Attachment D

Definition: Climate Modeling and Research System Benchmark Instructions

Solicitation Number 6400009257

Table of Contents

1. BENCHMARKS 3
1.1. CONVENTIONS USED IN THE CMRS BENCHMARK INSTRUCTIONS 3
1.2. BENCHMARK AVAILABILITY 3
1.3. BENCHMARK OVERVIEW 4
1.3.1. Benchmark Purpose 4
1.3.2. Benchmark Summary 4
1.4. CONDITIONS AFFECTING THE BENCHMARKS 4
1.4.1. Restrictions on Use of the Benchmark Suite 4
1.4.2. Storage and File System Conditions During the Benchmark Execution 4
1.5. CODE CHANGES AND AUDIT TRAILS 5
1.5.1. General Requirements 5
1.5.2. Code Changes to Achieve a Baseline 5
1.5.3. Code Changes to Achieve Optimized Results 6
1.6. PERFORMANCE DATA 6
1.6.1. Test System Equivalence to Proposed Solution 6
1.6.2. Use of Hardware Multithreading Features 6
1.6.3. Use of Hardware Undersubscription 6
1.7. MATERIALS RETURNED WITH THE SOLICITATION 6
2. SCALING BENCHMARK 7
2.1. GENERAL COMMENTS 7
2.2. RUNNING THE SCALING STUDY 8
2.3. CM-CHEM AND CM2-HR 8
2.3.1. General Information 8
2.3.2. Model Verification 9
2.3.3. Model Reproducibility 10
2.4. CAM/WACCM 10
2.4.1. Procedure 11
2.5. GFS 18
2.5.1. Model Reproducibilty and Validation 18
2.5.2. Parallelization of GFS 18
2.6. FLOW-FOLLOWING FINITE-VOLUME ICOSAHEDRAL MODEL (FIM) 19
2.6.1. Building FIM 19
2.6.2. Running FIM 20
2.6.3. Verification Procedure 20
2.6.4. FIM runtimes 21

3. THROUGHPUT BENCHMARK 21
3.1. GENERAL COMMENTS 21
3.2. THROUGHPUT BENCHMARK SCORING 23
3.3. THROUGHPUT BENCHMARK OUTPUT 23
4. I/O BENCHMARKS 24
4.1. METADATA OPERATIONS 24
4.2. FAST SCRATCH BENCHMARK 24
4.3. LONG TERM FAST SCRATCH BENCHMARK (LTFSB) 26
5. EPCC OPENMP MICROBENCHMARKS 26

1. Benchmarks

1.1. Conventions used in the CMRS Benchmark Instructions

The following conventions are used in the CMRS Benchmark Instructions. These conventions describe specific responsibilities of the Offeror as part of their response. The Offeror should take care to ensure that all items marked Required are properly addressed. Offeror proposals are encouraged to address Desired and Optional items as appropriate to ensure that those contributions to the benchmark results are well understood.

Attribute Priority	Description
Required [R]	The elements of the benchmark results that form the minimum baseline for the evaluation of the benchmark, and must be reported. The Offeror shall fully describe any exceptions to the reporting of items considered required. Benchmark Instruction Sections that include these items are marked with [R].
Desired [D]	The elements of the benchmark results that provide information above and beyond the baseline results that will assist in more fully describing the performance capabilities of the proposed CMRS. The Offeror is encouraged to provide as many such items as possible. Benchmark Instruction Sections that include these items are marked with [D].
Optional [O]	The elements of the benchmark results that provide information above and beyond the information contained in the baseline results and desired item reporting. The Offeror is encouraged to provide as many such items as possible. Benchmark Instruction Sections that include these items are marked with [O].
Information [I]	Items that are provided as additional information to an Offeror regarding information about the benchmark suite that may aid an Offeror in determining the most suitable system configuration. Benchmark Instruction Sections that include these items are marked with [I].

Figure 1. Conventions used in the CMRS Benchmark Instructions

1.2. Benchmark Availability

For the purposes of this Solicitation, ORNL has assembled a series of application codes and data sets that will comprise a benchmark suite. Due to its total size, this benchmark suite cannot be distributed by typical electronic content distribution methods (web, ftp, or similar). Requests for the benchmark suite must be directed in writing to the ORNL Subcontracts Administrator. For the duration of this Solicitation, the ORNL Subcontracts Administrator may be reached via email at hpcnoaaadm@ornl.gov. [I]

1.3. Benchmark Overview

1.3.1. Benchmark Purpose

The purpose of the benchmark suite is to establish the performance of the proposed Climate Modeling and Research System (CMRS) computer system(s) by executing simulation codes representative of the current and anticipated computational research conducted by NOAA. The system must successfully execute the elements of the benchmark suite at the desired scales and demonstrate accuracy within the limits specified for each separate benchmark code. [I]

1.3.2. Benchmark Summary

The CMRS benchmark suite is composed of 4 parts with the following goals:

1. Application scaling benchmarks for a range of weather and climate applications shall be used to evaluate system performance and resource requirements.

2. A workflow throughput benchmark, based on the GFDL coupled climate model, shall be used to evaluate both the capacity of the proposed system and the performance of the filesystem under load. This benchmark shall determine the maximum number of jobs that can be completed in a fixed wall clock time using the full system.

3. I/O benchmarks shall be used to evaluate filesystem aggregate I/O and metadata performance.

4. The EPCC OpenMP microbenchmarks shall be used to evaluate the performance of OpenMP on the proposed system. These are to be run on a single node.

1.4. Conditions Affecting the Benchmarks

1.4.1. Restrictions on Use of the Benchmark Suite

All material (code, data, scripts, etc.) that is distributed as part of this benchmark suite is the property of the Government. The benchmark codes and related confidential information may only be reproduced or copied by the Offeror for their normal use and analysis in conjunction with the benchmark testing. Any changes made by the Offeror to the benchmark codes shall become the property of the Government. [I]

1.4.2. Storage and File System Conditions During the Benchmark Execution

ORNL is interested in the sustainable performance of the system under typical operating conditions. Therefore, file system(s) supporting the benchmark runs shall be fragmented and filled to at least 60% capacity. Additionally, the throughput benchmark shall be executed with the storage system running in a degraded mode where at least 10% of the LUNS that contain the active file system(s) are being rebuilt. [R]

The Offeror shall describe the method(s) used to replicate these storage and file system conditions for the purposes of the benchmarks. The Offeror shall propose the method(s) by which file system fragmentation, 60% capacity and “degraded mode” shall be achieved for the Acceptance Test for both the FS and the LTFS. [R]

1.5. Code Changes and Audit Trails

1.5.1. General Requirements

The Offeror may make changes to benchmark source code under the following conditions:

1. Changes to scripts are permitted to account for architectural and queueing system differences. [I]

2. Changes to codes must be consistent with relevant standards (ie. ANSI language standards, MPI API standards, etc.). No assembly level recoding of source code is permitted. [I]

3. All source code modifications must be isolated for conditional compilation using pre-processor #if/#endif definitions. As an example,

```
#if (defined CMRS_"OFFEROR")
<modified source code>
#endif
```

   The Offeror shall consistently substitute their company name or initials for the "OFFEROR" keyword. [R]

4. The Offeror shall provide a copy of all modified source code and script files. The Offeror shall document each change, describe the reason for the change, and, as applicable, the impact to the performance associated with that change. [R]

1.5.2. Code Changes to Achieve a Baseline

ORNL is interested in coding styles, application optimization and optimization techniques. However, changes to the benchmark code may or may not produce improvements in the more general set of applications for which the benchmarks are surrogate. Therefore, the Offeror shall create a *baseline* that contains only those code changes that are necessary for the model to execute correctly. The reported results shall be clearly labeled as Baseline. [R]

Changes to the benchmark that are acceptable for the baseline results include:

