HPCG

Distribution

This document discusses HPCG (an open source benchmark and problem under a New BSD License) and results obtained on the unclassified Sandia National Laboratories' CTS-1 system on the restricted network. This document has undergone review and approval and is distributed as an unclassified, unlimited release (**UUR**) report (**SAND2020-3625 O**).

Summary Version

The preferred benchmark is HPCG version 3.1 Reference Code. In some instances (e.g., GNU GCC versions >= 9.0) this causes compile errors with `hpcg/src/ComputeResidual.cpp` (see additional discussion here). In this event, an updated HPCG version 3.1 w/ additional bugfix(es), denoted hereafter as 3.1 Updated Code, can be used. These details are listed below.

- **3.1 Reference Code**: download from http://www.hpcg-benchmark.org/downloads/hpcg-3.1.tar.gz, which has SHA256 `33a434e716b79e59e745f77ff72639c32623e7f928eeb7977655ffcaade0f4a4`.
- **3.1 Updated Code**: use Git to clone https://github.com/hpcg-benchmark/hpcg.git and checkout the commit with SHA-1 hash `e9e0b7e6cae23e1f30dd983c2ce2d3bd34d56f75` on `Mon Feb 17 12:04:33 2020 -0500`.

Purpose of Benchmark

Quantify performance of a conjugate gradient solver [1].

Characteristics of Benchmark

From [2],

> *HPCG is designed to exercise computational and data access patterns that more closely match a different and broad set of important applications, and to give incentive to computer system designers to invest in capabilities that will have impact on the collective performance of these applications.*

Run Rules

The run rules for HPCG are enumerated below. The sections that follow will provide additional clarification.

1. The figure of merit (FOM) is based on the rate at which HPCG instances complete.

2. Any number of instances may be used as long as they fit on a single node.
3. A single instance is a single rank of HPCG, i.e., `npx=npy=npz=1`.
4. Each instance must be run with `nx=ny=nz=128` (e.g., 1.9 GB/instance, 1m 40s run time for one instance, 9m 30s for all of 72 concurrent instances on CTS-1).
5. The clock for determining the rate of completion is started when the first instance begins execution and stops when the last instance completes.
6. The clock must reflect elapsed (i.e., wall clock) time.
7. HPCG must be compiled and run "as is" without source modifications or external libraries.
8. Compiler and linker optimizations are allowed as long as all optimizations used are publicly available.
9. HPCG may be compiled using a recent version of an OpenMP-enabled compiler and executed in multi-threaded mode.
10. Total wall time must exceed 5 minutes. There are no limits of how many back-to-back HPCG instances that can be run.

Building the Benchmark

The instructions to build HPCG are provided with its source code. It is relatively easy to build and has no external dependencies (except MPI and OpenMP). Ultimately, "`Makefile`" needs to be updated for local paths and compilers as needed. Once that is complete, it can be built with the `` `make clean && make all` `` command.

Running the Benchmark

There are two sets of command-line inputs that control the important aspects of the benchmark. The first is the size of the local memory mesh for each rank, expressed as `nx`, `ny`, and `nz`. For example, `nx=ny=nz=128` expresses a local mesh that is 128x128x128 (i.e., 2,097,152) points per MPI rank. The global mesh is the sum of the local meshes over all MPI ranks.

The second is the arrangement of ranks in the processor mesh, expressed as `npx`, `npy`, and `npz`. The processor mesh determines the number of total ranks and how efficiently they communicate. For example, `npx=4`, `npy=4`, and `npz=2` represent a processor mesh that is 4x4x2 (i.e., 32) MPI ranks in size. All three of these parameters are set to 1 due to the run rules.

An example Bash script with SLURM directives for running a *single* instance with 1 MPI rank and 1 thread per rank on CTS-1 is provided below.

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --time=00:30:00
#SBATCH --job-name=xhpcg-a

export OMP_NUM_THREADS=1
export MKL_NUM_THREADS=1

time srun --mpi=pmi2 --nodes=1 --ntasks=1 /path/to/hpcg-3.1/bin/xhpcg \
        --nx=128 --ny=128 --nz=128 --npx=1 --npy=1 --npz=1
```

An example Bash script with SLURM directives for running 72 separate instances concurrently on 36 cores, with 1 MPI rank per instance and 1 thread per rank on CTS-1, is provided below.

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --time=00:30:00
#SBATCH --job-name=xhpcg-a

export OMP_NUM_THREADS=1
export MKL_NUM_THREADS=1


export srun_invoke="srun --mpi=pmi2 --nodes=1 --ntasks=1 \
    /path/to/hpcg-3.1/bin/xhpcg \
    --nx=128 --ny=128 --nz=128 --npx=1 --npy=1 --npz=1"

function run_swarm {
    for (( i=0; $i < 36; i++ )) ; do
        eval $srun_invoke && eval $srun_invoke &
    done
    wait
}
time run_swarm
```

Outputs

The chief output is from the `` `time run_swarm` `` command in the example above. The resultant time is provided below (typically saved within the SLURM output file).

```
real  9m30.835s
```

Benchmark Verification

Each instance of HPCG must be successful. Success can be assessed by making sure that each instance produces the following items:

1. A text output file (e.g., "`HPCG-Benchmark_3.1_2019-06-06_06-32-32.txt`").
2. Each text file should contain a "VALID" GFLOP/s rating , e.g., "`HPCG result is VALID with a GFLOP/s rating of: 1.44214.`"

Figure of Merit (FOM)

The FOM is the number of successful HPCG instances completed per unit time on a single node. This is calculated by dividing the number of simultaneous instances of HPCG by the total elapsed time, i.e., the time from when the first instance begins until the last instance completes.

Recommendations for Sizing and Scaling Problems

The FOM was created to assess the total amount of work a single node can perform given the constraints of the problem size, which requires approximately 1.9 GB of memory per instance. The FOM will likely be highest by maximizing the number of simultaneous HPCG instances. If there is not enough memory to have one instance for each core, then running with more than one thread per MPI rank can be employed to speed up some or all instances.

If there is a scenario where some of the instances finish much sooner than others, then it is recommended to run as many back-to-back instances as it takes for all cores to complete at the same time. For example, if a node can run 16 HPCG instances however 8 of them finish in 2/3 of the time as the other 8, then the fast 8 could handle running 3 back-to-back instances while the slow 8 run 2 and then all 16 lanes finish at the same time with a total of 24 + 16 = 40 instances.

Figure of Merit Data on CTS-1

The FOM on CTS-1 was generated by running 72 instances, of which the final 36 instances finished at almost the same time. The total elapsed time was 9m30.835s. The FOM for this example is 72 instances ÷ 9.51391667 minutes = ***7.56786112 instances per minute***.

References

[1] Jack Dongarra, Michael A. Heroux, Piotr Luszczek, "**HPCG Benchmark: a New Metric for Ranking High Performance Computing Systems**," *Technical Report*, Electrical Engineering and Computer Science Department, Knoxville, Tennessee, UT-EECS-15-736, November, 2015.

[2] HPCG, https://hpcg-benchmark.org, accessed March 2020.

Change Log

- **2019-03-13**: Initial draft of this documentation was created.
- **2019-03-13**: SAND number (SAND2019-2805 O) assigned.
- **2019-06-06**: Revised to support HPCG 3.1 and new SAND number assigned.
- **2020-03-25**: Revised to support HPCG 3.1 with bugfix in SHA-1 hash bd34d56f75 and new SAND number assigned.