Checkpoint Restart Testing

Alan Scheinine
Lockheed-Martin, ERDC-ITL-MS

May 28, 2010

Summary

Programs built with Checkpoint Restart (CPR) and actually checkpointed and restarted, gave numerical results identical to programs not built for checkpointing. The impact with regard to performance is minimal, less than three tenths of one percent on average, except for OOCORE which had large fluctuations in execution time even without checkpointing.

Two important bugs that were not resolved as of the end of testing for this report are that (1) a job does not go into a hold state if the time needed to write the checkpoint dataset is greater than five minutes and (2) for any given job, the `qhold` command does not work after the first usage of `qhold` and `qrls`.

For four large-scale applications, the sizes of the checkpoint datasets and the write times were tabulated on onyx and can be used to extrapolate the impact of larger jobs on a larger machine.

The use of automatic checkpointing at regular intervals was tested. Since this feature is intended to protect long-running jobs from machine failure, further tests will need to be done with actual machine failures (on onyx) to demonstrate that the scripts and procedures are correct.

Contents

1 Introduction

Checkpoint Restart (CPR) was tested on the machine *onyx* [1], whose configuration is nearly identical to the Cray XT4 machine *jade* [2]. The goals of the testing were

- to assure that the numerical results would be the same when checkpointing is used (Sec. 3.1),
- to ascertain whether there would be a performance penalty for linking with CPR capability (Sec. 3.2),
- to determine if there are limitations such as the size of the checkpoint files (Sec. 3.3),
- and to determine if there were any problems, such as cleaning-up checkpoint files, or cleaning up PBS processes or problems with the underlying ALPS queue system if CPR should enter into an error state (Sec.s 3.4, 3.5 and 3.6).

1.1 Description of Tests

This is a brief description of the types of tests. Later sections will provide more details of the test results.

Three Sustained Systems Performance (SSP) codes were used for testing (CTH [3], OOCORE [4] & HYCOM [5]) using two different process counts for each application. In addition, the application WRF [6] was used as a test code. Also, a small program was used for initial testing.

For each test case (application and dataset size), three types of tests were run.

- The first was to build and run a test case without loading the CPR module (blcr).
- The second type of test was to run an executable linked for CPR without actually checkpointing, in order to determine the performance by comparing the run time with that of the first type of test.
- The third type of test was to stop and start again a running program several times "by hand" using `qhold` and `qrls`.

In addition, for one application code

- A fourth type of test was done concerning the use of automatic checkpointing at fixed intervals of time.

User programs often utilize libraries that have been compiled without loading the checkpointing module. The WRF program was chosen to test that the checkpointing module was only needed during linking. Specifically, WRF was linked with a NetCDF library and a HDF5 library that

were intentionally not compiled with checkpointing. The purpose of this test was to confirm that specializing an executable for CPR only requires loading the blcr module during compilation of the main program and linking.

In addition to the large applications, a small program was created with a separate file for MPI calls and a separate file for the writing of a file by each process. The object code from the MPI file and the file-writing file were created without loading the blcr module, whereas the main program was compiled and all object files linked after the blcr module had been loaded. The results of the small program were rigorously inspected. The successful run of this program and of WRF indicates that previously compiled libraries, such as libraries built by the PETTT ACE Team [1], can be used with checkpointing. See Note (1) for more comments concerning linking with libraries that were not compiled with the blcr module.

1.2 Description of Programs

Initial tests of CPR were done with a small, simple test program (named smalltest) that used both MPI and writing of files, and had a thorough test of the results. Four large-scale real-world parallel programs were used for testing. Three of these programs (CTH [3], OOCORE [4] & HYCOM [5]) were from the Sustained Systems Performance (SSP) Benchmark Suite, in particular, the versions used in the Department of Defense TI-07 Applications Benchmark Test Suite. For the SSP codes, testing was done using two different process counts: 64 & 288 processes for CTH and OOCORE and, 59 & 256 processes for HYCOM. (The maximum number of processes available on onyx is 312.) For the SSP codes, the two processor counts correspond to the "standard" and "large" cases. An additional application, WRF [6] (version 3.1.1), was used for testing the use of libraries (NetCDF and HDF5) not compiled with the blcr module. WRF was run using 256 processes.

