

# New Tools Using the Hardware Performance Monitor to Help Users Tune Programs on the Cray X-MP\*

D. E. Engert, L. Rudsinski  
Argonne National Laboratory

J. Doak  
Cray Research Inc.

September 25, 1991

## Abstract

The performance of a Cray system is highly dependent on the the tuning techniques used by individuals on their codes. Many of our users were not taking advantage of the tuning tools that allow them to monitor their own programs by using the Hardware Performance Monitor (HPM). We therefore modified UNICOS to collect HPM data for all processes and to report Mflop ratings based on users, programs, and time used. Our tuning efforts are now being focused on the users and programs that have the best potential for performance improvements. These modifications and some of the more striking performance improvements are described.

## 1 Introduction

The Cray X-MP at Argonne National Laboratory went into production in January 1988 under the UNICOS operating system. Its purpose is to provide supercomputing capabilities to the Laboratory.

The X-MP is, of course, capable of doing general-purpose computing. Its ease of use, good network access, security, and large file systems make it attractive for almost any type of computing. Our objective, however, is to use the X-MP as Argonne's scientific batch compute engine. Interactive access to UNICOS is intended mostly for interactive graphics, debugging, and simple data management. We encourage users to develop code, read mail, and do word processing on other machines.

The strong point of the Cray X-MP for scientific computing is its ability to vectorize floating-point calculations. Nevertheless, the performance of the X-MP depends heavily on

---

\*Work sponsored by U.S. Department of Energy under contract number W-31-109-Eng-38

code tuning. Currently we have about 500 enrolled users. Many of these users have simply ported their code from other computers and have relied on the compiler optimization for performance speedup. They have made little effort to use the Cray tuning tools to achieve maximum performance.

Aware of this situation, our division director asked the question, "Can someone tell me the megaflop (Mflop) rate of our X-MP?" Admittedly, the Mflop rating of a program is only a relative indication of its effective use of the machine. Programs that are not floating-point intensive may still be well suited for the Cray. They may be doing logical operations or integer calculations, or even using the Cray for its I/O performance. And we are aware that the Mflop rating can be artificially inflated. At one time we considered a "cpu guzzler" surcharge for processes showing low Mflop performance, but soon realized that that was not a good idea and probably would not work, since all users would have to do is add a small section of "do nothing" code that does vector additions and multiplications just to raise their Mflop rate. A user who has changed the HPM group from group 0, or who adds a few extra multiplications by 1.0 or additions of 0.0 to his loops can change the Mflop rating while not changing the overall CPU time of a process. (One such do-nothing test case gets 304 Mflops on a machine with a peak rating of 235.)

Nevertheless, we believed that an overall Mflops count was certainly of interest. We also knew that the Cray, unlike any other family of computers, has a Hardware Performance Monitor that can provide performance statistics on any user code. The HPM comprises of four groups of counters, of which one group may be active at a time. Group 0 and 3 can be used to count Mflops. At process switch time, the counters are read and reset. Three sets of counts are kept by the kernel: an overall count, a process count, and a multitasking group count. The user can invoke the `hpm` command to measure his individual program, but no command existed to show the overall count.

## 2 New Tools

In December 1989, we asked our CRI analyst, Jeff Doak, to tackle the Mflop overall count problem. Jeff changed the default HPM group from 1 to 0. He then wrote `hpmrpt`, which uses routines from `hpm` to report the Mflop rate since the last boot. The new command reads from `/dev/hpm_all` to get the systemwide counts, instead of reading from `/dev/hpm` which contains the data for a single process. A cron job and `hpmrpt` provides us with a daily Mflop rate for the X-MP. UNICOS 6.0 now has the `hpmall` command, which is a direct result of the work started here.

It soon became apparent that if we could get the overall Mflop rate and if a user could get his own Mflop rate, we should be able to get the Mflop rate for each process. That information would give us insight into how the machine was being used, who was using it well, and—more important—who needed our help in tuning a code.

Again, no UNICOS 5.1 tool existed to provide this information. This time, Bob Swanson of Cray's Libraries and Tools Group rescued us. He gave us a replacement routine for `hpmddump` which is in `libc` and is linked with every module. Jeff changed `hpmddump` to write one record per process to a common file only if it uses more than 15 seconds of CPU time. Each record contains `userid`, `date`, `time`, `process name`, `process number`, and `process HPM data`. The access method is simple; we do not use file locks or `retry`. Few access collisions occur, and the update of the common file takes just a fraction of a second. Moreover, the user program size is increased less than 500 words by the additional code to `hpmddump`. The new tool was installed in July 1990.

Since `hpmddump` is a process termination routine and part of the user's module, only those modules linked after this date will record the statistics. When building new versions of UNICOS, we install the `hpmddump` modification last, so that only the user programs and not all of the UNICOS utility processes have this modification. This approach is taken for performance and reliability reasons.

Jeff also wrote a new program `mflops` to read and report on the contents of the common file. It reports, by default, the recorded programs run for the current user on the current day. Options include reports on codes, users, specific dates, and CPU use. Our report program `mflops` was the basis for the UNICOS 7.0 command `hpmflop`. `Hpmflop` is our current report generator. Jeff has placed a version `hpmddump` and the report program `hpmflop` on the crayamid in `jdoak/hpm.cpio`.

### 3 Cray Optimization Project

With the development of the `mflops` program, we were in a position to focus our tuning efforts on programs that could benefit most from code optimization. In January 1991 we formed the Cray Optimization Project, headed by Larry Rudsinski.

