NASA CONTRACTOR REPORT 177464

Multitasking and Microtasking Experience on the NAS Cray-2 and ACF Cray-XMP

Farhad Raiszadeh
Sterling Software
Numerical Aerodynamic Simulation (NAS) Systems Division
NASA Ames Research Center, Mail Stop 258-5
Moffett Field, CA 94035
Tel: (415)-694-4322

Prepared for
Ames Research Center
Under Contract NAS2-11555

National Aeronautics and
Space Administration

NASA CONTRACTOR REPORT 177464

Multitasking and Microtasking Experience on the NAS Cray-2 and ACF Cray-XMP

Farhad Raiszadeh

CONTRACT NAS2- 11555
June 1987

NASA

MULTITASKING AND MICROTASKING EXPERIENCE ON THE NAS CRAY-2 AND ACF CRAY-XMP

Farhad Raiszadeh
Sterling Software
Numerical Aerodynamic Simulation (NAS) Systems Division
NASA Ames Research Center, Mail Stop 258-5
Moffett Field, CA 94035
Tel: (415)-694-4322

ABSTRACT

The fast Fourier transform (FFT) kernel of the NAS benchmark program has been utilized to experiment with the multitasking library on the Cray-2 and Cray X-MP/48, and microtasking directives on the Cray X-MP. Some performance figures are shown, and the state of multitasking and microtasking software is described.

INTRODUCTION

Most programs on Cray supercomputers are executed on one CPU. However, Cray supercomputers that have been installed at NASA Ames Research Center have four central processing units (CPU's). They include a Cray-2 at NAS and a Cray X-MP at the Advanced Computational Facility (ACF). To utilize the full potential of these supercomputers, programs can be executed simultaneously on these four processors. To achieve this improved performance for a single program, Cray Research Inc. (CRI) has developed some utilities for Fortran programmers. They are called multitasking and microtasking.

Multitasking is a mode of execution in a multiprocessor computer that provides for executing two or more parts of a single program in parallel. It uses feature routine calls from the Cray multitasking library. To achieve an efficiently multitasked program, the multitasked modules should have a large granularity (task size). Multitasking can be accomplished efficiently at any level of the program, as long as the overhead for the library calls does not overwhelm the gains from multitasking, and the data dependencies in different processors do not handicap the execution.

CRI offers another parallel processing approach for Cray X-MP called *microtasking* that permits multiple processors to work on a Fortran program at the DO-loop level.

Microtasking is not currently supported on Cray-2. It uses compiler directives rather than feature routine library calls, and can be implemented on modules with smaller granularity. In a microtasked code, synchronization overhead is very small, and tasks can be much smaller.

Following is an overview of different modes of parallel processing, and the highlights of their benefits and weaknesses.

PARALLEL PROCESSING

Normal Fortran programs on Cray machines are set up logically to run on a single processor. Programs that run on a single processor execute with considerable speed; however, in a four processor system, an even greater speed (approaching to four times faster) can be achieved if a single program utilizes all four processors concurrently.

Parallel processing can harness the power of all four processors for the execution of a single job. Utilization of parallel processing is not possible for all applications. Some programs and algorithms can make a reasonable use of parallel processing, while others do not lend themselves to this mode of execution. Before a user embarks on using parallel processing capabilities such as multitasking or microtasking, he should look at all other possibilities for improving the performance of his code. Most so-called *dusty decks* are not good candidates for parallel processing unless an attempt is made to optimize and vectorize the code before other improvements for parallel processing are implemented.