2 Compiling and Running Batch Jobs with CPR

Cray technical support provided the following information.

> 2.3.1 Requirements and/or Limitations for Checkpoint/Restart
>
> Specific third-party batch system software releases are required for CPR support. For more information, access the 3rd Party Batch SW link on the CrayPort website at crayport.cray.com. For information about accessing CrayPort, see CrayPort.
>
> CPR applications on Cray XT systems require a library that has integrated BLCR support; for that reason, applications must be linked with the Cray MPT 3.0.1 release or later libraries. Thus, only applications using the MPI and SHMEM programming models are checkpointable.
>
> For performance monitoring of applications that may be checkpointed and restarted, CrayPat 4.4 release is also required.

[1] User Productivity Enhancement, Technology Transfer and Training (PETTT) Program, Advanced Computational Environment (ACE) Team

I was unable to find this text on the Cray documentation website (`http://docs.cray.com/`). Moreover, onyx did not have manual pages for either "`cpr`" or "`blcr`". The command "`man qsub`" has some information on CPR.

All of the libraries compiled for users on jade have been compiled under a release greater than Cray MPT 3.0.1, so should be compatible with CPR. See Note (2) for more comments concerning the documentation.

The user's program should be built after loading the blcr module, for example,

```
module load blcr/0.7.3
```

To submit to PBS a program that can be checkpointed, the `LD_PRELOAD` environment variable must be passed to qsub. For example,

```
qsub -v LD_PRELOAD=/usr/lib64/libcr_run.so.0 batch-script.pbs
```

It can be assumed that after loading the module blcr, the environment variable `CPR_ENV` will be defined as the variable list shown above so the job submit command can be written

```
qsub -v $CPR_ENV batch-script.pbs
```

The value of `LD_PRELOAD` can be found using the command

```
/usr/bin/cr_run env | grep LD_PRELOAD
```

See Note (3) for more details concerning `CPR_ENV`.

The PBS batch script does not need to be modified, in particular, the `aprun` command does not need to be modified.

If two environment variables need to be passed, then a comma-separated list should be used as shown below.

```
qsub -v ${CPR_ENV},SSPROOT=${WORKDIR}/CTH/wtCPR1/ssp_test cth_large_0288.bat
```

The documentation shown by executing the command "`man qsub`" shows a space after the each comma in the variable list after the "`-v`" but the space after the comma results in a syntax error.

All CPR tests ran correctly when `LD_PRELOAD` was defined as one specific library. However, there may be other Cray libraries that need to be loaded for other applications, in which case utilization with CPR would require a more complex `LD_PRELOAD`. Submitting a job when `LD_PRELOAD` is defined will result in a job that can be placed on hold using the command

```
qhold <jobid>
```

and can be restarted using

```
qrls <jobid>
```

In order to have a job that is automatically checkpointed at fixed intervals during the run, the job submission command is either

```
qsub -v $CPR_ENV -c c=<minutes of CPU time> batch-script.pbs
```

or

```
qsub -v $CPR_ENV -c w=<minutes of wallclock time> batch-script.pbs
```

where <minutes . . . > is an integer describing the time interval in minutes of CPU time or wallclock time. Automatic checkpointing is intended to deal with the event of a machine failure, but is not intended to deal with a program failure such as a segmentation fault. See Sec. 3.5 for more discussion concerning automatic checkpointing.

3 Safety of CPR

The safety of CPR has several facets. In particular, the numerical results must be correct, the impact on the computational system and filesystem must be modest, and errors should not disable nodes or disable the batch queue system.

3.1 Consistent Results

Numerical comparisons show that checkpointing does not have an affect on the numerical results.

