IRS: Implicit Radiation Solver Version 0.9.1 Benchmark Runs

The zrad3d Test Deck

There is a single test deck which may be scaled to run on various size problems. The number of domains in the problem scales with the number of MPI tasks and threads. The number of zones per domain will remain constant. Thus, the problem scales, and the amount of work done per domain remains the same as more MPI processes are run with more domains.

The following command line argument must be set by the user as part of the test runs

- -def NDOMS=999

Where 999 is the number of domains to generate during the test run. NDOMS is used to set the number of domains in the problem. For pure MPI runs, this is set to the number of MPI processes. There is one limitation on this setting – it must be a number cubed. That is, NDOMS may be set to 8, which is 2 cubed, or 27, which is 3 cubed, or 64 which is 4 cubed. etc.

In addition, the following two command line arguments may be set for advanced runs, but for this testing, the default values will be acceptable.

- -def NCPUS=999

This specifies the number of CPU's to be used. This will default to NDOMS. When running with one domain per MPI task or per thread, this is the acceptable setting. For advanced domain overloading experiments (not part of this test plan), this setting may be used to specify that fewer CPUS than domains will be used when running the test problem.

The third setting is

- -def NZONES_PER_DOM_SIDE=999

The default setting is 25 zones per domain side. When cubed, this results in 15,625 zones per domain. This setting is available for experiments which wish to increase or decrease the amount of work done per domain. This may be useful in seeing the relationship between cache and work per domain. Again, this testing is not part of this document.

Running the Code

The code should be run in the following configurations.

- Parallel using MPI
- Parallel using OpenMP Threads
- Parallel using MPI and OpenMP Threads

For this testing, you should either disable the output from the non-master MPI processes, or redirect them to a parallel file system.

This is done via one of the following command line options like so:

- -child_io_off

- -child_outpath /path/to/parallel/dir

The first option will be used in my examples, and will simply turn off output from non-master MPI processes.

The second option may be used to redirect this output to some directory.

This code uses OpenMP threads and honors the OMP_NUM_THREADS environment variable when determining how many threads to run. Consequently, the examples for the threaded runs will rely on two additional items when compared to the pure MPI runs.

- Set the environment variable OMP_NUM_THREADS to 2, 4, 8, or however many threads are appropriate.
- -threads must be specified on the command line to tell IRS to enable threading.

The deck used for testing, zrad3d, is located in the ~/irs/decks directory. It will need to be copied to your run location for testing.

Running Parallel using MPI

This should be your fIRSt set of tests. These examples will use the LLNL slum srun utility to specify the number of processors and nodes to use when running. You will have to modify this for whatever method your parallel platforms use to control these parameters. In the following examples

these words refer to the LLNL slurm system and are not part of the IRS code.

- srun -N 9 -n 9

-N specifies the number of nodes and -n specifies the number of cpu's when running under slurm.

You should run a sequence or runs, increasing the number of domains from 2 cubed, to 3 cubed, to 4 cubed, etc. as high as your platform will go. You should specify a prefix to be used by IRS when it creates output files so that your output will be organized. This is specified using the -k xxx option. For the mpi runs, specifying -k 00008MPI for the 8 domain run, and -k 00064MPI for the 64 MPI run, etc. will be used.

- `srun -n 8   irs zrad3d -child_io_off -k 00008MPI -def NDOMS=8`
- `srun -n 27  irs zrad3d -child_io_off -k 00027MPI -def NDOMS=27`
- `srun -n 64  irs zrad3d -child_io_off -k 00064MPI -def NDOMS=64`

- `...`
- `srun -n 729 irs zrad3d -child_io_off -k 00729MPI -def NDOMS=729`

The 729 run is a 9 cubed domain run.

To verify the problem generation size, you can grep the output or hsp files for the string “INITIAL” like so the following grep after a 27 domain run

```
grep INITIAL 00027MPI-0000-hsp
INITIAL Setup NCPUS                   = 27
INITIAL Setup NZONES PER DOMAIN SIDE = 25
INITIAL Setup ZONES PER DOMAIN       = 15625
INITIAL Setup TOTAL NDOMS            = 27
INITIAL Setup TOTAL ZONECOUNT        = 421875
INITIAL Setup TOTAL NDOMS PER SIDE   = 3
INITIAL Setup TOTAL ZONES PER SIDE   = 75
```

This verifies that each domain 25 zones per side for a total of 15,625 zones to work on. There are 27 domains in the entire problem. The total number of zones is thus 15,625 * 27 or 421,875 zones.