1. Compiler command lines with performance-specific options including, but not limited to, automatic parallelization.

2. Use of commercially available and supported source pre-processors that are bid as part of the offering.

3. Use of compiler "directives" within the source. [I]

Changes to the benchmark that are generally acceptable for baseline results include:

- Use of commercially supported libraries that are bid as part of the offering. The level of effort required to introduce the library into the general source code base shall be evaluated. [I]

1.5.3. Code Changes to Achieve Optimized Results

The Offeror may submit a second set of benchmark results that makes additional changes beyond those changes needed to produce the Baseline Results. These results shall be documented in the same fashion as the changes necessary to achieve the Baseline Results. Should the Offeror submit this second set of benchmark results, these shall be clearly labeled as Optimized. [D]

1.6. Performance Data

1.6.1. Test System Equivalence to Proposed Solution

As already stated, ORNL requires the Offeror to provide a Baseline Result for the benchmark suite and strongly desires an Optimized Result. To correctly assess the Offeror's benchmark results, additional information is required. [I]

The Test System(s) on which the benchmarks are run and for which performance data is reported shall be as close as possible to the initial offered system. ORNL acknowledges that it may not be possible to use the offered system for the solicitation proposal. Therefore ORNL shall evaluate performance projections based on the characteristics of the test system (i.e. actual test system size, technology equivalence, etc), thoroughness of data gathering, projection methodology and Offeror history. [I]

The Offeror shall clearly mark any results that are projections to proposed systems rather than measured results. [R]

1.6.2. Use of Hardware Multithreading Features

If hardware multithreading technologies such as Simultaneous Multithreading (SMT) or Hyperthreading (HT) are available on the benchmarking system, ORNL would like to see results for the scaling studies with and without this feature. If the Offeror employs hardware multithreading as part of their benchmark efforts, the Offeror shall insert sheets or sections into the benchmark results Microsoft Excel spreadsheet for these hardware multithreaded results, and clearly indicate how many tasks, threads, physical cores and nodes were used for each run. [D]

1.6.3. Use of Hardware Undersubscription

Baseline results for the benchmark applications must use all cores on each socket and all sockets on each node. [R]

Additional results may be provided as part of the reported Optimized Results if improved performance and capacity are achieved in other configurations. [D]

1.7. Materials Returned with the Solicitation

The Offeror shall provide, in tar/gzip format, the source code and scripts used and the requested verification output for all aspects of the benchmark, as described in the benchmark instructions and README files, on ISO-9660 CDROM. All written responses and spreadsheets called for in these sections must be returned with the Solicitation Proposal in printed form and digitally on ISO-9660 CDROM. [R]

The Offeror shall provide a detailed but concise description of the benchmark system and system configuration. The format for this description may mimic and abbreviate the format for the description of the proposed subsystem(s), using the guidelines provided in the Instructions to Offerors. Concise statements where the Test System(s) are equivalent to the proposed systems may be used to eliminate duplicate material. [R]

The Offeror shall provide information on the benchmark system in the Offeror_info worksheet of the Benchmark_Results spreadsheet. This shall include but not be limited to:

- The number of physical and logical processors on the system.
- Processor characteristics, including
  - cycle time
  - socket configuration (number of cores per socket, availability of multi-threading, memory per core)
  - node configuration (number of sockets per node)
  - peak performance,
  - vector length
  - cache configuration
  - total and application memory available to each core, socket, and compute node
  - memory type
- A description of the "communication fabric" of the system
- The hardware and software supporting the file system(s) used for the benchmark
- OS version, user configurable kernel and system parameters [R]

The file "Benchmark_Results.xls" is distributed with the benchmark codes. Timings for all jobs shall be entered into this spreadsheet. [R]

The Offeror shall provide a complete, concise description of the data-gathering procedures, the data gathered, and any extrapolation methodology used. All timings shall be presented in whole units of seconds. Fractional timings that are less than 0.5 shall be rounded "down" to the nearest integer; timings that are greater than or equal to 0.5 shall be rounded "up" to the nearest integer. The Offeror shall report what compiler, compiler version, compiler options, libraries, and other pertinent information were used for each benchmark; the purpose of each option should be reported. [R]

Offerors are not allowed to change the floating-point precision of any of the benchmarks. [R]

With respect to the data describing the Test System, the Offeror shall describe how the proposed CMRS system shall differ from the benchmark Test System. Further, the Offeror shall describe how the data provided and the extrapolations from the Test System show that the installed system shall perform as offered. [R]

2. Scaling Benchmark

2.1. General Comments

The scaling benchmark comprises five applications:

- CM-CHEM
- CM2-HR

- CAM/WACCM
- GFS
- FIM

The goal of the Scaling Study is to measure individual application performance, scaling and resource requirements. Ideally, data for the Scaling Study should be collected using the same Test System used for the Throughput Benchmark. Lacking this consistency, detailed documentation of the system differences shall accompany Scaling Study data. Commentary concerning the scaling and performance implications of the system differences shall be provided as well. [R]

Descriptions of the individual benchmark experiments are provided for each of the benchmark codes. See the README files included with the benchmark source for additional details. [I]

2.2. Running the Scaling Study

Applications shall be run on as few processor cores as practical for the given experiment and shall be scaled to as many cores as possible. At some number of cores the performance improvement of an application with respect to a particular experiment may flatten and perhaps decline (the “rollover” point of the scaling curve). For CM-CHEM, CM2-HR, CAM/WACCM, and GFS, the Offeror shall provide data, documentation and projections as necessary up to and including either the rollover point or the proposed system or sub-system size, whichever is smallest. [R]

The CM-CHEM script with heavy I/O should be run on one of the core counts reported for the CM-CHEM scaling study. [R]

The Offeror shall include one of the scaling study points for CM2-HR at the core count and decomposition that is proposed for the throughput. [R]

For FIM, at least one processor count should have a run time of 1800 seconds or less for the FIM model portion of the job. [R]

In order to obtain a reasonable understanding of the scaling curve, the Offeror shall provide a minimum of 4 data points for each experiment. Note that the choice of core counts shall show the range of performance. [R]

Results of the scaling study must be entered into the appropriate sections of the benchmark_results.xls spreadsheet. [R]

2.3. CM-CHEM and CM2-HR

2.3.1. General Information

CM-CHEM is a coupled global model with a 2-degree, 48-level atmosphere with numerous chemical species and a 1-degree ocean. It represents near horizon climate research efforts. Given the resolution of the atmosphere and ocean models, the scalability of this application is currently limited to approximately 900 cores on the Cray XT systems located at ORNL. [I]

CM2-HR is much higher resolution in both atmosphere and ocean. The model is constructed from 0.5-degree atmosphere and a 0.25-degree ocean. In a time frame exceeding the time line for the system procurement, the target for this model is a 0.25 degree atmosphere and a 0.1-degree ocean. This model is currently under construction and does not yet contain the "physics" of WF1. While it is believed that

ultimately this job will scale to 8000 or more cores, the current early implementation (0.5 degree atmosphere + 0.25 degree ocean) appears to be limited to around 2100 cores on the Cray XT systems located at ORNL. The scaling limitation is believed to be related to the boundary layer exchange model grid. Work is ongoing to alleviate this scaling bottleneck but will not be available for the benchmark. [I]

CM-CHEM and CM2-HR are built on the NOAA/GFDL Flexible Modeling System (FMS). Both models use a cubed-sphere grid for the atmosphere and a tri-polar grid for the ocean. Both models break the total number of cores allocated to the application into 2 disjoint sets: a set running the atmosphere and land and a set running the ocean and sea ice. The sets "join" to perform the boundary layer calculations. [I]

Both models are scalable in "discrete units" only. [I]

Core counts for the atmosphere set are defined by 6*M*N where M and N are integers. As currently run in production, M is one of {N, N+1, N-1} (i.e. the sub-domain decomposition is either "square" or "slightly rectangular"). Other configurations with M and N as integers may also be possible. [I]