3.1.1 smalltest

For a number of processes on the order of 32, the process of rank 0 begins a round-robin cycle by using `MPI_Send()` to send the integer 1 to the the process of rank 1. Each process, in rank order (modulo the number of processes) uses `MPI_Recv()` to receive an integer from the lower-ranked process, increments the value by one then sends the integer to the next process, On the order of 262144 cycles are done. This can repeated 16 or more times, beginning again at 1 to avoid integer overflow. In addition, as each process does a receive–increment–send, the integer value received is written to a file, one file per process. A Perl script was written to read the files to check the accuracy. No errors were found when `qhold` and `qrls` were used to pause the job and write checkpoint files.

This test was also used to show that object files that might be contained in a library did not need to be compiled with the blcr module loaded. In particular, the use of `MPI_Send()` and `MPI_Recv()` was in a file separate from the main program and the writing of the integer value was in another file separate from the main program. These two files were compiled without loading the blcr module, then the main program was compiled and the executable linked after the blcr module was loaded.

3.1.2 CTH

For both the “large” and “standard” cases, for the three types of runs (specifically, the run with the executable compiled without CPR, the run with the executable compiled with CPR but no actual checkpointing, and the run checkpointed using `qhold` and `qrls`) the standard SSP validity test declared the results to be accurate. In addition, the files `ogcth.o`<jobid> were compared and only showed differences related to execution times.

3.1.3 OOCORE

For the same six kinds of tests described in the previous section, the validity checking script (`check_oocore_output.pl`) supplied with the SSP distribution found all the results to be valid.

3.1.4 HYCOM

For the same six kinds of tests described in the previous section, the validity checking scripts (`ti-07_hycom_standard_autocheck.pl` and `ti-07_hycom_large_autocheck.pl`) supplied with the SSP distribution found all the results to be valid.

In addition, the output files `summary_out.o`<jobid> were compared and found to be identical. (Comparing between 3 "standard" tests and comparing between three "large" tests.)

3.1.5 WRF

The application WRF provides a program diffwrf that compares two output files, showing numerical differences between various fields in terms of RMS, digits of accuracy and other measures. For the particular dataset and parameters used, the WRF output contains 35 fields such as wind velocity, humidity and temperature. For each run diffwrf was used to compare the output to a standard canonical program output for the same data. The results of this comparison were identical (the diff of the outputs of diffwrf) between the run with the executable compiled without CPR, the run with the executable compiled with CPR but no actual checkpointing, and the run checkpointed using `qhold` and `qrls`.

3.2 Execution Time

The increase in run time for an executable capable of checkpointing was found to be negligible, around $\frac{3}{10}^{th}$ of one percent, except for OOCORE which uses a large amount of file I/O, as shown in Table 1. For the "Linked with CPR" tests, as well as linking when the blcr module was loaded, before launching the job the environment variable
`LD_PRELOAD=/usr/lib64/libcr_run.so.0` was exported in order to force utilization of CPR-enabled routines.

The increase in execution time for OOCORE when linking with CPR is not definitive because the execution times were highly variable even though there were no other users on onyx. While running OOCORE, commands on the login node onyx01 became slow even though the CPUs of onyx01 were not used in the run. This suggests that the Lustre file system was stressed by OOCORE. As we can see from the table, for OOCORE the large variation in wallclock times also occured for the "plain" version without CPR.

The standard deviations shown are based on two runs for each case. Not all cases were done twice.

Application	Num. Procs.	Without CPR	Linked with CPR	% Increase
CTH	64	2343	2340	-0.1
CTH	288	4074.5 ± 1.5	4081.5 ± 2.5	0.2
OOCORE	64	1854	1867	0.7
OOCORE	288	2517 ± 210	2663 ± 123	5.8 ± 6.6
HYCOM	59	1786	1791	0.3
HYCOM	256	2736	2745	0.3
WRF	256	1775	1770	-0.3

Table 1: **Run times, without and with CPR.** The run times, in seconds, for executables linked without CPR compared to executables linked with CPR and run with the environment variable `LD_PRELOAD=/usr/lib64/libcr_run.so.0`. The executables linked with CPR were not actually checkpointed during the runs used to prepare this table, since the purpose was to measure the performance hit caused by linking with the BLCR library. The cases of CTH and OOCORE with 288 processes were done twice, so a standard deviation is shown. The increase of execution time when linked for CPR averaged around $\frac{3}{10}^{th}$ of one percent, except for OOCORE which uses a large amount of file I/O.

