LA-UR -92-1453

LA-UR--92-1453

DE92 013239

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

TITLE: **EVALUATION OF A CRAY PERFORMANCE TOOL USING A LARGE HYDRODYNAMICS CODE**

AUTHOR(S): Kenneth M. Lord and Margaret L. Simmons

SUBMITTED TO: Sixth International Conference on Modelling Techniques and Tools for Performance Evaluation
September 16-18, 1992
Edinburgh

DISCLAIMER

Los Alamos National Laboratory
Los Alamos, New Mexico 87545

EVALUATION OF A CRAY PERFORMANCE TOOL USING A LARGE HYDRODYNAMICS CODE

Kenneth M. Lord
Cray Research, Inc.
Eagan, MN 55121

and

Margaret L. Simmons
Los Alamos National Laboratory
Los Alamos, NM 87545

1. Introduction

Early efforts to decompose programs for parallel machines were very difficult and not always successful [1,2]. There were many possible paths that could be followed to partition a scientific code for parallelization. For example, early researchers [3,4] in compiler methods of parallelization proposed a clustering scheme that attempted to translate a Fortran program into its most basic form—a directed graph of dependences where nodes represent elementary operations and edges show the flow of data. One would then find clusters of nodes that could be computed independently, coalescing the clusters and increasing granularity until some desired degree of parallelization was reached. This approach was difficult to automate effectively, and so was never very useful.

Another method that seemed more accessible was to partition a code by hand based on some high-level knowledge of the application. This approach, based on a proposed "top-down" methodology [5], required the use of some sort of dependency analysis tools for success on any large, realistic scientific code. At that time, nearly five years ago, there were only a few such tools, mostly in the research stage [6,7]. These tools were cumbersome and difficult to use, partly because their conservative approach required them to present as dependences anything that was in any way questionable, creating more information than one could understand or use. It was clear that for truly successful partitioning of codes for parallel processing, tools not only had to accomplish this analysis automatically, but had to present the results of the analysis in a graphical, understandable format. This problem still continues. Although there are many more tools available today than five years ago, many still suffer from the problems mentioned above. Researchers and tool-builders still debate what to give the user and how to present the information [8].

This paper will discuss one of these automatic tools that has been developed recently by Cray Research, Inc. for use on its parallel supercomputers. The tool is called ATEXPERT; when used in conjunction with the Cray Fortran compiling system, CF77, it produces a parallelized version of a code based on loop-level parallelism, plus information to enable the programmer to optimize the parallelized code and improve performance. The information obtained through the use of the tool is presented in an easy-to-read graphical format, making the digestion of such a large quantity of data relatively easy and thus, improving programmer productivity.

In this paper we address the issues that we found when we took a large Los Alamos hydrodynamics code, PUEBLO, that was highly vectorizable, but not parallelized, and using ATEXPERT proceeded to parallelize it. We show that through the advice of ATEXPERT, bottlenecks in the code can be found, leading to improved performance. We also show the dependence of performance on problem size, and finally, we contrast the speedup predicted by ATEXPERT with that measured on a dedicated eight-processor Y-MP.

2. Overview of PUEBLO

The PUEBLO code is used to numerically model point explosions in space. The code uses a three-dimensional, time-explicit Lagrangian finite-difference numerical technique in which all hydrodynamic variables including velocities, are cell-centered. This technique is based on a form of the Godunov method, which uses a first-order Riemann solver. It also uses a Gamma-law Equation-of-State. The hydrodynamics cycle is split into a Lagrangian

phase and a rezone-advection phase in which conserved quantities are transferred from the Lagrangian mesh to an arbitrarily specified mesh.

The problem that we analyzed used two different mesh sizes: these were 32×32×32 and 64×64×64. In the code the three dimensions of the mesh are merged into a single one-dimensional data structure, so that the primary loop lengths are on the order of the cube of one dimension of the mesh. The problem that was run on the smaller mesh size involved both the Lagrangian phase and the rezone-advection phase. The problem that was run on the larger mesh involved only the Lagrangian phase.

3. Overview of CRI Tools

3.1. SCOUNT

SCOUNT is a benchmarking utility that counts the number of times each statement in a Fortran program is executed. SCOUNT produces a source listing with an execution tally next to each line of code. We used SCOUNT to ensure that during the initial phases of optimization, the concentration of effort was on those loops that our problem actually executed.

3.2. PROF and PROFVIEW

Through a method of timing by address range, the PROF utility indicates how much time is spent in various segments of code within routines. At regular intervals, the operating system records the address of the instruction being executed. Addresses are grouped in "bins" or "buckets," whose size is selectable; these bins can be associated with labels internal to a program.

The PROFVIEW utility generates reports in various formats from the raw data generated by PROF. Since the UNICOS 6.0 operating system release, PROFVIEW has provided an X-Window interface.