Conversion of a program to multitasking is usually a major programming task. In most cases a comprehensive study of the data dependencies between different modules of the entire program may be necessary. Since each task that is generated in a multitasking mode has an independent control sequence within the program, memory for variables can be allocated in two different modes. The memory can either be local to one task or it may be global to all tasks. In most programs a systematic analysis of the data dependencies and the flow of data between different tasks reveals that a program needs local memory as well as global memory. With the current compiler only global memory is available on the Cray-2, and the local memory for each task, which is dubbed `TASK COMMON`, is not available on the Cray-2. This handicaps multitasking capability. When local data are necessary for multitasking on the Cray-2, multiple definition of data for each task can resolve this shortcoming. However, this can be cumbersome, and in some cases it makes multitasking impossible to implement.

The order in which the statements are executed in programs run on only one CPU is well defined. Repetitive runs produce identical results because the same instructions are executed in the same order each time. Data and control dependencies are satisfied

simply by the location of statements in the program. Temporal ordering is implied by spatial ordering. Parallel processing introduces a new dimension to the order of execution. While the sequence of execution in each task remains well defined, the relative order of task execution has no default order. In fact, the order of execution may change from run to run, and the programmer must control the ordering to satisfy dependencies. Failure to manage the temporal ordering of tasks is a subtle error that may be difficult to identify. While there is no guarantee that more than one processor will be allocated to work on the tasks of a given job, there is also no guarantee of which of the parallel tasks will finish first. Multitasking is *nondeterministic* with respect to time but software processes must usually be made deterministic with respect to results. Communication between parallel tasks and protection of shared data must be taken care of by the programmer. If some data from one part of the program is needed to execute another part, they can not run concurrently. Parallel processing is possible only when the concurrent parts of the program can execute independently. To clarify these subtle rules, some typical classes of codes executed on the Cray-2 are described in the following paragraph.

Computation of fluid flows by finite difference techniques has traditionally been accomplished with a system of grid points. Recently, multiple grids have become popular with some researchers in the field. A multiple grid code with overlapping, or patched grid systems is a prime candidate for multitasking, because each patch of the grid can be solved by one processor, and at the end of each iteration the boundaries of different grids can be updated. This class of problems furnishes a high granularity for each task, and can utilize multitasking in an efficient manner. In contrast to overlapping or patched grid codes, composite grid codes with two systems of coarse and fine grids can not be multitasked in this level because the solution of the two grid systems are interdependent. Multitasking of this class of codes should be accomplished in a lower level with a smaller granularity. In a single grid problem, if a factorization scheme is used, different dimensions can be resolved concurrently. However in an implicit scheme the benefits of multitasking are limited by data dependency between different parts of the program, and the small granularity of the parts that are independent. In problems where each step is dependent upon the previous step, and individual steps are not time intensive, the multitasking library does not speedup the execution, and the only alternative for parallel processing is microtasking because it has smaller granularity. However, microtasking is not currently available on the Cray-2.

The main advantage of microtasking is its low overhead and ease of implementation. It works better when the task size is small. For example, it can be utilized in the DO-loop level, where the overhead for feature routine calls of the multitasking library would be inefficient. Another advantage of microtasking is that when dedicated (stand-alone) time is not available, the microtasked job can dynamically adjust to the number of processors that become available for short periods from time to time.

Following is a brief review of multitasking and microtasking of one of the test kernels

Table 1: Timings for FFT NAS Kernel on Cray-2 and Cray X-MP

Execution Mode	Cray-2 Total CPU	Cray X-MP Total CPU	Cray-2 CPU Multitasked section	Cray X-MP CPU Multitasked section
Uni-tasked	18.01 sec	23.55 sec	1.12 sec	0.997 sec
Multitasking	17.55 sec	21.97 sec	0.28 sec	0.361 sec
Speedup Factor	1.03	1.07	4.	2.76

(CFFT2D), which is a part of the NAS Kernel Benchmark program.

UTILIZING THE MULTITASKING LIBRARY

The fast Fourier transform kernel of the NAS Benchmark program was multitasked as a test of the multitasking library. Multitasking library consists of a set of feature routine calls that control the execution of the code and the flow of data between different tasks. To test the robustness of the multitasking library, the multitasked program was executed on the Cray-2 and Cray X-MP. There was no attempt to achieve maximum possible parallel processing. In fact, this exercise was a feasibility run to verify that the libraries work.

