HACC

Summary Version
1.7 Update Feb. 14, 2018

Purpose of Benchmark

The Hardware Accelerated Cosmology Code (HACC) framework uses N-body techniques to simulate the formation of structure in collisionless fluids under the influence of gravity in an expanding universe. The main scientific goal is to simulate the evolution of the Universe from its early times to today and to advance our understanding of dark energy and dark matter, the two components that make up 95% of our Universe**.**

Characteristics of Benchmark

The HACC framework has been designed with great flexibility in mind – it is easily portable between different high-performance computing platforms. An overview of the code structure is given in Habib et al. Journal of Physics: Conf. Series, 180, 012019 (2009) and Pope et al. Comp. Sci. Eng. 12, 17 (2010). HACC has three distinct phases in the computation - their relative ratios to total run time strongly depend on the parameters of the simulation. The short force evaluation kernel is compute intensive with regular stride one memory accesses. This kernel can be fully vectorized and/or threaded. The tree walk phase has essentially irregular indirect memory accesses, and has very high number of branching and integer operations. The 3D FFT phase is implemented with point-to-point communication operations and is executed only every long time step; thus significantly reducing the overall communication complexity of the code.

Mechanics of Building Benchmark

The HACC code is hybrid MPI-OpenMP and depends on external FFT library. Both FFTW version 3.2 or later and IBM ESSL version 5.1 and later can be used. To compile the code, one needs first to modify the src/env/bashrc.vesta file accordingly. We recommend to keep “-D__bgq__” compilation flag and uncomment “#undef __bgq__” line in src/halo_finder/BGQStep16.c and src/halo_finder/BGQCM.c source files. Offeror may modify src/simulation/driver_hires.cxx file and insert equivalent function calls to start and stop counting performance data and/or time cycles. Both, flop rate and run time measurements are desired. Once proper changes are made, one needs to "source bashrc.vesta" the file, type "make clean" from the top “src” source tree level to ensure the clean build, after which type “make” in the directory “src/cpu”. The executable will be build as “src/cpu/vesta/hacc_tpm”.

Mechanics of Running Benchmark

The main input parameters are given in “indat” data file. For an example “00512_16x16x16_1112” run case, the major parameter of the run is “np = 2560”, the number of particles per dimension. This parameter determines the total number of “alive” particles and therefore, the size of the run: N = np**3 = 2560**3 = 16.78 Billions of particles. For this problem set up, the number of grid points per dimension, “ng”, should match the number of particles per dimension; even though the code can handle other use cases. Finally, the two parameters, the number of time steps “nsteps”, and the number of sub-steps, “nsub”, are used to set up the duration of the run. The “nsteps” argument controls the number of long force evaluations and therefore determines the communication intensity of the run. The “nsub” argument controls how many time steps to proceed between the long force evaluations, and therefore determines the computational intensity of the run. The FOM must be reported for nstep=3 and nsub=5. The typical run command is given below:

hacc_tpm indat cmbM000.tf m000 INIT ALL_TO_ALL -w -R -N 512 -t GEOMETRY

In here, all arguments should stay as shown except the “-t GEOMETRY” argument, that must be defined accordingly to the number of grid points, the number of MPI processes, and the topology of underlaying physical partition. This arguments defines the 2D and 3D grid decompositions of the simulation. For an example discussed above with 512 nodes and 8 MPI processes per node, the physical topology is going to be 4x4x4x4x2x8, which can be fold into (4x4) x (4x4) x (2x8) = 16x16x16, a perfect 3D decomposition. Each dimension must of the decomposition must be a deviser of the “ng” value. Internally, we use Cart_create function call to obtain a 2D Cartesian topology. The same restriction for the dimensions

applied as well. Here we use ng = 2560, therefore 1) "-t 16x16x16" can be applied as 16 divides 2560, 2) Most applicable 2D Cartesian topology for 4096 MPI processes is going to be 64x64, and 64 divided 2560.

HACC is intensively using thread stacks for local and private data; therefore the default stack size that most OS provide may be insufficiently small for successful run. We recommend to use the thread stack size around 4 MB. Additionally, the offerer may need to increase the value VMAX in file src/halo_finder/RCBForceTree.cxx, currently set to 16384. This variables controls the size of the neighborhood list in a process; at smaller total number of processes, the neighborhood list is increasing.

The result of the run is produced in file "m000.pk.fin" which presents the integrated spectrum. The standard output will also contain a set of performance and timing values.

Example runs:

Small problem: single node, 8 MPI processes per node
np = ng = 320, physical box = 252, -t 2x2x2

Medium problem: 2048 nodes, 16 MPI processes per node, 32768 MPI processes total
np = ng = 4096, physical box = 3200, -t 32x32x32

Large problem: 98304 nodes, 16 MPI processes per node, 1572864 MPI processes total
np = ng = 15360, physical box = 11630, -t 192x128x64

CORAL class problem:
np = ng = 24576, physical box = 18460

These examples are given only as suggested values. The offerer may provide the figure of merit (FOM) for problem sizes larger than CORAL class problem size. The reported figure of merit is calculated as the total number of particles (np*np*np) divided to the run time.

Verification of Results

The result of the benchmark is produced in file "m000.pk.fin" which presents the integrated spectrum. The standard output will also contain a set of performance and timing values. It is advisable to provide the performance estimates of the FOM for the problem sizes, that maximize utilization of the hardware from both computing capability and memory capacity.