Memory, I/O and Multiprocessing Considerations on ABAQUS and MATLAB

Dong Ju Choi and *Mike Ess*, ARSC, Fairbanks, Alaska

ABSTRACT: *In this paper we present results for Abaqus and Matlab jobs running on the Cray-YMP. Beyond the simple job submission, the user on the Cray YMP has several options such as defining I/O characteristics and using multiple CPUs. Using ABAQUS(5.4) and MATLAB(4.1.1) as examples, we evaluate some of these strategies in terms of the trade-offs between memory, I/O CPU time and wall clock time. We also evaluate how these trade-offs effect resource accounting.*

Introduction

Many users of the Cray YMP at ARSC are not programmers but “users of packages”. These packages are applications available to users only as binaries and are treated by the user as “black boxes”. They perform a function for the user but the details of how that function is implemented is hidden from the user. Abaqus for finite element problems and MATLAB for mathematical problems are two such examples.

Even without access to the source, the users of these packages have options to effect performance on the YMP. In the case of Abaqus, the user may alter how I/O is handled. For both Abaqus and MATLAB, the user may specify how many YMP CPUs will be used during execution. These decisions have complicated effects on wall clock times, CPU times and allocation resources. These decisions are further complicated by the interaction of the batch environment with the user’s job.

This paper describes some experiments with these two packages on the ARSC YMP. Variations in performance due to the batch environment are specifically measured and illustrated.

Abaqus Results

Abaqus is a Finite Element package supplied by Hibbit, Karlsson & Sorensen, Inc.[1], ARSC has version 5.4. It solves a wide range of problems in statics, dynamics and heat transfer. Abaqus coupled with the 1GW of memory on the ARSC YMP is a unique computational resource. And accordingly, Abaqus is one of the most popular application at ARSC, both in number of users and in total CPU time consumed.

By default, Abaqus is a multitasked program, meaning that it was compiled with -Zp or is linked with library routines that are multitasked. In this situation, the value of the user’s environmental parameter NCPUS determines whether multiple CPUs are used or not. Usually the NCPU environmental parameter is set in the abaqus script that users define when they submit an Abaqus job. The sample scripts, provided by the suppliers of Abaqus, have the NCPUS environmental parameter set to 1 but it is easily changed. The ARSC YMP is a heavily loaded machine and we are concerned about the overhead of multitasked jobs on the throughput of the YMP. This was the motivation for experimenting with Abaqus.

We began with a benchmark problem provided by Hibbit, Karlsson and Sorensen, Inc[2]. From that document we have the following description: “This is an eigenvalue analysis consisting of shell elements of the type S8R. The problem does nine back-substitutions for nine iterations to extract 15 eigenvectors”. This problem has a very large I/O component because of the multiple back-substitutions. In particular the I/O wait time is approximately equal to the CPU time.

Using the CRI tool procstat, it was easy to narrow in on the file that accounted for most of the I/O wait time. Below are the typical results from procview run on the results collected by procstat collected during a benchmark run:

Table 1. Procstat statistics for Abaqus Experiment

File Unit	FIle Size(MWs)	I/O wait time(seconds)
2	23	246
1	3	34
19	3	51
23	3	0
21	2	4
10	2	100
22	1	0
25	1	1
30	1	1

The experiments that we decided to try were:

1. Vary the number of CPUs by changing the environmental parameter NCPUS.
2. Change the I/O behavior with ASSIGN commands
   a. Assign file FT02 to the ldcached /tmp file system
   b. Assign file FT02 to be a memory resident file
   c. Use the ABAQUS EIE FFIO I/O layer to manage file FT02

At ARSC, the /tmp file system is ldcached with 1 MB of the 1 GW DRAM memory allocated as a buffer for files on the /tmp file system. Another I/O optimization allocates memory in the user's job to act as a RAM disk for the entire file. This is the strategy of making the most active files "memory resident".[7] A third alternative is to use the EIE FFIO layer provided by CRI that specifically optimizes I/O on Abaqus jobs[8].

The results from the utility "ja" of these experiments are summarized in Tables 2 and 3 below.