Core counts for the ocean set can be more flexible. Experience has demonstrated that with most compilers, a completely malleable executable performs less well than an executable compiled such that almost all array sizes are known at compile time (the latter being known as a "static memory size" executable). Decompositions for "static memory" executables are limited to core counts which produce the same number of sub-domain points on every process. An equivalent way to define this is that the number of cores used for each of the "X" and "Y" directions must divide the global grid evenly in each direction where X*Y = the total number of cores devoted to the ocean component. [I]

Finally since all processes must synchronize to perform the boundary layer calculations as a single group, load balance considerations limit the number of atmosphere/ocean decompositions that can be used together effectively. [I]

The model contains functions that report the Total runtime, Initialization, Main loop and Termination timing in terms of the minimum process time (tmin), the maxium process time (tmax) and the average process time. The Offeror shall report the maximum process time (tmax) for the Total Runtime, Initialization, Main loop and Termination for each study instance in Benchmark_Results.xls. [R]

The model writes to stdout (standard out - i.e. the "screen"). This information shall be captured (such as by piping to a file) and returned for all model instances. Only ASCII output shall be returned. [R]

ORNL requires that one of the scaling study points for CM2-HR shall be provided at the core count and decomposition proposed for the throughput. [R]

2.3.2. Model Verification

The scripts CM-CHEM-verification and CM-HR-verification are set up to run 2-day simulations and print information needed for verification. These should be run for both the reproducible executable and the higher optimization level by changing the value of “set executable” in the script. [R]

2.3.3. Model Reproducibility

The output from CM-CHEM and CM2-HR (i.e. the history and restart files) is bitwise "reproducible" as defined below. ORNL requires that there exist one or more sets of compilation flags for which each of these reproducibility characteristics can be maintained. [R]

2.3.3.1. Reproducibility Across Core Count and Problem Decomposition.

In this mode, the history and restart files resulting from a given model segment bitwise reproduce the results from the same code compiled with the same compiler version and settings and run on the same type of hardware (i.e. compute nodes and communication infrastructure) with the same runtime libraries regardless of run core count or decomposition. In addition to possible compiler settings necessary to produce this behavior, the model must be run with the xgrid_nml fortran namelist variable set to: make_exchange_reproduce=.true. [R]

2.3.3.2. Absolute Reproducibility at the Same Core Count.

In this mode, the history and restart files resulting from a given model segment bitwise reproduce the results from the same code compiled with the same compiler version and settings and run on the same type of hardware (i.e. compute nodes and communication infrastructure) with the same runtime libraries. Executables compiled from identical source code bases with the same compiler version, compiler and linker settings to the same libraries shall produce bitwise identical restart and history files when run on the same model segment input files and parameter settings at the same core count and problem decomposition. This implies that an existing executable run in an identical environment on the same input at the same core count and problem decomposition shall bitwise reproduce history and restart files at all times. [I]

The infrastructure supporting CM2-HR and CH-CHEM is continually tested to meet the reproducibility criteria. By construction, it is intended to meet the criteria for all input datasets. At various points in the life span of this infrastructure, failure to meet the reproducibility criteria has been traced to a number of system hardware and software failures as well as errors within the application itself. All such failures are investigated until the root cause is found. [I]

To verify reproducibility across core count, the Offeror shall run the model for 1 simulated month with make_exchange_reproduce=.true. for two different problem decompositions along with any compiler flags and environment variables necessary to achieve reproducibility. See scripts CM-HR-repro and CM-CHEM-repro. The reproducibility of the atmospheric and ocean components of the model may be verified through a series of checksums and global integrals written to stdout at the end of the run. Note that this can only check for platform "self consistency"; it is not expected that the Offeror shall bitwise reproduce the Government provided output. [R]

2.4. CAM/WACCM

CAM is the CCSM Community Atmospheric Model, the atmospheric model component of the Community Climate System Model (CCSM), which was jointly developed in cooperation with the National Science Foundation by NCAR, the Department of Energy, and the National Aeronautics and Space Administration. The current version of CAM is CAM3. [I]

WACCM is the Whole Atmosphere Community Climate Model, based on the dynamics of the CAM3 model, the chemistry and related processes from the Model for Ozone and Related Tracers (MOZART) and with additional chemical and physical processes needed to represent the mesosphere and thermosphere from the Thermosphere Ionosphere Mesosphere Electrodynamics General Circulation Model (TIME-GCM). [I]

There are five separate tests in the CAM/WACCM suite consisting of four "correctness tests" and one "performance test":

1. cam-pergro - A 2-day solution separation test to validate that CAM running on the Offeror's system produces answers within round-off of CAM running on the NCAR production platform bluesky system.

2. cam-toa - A 15 month CAM simulation of which the last 12 months are analyzed to validate a correct top-of-atmosphere energy balance.

3. waccm-rst - A combination test to validate exact restarts, and that answers are independent of the grid decomposition. Both 1D and 2D grid decompositions are employed. This test uses the WACCM configuration of CAM.

4. waccm-1mo - A 1 month WACCM simulation which generates a history file containing O3, H2O, U, and T. A field significance test is applied to this file to determine whether the same month computed on the Offeror's system is statistically the same as what is obtained on bluesky.

Note: All previous CAM/WACCM tests (1 through 4 above) can be run at the optimization chosen by the Offeror to produce correct results. [I]

5. waccm-perf - A 10-day performance test of WACCM.

Note: For the Baseline Results, this test must be run at the lowest of the optimization levels required to obtain correct results for all preceding CAM/WACCM tests (1 through 4 above). [I]

2.4.1. Procedure

The CAM/WACCM benchmark code and data is in the cam_waccm directory. Follow these steps to build and run the tests. [I]

There are two compressed tar files in the cam_waccm subdirectory. The cam319wa7.tar.gz file contains scalar code and the cam319wa7vec.tar.gz file contains vector code. The Offeror should select which of these code distributions is most appropriate to the Offeror's proposed solution and use it for performing and reporting the results for the CAM/WACCM tests. The Offeror should uncompress and extract the code into the cam_waccm subdirectory. The top level directory for the source code distribution is either cam_waccm/cam319wa7 for the scalar code version, or cam_waccm/cam319wa7vec for the vectorized version. All subsequent instructions will refer to either of these top level directories as $TEST_ROOT. [I]

The file inputdata.tar contains all the necessary datasets for running the CAM/WACCM tests. It should be untarred into the $TEST_ROOT subdirectory. The file validation-data.tar contains all the necessary datasets for validating test results. It should also be untarred into the $TEST_ROOT subdirectory. [I]

Shell scripts for driving each of these tests are in the $TEST_ROOT/tests/ directory. The scripts are designed to be executed from the tests directory. The shell script for each test will need to be edited for the particular environment of the test machine before being executed. [I]

Each test runs in it's own subdirectory of $TEST_ROOT/tests/. Approximate disk space requirements for each of the top level directories where tests are run is (in KBytes):

```
 323680  test 1   cam-pergro
1867168  test 2   cam-toa
9442784  test 3   waccm-rst
  88928  test 4   waccm-1mo
 154080  test 5   waccm-perf [I]
```

All test scripts are designed to be submitted to a batch queue system to build and run the job. Each test script can be run interactively (without using a batch system) with the "-b" option to just build the executable and then exit. The script can then be submitted to the batch queue system to run the job. The scripts are set up with batch queue commands appropriate for NCAR's bluesky system, which uses IBM's LoadLeveler® batch subsystem. This queuing system sets the current directory for the script to be the one from which the job was submitted. [I]

CAM's configure script (in $TEST_ROOT/ /models/atm/cam/bld/) produces a Makefile which is used to build the model. The configure script uses the file $TEST_ROOT/models/atm/cam/bld/Makefile as a template for the custom Makefiles it produces for a specific configuration. On a new platform it is quite likely that Makefile modifications will need to be made. If these modifications are made to the file $TEST_ROOT/models/atm/cam/bld/Makefile then each invocation of configure will pick up the changes. It is not advisable to edit the file produced by configure. Also note that configure has many command line options to allow for common customizations such as defining CPP macros or adding libraries to the Makefile's LDFLAGS macro without having to edit the Makefile template. [I]

