POLLY code and the autotasked X3D code are presented in Section 4. The work on these two codes has been an interesting learning experience and the lessons learned, as well as code design recommendations, are given in Section 5.

2. MULTITASKING POLLY

POLLY was a large 3-D, time-dependent Free-Lagrange hydrodynamics code that had a number of features that that were not present in earlier codes that we had multitasked. The physics portion of the code, excluding the input generation, graphics, and control sections was about 40,000 unique lines of code. In addition the code made extensive use of the solid state disk

(SSD). The number of words written to the SSD was not known a priori due to the time-dependent nature of the mesh. Dynamic memory management[7] was used in all levels of the code. In smaller codes multitasking can frequently be done by using temporary arrays, but POLLY was memory bound as well as taking a very large amount of time to execute, and thus additional memory could not be used in the multitasking. The multitasking was done by dividing the mesh into NP sections, where NP is the number of available processors. Each task worked on the identical coding, but on different data.

One of the most serious problems encountered in multitasking POLLY involved the codes use of pointers. Pointers were used in low level subroutines in a dynamic manner; that is arrays were allocated and deallocated during each time step. In the original serial code, one large array was used in some cases and pointers into different parts of the array were used. In the multitasked version, this created chaos. Separate pointers were needed for each task so that arrays do not write over each other. This can be easily done with the Los Alamos Memory Management System(MMS)[7] since this system allows for both an array name and a partition name. The task name was used as the partition name.

3. AUTOTASKING X3D

Due to the extreme time that it took for POLLY to execute the code was rewritten and renamed X3D. In X3D an effort was made to keep serial sections out of the core of the code. In addition the current version of X3D does not use the SSD. X3D has 100,000 unique lines of code and 561 subroutine currently. A number of real code features exist that still limit the fraction of the code that can be run in parallel. These include the need to periodically check for user interaction from the terminal, periodic graphics and restart dumps and normal output of the results.

In order to determine where the execution time is being spent in the serial code, and thus which routines should be autotasked first, timing studies were done using the TALLY utility[6]. Figure 1 shows a partial list of the TALLY results. This figure shows that the main hydrodynamics routine, HYDROM, uses 62.5 percent of the execution time and the subroutine that determines the time step for the next cycle, TIMSTP, uses 32.38 percent. Therefore about 95 percent of the time is spent in these two routines and if they were

made entirely parallel, Amdahl's Law, Figure 2, shows that the speedup would be a maximum of 3.48 on 4 processors. It is significant that the next most heavily used routine is the vector square root. Thus it is not practical to increase the fraction parallel much above 95 percent since, after HYDROM and TIMSTP, none of the X3D subroutines use enough time to make it practical to autotask them.

Autotasking is done by using a number of constructs in front of Fortran statements in the code. One advantage of autotasking is that these constructs are comment cards if the code is not run through a preprocessor. This is a great advantage in a code such as X3D which is under continuous development. In the multitasking done on the POLLY code it was necessary to restructure large parts of the code and therefore it was not possible to keep up-to-date with the latest code versions. Figures 3-6 show the most important autotasking constructs used in X3D.

4. RESULTS

The POLLY results for four processors during dedicated system time (DST) are shown in Figure 7 for the Noh test problem[8]. The mesh consisted of 48,400 mass points. Only 21 time-step cycles were run in DST due to the large execution time required for POLLY. Amdahl's Law is also shown so that progress can be judged as greater portions of the code were multitasked. The speedups are for the physics portion of the code. Initialization and final dumping and cleanup are excluded since this code took so long to run that initialization and final cleanup overwhelm the results if they are included.

During the initial multitasking work at Los Alamos, problems occurred in virtually every aspect of the work. This included problems in the operating system, libraries, compiler, debugger; and, of course, in the code itself. The change in the speedup, for a fixed fraction of parallel code, is shown for two different versions of the Cray Time Sharing System (CTSS) and the same version of POLLY. This CTSS bug fix resulted in a significant increase in the speedup.

The first step in multitasking or autotasking is to go from a static code compilation to a stack (reentrant) based code. This caused a number of problems since, in a given subroutine, local variables no longer exist after exiting the subroutine, in stack based code. This is not the case in static

code. These local variables had to be found and changed to global variables in a common statement in those cases where it was expected that they would be needed in succeeding time steps. In one case a buffer was defined in a subroutine by using a dimension statement and then used again in a later part of the code. The array no longer existed in the stack code.