This FFT routine is not the best possible candidate for multitasking, because in each iteration there exists some data dependency with respect to previous computations. Consequently, the multitasking could not be accomplished in the highest level of the subroutine calls. In fact multitasking was only utilized in the most computationally intensive DO-loop. The number of library calls to TSKSTART and other multitasking routines were proportionate to the number of iterations. However, this test demonstrated that the Fortran programmer can obtain benefits from multitasking if his algorithm and data structure are suited to parallel processing. This exercise also demonstrated the effort necessary to utilize multitasking library on both Cray-2 and Cray X-MP. The amount of man-hours necessary for the multitasking of a code is highly dependent on the experience of the programmer. For an experienced analyst who is novice in the usage of multitasking, approximately forty hours of work is necessary to convert a code like the NAS benchmark FFT kernel to a multitasked version. This does not include the time necessary for the initial familiarization with the library.

Table (1) shows the overall run time and run time for the multitasked portion of the program on the Cray-2 and Cray X-MP. Total CPU time is defined as the execution time

based on the wall clock in a batch mode. These times are presented for the Cray-2 and Cray X-MP, with and without multitasking. These values are not exact and may vary up to ten percent for each run due to memory bank conflicts and swapping, but usually it varies only a few percentage points. Multitasked section CPU time is the time spent on the DO-loop that was multitasked. This CPU time does not include the overhead for the library calls; it is the CPU time spent inside the DO-loop. This table also shows the improvement that was achieved by parallel processing. As displayed in the table, the speedup for the multitasked portion of the program is four times for the Cray-2 and 2.76 times for Cray X-MP. This speedup is only for the multitasked portion of the program, and the overall speedup is much lower, because only a small portion of the program was multitasked. In fact, the serial portions that could not be multitasked used most of the execution time, as is reflected in the performance table.

The overhead for multitasking libraries was rather large in this example, because of the small granularity of the multitasked section of the code. However, the overall run time was reduced, but not to the extent that one would expect from a four processor system. Another detriment to the improvement was that only some parts of the program could be multitasked, and to the extent that the program runs with one processor, it could not benefit from multitasking.

Multitasking is not applicable to inherently sequential algorithms, and a systematic study of the code is necessary before any attempt is made to multitask a code. Also before a user embarks on multitasking, he should try to optimize and vectorize his code. These test runs demonstrated that the multitasking library is an important tool for parallel processing of the Fortran codes on both the Cray-2 and Cray X-MP. However, some enhancements to the library calls are greatly needed before we can expect general acceptance of multitasking by the users. Of special interest is inclusion of `TASK COMMON` in the Cray-2 version of the multitasking library. With such enhancements it is expected that multitasking will get a wider acceptance by the users.

MICROTASKING

Microtasking permits multiple processors to work on a Fortran program at the DO-loop level or in the subroutine level. Microtasking has a very low overhead; and if it is used at the DO-loop level, the user does not need to be concerned with the synchronization analysis. Microtasking can be a very handy tool, because it usually does not require the extensive data dependency analysis that one would be expected to address in a multitasking job. Microtasking consists of some compiler directives that invokes parallel processing of the microtasked code. The absence of library calls reduces the overhead for microtasking significantly. Hence, microtasking can be utilized for parallel processing of code sections

with smaller granularity than the feature call multitasking library. This usually reduces the scope of data dependency analysis that has to be addressed in microtasked code. Therefore, microtasking can usually be accomplished with much less effort than multitasking library calls. In fact the microtasking of the fast Fourier transform kernel of the NAS benchmark program was microtasked with about eight man-hours of effort by an analyst with limited experience in microtasking. This is significantly less than the necessary effort for multitasking of the same code.