3.3 Amount of Disk Space

The total size of the dataset needed for checkpointing was determined for four applications. In addition, the time needed to write the files was measured. CPR is configured so that the checkpoint dataset is written to the directory `/work/PBS/cpr/<jobid>.pbs.CK`. The command "`du -sm *`" was used to determine the dataset size. During early tests it was noted that execution does not pause if the time needed for writing the checkpoint dataset exceeds five minutes. Though attempts were made by the ERDC Cray Support Staff to solve this problem, during the final week of testing the problem remained. In Table 2 the dataset size is shown in parenthesis when the write time was greater than five minutes, since it is not known whether the size would be the same if CPR worked correctly. The table entry "> 300" denotes CPR failure because the job continued to run rather than going into the "hold" state. For jobs that were successfully put into the "hold" state (`qhold` <jobid>) then released (`qrls` <jobid>), and also for jobs that continued to run despite use of the `qhold` command, the program results were correct when the run finished. Moreover, for every test, whether successful or not, the directory containing the checkpoint dataset was automatically deleted when the job finished. Table 2 is correct in showing that the case of CTH with 64 processes required more time for writing the checkpoint dataset than CTH with 288 processes.

Application	Num. Procs.	Checkpoint Write Time (secs)	Checkpoint Dataset Size (MB)
CTH	64	> 300	(26663)
CTH	288	167	55041
OOCORE	64	125	43351
OOCORE	288	122	34955
HYCOM	59	80	26239
HYCOM	256	283	93783
WRF	256	75	24205

Table 2: **Checkpointing Time and Dataset Size.** This table can be used for projecting the amount of disk space needed for a large job on a machine such as jade. On jade each node has 8 GBytes of memory and 4 cores per node, so for a program that fully utilizes memory, if it runs on 1000 cores, then 2 TeraBytes of memory would be needed for the checkpoint dataset. In practice, the worst case for these tests was HYCOM for which the total size of the checkpoint files for a run with 1000 cores can be projected to be 0.37 TeraBytes.

3.4 Checkpointing with qhold and qrls

Two problems were identified when `qhold` and `qrls` was tested. First, a job would not enter the hold state when `qhold` was used if the time needed to write a checkpoint dataset was greater than five minutes. Second, `qhold` and `qrls` could only be used once, the second time `qhold` was used the job did not enter the hold state.

3.4.1 Checkpoint Write Time Limit of 5 Minutes

Early on, during testing it was discovered that if the time needed to write the checkpoint dataset was more than five minutes then the `qhold` command would return an error message and the job did not enter the hold state "H" but rather continued running to completion. The ERDC Cray Support Staff was informed of the problem. They increased the default timeout limit for the checkpoint action. However, the problem continued to occur. The ERDC Cray Support Staff was informed. Evidently there is another parameter that needs to be changed. In these cases the jobs gave correct results and the checkpoint datasets were cleaned-up automatically at the end of the jobs, the jobs just did not go into a hold state.

During the last week of testing, `qhold` command showed a different behavior. The `qhold` command never returned a prompt. The CPR files were written but the jobs never went into a hold state, but rather ran to completion.