Other problems occurred because only one channel exists in CTSS to the solid state disk. The SSD was used in such a manner in the serial code that each processor had to use it in the multitasked code. Thus a queue, at a low level in the code, existed and processors had to wait. Significant problems existed because of the use of dynamic memory management. It was necessary to carefully sort out the pointers that needed to be local to each task and those that needed to be global to all tasks.

Because of the above problems, numerous critical and sequential sections were necessary in the core of the physics part of the code and the best speedup achieved was 2.82 on 4 processors. This is not an acceptable use of the resources on a X-MP/416. Because of the overall speed of POLLY, it was decided to rewrite the code.

The hydrodynamics core of X3D was designed with consideration given to autotasking the code and thus it was very much easier to autotask it than it had been to multitask POLLY. This was partially due to the fact that autotasking is inherently easier to implement than multitasking. In this case since the SSD is not being used only 6643 mass points were used. X3D runs fast enough that the entire Noh problem of 760 time cycles can be run during DST. The Noh problem consists of a sphere of ideal gas with an initial velocity of one inward. At time 0.6, Noh obtained an analytical solution. The density as a function of the radial position has a value of 64 and than tapers off at large radius. The initial and final meshes are shown in Figure 8 and the density as a function of radius is shown in Figure 9.

TALLY results for four processors are shown in Figures 10-12. These results were obtained during DST since the multitasking version of TALLY is not valid for non-DST runs. These results show that a significant portion of the run time was spent with three processors waiting (10.41 percent in the MTHELP autotasking subroutine, Figure 10 and the waiting percentages in Figure 11). A more detailed timing analysis showed that most of the waiting time was in the time step subroutine (Figure 12 , W at statement 500B, which is the end of the do 500 loop). Work was done to reduce the waiting time and the results are shown in Figure 13. The serial time, the autotasking time

during DST for 4 processors and the speedup are shown for three versions of X3D over a two week period. The speedup remained almost constant; at about 3, but significantly the serial code was speeded-up 27 percent in the two week period. A speedup of three was less than we were hoping for, but in terms of the total code execution time, the project has been very successful.

5. LESSONS LEARNED AND CODE DESIGN RECOMMENDATIONS

In this section we will discuss the lessons we have learned in converting a code written for a serial machine, namely a Cray 1, to a multitasked code for a Cray X-MP/416 in the hopes that other computational physicists can avoid some of the problems we encountered. Code design recommendations are also given.

As discussed above the first step is to produce a correct serial, stack based code. Variables that must be local have to be carefully distinguished from global variables. Dynamic memory management should be kept at the highest levels possible since memory adjustments are a serial process. If memory management must be done in multitasked parts of the code, pointers need to be defined for each task. In the Los Alamos Memory Management System, arrays can be separated into partitions so that the same array can be defined for each task and each task has its own copy.

A desirable feature in future machines would be a separate channel to the SSD from each processor so that SSD requests would not have to enter a queue.

A problem that existed with the X3D code is that the code is under continuous development by a physics team, most of whom have no multitasking or autotasking experience. This results in new code packages being added that are not optimum for subsequent autotasking. If autotasking is going to be done on a code then the entire code team needs to be throughly trained in the philosophy of parallel coding.

Better software tools are needed for parallel code development. Large code development requires a good dynamic debugger in which break points can be set in all tasks, not just the root task. This can be done in the Los Alamos Dynamic Debugging Tool (DDT). Better tools are needed to determine code hot spot areas for both the serial and the parallel code. Determining load balancing problems is essential. One of the hardest parts

of converting serial codes to efficient parallel codes is differentiating between local and global variables. Efficient tools to do this are needed. For our codes these tools must be able to handle pointers.

If resources are available, redesigning and rewriting the entire code is the best course of action in order to obtain an efficient parallel code. From Amdahl's Law it is clear that getting high efficiencies is extremely difficult with older codes. The serial sections must be reduced to an extremely small percentage, which is best done by redesigning the code.

APPENDIX - REFERENCES

1. H. E. Trease, "Three-Dimensional Free Lagrangian Hydrodynamics", in Lecture Notes in Physics, eds. M. J. Fritts, W. P. Crowley and H. Trease, Springer-Verlag, New York (1985).

2. E. Williams and F. Bobrowicz, "Speedup Predictions for Large Scientific Parallel Programs on Cray X-MP-Like Architectures", Proceedings of the 1985 International Conference on Parallel Processing, St. Charles, Ill., (August 20-23, 1985).