3.3. ATEXPERT

ATEXPERT is a tool developed by Cray Research, Inc. for accurately measuring and displaying information on the Autotasking performance of a job that is run on an arbitrarily loaded system. It predicts speedups that would result during dedicated execution from data collected while running a code on a nondedicated system. It provides a wealth of information on the code under consideration enabling the programmer to find those spots in the code that may be contributing to performance bottlenecks. ATEXPERT provides an X-Window interface as well as an interactive and batch ASCII interface.

ATEXPERT is actually more than a single command; it is composed of three phases:

(1) an instrumentation phase,

(2) a data-gathering phase, and

(3) an analysis phase.

During the instrumentation phase, the FMP preprocessor (from the CF77 compiling system) adds additional timing code to the regions of the code determined to be parallelizable, that is then compiled into a user's program. Figure 1 shows schematically how this is done. During the data-gathering phase, the program is executed and raw timing information is gathered. In addition, the instrumentation also records the number of unitasked scalar iterations for each loop, the number of concurrent iterations for each parallel loop, plus other relevant information associated with each loop. When the program terminates, this information is written to a file. In the analysis phase, this file is read by ATEXPERT which then displays through its X-Windows graphics interface program the Autotasking performance data thus collected. An ASCII display format is also available.

Timings

bp -	*time required to begin a parallel region*
bcs -	*time to begin a control structure*
top of loop -	*time necessary to get to the top of a control structure*
bot of loop -	*time necessary to get to the bottom of a control structure*
ls -	*time required to do loop synchronization*
il -	*time to get the next control structure started; interloop time*
ep -	*time required to end a parallel region*

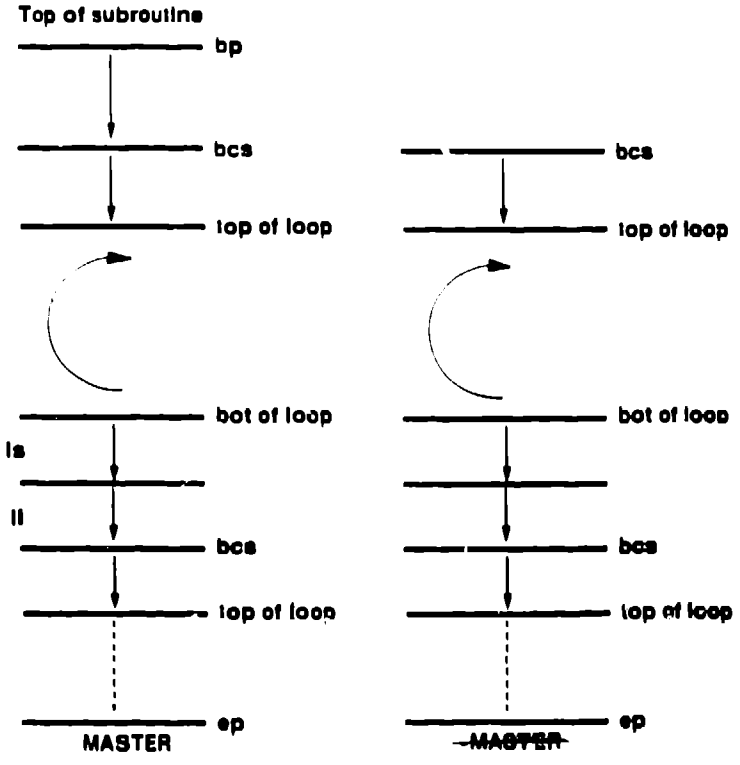

Figure 1. Schematic of timing-call insertion by ATEXPERT.

ATEXPERT decomposes the execution time of a program into parallel region time and preceding serial time (time spent outside of parallel regions). Parallel region timing is obtained for unitasked execution of each parallel region as well as multitasked execution of the region. Multitasked execution is further decomposed into measurement of distribution of work among processors and measurement of overhead costs for parallel execution.

Overhead, in this case, is defined [7] as the difference in speedup between that predicted by Amdahl's Law and that measured or projected from an actual Autotasking run. Amdahl's Law [8], as quantified by the Ware model [9], and extended to multiprocessing is given by

$$S(P) = [(1 - f(p)) + f(p)/P]^{-1}$$

where

$S(P)$ = maximum expected speedup from multitasking,

P = number of processors available for multitasking,

$f(p)$ = fraction of program that can execute in parallel, and

$1 - f(p)$ = fraction of program that is serial ($=f(s)$).