In the test scripts, the parallel execution environment is set using LoadLeveler® and OpenMP variables. The total number of MPI processes used for a job is the number of nodes (#@node) times the number of tasks per node (#@tasks_per_node). The number of OMP threads per MPI process is set by the environment variable OMP_NUM_THREADS. Not all platforms support OpenMP, and some that do support OpenMP find that its use does not provide increased performance of CAM/WACCM. In that case the model can be configured to run in an MPI only mode; just replace the -smp argument to configure by -nosmp. [I]

2.4.1.1. Test 1: cam-pergro

The utility cprnc is required to compare CAM's output file against a control. To build the utility, cd to the directory $TEST_ROOT/models/atm/cam/tools/cprnc, and type make (must be a GNU make). The main issue for a successful build is to specify the directories containing the netCDF library and include files. These may be specified by setting the environment variables LIB_NETCDF and INC_NETCDF. [I]

cd to the $TEST_ROOT/tests/ directory. Edit the script cam-pergro.sh and modify it for execution on the target platform. After appropriate modifications have been made, execute the script cam-pergro.sh. [I]

A summary of the run is written to stdout by the run script. Check the output to see if the run passed the initial evaluation. If so the output file $TEST_ROOT/tests/cam-pergro/RMST_cmp_bluesky.out must be

sent to ORNL for the final evaluation. If the initial evaluation does not pass then the run on the test platform is not within roundoff of the run on NCAR's production bluesky system. A suggested strategy for addressing this problem is to reduce the compiler optimization level and rerun the test. [I]

The Offeror shall provide the information requested in the benchmark results spreadsheet. [R]

2.4.1.2. Test 2: cam-toa

The utility toa_energy is required to analyze CAM's output from this test. To build the utility, cd to the directory $TEST_ROOT/tests/check-toa and edit the Makefile so that the netCDF library and module files will be found. This utility uses the Fortran90 interface to netCDF, thus must be able to access the netcdf module file in the directory specified by the NETCDF_MOD variable. Then build the utility by issuing a make command (must be a GNU make). [I]

cd to the $TEST_ROOT/tests/ directory. Edit the script cam-toa.sh and modify it for execution on the target platform. After appropriate modifications have been made, execute the script cam-toa.sh. This run requires about 5 hours of wall time on one 32-processor bluesky node. [I]

A summary of the run is written to stdout by the run script. Check the output to see if the run passed the TOA energy balance. The test involves checking three separate conditions. All must pass for a successful test. If the test does not pass, we suggest reducing the compiler optimization level and rerunning the test. We expect that the optimization level used for a successful test 1 (cam-pergro) should allow this test to pass as well. However, we have encountered compilers that pass test 1 at a higher optimization level than was required to pass this test. [I]

The Offeror shall provide the information requested in the benchmark results spreadsheet. [R]

2.4.1.3. Test 3: waccm-rst

This test requires the utility cprnc. See the directions from test 1 (cam-pergro).

cd to the $TEST_ROOT/tests/ directory. Edit the script waccm-rst.sh and modify it for execution on the target platform. After appropriate modifications have been made, execute the script waccm-rst.sh. This script does three short runs of WACCM. The first two use the 1D decomposition and will make use of as many MPI processes as are assigned to the job (up to a maximum of 32). The final run uses the 2D decomposition and this is set via the namelist variable npr_yz. The test script is set up assuming that 8 MPI processes are assigned to the job. If a different number of processes is assigned then npr_yz must be adjusted accordingly (see notes below on configuring CAM/WACCM for peak performance). [I]

A summary of the run is written to stdout by the run script. Check that the output contains the message "Restart test PASSES". [I]

The Offeror shall provide the information requested in the benchmark results spreadsheet. [R]

2.4.1.4. Test 4: waccm-1mo

The utility fld_sig.exe is required to analyze WACCM's output from this test. To build the utility, cd to the directory $TEST_ROOT/tests/check-fld-sig and edit the Makefile so that the netCDF library and module files will be found. This utility uses the Fortran90 interface to netCDF, thus must be able to

access the netcdf module file in the directory specified by the NETCDF_MOD variable. Then build the utility by issuing a make command (must be a GNU make). [I]

cd to the $TEST_ROOT/tests/ directory. Edit the script waccm-1mo.sh and modify it for execution on the target platform. After appropriate modifications have been made, execute the script waccm-1mo.sh. This run requires about 5.5 hours of wall time on one 32-processor bluesky node. [I]

A summary of the run is written to stdout by the run script. Check the output to verify that the field significance tests pass for each of four fields. Here is an example of the output from a successful test:

```
H2O: PASS - Percentage of points outside 2sd limit is  4.94002525252525260
O3: PASS - Percentage of points outside 2sd limit is  3.94570707070707050
T: PASS - Percentage of points outside 2sd limit is  3.59848484848484862
U: PASS - Percentage of points outside 2sd limit is  2.91982323232323226 [I]
```

Note that since this is a statistical test, it is possible that a valid simulation will contain fields that fail this test. If only one or two fields report FAIL, and the percentage of points outside the 2sd limit is between 10% and 20%, there is a reasonable chance that the test will be judged successful. In the event of any reported failures the output file $TEST_ROOT/tests/waccm-1mo/waccm-1mo.h0.nc must be returned to with the Offeror's proposal for a more detailed analysis. [I]

The Offeror shall provide the information requested in the benchmark results spreadsheet. [R]

2.4.1.5. Test 5: waccm-perf

cd to the $TEST_ROOT/tests/ directory. Edit the script waccm-perf.sh and modify it for execution on the target platform. After appropriate modifications have been made, execute the script waccm-perf.sh. The purpose of this test is to demonstrate peak performance running the WACCM model, observing the constraint that the compiler optimization may not exceed the lowest of the levels required to pass the preceding correctness tests (tests 1 through 4). The script will need to be modified to configure a WACCM run that demonstrates peak performance (see notes below on configuring CAM/WACCM for peak performance). [I]

A summary of the run is written to stdout by the run script. Included is a performance summary obtained from the output of timing routines that are built into the executable by default. [I]

This timing information should be returned with the Offeror's proposal. [R]

Here is an example timing summary from this script running on two 32-processor nodes of NCAR's bluesky system:

```
Number of processes:                    16
Max number of threads per process:       4
Overall performance (sim-years/day):     0.53

Wallclock times:  per process (max|min)
stepon             4425.7 | 4425.6
  d_p_coupling      106.3 |   79.9
  bc_physics        995.4 |  549.2
  atmlnd_drv        485.2 |   40.0
  ac_physics       1350.6 |  794.5
  p_d_coupling      977.7 |  421.6
```

The test script is set up to do a 10-day simulation; this should not be changed. [I]

On some platforms the built-in timing library may itself cause an unacceptable performance penalty. In that case the timing library can be disabled by adding "-cppdefs -DDISABLE_TIMERS" to the command line arguments of the configure command, and the Offeror should use an alternate and appropriate methodology to obtain the wall time for the simulation and report the performance as "simulation years/wallclock day". [I]

The Offeror shall provide the timing summary information requested in the benchmark results spreadsheet for the run that demonstrates the peak performance of the waccm-perf test on their system. [R]

ORNL requests information that will allow determination of how the performance is scaling in the peak performance region, where "peak performance" is defined as the largest achievable value of "sim-yrs/day". Thus, if resources are available, the Offeror shall submit performance numbers for processor counts of approximately 1/4, 1/2, and X times the number of processors used for the peak performance result, where X is either:

- greater than 1 if less than all batch processors on the benchmark system are required for peak performance or
- approximately 3/4 if all batch processors on the benchmark system are required for peak performance. [D]