3. F. Bobrowicz, "Autotasking on Cray X-MP Supercomputers", in preparation.

4. G. M. Amdahl, "Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities", AFIPS Conference Proceedings, Volume 30 (1967).

5. W. H. Ware, "The Ultimate Computer", IEEE Spectrum, pp. 84-91, Volume 9, Number 3 (1972).

6. TALLY, Computer Documentation Group, Los Alamos National Laboratory, Report CTSS-131 (1981).

7. W. H. Spangenberg and F. W. Bobrowicz, The Memory Management

System for the Los Alamos Cray X-MP Environment, Los Alamos National Laboratory Report LA-UR-86-1254 (1986).

8. W. F. Noh, Artificial Viscosity(Q) and Artificial Heat Flux(H) Errors for Spherically Divergent Shocks, Lawrence Livermore National Laboratory Report UCRL-89623 (1983).

Figure 1: Partial List of TALLY Results For The Serial Version of The X3D Code

PARTIAL TALLY RESULTS

ROUTINE	HITS	PERCENT	
UNPACKTP	535	0.31	*
HYDROM	107513	62.50	**********
ICONV	176	0.10	*
TIMSTP	55698	32.38	**********
ISAMAX	503	0.29	*
%SQRT	2539	1.48	*

Figure 2: Amdahl's Law

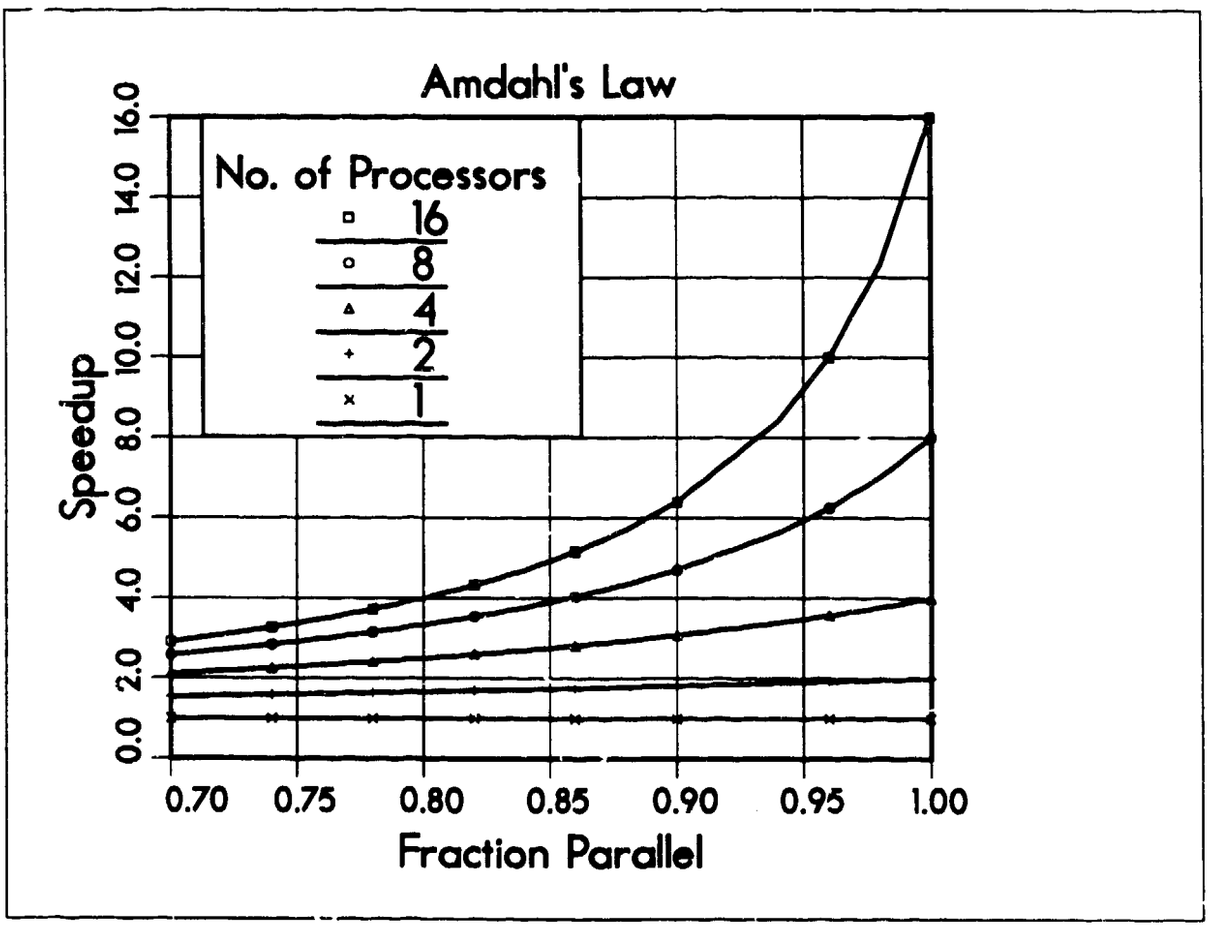

Figure 3: Subroutine Construct For Autotasking

AUTOTASKING

CMLT$ MULTI

subroutine hydrom

*

*

*

Figure 4: Non-Vector Do Loop Construct For Autotasking

AUTOTASKING

```
CMLT$ DO MULTI
      do 1000 n=1,nblocks
         *
         *
         *
```

Figure 5: Vector Do Loop Construct For Autotasking

AUTOTASKING

```
CMLT$ DO MULTI 64 2001
*dir$ shortloop
      do 2000 n=1,np(5)
         *
         *
         *
```

Figure 6: Sequential and Critical Section Constructs For Autotasking

AUTOTASKING

```
CMLT$ SEQ
          if (... ) ...
               *
               *
               *
CMLT$ END SEQ
               *
               *
               *
CMLT$ CRITICAL
          sum = sum + ...
               *
               *
               *
CMLT$ END CRITICAL
```