ATEXPERT obtains the fraction that can execute in parallel ($f(p)$) from exploited parallelism rather than from existing parallelism. This is an important difference because it is a measure of the detected parallelism rather than of potential parallelism. ATEXPERT cannot currently detect potential parallelism, due in part to CF77's inability to carry out interprocedural analysis. A user can change this fraction from potential to detected parallelism by use of inserted directives. Overhead thus calculated (or projected) is further decomposed into various contributing factors associated with the Cray Autotasking System, such as Begin Parallel overhead, Slave Arrival overhead, Convoy Time, and others. It is interesting to note that this idea of overhead as the difference in predicted versus measured speedup was first proposed as a extension to the Ware model by Buzbee [10] in 1984.

4. Results

The goal of this project was to take the serial, but highly vectorizable program, PUEBLO and using various CRI tools to parallelize the code, obtaining the best possible speedup through the use of information provided by the tools. A constraint was not to change the algorithm and to allow only the minimum changes in the code necessary for successful execution. We will first detail the results from the 32×32×32 mesh problem and then give results from the larger mesh.

After some initial information-gathering runs using SCOUNT that provided information about which loops were actually being executed by our problem, we began our analysis with the use of PROF and ATEXPERT. The initial PROF runs showed that a routine named ISMIN took 11.7% of the runtime. When we allowed the compiler to replace ISMIN with a more efficient version from the Cray scientific library, SCILIB, the time spent in ISMIN dropped to an insignificant amount and the total execution time improved by 10%. This improved version was run through the CF77 compiling system to enable automatic parallelization, called Autotasking. A Profile from this step shows in Figure 2 that the subroutines RIEMAN and ADVECT are the most heavily used routines. This shows us where we need to look first to improve parallelism. In this code, RIEMAN dominates in both serial and parallel mode; it is clearly an important subroutine in the code.

Next we ran the code using ATEXPERT. The results of this effort are shown in Figure 3. Notice in the plot on the left side of the figure that the predicted speedup is only 2.5 out of a possible 4.8, (assuming 8 processors) using the Amdahl's Law calculation described above. Since a highly vectorizable code implies many loops that should be amenable to parallelization, this result is both puzzling and disappointing. Further investigation of the plot in Figure 3 shows us that the problem begins when more than three processors are used. By inspecting additional information provided by ATEXPERT, we find that loop 20 in the most heavily used subroutine, RIEMAN, is performing poorly. Clicking on "Source Files" in the command menu allows us to bring up a window containing source code for RIEMAN, zeroing in on loop 20. Figure 4 shows the fragment of code representing this loop. This fragment makes the problem obvious. The Autotasking system, by default, tries to vectorize inner loops and multitask outer loops. The outer loop in this piece of code has an upper limit of 3. The inner loop, however, has an upper limit of 32768 (for the small problem)! If we could run the inner loop as concurrent vector, that is, sending "chunks" of the inner-loop vector to each of the processors, the performance would improve. Checking our Autotasking manual, we see that there is a Cray microtasking directive that lets us do just that. By inserting a directive of the form *cmic$ do parallel vector* we get the long inner loop partitioned across the eight processors, thus allowing both vectorization and full parallelization, improving our granularity and giving a better speedup.

Since PUEBLO is a three-dimensional code, the upper limit of three on outer do-loops should be quite common, and a check of other subroutines that contribute heavily to the runtime is probably a good idea. Doing so shows several more instances of the same problem. Adding directives to these subroutines gives us the results seen in Figure 5.