2.4.1.6. Configuring CAM and WACCM for peak performance

This section provides guidance on the main aspects of the parallel environment, the build, and the runtime settings that the Offeror can control to configure CAM or WACCM for peak performance. In addition there are communication options that are both build time and run time configurable that are beyond the scope of this discussion. [I]

The specific numbers in this section all assume the grid contains 96 latitudes, 144 longitudes, 26 vertical levels for CAM, and 66 vertical levels for WACCM; which is the resolution of the tests in this suite of CAM/WACCM tests. [I]

CAM uses different grid decompositions in the dynamical core (the "dycore") and in the physical parameterization package (the "physics"). Both can be configured by the user. [I]

CAM's finite volume (FV) dycore has two options for the grid decomposition, called 1D and 2D. The 1D decomposition (default) is by latitude, and requires a minimum of 3 latitudes per MPI process. The resolution used in all tests in this suite contain 96 latitudes, thus a maximum of 32 MPI processes may be used. The work done by each MPI process can be shared by multiple processors by enabling SMP threading which is implemented using OpenMP. The code does not do any dynamic thread management, so the number of threads in each process is determined by the environment variable OMP_NUM_THREADS. The WACCM code is currently being run successfully with 256 processors by using a 1D decomposition with 32 processes and 8 threads per process. [I]

To run with more than 32 MPI processes the 2D decomposition must be used. The 2D decomposition is actually two 2D decompositions, one in latitude/height (YZ), and the other in longitude/latitude (XY).

Because of the way these decompositions are specified in the namelist, it appears that there are 4 independent parameters to set in addition to the number of MPI processes. But practical constraints reduce the number to 1 (assuming the total number of MPI processes is set independently). The namelist variable is npr_yz, and it is specified as an integer array of length 4, (npr_y, npr_z, nprxy_x, nprxy_y), where

npr_y	number of latitude blocks in the YZ decompositions
npr_z	number of vertical blocks in the YZ decompositions
nprxy_x	number of longitude blocks in the XY decompositions
nprxy_y	number of latitude blocks in the XY decompositions [I]

The components of npr_yz must satisfy the condition

nprocs = npr_y*npr_z = nprxy_x*nprxy_y,

where nprocs is the total number of MPI processes. In addition, it is beneficial to satisfy

npr_y = nprxy_y and
npr_z = nprxy_x

as this minimizes the cost of the transpose between the decompositions. [I]

If we consider npr_y and npr_z to be the independent variables, this sets both the total number of MPI tasks and the number of blocks in each dimension of the XY decomposition. The notes that follow will just consider setting npr_y and npr_z. [I]

Just as the 1D decomposition has the restriction that each block contain at least 3 latitudes, in the 2D decomposition all blocks have the restriction that each dimension must be at least 3. For CAM (which uses 26 levels) the maximum values of npr_y and npr_z are 32 and 8, which means a maximum of 256 MPI processes can be utilized. The namelist specification for this configuration is `npr_yz=32,8,8,32`. For WACCM (which uses 66 levels) the maximum process count is 32x22 or 704 (setting `npr_yz=32,22,22,32`). Again, the total PE count is increased by threading in each process. [I]

The physics decomposition is based on data structures called "chunks" which are arbitrary collections of vertical columns; no assumptions about spatial contiguity of the columns are made. The main cause of load imbalances in the model are due to calculations of the physical parameterizations in daylight columns being more expensive than the calculations in nighttime columns. The fact that chunks don't require columns to be spatially contiguous allows the assignment of columns to chunks in a way that balances the number of night and day columns. But the best schemes for load balancing incur the largest communication cost in transposing between the dycore and physics decompositions. Hence several load balancing options have been implemented which embody various tradeoffs between load balancing and communication costs. These options are specified at runtime via the setting of the namelist variable phys_loadbalance to one of the following values (default is 0):

6. Chunks may cross block boundaries, but retain same processor mapping as blocks. If possible, processors assigned as day/night pairs. Columns (or pairs) are wrap-mapped. This option tries to mix up the vertical columns in order to assign equal numbers of daylight columns to each chunk, but without sending data between different MPI tasks.

7. Chunks may cross block boundaries, but retain same SMP-node mapping as blocks. If possible, processors assigned as day/night pairs. Columns (or pairs) are wrap-mapped, same as 0, but now tasks are allowed to trade columns as long as they are assigned to the same SMP node (where, hopefully, communication is cheap).

8. Two-column day/night and season column pairs wrap-mapped to chunks to also balance assignment of polar, mid-latitude, and equatorial columns across chunks. Scientifically, this is the near optimal option. It assigns pairs of columns with latitudes reflected over the equator and 180 degrees different in longitude. It then mixes up these pairs, assigning a mix of latitudes (polar, mid-latitude, equatorial) to each chunk. Computationally, this option requires an MPI_ALLTOALLV communication, maximizing the communication requirements.

9. Same as 1 except that SMP defined to be pairs of consecutive processors. This option allows neighboring processors to swap data, but no communication except between these pairs. When this option is used, the latitude assignment is modified so that the mirror of a latitude slice assigned to processor *i* is assigned to processor *i+1* (or *i-1*), so that good column pairs can be created. [I]

By way of example, experience has shown that the default value phys_loadbalance=0 works best on systems like NCAR's bluesky system, while phys_loadbalance=2 has achieved the best results on systems having superior MPI all-to-all communications.

Another user configurable aspect of the physics decomposition is setting the maximum chunk size (number of columns in the chunk) via the –pcols argument to configure. The default value of 16 gives good performance on scalar architectures, while vector architectures will probably give better results with a larger setting. [I]

Within each MPI process additional parallelization is available for the physics calculations via threading of loops over the number of chunks in the process. The number of chunks assigned to each MPI process depends on the number of columns (which is determined by the dynamics decomposition) and the chunk size. For a concrete example, consider using the maximum number of processes that are available to a 2D decomposition of the WACCM grid, i.e., 32x22. That implies the number of columns in a process is 21 at most (3 latitudes by the 6 or 7 longitudes that result from dividing 144 longitudes into 22 blocks), which means that using the default chunk size of 16 will result in only 2 partially filled chunks per process. In this case assigning any more than 2 processors per process will result in idle processors during the physics calculation. Also, since the physics is vectorized over the number of columns in a chunk, a vector machine will need to use far less than the maximum number of MPI processes in order to have reasonable size chunks for vectorizing the physics. [I]

2.4.1.7. Offeror Response

The following files should be returned with the Offeror's proposal:

1. A compressed tar file of the Offeror's $TEST_DIR containing all source code and scripts, but no data input or output files, used by the Offeror to run the CAM/WACCM tests, along with an Offeror README file explaining any code modifications.

2. test1 – cam-pergro: stdout from the successful cam-pergro.sh run and the $TEST_ROOT/tests/cam-pergro/RMST_cmp_bluesky.out file.

3. test2 – cam-toa: stdout from the successful cam-toa.sh run.

4. test3 – waccm-rst: stdout from the successful waccm-rst.sh run.

5. test4 – waccm-1mo: stdout from the successful waccm-1mo.sh run and, if any failures are reported, the file $TEST_ROOT/tests/waccm-1mo/waccm-1mo.h0.nc.