Table 2. Wall clock time for Abaqus experiments(seconds)

Number of CPUs	1	2	4	8
Using /tmp	819	589	750	1123
Memory Resident	441	580	454	477
EIE FFIO layer	428	427	432	558

All of these runs were made during the typical batch time at ARSC. From these results we can see that using more than the default I/O can have significant reductions in wall clock time but that multiprocessing did not always reduce wall clock time. In particular, using 8 CPUs always increased the time that the user waited for their job to finish.

Wall clock time and CPU time are only two metrics that concern our users. Most ARSC users are given a certain number of "Service Units" and use of that unit is the quantity to minimize. At ARSC we have a particularly simple algorithm for computing SUs (Service Units):

SUs =
= 1.000 * YMP time (CPU hours)
+ 0.050 * T3D timetable hours)
+ 0.005 * Memory*YMP time (MWords hours)
+ 0.003 * Disk real time (GByte hours)
+ 0.003 * DMF Silo real time (GByte time)
+ 0.000 * CRL tape time

Using the statistics from the utility ja we can also produce a table of the SU charges for each of the above experiments. The only other term relevant to the above algorithm is the time-memory integral as shown in Table 3:

Table 3. CPU time-memory integral (Mw-seconds) for Abaqus exper.

Number of CPUs	1	2	4	8
Using /tmp	4237	4237	2395	2121
Memory Resident	9648	9886	9674	9626
EIE FFIO layer	11643	11631	11714	11735

Combining the results of Table 2. and Table 4. with the SU charging algorithm we can produce the SU charges for each of the Abaqus experiments:

Table 4. SU charges for Abaqus experiments

Number of CPUs	1	2	4	8
Using /tmp	484	486	494	497
Memory Resident	462	475	474	479
EIE FFIO layer	462	465	477	488

From these experiments, ARSC is able to suggest to its many Abaqus users the following recommendations:

1. Using either memory resident files or the EIE FFIO layer can greatly reduce I/O wait times (and therefore wall clock time) on large Abaqus runs.
2. The memory resident file produces a lower time-memory integral than the EIE FFIO file. However the EIE FFIO file uses less system time than the memory resident file.
3. On ARSC's YMP, both the memory resident file and the EIE FFIO file perform equally well, but given that the memory resident file is much easier to use, we recommend using memory resident files. On other YMPs, that do not have 1 GW or memory, this recommendation may not be valid
4. SU charges are always higher for multiprocessor Abaqus jobs, but the wall clock times are not always shorter.

MATLAB Results

Another package in use at ARSC is MATLAB, ARSC has version 4.1.1. From the MATLAB User's Guide[3] we have: "MATLAB is a technical computing environment for high-performance numerical computation and visualization." At ARSC the combination of SGI workstations has an interface to MATLAB running on the YMP is an attractive combination. But as is well known in the CRI world, not all problems are well solved by one computational environment.

The version of MATLAB that was installed at ARSC was a multitasked version that used multiple CPUs if the user allowed it. The number of CPUs specified by the NCPUs environmental parameter determined how many CPUs were used for each MATLAB session. By default at ARSC, NCPUS is set to 4 so unless the user specified otherwise, MATLAB was always running with 4 CPUs.

For some problems this maybe appropriate, in particular linear algebra problems on CRI machines can produce good speeds and speedups on multitasked code. To investigate whether this was the case with multitasked MATLAB we chose the two LINPACK[6] Benchmark problems to be solved with MATLAB and various other libraries. The problems factor and solve two systems of linear equations, one system has 100 equations, the other has 1000 equations.

There are several methods available for solving such systems of linear equations and because the linpack benchmarks have been around for more than 15 years, products have developed features to solve these specific problems. The six methods of solving these problems that we measured were:

1. A simple compilation of the Linpack Benchmark source code with the -Zp option. This way is insufficient to achieve either good vectorization or any multitasking.
2. A replacement of the Fortran Linpack and Blas routines of the Linpack Benchmark with the optimized versions in Libsci.
3. A compilation of the Linpack Benchmark source code with inlining turned on, so that the Blas routines are inlined into the Linpack routines. This provides the compiler with a nest of three DO loops that can be efficiently vectorized and multitasked.
4. A version of the Linpack Benchmark with the calls to Linpack factor and solve routines replaced with similar calls to the Lapack factor and solve routines. These Lapack routines are supplied from Libsci and are well vectorized and multitasked.
5. Using the IMSL[5] routine lsarg.
6. Using the simple MATLAB script:

```
n = 1000;
A = randn(n);
for i = 1: n
    B(i,1) = 0.0;
    for j = 1: n
        B(i,1)=B(i,1)+A(i,j);
    end
end
C = A \B;
```

In tables 6 and 7 below we summarize the mflop/s rates for each method on 1, 2, 4 and 8 CPUs on the 1000 by 1000 case and in tables 8 and 9 below we summaries the results for the 100 by 100 case.