3.4.2 Second qhold Has No Effect

Normally, the command `qhold` <jobid> causes the job state to be written into a directory `/work/PBS/cpr/`<jobid>`.pbs.CK` and causes the job enter the hold state — shown as "H" in the queue status. The `qhold` command does not return until the checkpoint dataset has been written. The next step, of course, was to use the `qrls` command to restart the job, which worked fine. It is noteworthy that the `qrls` command returns immediately and the queue status shows that the job is in the run state "R" immediately, though one would certainly expect that some time would be needed to restart a job. For these tests, after the command `qrls`, the second use of `qhold` was done after waiting a length of time at least a minute longer than the time needed to write the checkpoint dataset. The second `qhold` command always failed. A typical error message is shown below.

```
scheinin:onyx01% qhold 5670
qhold: Request invalid for state of job 5670.pbs
```

In nearly all cases, the final results were correct and the checkpoint dataset was deleted automatically at the end of the job. With regard to the checkpoint datasets, the typical directory structure after the second `qhold` failed is shown below.

```
scheinin:onyx01%  du -sm * ; date
1       5670.pbs.CK
26195   5670.pbs.CK.old
1       cprerror.log
```

The checkpointing procedure begins to write the second checkpoint dataset but fails shortly after starting. Of course, the ERDC Cray Support Staff was informed of all problems as soon as they were discovered.

3.5 Checkpointing at Fixed Time Intervals

Tests using the small program showed that checkpointing at regular intervals ran successfully, a new checkpoint file was written every 3 minutes when the option "`-c w=3`" was specified. Moreover, though a second checkpointing by hand (a second use of `qhold` on a running job) was found to fail, the automatic checkpointing wrote a new checkpoint file every three minutes, not only after the first three-minute interval.

Automatic checkpointing is specifically for protecting a program from a machine failure. In contrast, if a user program exits due to an error (such as dividing by zero or attempting an invalid memory access) then the PBS job will end and the checkpoint files will be deleted. The use by a user of checkpoint files would require more study and may not be practical. Comments by Marty Bullock (ERDC Cray Support Staff) describe the situation with great clarity.

> The difficulty in providing a simple tool or method for this is in the implementation. At the system level, it requires privileged accounts in order to be able to read all of the node memory and write it to disk. In doing so, the location of the data and ability to manage resources within the systems internal scheduler and the WMS also require privileged account access. The result is that the configuration of the CPR capabilities must be tightly controlled and somewhat limited in scope.
>
> Since PBS simply hands off most of the work to a custom "handler" program or script, it may be possible to eventually adapt the process to more user-friendly configurations.

It should be emphasized that checkpointing using blcr has the drawback of creating very large checkpoint datasets, which may cause degradation of network bandwidth and filesystem bandwidth for all users. For example, on jade where each node has 8 GBytes of memory and 4 cores per node, for a program that fully utilizes memory, if it runs on 1000 cores, then 2 TeraBytes of memory would be needed for the checkpoint files. As described in Table 2, in practice the worst case with regard to checkpoint dataset size for these tests (HYCOM) can be used to extrapolate the need for 0.37 TeraBytes for a job of 1000 cores. Whenever possible, it would be preferable for an application to have, as part of the code, a checkpointing procedure that writes all relevant data at intervals that are natural for the algorithm, rather than capturing the entire program state at arbitrary times. Of course, even when the application does the checkpointing based on a procedure specific to the code, it is possible that very large files will need to be written. Many programs already do write "restart" datasets at regular intervals.

More testing will be needed in order to be certain that checkpointing at regular intervals will serve the intended purpose. In particular, a machine can fail in many ways. If a machine fails due to a Lustre filesystem error, then the checkpoint files may be corrupted. Moreover, though the old checkpoint directory is not erased until the new directory has been written, the machine restart procedure will need to deal with the case that the machine failure occured while the checkpoint

directory was being written. In addition, machine shutdown and startup scripts must be written **and tested** for dealing with the checkpoint files. For example, the shutdown of PBS in anticipation of the shutdown of a machine due to the failure of a major component must not invoke a standard epilog that erases the checkpoint files. But also, unused checkpoint files will eventually need to be deleted when the machine is restarted but not all jobs are able to run. Similarly, the startup procedure needs to restart the checkpointed jobs. The startup procedure may be complicated if the machine shutdown was abrupt rather than a clean shutdown. Testing will require shutting-down and restarting onyx and taking into consideration various modes of failure.