6. test5 – waccm-perf: stdout from the successful waccm-perf.sh run and, if the Offeror does not use the built-in timing routines, an explanation of the methodology used by the Offeror to obtain "simulation years/wallclock days". [R]

2.5. GFS

GFS is a global spectral weather model developed and used at NOAA NCEP. To build the GFS executable the Offeror will need to download and build ESMF version 2.2.2 release date 03/16/06 from http://www.esmf.ucar.edu/download/releases.shtml. The Offeror may reference instructions provided in the README file of the GFS directory. [I]

The initial condition files for this application are binary big endian. To generate little endian files, NOAA has provided a byteswap program. See fendian_conv.c provided with the application. [I]

Two different resolution jobs are provided, T190 and T510. The T190 job is provided for porting purposes. The scaling benchmark should be run using the T510 files. [I]

2.5.1. Model Reproducibilty and Validation

The GFS is bitwise reproducible with varying MPI task count and threads. Results should reproduce using the same code compiled with the same compiler version and settings and run on the same type of hardware with the same runtime libraries regardless of run core count or decomposition. [I]

Scripts and reference output files are provided for validation of results. Results for RMS values for various output fields at each vertical level are produced. The results after 24 hours should match to 5 digits accuracy for surface pressure and temperature and the RMS difference of the temperature fields should be less than 0.5. See the README file in the gfs/verify directory for detailed instructions. [I]

2.5.2. Parallelization of GFS

GFS is a hybrid model utilizing both OpenMP and MPI for parallelization. The MPI decomposition is 1-dimensional, which limits the MPI scalability to about 2/3 of the wave truncation. T510 is thus limited to about 340 MPI tasks. The OpenMP scalability is linear to four threads and good to eight threads on the IBM Power systems at NOAA NCEP. As a result, the T510 job scales to about 2600 cores on the NCEP CCS IBM Power6 nodes. The model can be run with any MPI task count up to the scalability limit and will run as a single MPI task if sufficient memory is available. It cannot be run without MPI. Due to the I/O design, the memory used by each MPI task is about the same for all tasks except for the last task, which uses much more memory. The last MPI task gathers the entire model state and writes it to secondary storage asynchronously while model integration continues on the other tasks. As a result, the last MPI task may need to be run on a node with fewer other tasks. [I]

ORNL would like to see results of a scaling study run in hybrid mode (OpenMP plus MPI). [D]

The Offeror shall report the run time for each core count in the Benchmark_Results.xls spreadsheet. [R]

The table below is an example of runtimes from NOAA/NCEP's IBM Power 6 system. [I]

MPI Tasks	Threads	Number of Nodes	Runtime (seconds)
288	2	9	539
288	4	18	296
288	8	36	226

Figure 2. NOAA/NCEP GFS Results for IBM Power6

2.6. Flow-following Finite-volume Icosahedral Model (FIM)

The flow-following finite-volume Icosahedral Model (FIM) from NOAA/ESRL is a global weather prediction model currently under development in the Global Systems Division of NOAA/ESRL. The FIM employs an Arbitrary Lagrangian-Eulerian (ALE) vertical coordinate running on a dynamic icosahedral horizontal grid. This ALE vertical framework is based upon a 'hybrid' structure, utilizing σ terrain-following levels near the surface and isentropic coordinates in the free atmosphere. The horizontal resolution of the icosahedral elements (which are primarily hexagons, with the exception of 12 pentagons) and hybrid levels are dynamically configurable at model run time. [I]

Physical parameterizations in FIM match those used in the operational Global Forecast System (GFS) developed at the National Center for Environmental Prediction (NOAA/NCEP). This allows FIM forecast initialization from the GFS analysis. Hybridization in the vertical coordinate is achieved through a unique flux corrected transport scheme. [I]

2.6.1. Building FIM

FIM utilizes the Scalable Modeling System (SMS) parallel programming package, an open source parallelization toolkit, also developed at NOAA/ESRL/GSD. SMS was developed to simplify development and parallelization of atmospheric and oceanic models. This package provides a portable, directive-based parallelization library, simplifying the message passing required to support NWP models running on distributed (or shared) massively parallel computer systems. To enable development of the icosahedral grid utilized by FIM, SMS was enhanced to support an indirect addressing mode for non-Cartesian grids. [I]

To build the SMS components, locate the sms directory and follow the directions detailed in the 'INSTALL' text file located at the top level of the sms directory. This procedure employs the familiar 'configure – make – make install' paradigm common to many open source packages. Note that installing this package in a non-standard location is supported by specifying a –dir option during the configure step (e.g. ./configure –dir my_sms_dir). Running the regression tests ensures your SMS installation is correct. [I]

To build the FIM model and associated utilities change your current directory to the fim directory and follow the directions detailed in the 'README.FIM_benchmark' text file at that location. You will need the makedepf90 utility. [I]

2.6.2. Running FIM

Many of FIM's runtime variables are controlled through use of a namelist (FIMnamelist). This namelist provides a mechanism to control a wide variety of parameters used by FIM at runtime, including location of data files, number of compute processors, use of I/O processors, horizontal and vertical resolution, and many more. Note that many of the variables (marked with 'notused') have been disabled to simplify porting. Also note that the number of compute tasks used by the fim model is specified in the FIMnamelist via "ComputeTasks". The namelist is read by both the pre-processor and the model, and must not be modified in between. This is done because the serial pre-processor programs may optionally re-order fim input data based upon the number of compute tasks to be used by the model. Once you have run the pre-processor for a desired horizontal grid (glvl), number of vertical levels (nvl), number of compute tasks (ComputeTasks), and write task scheme (write_task_scheme) you can re-use the fim model input files for multiple fim model runs. Note that the number of cores requested in the job submission must take in to account the need for 32 additional cores assigned to file I/O. [I]

Two variants of FIMnamelist have been provided in FIM_bm/fim/FIMrun to ease porting. FIMnamelist.small_case sets up a very small test case that can be run on a single task on most machines. FIMnamelist.10km sets up the global 10km benchmark run that is to be used to report all benchmark timing. Copy the desired file into FIMnamelist prior to running FIM. Then modify ComputeTasks if needed. [I]

2.6.3. Verification Procedure

To ensure the model ran correctly and produced a reasonable answer, examine the reported precipitation and mass figures reported near the end of this file. The contents of fim_out_* files should be identical across different numbers of processors on any machine. Printed diagnostic sums will exhibit small differences across various MPI task counts, architectures and compilers. We expect the 'Global 3D mass' and 'Global 3D water vapor' results at time step 5760 (1 day) to be within 3 significant digits accuracy of the results in the sample (5.1232E+18, and 1.3697E+16 respectively). See FIM_bm/fim/FIMrun/sample_runs/fim9_64_880_21/fim/ for output from a sample 10km run. [I]

Bit wise reproducibility across processor counts is expected at the lowest optimization levels. [I]

The Offeror shall provide the run time for the fastest case where bit-wise-identical results can be achieved across all processor counts. [R]

The Offeror may provide the run time for the fastest case where bit-wise identical results are not necessarily achieved, but the accuracy tolerance is still met. [D]

ORNL will also require the following files to verify your results:

FIM_bm/fim/FIMrun/sample_runs/fim9_64_880_21/fim/fim_out_2D000024

FIM_bm/fim/FIMrun/sample_runs/fim9_64_880_21/fim/fim_out_hg000024 [R]

Note that the name of the directory in FIMrun will reflect the number of cores used for computation and I/O in your run (replaces "sample_runs/fim9_64_880_21" in the paths above). [I]

2.6.4. FIM runtimes

The Offeror shall provide runtimes as reported in the stdout file produced by the fim run for at least 4 runs in a range of processor counts. At least one processor count should have a run time of 1800 seconds or less for the FIM model portion of the job. The table below is an example of runtimes from NOAA/ESRL's Intel Nehalem processor-based MPP system. Run times will appear at the end of FIM_bm/fim/FIMrun/fim*/fim/stdout in lines that look like this:

Total time = 5986.69788503647 [I]

If two or more times appear (optional write tasks report their own times), please report the largest.

