SNAP as a Benchmark for CTS-2

Structured mesh, linear radiation pseudo-transport proxy application

Benchmark Problem Description and Specifications

SNAP solves a structured mesh, linear radiation pseudo-transport problem. This specific problem is of a fixed, 3-D spatial domain with a fixed discretization in the x-dimension. The y- and z-dimension discretization are weakly scaled with the number of nodes to create the benchmark at varying resource allocations. The number of energy groups and angular directions for mesh sweeps are fixed at 54 and 48, respectively. The material map and source map are fixed. The problem will run for 2 time steps.

The job has been designed to take approximately half of the memory capacity requirement. To assist in verifying this, output was added to SNAP to report the number of words allocated per rank.

Code Access and Compilation Details

Use the updated version v1.09 of SNAP, publicly released at GitHub. SNAP v1.09 has been specifically tagged and can be found at https://github.com/lanl/SNAP/releases/tag/ver1.09-04182019

SNAP should present no issues for Fortran compilers with OpenMP extensions. It is permissible to modify the Makefile and the Fortran flags to obtain the best performance on the reference node, and to run SNAP in a way which makes the best use of the node under test. The SNAP source code should not be modified in any way without a written approval. Any approved changes become the intellectual property of LANL.

Execution Details

Results in this document are given for 1-node problems run on CTS-1 and the Haswell partition of Trinity. Each of these types of runs was done with MPI only, 2 threads/rank, and 4 threads/rank. Affinity was set at the core level for threads. That is, as threads were increased, MPI ranks were not fixed; an increase in the number of threads per rank was met with a proportional decrease in the number of ranks. Each run was performed multiple times and the results shown below are the median of those runs. Noise in execution time was generally very small except for a couple extreme outlier cases that were assumed to be the result of some system perturbation/slow node issue.

It is expected that SNAP would be run in an advantageous mode for the node under test, whether through a scheduler or by using `mpirun` (or analog) directly. Prior experience shows that running SNAP with 2 OpenMP threads per MPI rank delivers good results.

An `mpirun` example that runs SNAP configured for 16 Open MPI ranks and 2 threads/rank is

```
mpirun -n 16 --map-by socket:PE=2 --bind-to core SNAP input output
```

If SLURM is used to launch the job, here are two alternative `srun` command examples for setting rank/thread mapping and affinity:

```
srun –n #ranks –c #threads/rank –cpu_bind=cores ...
srun –n #ranks --hint=nomultithread --cpus-per-task=#threads/rank ...
```

These command lines could be within a scheduler script. The runtime would normally be under 10 minutes per run.

Correctness Verification

The number of iterations increases as the problem is weakly scaled, probably because of the fabricated numerical formulas used in SNAP. To ensure correctness, please run the provided input file (16 ranks, 2 threads/rank) and verify that the number of iterations matches 2226 within some very small leeway (+/-10 iterations). Because SNAP does not solve a real problem, further results do not reveal much additional insight into the "correctness" of the code. However, large deviation in the number of iterations likely indicates some aspect of the code is not working correctly and should be further investigated. The line in the SNAP output can be found by searching for the unique string, "Total inners", which reports the total number of full transport mesh sweeps performed (the most important kernel).

Figure of Merit

The inverse of the reported "Grind Time" ("Inv. Grind" in the table above) is the Figure of Merit (FOM), with correctness checked through the total number of iterations and staying close to the memory per node target of about 1 GiB per core. This provides the most relatable sense of how the code is performing. Inverse grind time can be considered how many degrees of freedom are solved per unit of time.

The grind time to compute the FOM and constraints can be found in the SNAP output:

```
grep "Grind Time"
grep "Total inners" [must be 2226 +/- 10 for 16×2 input only]
grep "Allocated words" [approximately 1 GiB/core]
```

The provided input below has been run with 16 ranks and 2 threads/rank on a CTS-1 node (using the `mpirun` command from above) and produces the inverse grind time

FOM = 1.50 ns^{-1}.

It was modified slightly for load balance by rank and run again using all 36 cores of a CTS-1 node with 18 ranks and 2 threads/rank, producing the inverse grind time

FOM = 1.68 ns^{-1}.

See the discussion below for further details about how the problem was modified for this alternative FOM measure.

Reporting

For this benchmark, include the SNAP version number (including any approved source code changes), makefiles used to build on the target platform, input files, run scripts, and all standard output files for both correctness verification using the nominal input and for performance measurement using your node matched input. The node under test should be described in sufficient detail, particularly the processor and memory characteristics.

Input file

The nominal input file which currently runs on CTS-1 systems (2×18 Intel Broadwell cores per node), simplified to run only 16 ranks and 2 threads/rank and leave a few cores to other tasks:

```
! Input from namelist
&invar
  nthreads=2
  nnested=1
  npey=4
  npez=4
  ndimen=3
  nx=640
  lx=32.0
  ny=16
  ly=16.0
  nz=16
  lz=16.0
  ichunk=64
  nmom=4
  nang=48
  ng=54
  mat_opt=1
  src_opt=1
  timedep=1
  it_det=0
  tf=0.02
  nsteps=2
  iitm=5
  oitm=100
  epsi=1.E-4
  fluxp=0
  scatp=0
  fixup=1
  soloutp=0
  popout=0
  swp_typ=0
  angcpy=1
  multiswp=1
/
```

This input runs 16×2 threads and produces the inverse grind time of **1.50 ns^{-1}** on a CTS-1 node. This nominal input must be used for correctness verification. For performance measurement, it may be modified to maximize performance of the node, e.g., by scaling the number of cores (and potentially the number of spatial mesh zones) in use to fit the tested node. The product of input parameters `npey` and `npez` gives the number of ranks in use. It is recommended that `npey` and `npez` be reasonably close to equal whole numbers. When making these changes, keep `ny=4*npey` and `nz=4*npez` to maintain the same memory footprint per MPI rank (2.12 GiB, which is 1.06 GiB per core in use). Other parameters should not be modified without a written approval. The product `npey*npez*nthreads` gives the number of cores in use and must fit the node under test, perhaps with a few cores left to other tasks. With these changes, the number of iterations is may change, so only the grind time is of interest. As an example, choosing `npey=3, npez=6, ny=12,` and `nz=24` to run 18×2 threads produces the inverse grind time of **1.68 ns^{-1}** on the same CTS-1 node.

Additional Execution Results

As a further example of the effect of swapping MPI ranks for OpenMP threads for a fixed number of resources, the provided input was additionally run on the Haswell partition of the Trinity supercomputer. These runs used the first example of the `srun` command above. The table further indicates the total Solve time for some sense of wall-clock time.

Nodes	Ranks	Threads/ Rank	Iterations	Solve (s)	Grind (ns)	Inv. Grind (ns^{-1})
1	32	1	2226	102.06	7.29e-1	1.37
1	16	2	2226	98.27	7.02e-1	1.42
1	8	4	2226	99.99	7.14e-1	1.40