From the `mflops` statistics we selected the top ten users whose codes were using the most CPU time on the Cray and whose Mflop rate was less than 25. We contacted these users and discussed with them the performance of their code. Throughout, we emphasized how an improvement in performance would allow them to do more science—or even new science—for the same computer cost. Our approach was to make the offer so good that the user could not turn it down.

Admittedly, most scientists that we contacted were a bit apprehensive that some stranger was going to obtain a copy of their code and analyze it. But we continued to play down this point by stressing that we provide the computer performance insight and they provide the scientific insight. And yes, the real selling point was that the service was free.

After gaining the support of the scientist, we obtained a copy of the source code and all

data files needed to run a representative case. Generally, this involved creating a directory in `/tmp` (i.e., `/tmp/1er`), setting the permissions so everyone can read and write into the directory, having the scientist copy the files to that directory and give read permission to the files, and then storing the needed files in a working directory. This approach might seem cumbersome, but it is certainly less annoying than finding out that some essential file is read protected or missing.

Once all the needed files were in the working directory, we created a subdirectory for the source code. We used the Cray tool `Fmaker`, which splits the source file into several files, with each subroutine or function contained in its own file. `Fmaker` then builds the `make` file needed to build the executable (i.e., `a.out`) file. The parameters for both the compiler and loader can be set on a single line in the `make` file. After the first use of `make`, only the routines that are being changed need to be recompiled, thus reducing compilation time and cost.

## 4 Challenges Met

An early challenge in the Cray Optimization Project was scaling down a production run to get a reasonable test case. We needed to execute the code long enough to get meaningful statistics and yet not consume significant Cray resources. Even when the file itself executed in only 10 minutes, the additional overhead needed for accumulating statistics from the analysis tools such as `PERFTRACE` and `PROF` would cause the `NQS` time limit to be exceeded and the statistical output from `PERFTRACE` or `PROF` to be lost. Therefore, we needed a way to generate performance statistics even if the code terminated with a time limit. Otherwise, the run was a waste of our time as well as Cray CPU time.

Jeff responded by writing two routines: `cleanstop.c` and `cleanstop.f`. When the task time limit is reached, `cleanstop` receives the `sigcpulim` signal and terminates the code gracefully, saving the statistical output from `PERFTRACE` or `PROF`. The two routines have been placed on the `crayamid` in `jdoak`.

The statistics generally indicated that two or three routines were consuming most of the execution time. We typically used various tools from the `Toolpack` library (A library of Fortran 77 tools developed at Argonne) to clean up these routines—indenting `do`-loops and `if`-structures, resequencing statement labels, and rewriting `IF-GO TO` Fortran syntax to the more readable `IF THEN ELSE ENDIF` format. The routines were then compiled with `cft77` using the `m` option. This option annotates the listing file, indicating which loops vectorize and which ones do not. Finally, we made the appropriate optimization and vectorization changes.

In making such changes, it is important that we can tell not only how much the changes decrease the execution time, but also how much they increase the `Mflop` rate for

that particular section of code. Again, Jeff wrote a small C program, called `floprpt`, that provides the desired information. The code of interest is bracketed by two calls to `floprpt`. The first call, with zero as an argument, initializes the Mflop counters; the second call, with an argument of one, prints the results from the first call. The following is an example of the code and the results from using `floprpt`.

```

        parameter(ndim=75)
        dimension a(ndim), b(ndim), c(ndim)
        data s /.25/
        data n /ndim/
        write (6,*) n
c
        do 1 i=1,n
        b(i) = 10.0*ranf()
        c(i) = 23.2*ranf()
1   continue
c
        call floprpt (0)
        do 2 i=1,n
        a(i) = b(i) + s*c(i)
2   continue
        call floprpt (1)
        write(6,*) a(1),a(n)
        stop
        end

% hpm a.out
add/mult/recip/clocks/mflop: 50 50 0 140   84.03

```

We mention a caveat in measuring small loops in a test case as shown here. When the compiler compiles `loop2` in the main program, if it does not think the output from the loop is ever used, it does not generate any code for that loop. Thus, we added the `write` statement to write out the first and last element of the array `a`.

## 5 A Win-Win-Win Situation

The results of the Cray Optimization Project have been excellent. After we obtained the various statistics and discussed with the users where their code was spending most of its time, several users completely rewrote the routines, gaining significant improvement in

performance and much better Fortran. In other cases, only minor changes were necessary to get the compiler to do a better job of vectorizing the code. We have worked with over ten users in the first eight months of the project, seeing improvements from factors 1.5 to over 11 in the execution time.

The project has proven to be a WIN - WIN - WIN situation. The users WIN by having codes that perform better, thus allowing researchers to do more science for the same cost, with better turnaround. The computer center WINS by having better rapport with our users and by having more computing capacity available for them. CRI WINS by having codes that are tuned for their computers, thereby strengthening CRI's position as the supercomputing market leader and putting the organization in an excellent position for future acquisitions.

In fact, anyone can WIN who has a diverse user community where little is known about the users or their codes. These modifications can be obtained by a center's CRI analyst from the crayamid system. They are located under jdoak.

## Note

Jeff Doak is no longer our site analyst but has moved on to another assignment within CRI. We thank Jeff for his hard work on this project and expect to see his name on other Cray improvements in the future.

For more information we can be contacted by email at:

Doug Engert	b17783@anlvm.ctd.anl.gov
Larry Rudsinski	b26605@achilles.ctd.anl.gov
Jeff Doak	jdoak@birch.cray.com

## DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of author expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

**END**

**DATE  
FILMED**

**11 127 191**