Compute Tasks	Total processors	Runtime (seconds)
560	592	8793.6
800	832	6094.4
1040	1072	4944.9
1280	1312	4180.5
1520	1552	3737.1
1760	1792	3047.3
2000	2032	2865.1

Figure 3. NOAA/ESRL FIM Results for Jet MPP

3. Throughput Benchmark

3.1. General Comments

The throughput benchmark shall measure system performance under workload and runtime environment. The proposed system shall maximize the overall throughput and minimize the execution time as measured by the number of CM2-HR workflow instances that can be completed in the allotted time and the reduction in wall clock time for each instance as compared with the baseline measurement provided by ORNL. [I]

The workflow is a sequence of steps designed to represent the complete end-to-end execution of a single modeling application. The procured system shall be responsible for only a subset of the full end-to-end workflow processing. In particular, it is anticipated that the procured system shall

- Set up the workflow run directory from data copied to ORNL for the purposes of the specific run and from data located at ORNL
- Run the model using the requested resources
- Move/copy the run output to
    - The long term fast scratch filesystem
    - The fast scratch directory used for the next segment of the model run [I]

For throughput benchmarking, a “workflow instance” is comprised of 2 consecutive CM2-HR simulation segments. The wall clock time of these steps as measured from a single script shall be used to evaluate the proposed system performance. [I]

For the purpose of timing the benchmark, it is assumed the initial run directories exist and the model input data are already in place. The run script provided will initiate the first segment and run it to completion. It will then move files, set up the run directory for the second segment, run that segment and move the resulting files. [I]

The Offeror shall report the timings produced by each segment as well as the time for the complete job script including all data movement in the Benchmark_Results.xls file. If benchmark runs are done using only one filesystem, data shall be copied and then deleted to ensure that blocks are actually moved. [R]

The Offeror may modify the script to use whatever sequence of operations they define as optimal for use in a production setting such that the run directory for the second segment is setup in such a way that running the segment will not overwrite the history files from the previous segment. The next segment may begin as soon as the model output from the previous segment is in a state where it will not be overwritten by the new segment. During the acceptance test, the run script must initiate the movement of the history files from the FS to the LTFS in a manner the selected Offeror recommends for the production setting. [I]

A "completed work flow" shall mean that the next segment (segment 3) of the model run is ready to run. This implies that the new segment run would not overwrite history files from the previous segment. It also implies that the restart files from the previous segment (which were written to the RESTART/ subdirectory) are now readable from the INPUT/ directory. This can be accomplished (for example) by an Offeror-designed set of move, copy and/or link operations. These operations need not be completed prior to start of the next segment. For example, the Offeror may design a set of operations that starts the next model segment in a new directory having spawned a set of processes executing concurrently to clean up the old run directory. [I]

For reference, current methodologies move the job segment history files and copies of the history files to the equivalent of the LTFS. The scripts also push a copy of the history files into the run INPUT/ subdirectory. All operations complete serially before the next segment initiates. Alternate approaches are encouraged. For example, a different approach might simply setup a new run directory and initiate asynchronous data movement processes for the history files from the previous segment. The new segment may then start as soon as its run directory is established. [I]

Performance of the post-run data movement from the FS to the LTFS must be consistent with FS disk space availability requirements to maintain the full production workload. [R]

In the ideal case, throughput benchmark measurements are taken on the systems proposed for delivery using the same queuing and scheduling software being proposed for the installed system. It is understood that realization of the ideal case is unlikely. Thus, it is generally expected that the Offeror shall take performance measurements on systems with the software scheduling and queuing infrastructures currently available to them. After evaluating interactions between instances competing for resources, projection methodologies may be used to produce timings for the proposed configuration. [I]

As part of the Acceptance Test, the Offeror shall be required to demonstrate the proposed system capacity and work flow instance wallclock performance. [I]

3.2. Throughput Benchmark Scoring

The Throughput Benchmark must complete in no more than 3.5 hours. The Offeror shall measure and report the workflow component timings as described below. Based on this data and other aspects of the offered system(s), the Offeror shall propose the number of workflow instances that can be completed within the 3.5-hour benchmark time on the proposed system. This defines the system's Capacity. Improvements in job segment time as well as total wallclock time to completion are important so Offerors should run each instance on the number of cores that gives a performance sweetspot and an instance runtime between 3 and 3.5 hours. [R]

ORNL will extrapolate from the number of instances that can be completed within the 3.0-3.5 hour throughput benchmark time to the number that can be done per day by extrapolating the segment runtime to segment lengths that represent typical production runs. [I]

As part of the Acceptance Test, the selected Offeror shall supply and launch run scripts compatible with the offered queuing system for all work flow instances. If asynchronous data movement techniques are employed to clean up from a previous run while overlapping with a new segment start, it must be demonstrated that the proposed workflow performance can be achieved even in the presence of data moving processes. It is essential that the Offeror account for potential interactions between the work flow instances in proposing the offered throughput time.[I]

The Offeror shall verify reproducibility of the model at the same decomposition and core count. The reproducibility of the atmospheric and ocean components of the model may be verified through a series of checksums and global integrals written to stdout at the end of the run. Note that this can only check for platform "self consistency"; it is not expected that the Offeror will bitwise reproduce ORNL provided output. [R]

3.3. Throughput Benchmark Output

The file “Benchmark_Results.xls” has been distributed as part of the solicitation benchmark. In this file, an Excel spreadsheet is provided for the Throughput Benchmark. The CM2-HR model contains functions that report the Total runtime, Initialization, Main loop and Termination timing in terms of the minimum process time (tmin), the maxium process time (tmax) and the average process time. [I]

The Offeror shall report the maximum process time (tmax) for the Total Runtime, Initialization, Main loop and Termination for each segment of each work flow instance, as well as the job runtime (which includes data movement time) in Benchmark_Results.xls. [R]

The model writes to stdout (standard out - i.e. the "screen"). This information shall be captured (such as by piping to a file) and returned for all model instances. Only ASCII output shall be returned. The run scripts used for all throughput measurements shall be returned with the benchmark output. [R]

4. I/O Benchmarks

4.1. Metadata Operations

To verify the metadata performance of the file system, the Offeror will need the mdtest benchmark, available from http://sourceforge.net/projects/mdtest. For instructions on building and running this benchmark, refer to README file provided in the source code distribution obtained from this link. [I]

The mdtest benchmark must be run with the following parameters for 4, 16, 32 and 64 processes:

20 iterations (-i)

100 creat/stat/remove per process (-n)

single target directory (-d) on FS or LTFS

All output of this test shall be reported. [R]

4.2. Fast Scratch Benchmark

The fast scratch benchmark (FSB) is a synthetic workload meant to mimic the I/O workload of CM-CHEM. This benchmark utilizes the IOR benchmark and runs multiple concurrent invocations of the IOR benchmark each with different request sizes and is run on the CMRS compute nodes. While much of the workload is large block (1MB) reads and writes, a substantial percentage of the workload is small block I/O. Figure 4 illustrates the baseline request size distribution during a run of the CM-CHEM application. [I]

IOR is available from http://sourceforge.net/projects/ior-sio. For instructions on building this benchmark, refer to README file provided in the source code distribution obtained from this link. For instructions on how to run this benchmark, refer to the README file found in the tar file provided by ORNL. [I]