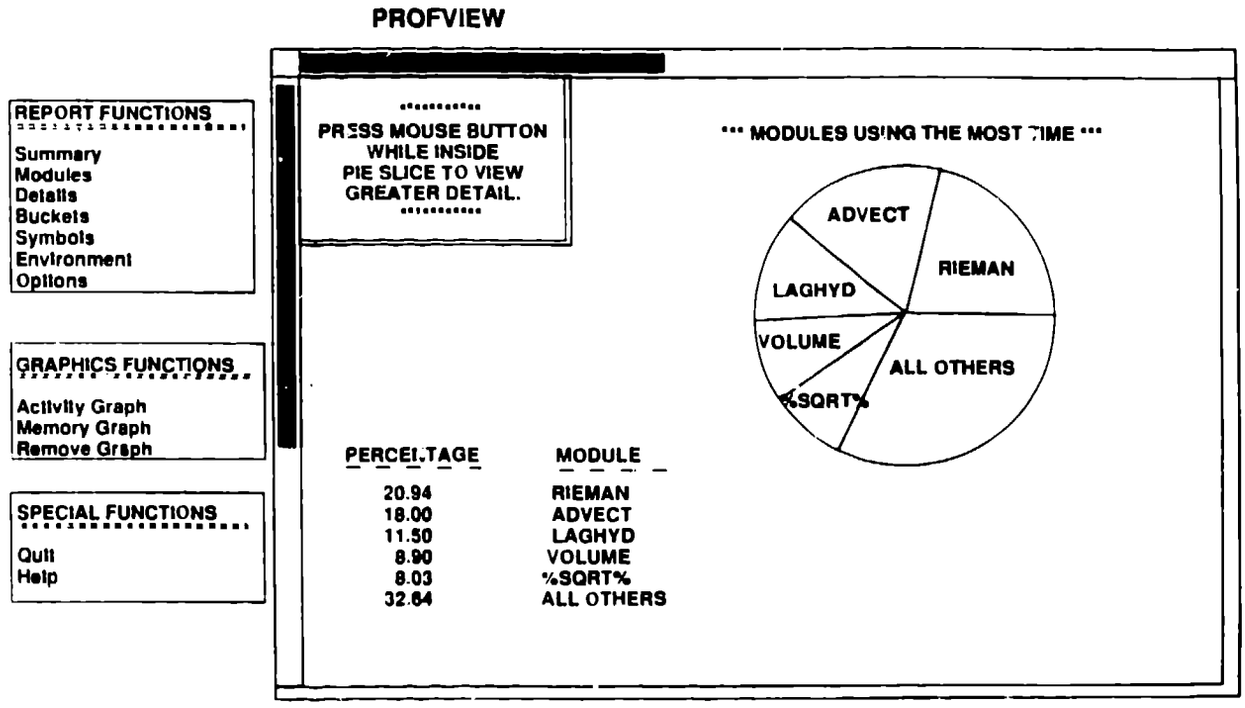

Figure 2. Graphical output from *prof/profview* showing where the percentage of execution time is spent in the intial run of PUEBLO.

Note that the speedup is now 6.4 out of a possible 7.2, or 90% of the Amdahl's Law prediction. By the judicious use of these directives, we have improved both the predicted parallelism and the measured parallelism. Remember, however, that the fraction of the code that can execute in parallel is obtained from detected parallelism rather than from potential parallelism. The box at the bottom of Figure 5 (left side) provides us with some potential problems that may be inhibiting parallelism in the code. By making use of this additional information we may be able to continue to improve the Amdahl's prediction as well as the actual speedup.

ATEXPERT uses measurements, sophisticated projection algorithms, and expert systems heuristics to arrive at the various statistics that it provides. In order to test the accuracy of this system, we ran PUEBLO on a dedicated YMP8/8128 using various numbers of processors and measured the actual speedup using the CF77 compiling system and autotasking. When we compared the results of this test with what ATEXPERT predicted, we found that at all levels of optimization, the differences were less than 10%. For example, at the highest level of optimization that we achieved (Figure 5), ATEXPERT predicted a speedup of 6.4, and we measured a speedup of 6.1 based on the sequential code using our version of ISMIN and 5.9 with the SCILIB version of ISMIN. This is a difference of 5% and 8%, respectively. The variation from what is predicted probably stems from several causes. One is there is a variable amount of work done in some of the loops in PUEBLO. This is known to affect the accuracy of the predictions from ATEXPERT. Another is the magnitude of the effect of memory contention, which is also known to be present in the code. The first effect could cause ATEXPERT to predict either higher or lower than what is measured; the second effect would cause the measured time to always be higher and therefore the speedup would be lower.

The success that we had in parallelizing PUEBLO came without nearly as much effort as had been required in the past and we decided to try nearly the same problem on a larger mesh size. The advantages of this would be that larger mesh sizes should give us longer vector lengths and better speedups. We used the same optimization directives that had been used on the smaller problem. We first used the ProfView tool to determine that the relative