These considerations suggest that checkpointing is most appropriate for smaller, longer running jobs. While on the subject of heavy filesystem usage for checkpointing, aside from checkpointing automatically at fixed intervals, checkpointing by a system administrator (utilizing `qhold`) could be used when it is necessary to stop a job temporarily in order to push through a higher priority job. In this case as well, longer running jobs that are relatively small would be the best candidates for checkpointing.

3.6 Remaining Problems

During initial testing of CPR, the job queue system would often fail, for example, due to errors in the usage of CPR. Typically, the commands `qstat` or `qview` would show that all cores were available but any job submitted would remain in a wait state (status "Q"). This may have been a problem with the underlying ALPS queue system.

During this testing phase Marty Bullock worked on the checkpoint scripts to ensure that they didn't exit prematurely and completed the ALPS cleanup routine. Moreover, he added a cron job to look for "orphaned" job reservations in ALPS that had no associated PBS job. As he explained, "In the latter case, the cron job will cancel the leftover reservations and release the resources."

Evidence of a more robust CPR system is the following. Initially, using `qhold` or another CPR command on a non-CPR job (not linked with the blcr module) would crash the queue system — or at least incorrectly remove nodes from the queue system. A test near the end of this testing period using `qhold` on a non-CPR job showed no impact on the queue system. More tests concerning the misuse of CPR will be needed in order to be certain that it is ready to be used on a production machine.

4 Conclusions

The use of Checkpoint Restart (CPR) on the machine onyx showed that the use of checkpointing had absolutely no effect on the numerical results for the four large applications used for testing. In addition, the execution time was increased by a negligible amount.

On the other hand, the CPR procedure has some serious bugs that need to be resolved. Specifically, checkpointing does not occur if the time needed to write the checkpoint dataset is greater than five minutes. Also, the use of the command `qhold` to put a job into a hold state only works once on any given job.

When these basic problems are fixed, additional testing will be needed to assure that erroneous

use of CPR does not damage the queue system. For example, during the initial month of testing, erroneous use of checkpointing caused the batch queue to fail. In particular, the allocated nodes would remain allocated so that a new job could not start. It would be easy for a user to make the mistake of trying to checkpoint a job that was not built for checkpointing, so one would want such a mistake to have no deleterious effect on the queue system. During this time period of testing, the queue system was modified by Marty Bullock so that it became robust against this type of error. Nonetheless, further testing of incorrect usage will be needed in order to be certain that CPR can be put onto a production machine.

Also, if automatic checkpointing is to be used for protecting long-running jobs from a machine failure, then the procedure will need to be testing using events that replicate typical modes of machine failure.

One must also consider that impact on the Lustre file system of the writing of large checkpoint datasets.

5 Notes

(1) All of the large applications used for testing have a single command for building the executable. Before building the executable, the blcr module was loaded. Doing so does not resolve the question of whether libraries previously compiled without module blcr can be used with CPR. The question was resolved by tests with WRF and the small test program.

Clearly, *executables* already built without blcr, for example executables built by the PETTT ACE Team, will need to be built again if checkpointing capability is desired. The question that requires testing is whether *libraries* built without blcr, for example libraries already built by the PETTT ACE Team, can be linked to programs built with blcr. A Cray technical representative assured me that the libraries already built can be linked to create checkpoint-able executables. This was confirmed by testing with WRF and the small test program.

With regard to WRF, though one command launches both compiling and linking, WRF requires the external libraries HDF5 and NetCDF. The two libraries were build *without* blcr then WRF was built after loading blcr. With regard to the small test program, it was intentionally dividing into three files: a main program, a routine dedicated to writing to a file, a routine dedicated to MPI send and receive. The latter two routines were compile *without* having loaded the blcr module. Then the main program was compiled and the object files linked to form an executable after loading the blcr module. All programs, including WRF and the small program, gave correct results.