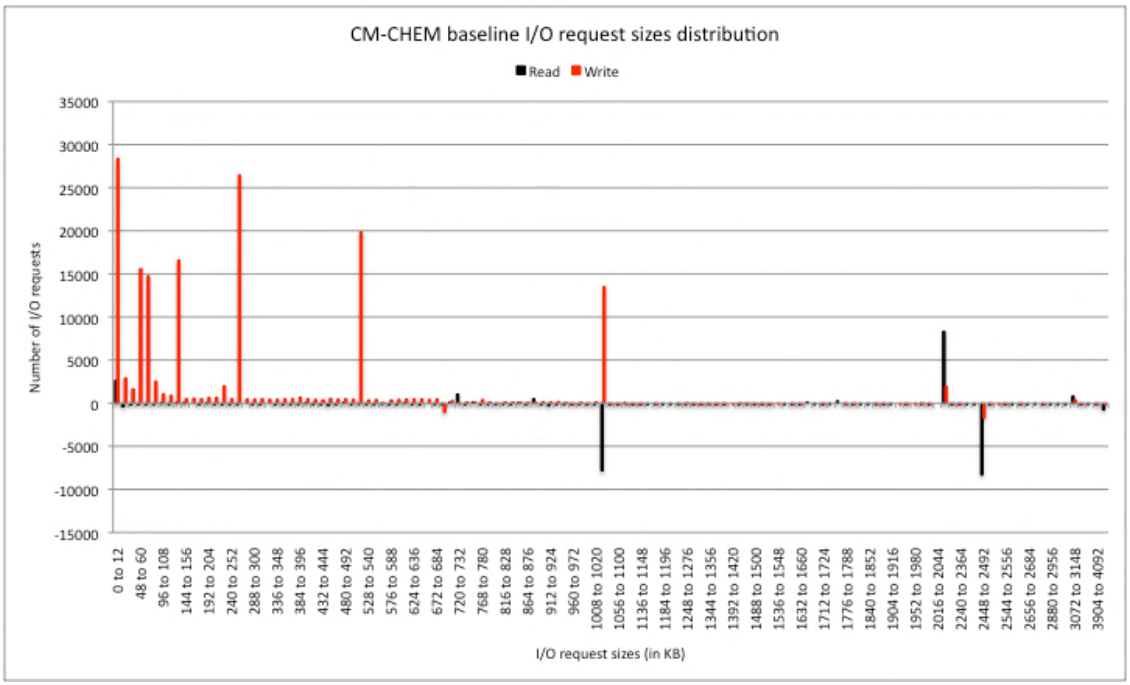

Figure 4. Baseline Request Size Distribution During CM-CHEM

Figure 5 illustrates the single restart request size distribution during a run of the CM-CHEM application during the checkpoint phase. [I]

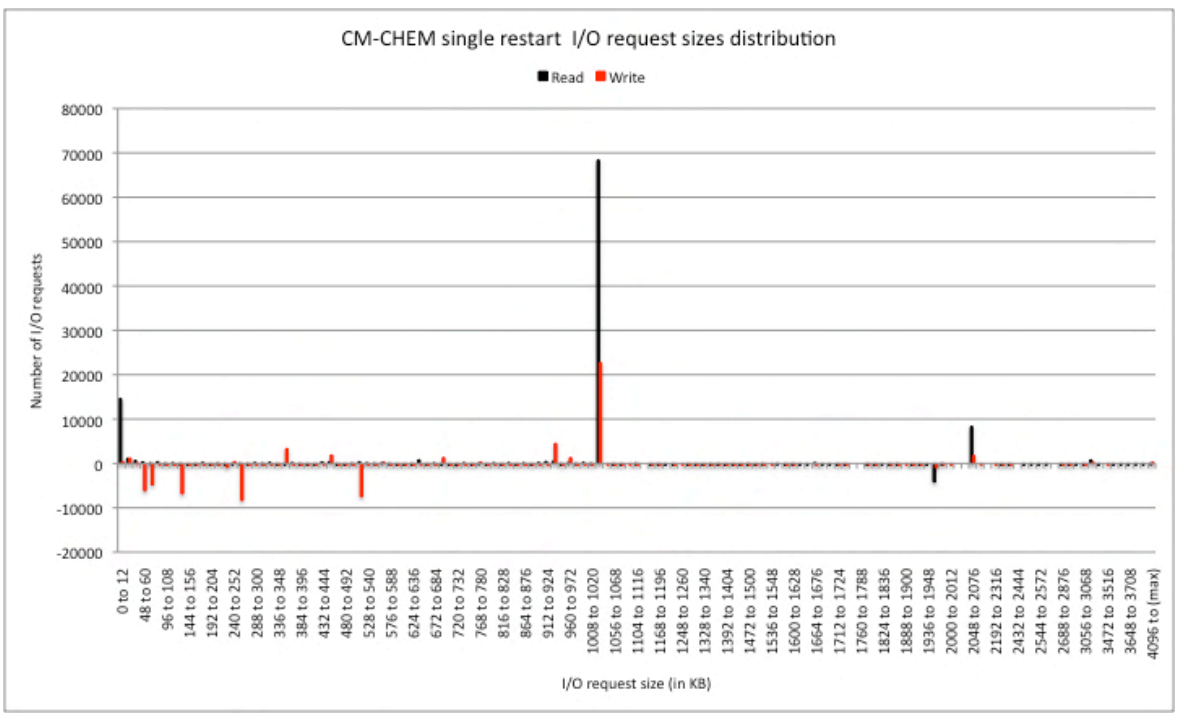

Figure 5. Single Restart Request Size Distribution During CM-CHEM

4.3. Long Term Fast Scratch Benchmark (LTFSB)

The LTFSB simulates the workload of moving multiple files between the FS and LTFS (a critical component of the CMRS workflow). This benchmark uses the IOR benchmark suite to simulate this workload and is run on the LDTNs. For instructions on how to run this benchmark, please refer to the README file found in the tar file provided by ORNL. [I]

Figure 6 illustrates the file size distribution for the CM2-HR application.

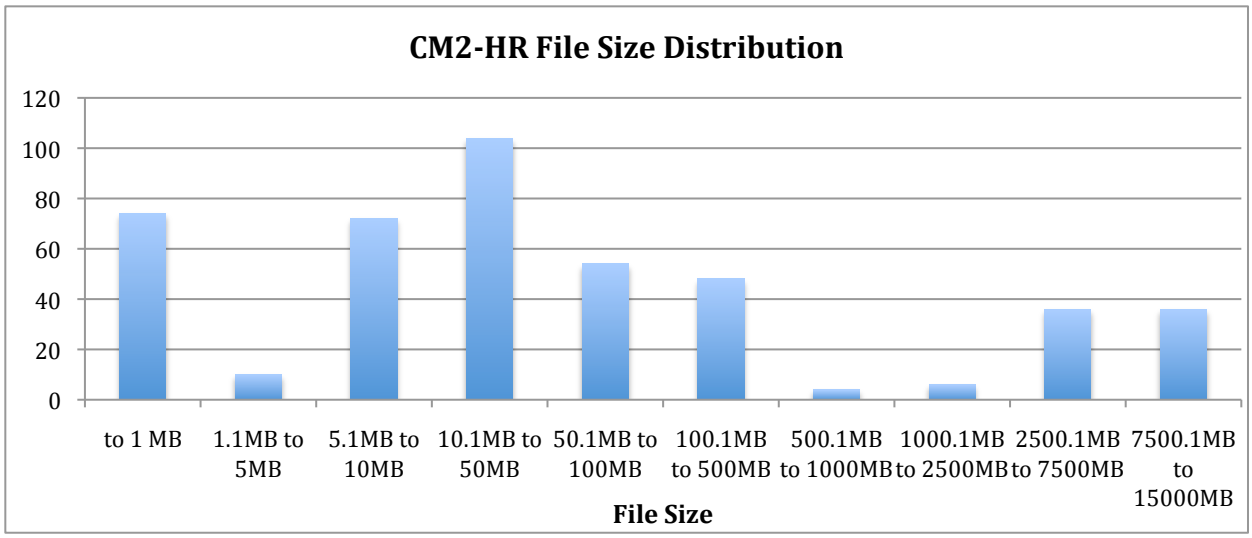

Figure 6. File size distribution for CM2-HR

5. EPCC OpenMP Microbenchmarks

To verify the OpenMP performance of the proposed system, The Offeror shall execute the C and Fortran90 versions of the EPCC OpenMP Microbenchmark Version 2.0. These are available from http://www2.epcc.ed.ac.uk/computing/research_activities/openmpbench/openmp_index.html. These benchmarks are designed for use with an OpenMP 2.0 compatible compiler. Instructions for building and running these benchmarks are provided on the web site. The synchronization and scheduling benchmarks must be run on a single node in two modes for each language:

1. Using all cores on the node

2. Using half the cores on the node, spread evenly across sockets [R]

To obtain accurate results, care is needed in the choice of clock routines. The Offeror shall choose a timer that is accurate and precise enough to produce valid results on this benchmark and document the timer chosen. Results should be entered into the benchmark spreadsheet. [R]