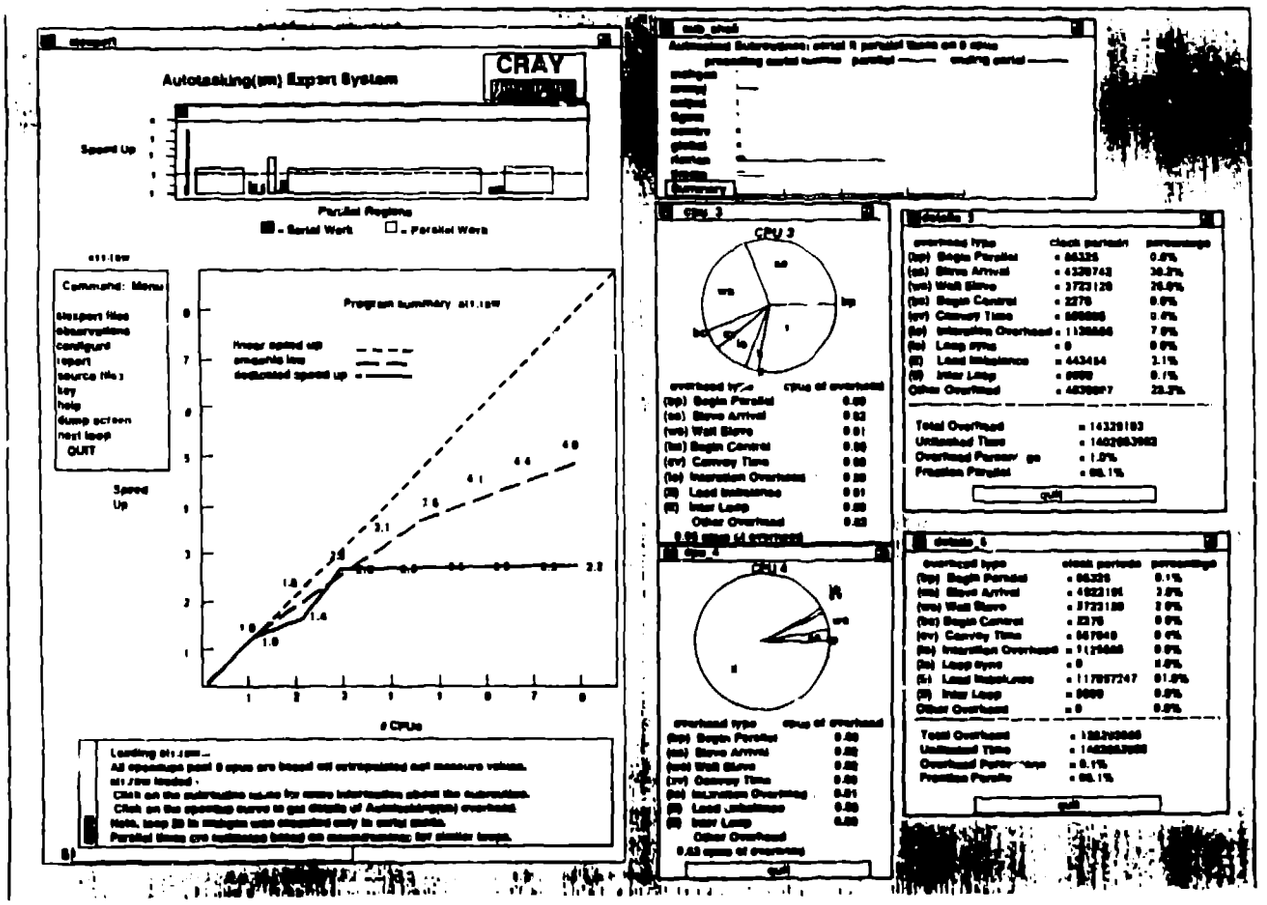

Figure 3. Graphical output from ATEXPERT showing the predicted speedup of 2.5 for the Initial Autotasking run of PUEBLO. This figure also shows the overheads associated with running the code.

subroutine usage had not changed. The results of that are shown in Figure 6, and we see that subroutine RIEMAN is still the most heavily used routine. However, because we are no longer doing the rezone-advection phase of the calculation, subroutine ADVECT is replaced by LAGVEL, which calculates the Lagrangian vertex velocities, as the second most heavily used routine. The results of the 64×64×64 size are shown in Figure 7. Note that both the Amdahl's Law prediction and the ATEXPERT prediction for speedups have improved. This is due, of course, to the fact that the vector lengths are now 262144; thus a larger percentage of the execution time is spent in the parallel parts of the code. These longer vector lengths also enable us to amortize more of the overhead associated with multitasking, and we see a decrease in predicted overhead from 0.8 cpus for the smaller problem to 0.6 cpus for this one, a 25% improvement. When we ran this version of the code on a dedicated system (again, a YMP8/128), the measured speedup for eight processors was 6.6. Again, this measured speedup is within 10% of the speedup predicted by ATEXPERT.

view

```
      do 10 l=1,lendv(lr)
      w(1,1) = 0.5e0*ss(l)/ra(l)
      w(1,2) = rho(l)*ra(l)
   10 continue

      do 20 m=1,3
      mcn = lcn(m,lr)
      do 20 l=lstrt(lr),lendv(lr)

c        compute the normal projections of the cell-centered velocities.

      unl = ((uc(l+mcn,1)*fn(l,m,1) + uc(l+mcn,2)*fn(l,m,2))
     &     +  uc(l+mcn,3)*fn(l,m,3))
      unr = ((uc(l,1)*fn(l,m,1) + uc(l,2)*fn(l,m,2))
     &     +  uc(l,3)*fn(l,m,3))

c        solve for the pressure and normal velocity of the face.
      umax = unl + w(l+mcn,1)
      umin = unr - w(l,1)
      plmin = pr(l+mcn) - w(l+mcn,2)*w(l+mcn,1)**2
      prmin = pr(l)     - w(l,2)*w(l,1)**2
      bl    = w(l+mcn,2)
      br    = w(l,2)
      a     = (br - bl)*(prmin - plmin)
      b     = br*umin**2 - bl*umax**2
      c     = br*umin - bl*umax
      d     = br*bl*(umin-umax)**2
      d     = sqrt(max(0.e0,d - a))
```

Figure 4. Code fragment from subroutine RIEMAN showing a loop that contributes to low-performance figure.