Figure 7: POLLY Multitasking Results

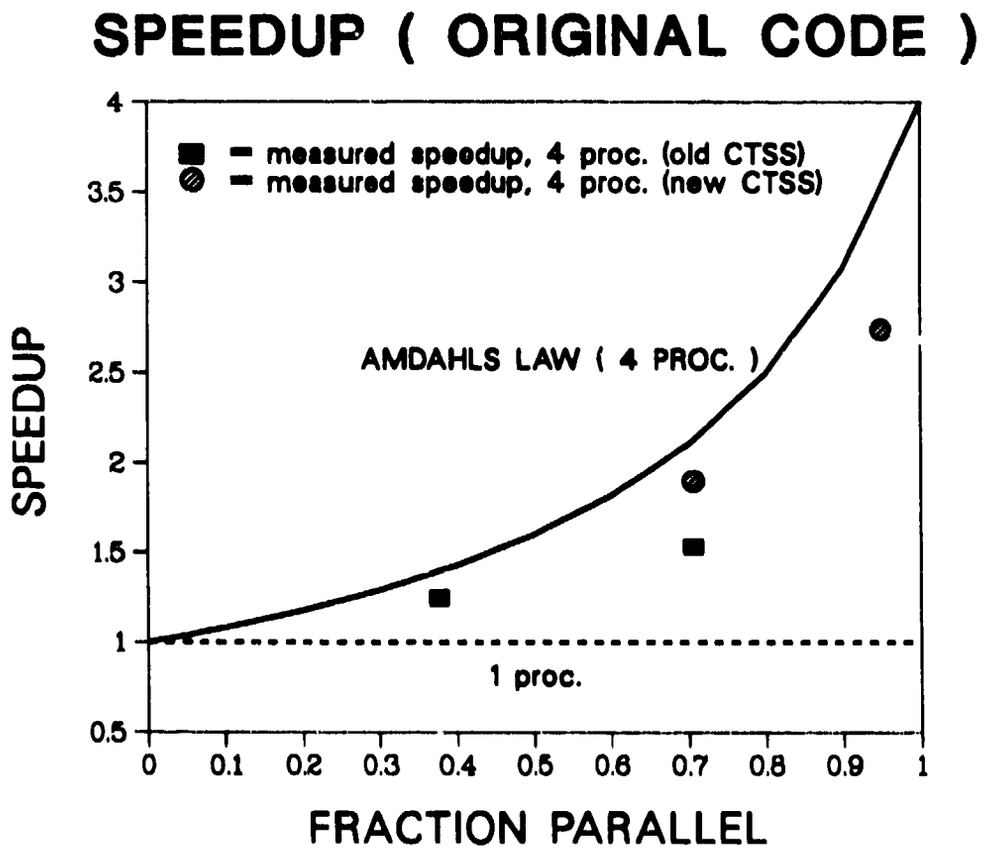

Figure 8: 3-D Spherical Noh Problem - Initial & Final Meshes

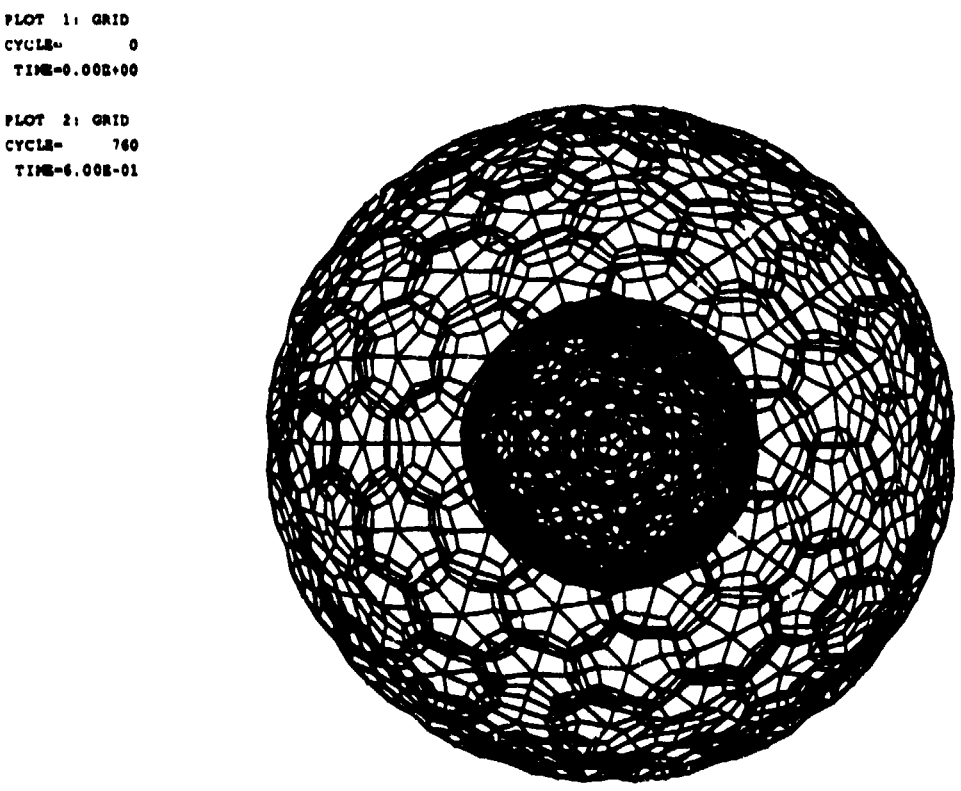

Figure 9: X3D Density VS. Position Results For The 3-D Noh Problem

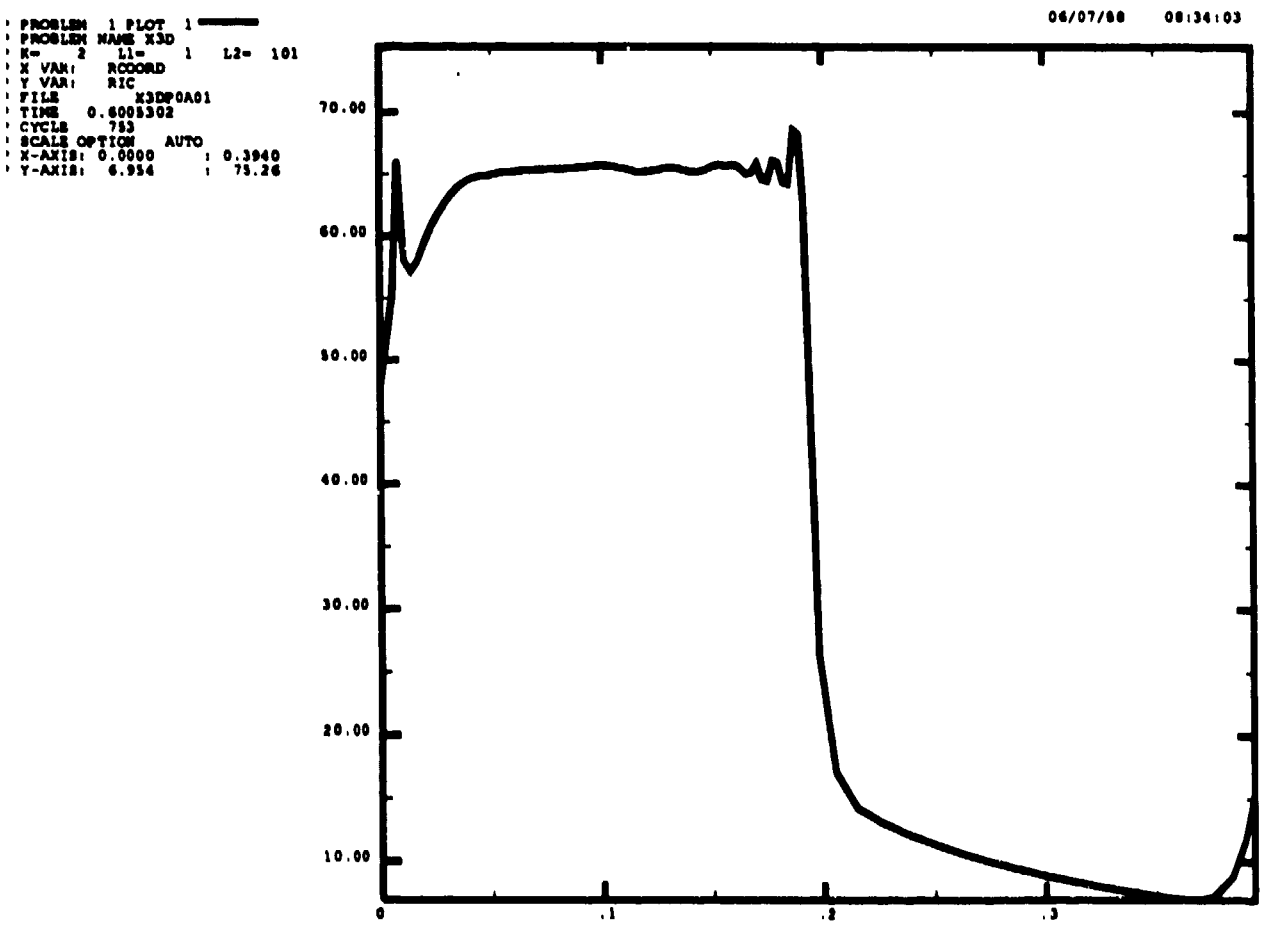

Figure 10: Partial Autotasking TALLY Results

PARTIAL TALLY RESULTS - 4 PROC.

	% FOR EACH PROC.					
ROUTINE	0	1	2	3	TOT	
MTHELP	0.00	3.46	3.48	3.47	10.41	**********
UNPACKTP	0.16	0.04	0.03	0.00	0.23	*
HYDROM	12.35	12.71	12.67	12.70	50.43	**********