Currently, microtasking is not supported by Cray-2 and it is only available on Cray X-MP. However, it does not run on X-MP under the current operating system (COS 1.14). In fact, the simplest microtasking job crashes the system. As a test of microtasking, the FFT code that ran with one processor was chosen for microtasking. One of the simplest microtasking jobs is a microtasked DO-loop. Here the modifications consisted of initiation of multiple CPU's, a directive for the beginning of the microtasked subroutine, and a single microtasked DO-loop. These modifications are as follows:

```
CMIC$ GETCPUS 4        ! to initiate the four CPU's
CMIC$ MICRO            ! to begin the subroutine with microtasked code
CMIC$ DO GLOBAL        ! the microtasked Do-loop
```

This is the simplest possible microtasking job that one may execute on Cray X-MP. But even this job crashes the system. This experience demonstrates that the microtasking directives need much improvement.

At this time only a handful of users have attempted to utilize microtasking on ACF's Cray X-MP. After consultation with these users, it is clear that the robustness of microtasking is questionable. It is expected that COS 1.16 will be installed on the Cray X-MP in the Spring of 1987. NASA Ames will be a beta test site for this version. According to John Avila microtasking has worked with COS 1.16 at the Chevron facility in Los Angeles. When the new operating system is installed on Cray X-MP, the microtasked FFT code will be used as an acid test for the new version of the operating system.

CONCLUSION

Multitasking and microtasking software are important tools for parallel processing. Currently, the multitasking library is supported on both Cray-2 and Cray X-MP. These library calls are not user friendly; but if the code lends itself to multitasking, a high degree of parallel processing can be achieved when the rules on data dependency and load balancing

are followed. If the benefits of parallel processing are to be realized, some enhancements to the multitasking library are necessary. One of the more pressing needs is TASK COMMON, which is not yet available on Cray-2. This is especially beneficial for the large codes that have many different data structures.

Microtasking is easy to implement, and has a very low overhead. However, it has not been a very reliable piece of software; furthermore, it is not supported on the Cray-2. NAS should begin talking to CRI to accelerate development of reliable microtasking software on the Cray-2. Once microtasking becomes a reliable utility, users will most likely prefer it to the multitasking library. CFD researchers may not want to spend weeks analyzing and modifying their program with multitasking library calls. But microtasking can be utilized in a timely and convenient manner that should be appealing to a user. Finally, microtasking can be utilized with less effort, and with a low overhead for most programs, while multitasking is not as yet applicable to many algorithms.

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
CR 177464		

4. Title and Subtitle	5. Report Date
Multitasking & Microtasking Experience on the NAS Cray-2 and ACF Cray-XMP	July 1987
	6. Performing Organization Code
7. Author(s)	8. Performing Organization Report No.
Farhad Raiszadeh	
9. Performing Organization Name and Address	10. Work Unit No.
Sterling Federal Systems, Inc. 1121 San Antonio Road Palo Alto, CA 94303	K 1707
	11. Contract or Grant No.
	NAS2-11555
12. Sponsoring Agency Name and Address	13. Type of Report and Period Covered
National Aeronautics & Space Administration Washington, D.C. 20546	Contractor Report
	14. Sponsoring Agency Code

15. Supplementary Notes

Point of Contact: Robert A. Carlson, MS: 233-15, NASA/Ames Research Center, Moffett Field, CA 94035
(415) 694- (FTS464) 6036

16. Abstract

The fast Fourier transform (FFT) kernel of the NAS benchmark program has been utilized to experiment with the multitasking library on the Cray-2 and Cray X-MP/48, and microtasking directives on the Cray X-MP. Some performance figures are shown, and the state of multitasking and microtasking software is described.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
Fast Fourier Transform (FFT) multitasking library microtasking directives Cray	Unclassified, Unlimited STAR Category - 61

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price*
Unclassified	Unclassified	9	

*For sale by the National Technical Information Service, Springfield, Virginia 22161