5. Conclusions

The conclusion that one can draw from this is that based on our experiences with PUEBLO, ATEXPERT does an excellent job of assisting in parallelizing a large code. The fact that this code is nearly 100% vectorizable helps because the analysis needed to determine vectorizability is essentially the same as that needed to determine loop-based parallelism. ATEXPERT's analysis provides the user with more information than previous tools have provided. Furthermore, this information is presented in several easily-understood formats. We were able to take a large sequential, but highly vectorizable code, and with a modest amount of effort parallelize the code obtaining predicted speedups of between 6.1 and 7.2. When the code was run on an actual eight-processor YMP, the speedups obtained were within 10% of those predicted by ATEXPERT. The effort that would have been required before the CF77 system and the advent of tools such as ATEXPERT would have been much greater. The information provided by ATEXPERT has also given us a better understanding of the performance characteristics of PUEBLO.

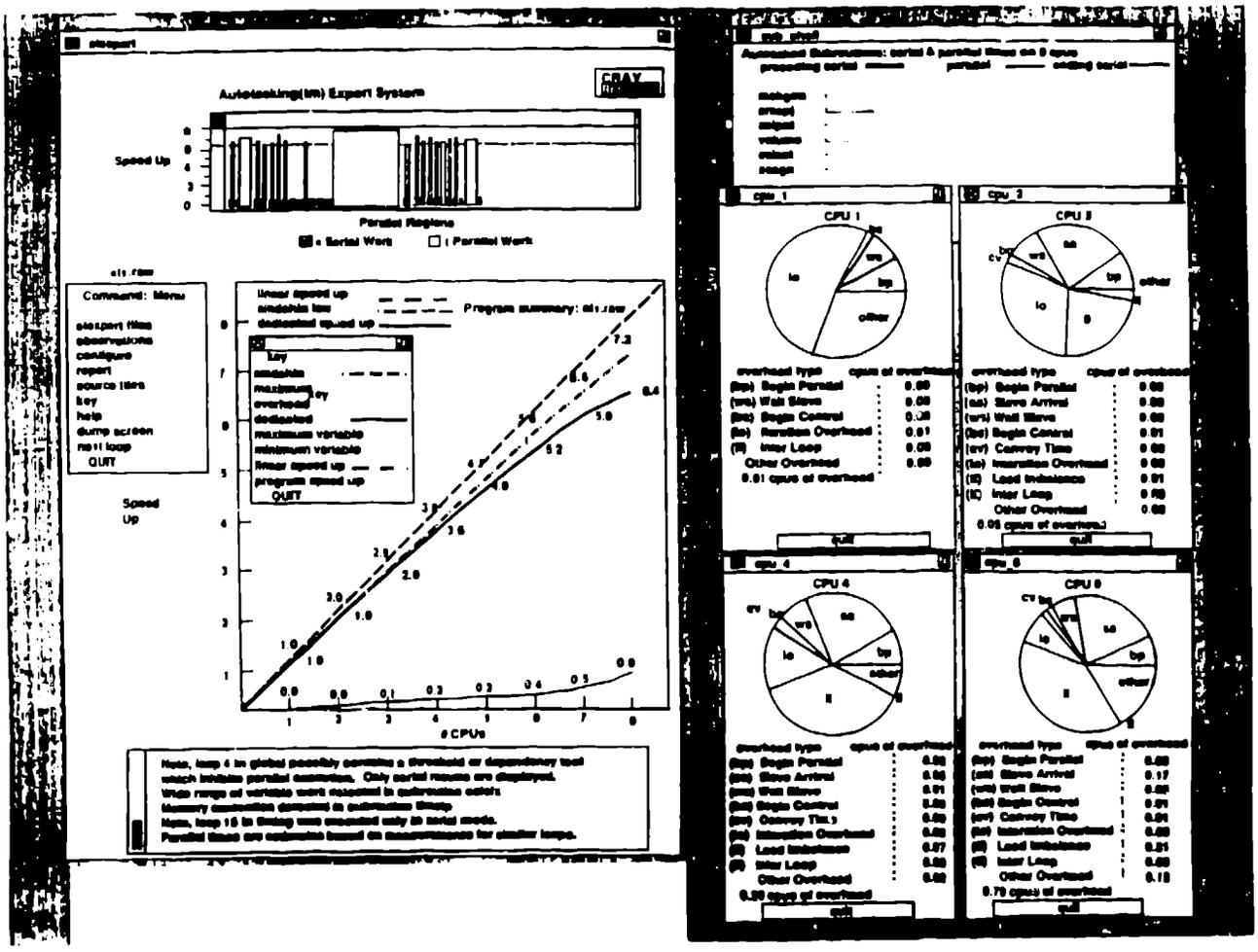

Figure 5. Graphical output from ATEXPERT showing the predicted speedup of 6.4 for the optimized autotasking run of PUEBLO. Overheads are also shown.