For most uniprocessor run we used the CPU timer second() to solve the factor and solve sections of code. For MATLAB there was no CPU timer so we used statistics from "ja" and subtracted away the overhead of matrix generation and output. For most multiprocessor runs, we used the real time clock timer RTC() and for MATLAB there is a built in pair of functions to measure wall clock time(tic/toc).

Several observations can be made from the tables of data:

1. On the ARSC YMP M98 there is a tremendous variance in uniprocessor times, depending on whether the run is in dedicated mode or batch mode. This is most likely due to the DRAM memory that is shared by all users in batch mode.
2. The results for both dedicated and batch runs of the 100 by 100 problem reinforces the contention that multitasking has no advantage for small problems.
3. For the 1000 by 1000 case we see decreases in wall clock times in both dedicated and batch mode for those options that multitask well: Libsci linpack routines, Libsci Lapack routines and compilation for multitasking after inlining.
4. IMSL and MATLAB do contain some multitasking capability but it is not as good as the other methods provided by CRI.
5. MatLAB and the Linpack and Blas solution behave similarly on the large problem, but MATLAB shows less performance on the smaller problem because of a larger overhead.

Table 5. Results(mflop/s) for 1000x1000 Solution in dedicated mode

Number of CPUs	1	2	4	8
cf77 -Zp	135	135	135	135
Libsci Linpack + Blas routines	131	251	442	647
cf77 -Zp -Wd-68	275	472	750	886
Libsci Lapack + Blas routines	300	555	938	1285
IMSL	166	238	302	306
Matlab	126	232	391	535

Table 6. Results(mflop/s) for 1000x1000 Solution in batch mode

Number of CPUs	1	2	4	8
cf77 -Zp	91	89	93	96
Libsci Linpack + Blas routines	92	177	324	529
cf77 -Zp -Wd-68	172	315	630	500
Libsci Lapack + Blas routines	261	442	647	781
IMSL	148	200	217	230
Matlab	86	134	312	475

Table 7. 100x100 Results(mflop/s) for Solution in dedicated mode

Number of CPUs	1	2	4	4
cf77 -Zp	32	32	32	32
Libsci Linpack + Blas routines	84	105	120	142
cf77 -Zp -Wd-68	151	151	151	151
Libsci Lapack + Blas routines	157	148	148	148
IMSL	23	24	24	24
Matlab	57	60	61	61

Table 8. Results(mflop/s) for 100x100 Solution in batch mode

Number of CPUs	1	2	4	8
cf77 -Zp	26	26	27	26
Libsci Linpack + Blas routines	67	91	89	65
cf77 -Zp -Wd-68	122	118	127	112
Libsci Lapack + Blas routines	126	107	110	119
IMSL	17	20	20	20
Matlab	35	41	48	42

References

1. Abaqus Introduction to ABAQUS/Standard, Hibbit, Karlsson & Sorensen Inc., Pawtucket, RI,1994.
2. Abaqus Version 5.3 Timing Runs, Hibbit, Karlsson & Sorensen Inc., Pawtucket, Rhode Island, July 26, 1994.
3. MATLAB User's Guide, The MATH WORKS Inc., Natick, Massechusetts,1992.
4. Linpack User's Guide, JJ. Dongarra, C.B. Moler, J.R. Bunch, G.W. Stewart, SIAM, Philadelphia 1979.
5. IMSL User's Manual, Visual Numerics Inc., 1994.
6. Jack Dongarra. Performance of Various Computers Using Standard Linear Equations Software. Computer Science Department University of Tennessee, Report CS-89-85.
7. Neil Storer. I/O Optimization Under UNICOS. Proceedings of the Cray User's Group Meeting, Spring, 1994.
8. Jeff Zais and John Bauer, I/O Improvements in a Production Environment. Proceedings of the Cray User's Group Meeting, Spring, 1994.