AUTOTS%	0.03	0.03	0.04	0.02	0.11	*
AUTOFN%	0.00	0.01	0.00	0.00	0.02	*
AUTOLN%	0.00	0.00	0.01	0.01	0.02	*
TIMSTP	8.23	8.29	8.27	8.28	33.07	**********
ISAMAX	0.03	0.03	0.03	0.02	0.11	*
HELOCVAL	0.07	0.00	0.00	0.00	0.08	*
%SQRT	0.23	0.18	0.20	0.22	0.83	*

Figure 11: Percent of Time That Each Logicial Processors Was Waiting

PARTIAL TALLY RESULTS
WAITS PER PROCESSOR

LOGICAL PROC. WAITS

CONNECTOR 0: 694 WAITS (8.55%)
CONNECTOR 1: 2073 WAITS (25.54%)
CONNECTOR 2: 1973 WAITS (24.30%)
CONNECTOR 3: 2036 WAITS (25.08%)

Figure 12: X3D Autotasking TALLY Results For Subroutine TIMSTP

PARTIAL TALLY RESULTS - 4 PROC.
SUBROUTINE TIMSTP

STMT NO.	ADDRS1	ADDRS2	0	1	2	3	TOT (ABS%)
	00304345	00304354	0.07	0.07	0.06	0.04	0.23 ***
	00304355	00304364	0.04	0.03	0.05	0.02	0.14 *
500B W	00304365	00304374	0.65	0.83	0.71	0.80	3.00 **********
	00304375	00304404	0.00	0.00	0.00	0.00	0.01 *
00019	00304405	00304414	0.00	0.01	0.00	0.00	0.01 *
	00304415	00304424	0.00	0.00	0.00	0.00	0.00 *
600B	00304425	00304434	0.00	0.00	0.00	0.00	0.00 *

Figure 13: Speedup Results For Three Versions of X3D