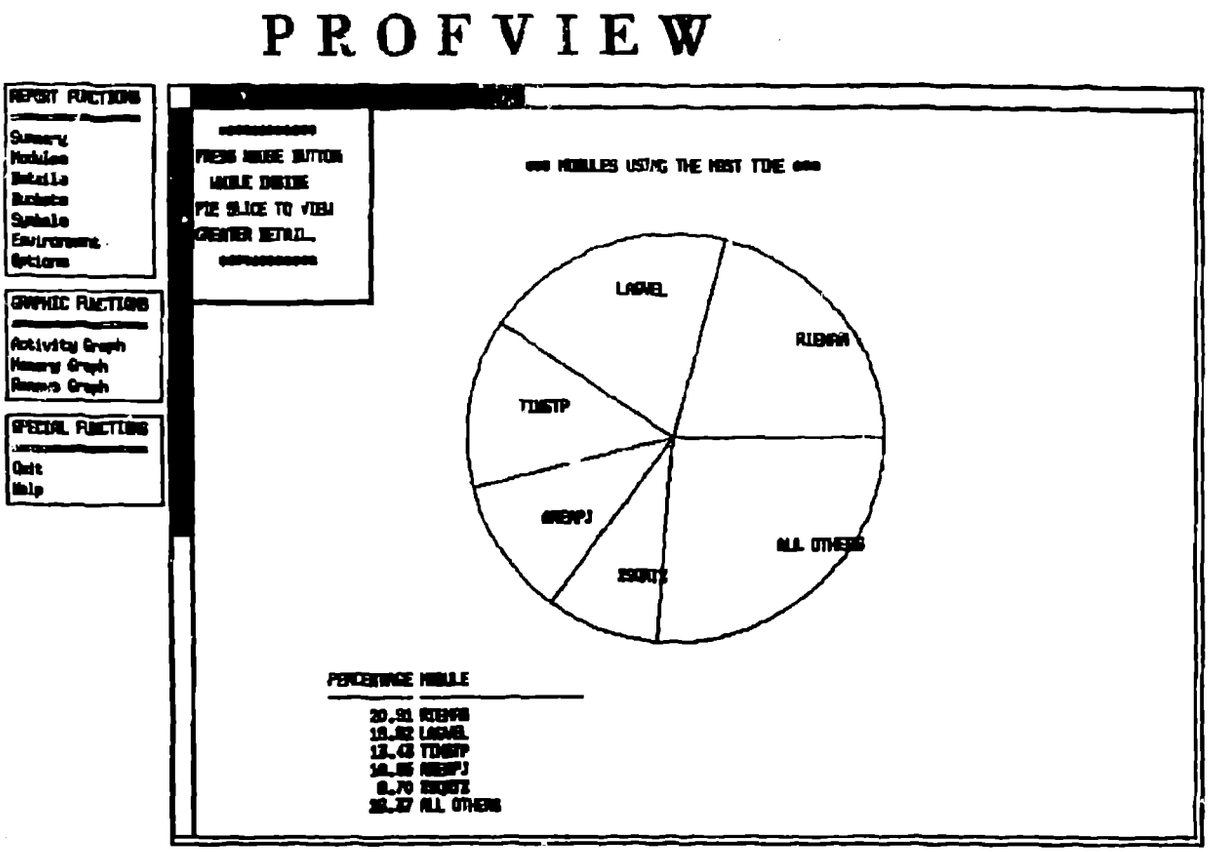

Figure 6. Graphical output from PROF/PROFVIEW showing where the percentage of execution time is spent in the larger PUEBLO problem.