To view the benchmark results for each run, you can grep the output or hsp files for the string “BENCHMARK” like so:

```
dawson@zeus287/build > grep BENCHMARK 00027MPI-0000-hsp | grep -v echo
BENCHMARK microseconds per zone-iteration = 0.5621875
BENCHMARK FOM = 48026681.489717
BENCHMARK CORRECTNESS : PASSED
```

With ideal scaling, the “ microseconds per zone-iteration “ benchmark number will remain flat as the problem is scaled in size.

The FOM number is based on this microseconds per zone-iteration number and is simply (NCPUS * 1.0e6) / microseconds per zone-iteration.

And of course the last BENCHMARK number should say PASSED, if it says FAILED then the test run did not exhibit adequate convergence.

Running Parallel using OpenMP Threads

On a single node, you should be able to run with only a single process and as many threads as are available. For existing LLNL machines, this is typically only 4 or 8 CPUs. Still, it is a useful test to ensure that threading is actually working and that the microseconds per zone-iteration number decreases as you apply more threads to a fixed size problem.

Set the NDOMS to the number of CPU's you have available on a node. Then run a series of threaded tasks, starting with a single thread, and working your way up to NDOMS thread. During these runs the microseconds per zone-iteration benchmark number should decrease.

Here are runs on the LLNL 'zeus' machine, which has 8 cpus on a node. These test runs, and the output benchmark number show the effect of increased threads on a fixed size, 8 domain problem.

export OMP_NUM_THREADS=**1**

srun -n 1 irs zrad3d -child_io_off -threads -def NDOMS=8 -k 01THREAD

BENCHMARK microseconds per zone-iteration = **3.8510073710074**

export OMP_NUM_THREADS=**2**

srun -n 1 irs zrad3d -child_io_off -threads -def NDOMS=8 -k 02THREAD

BENCHMARK microseconds per zone-iteration = **2.8367567567568**

export OMP_NUM_THREADS=**4**

srun -n 1 irs zrad3d -child_io_off -threads -def NDOMS=8 -k 04THREAD

BENCHMARK microseconds per zone-iteration = **1.8539557739558**

export OMP_NUM_THREADS=**8**

srun -n 1 irs zrad3d -child_io_off -threads -def NDOMS=8 -k 08THREAD

BENCHMARK microseconds per zone-iteration = **1.5473218673219**

As you can see, the microseconds per zone-iteration per zone iteration decreases as more threads are applied to the problem. It is not perfect scaling, but additional threads are working.

Running Parallel using MPI and OpenMP Threads

These types of run enable the use of MPI across platform nodes, and threading within the node. Typical runs will place 1 MPI process on each node, and use threading to access the CPUS withing the node.

Variations on this may also be useful. Possible variations may include

- Running 2 MPI tasks per node, and setting the number of threads to one-half the number of CPUs per node.
- Running 1 MPI task per node, and setting the number of threads to one

less than the number of CPUS per node. This variation may idle a CPU, which be used by the O/S. In past hardware, runs like this have produced better results than runs which use all CPUS on the node.

At a minimum, you should run the standard test of 1 MPI task and setting the number of threads to use all CPUS on the node.

However, other combinations of MPI and threads may also be tested as desired. For large runs, we are interested in any runs which show the combinations which best use the machine.

The following shows a sequence of runs on the thunder machine at LLNL which as 4 cpus per node. These runs use 1 MPI task per node, and 4 threads per task, to run an 8 domain test and then a 27 domain test.

```
export OMP_NUM_THREADS=4

srun -N 2 -n 2 irs  zrad3d -child_io_off -threads -def NDOMS=8  -k 002MPI_004THR
BENCHMARK microseconds per zone-iteration = 0.82125307125307
BENCHMARK FOM = 9741211.6679133
BENCHMARK CORRECTNESS : PASSED

srun -N 7 -n 7 irs  zrad3d -child_io_off -threads -def NDOMS=27 -k 007MPI_004THR
BENCHMARK microseconds per zone-iteration = 1.0239583333333
BENCHMARK FOM = 26368260.427264
BENCHMARK CORRECTNESS : PASSED
```

In the second run, 7 nodes of 4 CPUS were allocated. Of these 28 CPUS, only a maximum of 27 will be used, since there are 27 domains. This means that even though up to 4 threads may be used on each node, one of the nodes only had 3 domains, so a CPU was idled. Due to this , during issue, the code generated the following warning message, which may be ignored.

```
NOTICE: [calculate_mappings] Number of blocks is not evenly divisible among
processors
        Winging it.
```

Results to save

For all runs performed, the command lines used to run the code should be captures. The primary interest is the benchmark number for each set of runs. The number of MPI tasks, the number of threads, and the number of domains for each run needs recorded along with the benchmark numbers. The generated *hsp files should also be saved.

UCRL-CODE-2001-010