(2) The BLCR documentation from Berkeley Lab is based on dynamic loading of shared libraries. In contrast, a Cray XT executable is statically linked. However, it appears that the statically linked executable has an undefined symbol that causes the appropriate blcr routines to be added at run time. Though the Berkeley and Cray methods are similar, the differences result in differences usage and differences in documentation.

The Cray Linux Environment (CLE) 3.0 Software Release Overview recommends the reference

`Checkpoint/Restart (BLCR)`

```
Documentation for BLCR is available from Berkeley Lab at
http://upc-bugs.lbl.gov/blcr/doc/html/
```

However, a Cray system administrator explained, "This is not an out-of-the-box implementation of BLCR. There are significant modifications in ALPS and MPT to enable the support and functionality on XT systems." For questions I had about CPR documentation or about features not well documented, I would like to thank Alan Minga and Marty Bullock for their help. It would be desirable to have detailed documentation from Cray.

(3) During testing of CPR on onyx, the value of `CPR_ENV` was not initially defined, then later the system administrator defined it. But then blcr was updated and `CPR_ENV` was left undefined. When reminded, the system administrator again defined `CPR_ENV`. If `CPR_ENV` is not defined when the blcr module is loaded, then the user can have in his or her batch script

```
CPR_ENV=`/usr/bin/cr_run env | grep LD_PRELOAD`

qsub -v $CPR_ENV smalltoo.pbs
```

If `CPR_ENV` is defined, then the user cannot use the `grep` command shown above to define it because the result would be to set `CPR_ENV` to the pair of lines

```
LD_PRELOAD=/usr/lib64/libcr_run.so.0
CPR_ENV=LD_PRELOAD=/usr/lib64/libcr_run.so.0
```

The point is that defining `CPR_ENV` is an additional system administration step that is not automatic as part of the installation of blcr. To summarize what is described above, a simple one-line method of defining `CPR_ENV` in a user's script (that is independent of the version of blcr) becomes invalid if `CPR_ENV` is already defined. So a consistent policy would be to never define `CPR_ENV` when module blcr is loaded or always define `CPR_ENV` when module blcr is loaded. The main body of the text of this report assumes that `CPR_ENV` is always defined when module blcr is loaded because this policy simplifies the user's scripts.

Acknowledgements

I would like to thank the following persons. Christopher Borchert for coordinating the work. Robert Alter for supplying the SSP codes. Alan Minga of Cray who provided technical information. Jeremy Duckworth of the ERDC Cray Support Staff for help with using the machine onyx. Most of all, I want to thank Marty Bullock of the ERDC Cray Support Staff for responding to the various problems that I have encountered when testing CPR and for providing technical information.

References

[1] ONXY: Cray XT4, 78 compute nodes, 4 cores/node (2.1 GHz AMD Opteron Quad-Core), 8 GB memory/node; O/S xt-os/2.2.41A_PS05, parallel environment xt-mpt/3.5.0, compiler pgi/9.0.4, CPR blcr/0.7.3.

[2] JADE: Cray XT4, 2146 compute nodes, 4 cores/node (2.1 GHz AMD Opteron Quad-Core), 8 GB memory/node; O/S xt-os/2.2.41A_PS05, parallel environment xt-mpt/3.5.0, compiler pgi/9.0.4.

[3] CTH is a code used to study the effects of strong shock waves on a variety of materials using many different models. The software set CTH was developed by SANDIA under the contract with DOE.

[4] Out-Of-Core Solver (OOCORE) is a code developed by the ScaLAPACK project, a collaborative effort in producing high-performance linear algebra routines for distributed-memory message-passing MIMD computers and networks of workstations.

[5] The HYbrid Coordinate Ocean Model (HYCOM), is a primitive equation ocean general circulation model developed in a collaborative effort between the University of Miami, the Los Alamos National Laboratory, and the Naval Research Laboratory.

[6] The Weather Research and Forecast model (WRF) is a public domain code. WRF was developed at the National Center for Atmospheric Research (NCAR) which is operated by the University Corporation for Atmospheric Research (UCAR).