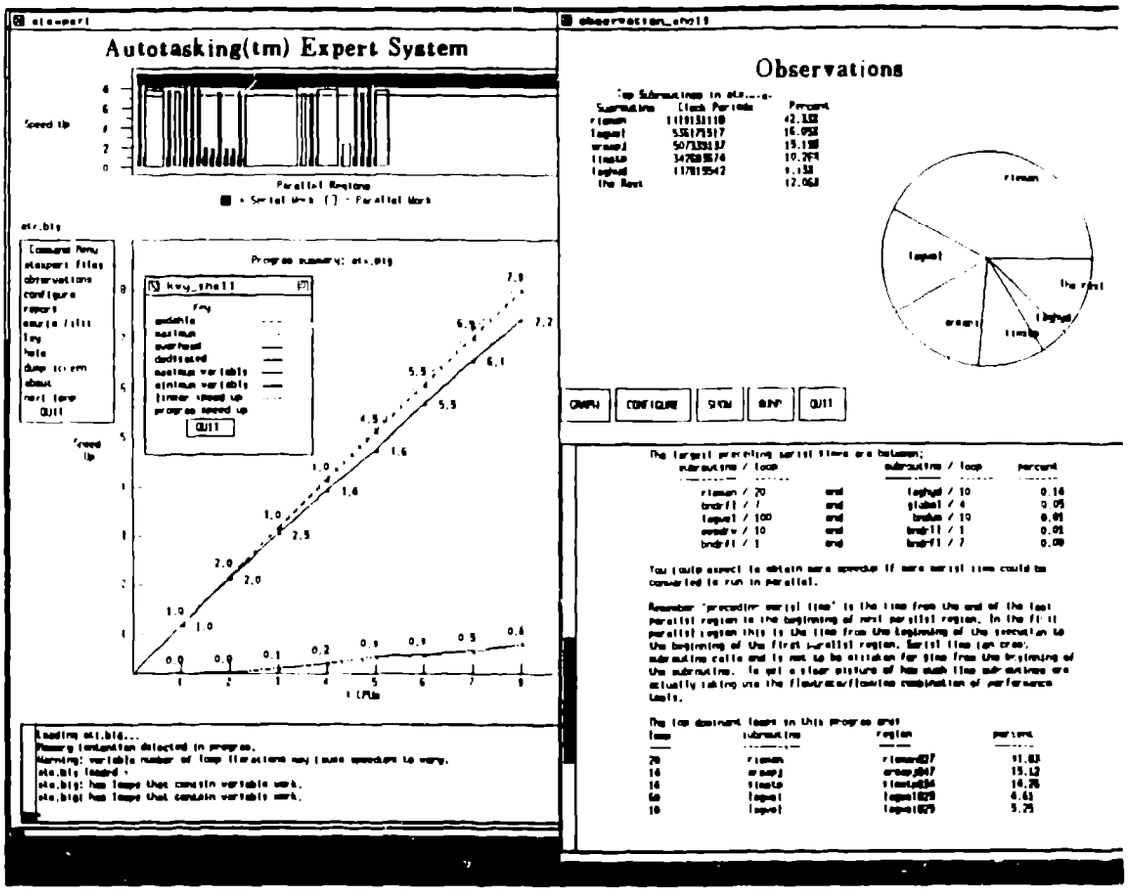

Figure 7. Graphical output from ATEXPERT showing predicted speedup of 7.2 on larger PUEBLO problem. Overheads are also shown.

References

(1) Simmons, M. L., "Using UCSD Multitasking Macros," Proceedings of the Cray Users Group, Spring, 1985, page 40.

(2) Lusk, Ewing L. and Ross A. Overbeek, "Implementation of Monitors with Macros: A Programming Aid for the HEP and Other Parallel Processors," Technical Report ANL-83-97, Argonne National Laboratory, Argonne, IL (December 1983).

(3) Ottenstein, Karl J., "A Brief Survey of Implicit Parallelism Detection," in Parallel MIMD Computation, The HEP Supercomputer and its Applications, edited by J. S. Kowalik, MIT Press, 1985.

(4) Nicolau, Alexandra, "Uniform Parallelism Exploitation in Ordinary Programs," Proceedings of the 1985 International Conference on Parallel Processing, August 1985.

(5) Lubeck, Olaf M. and Margaret L. Simmons, "An Approach to Partitioning Scientific Computations for Shared Memory Architectures," Proceedings of the First International Conference on Supercomputing Systems," pp 507-509, 1985.

(6) Applebe, William, Kevin Smith, and Charlie McDowell, "Static Debugging of Multitasking Programs," Proceedings of the Cray Users Group, Fall, 1985.

(7) Henderson, Leslie, Robert E. Hiromoto, Olaf M. Lubeck, and Margaret L. Simmons, "The Usefulness of Dependency-Analysis Tools in Parallel Programming: Experiences Using PTOOL," The Journal of Supercomputing 4, pp 83-96, 1990.

(8) Hayes, Ann H., Margaret L. Simmons, and Daniel Reed, "Workshop Summary Parallel Computer Systems: Software Performance Tools", Los Alamos National Laboratory Technical Report, 1992.

(9) Kohn, James, Cray Research, Inc., private communication.

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

(11) Ware, W., "The Ultimate Computer," IEEE Spectrum, March, 1982.

(12) Buzbee, B. L., "The Efficiency of Parallel Processing," Computer Design, June, 1984.