Nekbone

Summary Version
2.3.4.1

Purpose of Benchmark
Nekbone captures the basic structure and user interface of the extensive Nek5000 software which is a high order, incompressible Navier-Stokes solver based on the spectral element method. Nekbone solves a standard Poisson equation using a conjugate gradient iteration with a simple preconditioner on a block or linear geometry. Nekbone exposes the principal computational kernel to reveal the essential elements of the algorithmic-architectural coupling that is pertinent to Nek5000. More details on the benchmark are provided in the readme.pdf file included in the distribution.

Characteristics of Benchmark
Nekbone solves a standard Poisson equation using the spectral element method with an iterative conjugate gradient solver with a simple preconditioner. The computational domain is partitioned into high-order quadrilateral elements. Based on the number of elements, number of processors, and the parameters of a test run, Nekbone allows a decomposition that is either a 1-dimensional array of 3D elements, or a 3-dimensional box of 3D elements. The benchmark is highly scalable and can accommodate a wide range of problem sizes, specified by setting the number of spectral elements per rank and the polynomial order of the elements by editing the appropriate build and input files.

The benchmark consists of a setup phase and a solution phase. The solution phase consists of conjugate gradient iterations that call the main computational kernel, which performs a matrix vector multiplication operation in an element-by-element fashion. Overall each iteration consists of vector operations, matrix-matrix multiply operations, nearest-neighbor communication, and MPI Allreduce operations. The benchmark is written in Fortran and C, where C routines are used for the nearest neighbor communication and the rest of the compute kernel routines are in Fortran. Note that the CORAL version of the benchmark is implemented using OpenMP and MPI and may be run with a varied mixture of threads and processes.

The benchmark is intrinsically well load balanced, with each process having the same number of spectral elements and therefore the same amount of computational work. Communication consists of nearest neighbor point-to-point communication with up to 26 surrounding processes and MPI Allreduce operations. While the amount of data communicated with neighboring processes can vary somewhat between processes, depending on proximity to the domain boundaries, on most systems the effects of this imbalance has been observed to be minimal. For systems where MPI reduction operations scale and point-to-point communications are non-interfering between network nodes the performance of the benchmark has been observed to scale nearly linearly with increasing

number of nodes across several orders of magnitude. At the node level elements are distributed across threads and communication gather-scatter operations are performed cooperatively between threads. Good scaling to ten's of OpenMP threads has been observed when elements are distributes evenly across threads and times for OpenMP synchronization operations in the gather-scatter and MPI reduction operations are not excessive.

Mechanics of Building Benchmark:

- Change to the nekbone *test/example1* directory
- Edit the *SIZE* file, if needed, to:
  - set the maximum number of MPI ranks, *lp*
  - set the maximum number of elements per rank, *lelt*
- Edit the *makenek* script and set:
  - SOURCE_ROOT to the path to the nekbone source code
  - F77 to the name of the Fortran compiler
  - CC to the name of the C compiler
  - The code must be compiled to use 8 byte floating point values - the necessary compiler flags to promote 4 byte reals to 8 byte doubles must be specified.
  - Uncomment IFMPI flag if not using MPI
  - To use OpenMP specify the compiler specific OpenMP compiler flags
  - Uncomment and specify any link flags in USR_LFLAGS
  - Uncomment and specify optimization flags, OPT_FLAGS_STD and OPT_FLAGS_MAG
  - Enable/Disable routine timers by adding/removing "TIMERS" pre-processor symbol in PPLIST. Routine timers measure and report the time for various solver operations.
- Run the *makenek* script. If *makenek* fails to recognize the compiler it will generate a *makefile* and stop, the *makefile* may then be edited manually and the code compiled by running *make.*
- Run '*make clean*' to remove previous build

Mechanics of Running Benchmark

1. General:
   - Compile as described above with or without MPI and OpenMP
   - Edit data.rea:
     - "ifbrick" should remain set to "true"
     - Set iel0, ielN, and ielD to the minimum, maximum, and step between solves in spectral elements per MPI rank.
     - Set nx0, nxN, nxD to desired minimum number of grid points in each direction per element, maximum number per element, and grid point step between solves.

    - Optionally set npx, npy, npz to desired process decomposition in x, y, z. If the product of npx*npy*npz equals the number of rank the specified process decomposition will be used, otherwise a process decomposition will be generated.
    - Optionally set mpx, mpy, mpz to desired per process element decomposition in x, y, z. If the product of mpx*mpy*mpz equals the number of elements per process the specified decomposition will be used, otherwise an element decomposition will be generated.
  - run *nekbone* executable. Number of MPI ranks and OpenMP threads are specified using standard MPI and OpenMP flags and variables.
  - *Note: For CORAL problem values used in data.rea must meet the specification of the CORAL problem defined below.*
2. Small problem: single node and/or single CPU
  - Compile without MPI by setting IFMPI flag in makenek to 'false'
  - Edit data.rea file:
    - Set iel0, ielN, ielD to specify the number of element per rank – the values should be set to something greater or equal to 1 but genrally small to keep the problem size small. iel0 and ielN can be set to the same value, and ielD can be set to 1 to solve for only one element count.
    - Set nx0, nxN, nxD to 9, 12, 3 respectively to use CORAL grid point counts.
    - Set npx, npx, npz to 1
    - Optionally specify the mpx, mpy, mpz element decomposition
  - run *nekbone*
3. Medium problem:
  - Compile as described above with MPI enabled and IFMPI flag commented out.
  - Edit data.rea:
    - Set iel0 and ielN to the desired number of elements per rank.
    - Set nx0, nxN, nxD to 9, 12, 3 respectively to use CORAL grid point counts.
    - Optionally specify the npx, npy, npz process decomposition
    - Optionally specify the mpx, mpy, mpz element decomposition
  - run *nekbone* with any combination of processes and threads.
4. Large Sequoia problem:
  - The Large Sequoia problem is defined as having 50,331,648 spectral elements. The values in the data.rea file that determine the polynomial orders (nx0, nxN, and nxD) are to be set to 9, 12, and 3 respectively. On Sequoia the problem was run using 3,145,728 ranks and 2 threads per rank in a 192x256x64 (npx,npy,npz) process distribution with 16 (iel0=16, ielN=16, ielD=1) elements per rank in a 2x2x4 (mx,my,mz) decomposition. The number of MPI ranks, OpenMP threads, and elements per rank used maybe any value so long as total spectral element count

remains 50,331,648 elements and the ratio of the largest to smallest value in the set (npx, npy, npz) is not greater than 5. The values controlling the polynomial orders must remain fixed at 9,12, and 3.

- Compile as described above with MPI enabled and altering *lp* and *letl* in SIZE as required
- Edit data.rea to set the appropriate number of elements per process.
- run nekbone

5. CORAL class problem:
    - The CORAL problem is defined as having approximately 2.5 million spectral elements per petaflop of theoretical peak performance of the system. The number of elements used may be any value within 15% of the approximate value calculated for the system. A 100 PF system therefore should use a total element count within 15% of 250 million elements.
    - The total element count is the number of MPI ranks multiplied by the number of elements per rank. The number of MPI ranks and number of elements per rank must be chosen such that the total element count meets the criteria specified above.
    - The number of elements per rank is specified in the data.rea file by setting values for iel0, ielN, and ielD. The values set for iel0 and ielN should be the same and ielD should remain set to 1.
    - The number of MPI ranks must be chosen such that the 3D process distribution used has process counts in the X, Y, and Z dimensions (reported as npx, npy, and npz in the standard output) that are all greater than 2 and the ratio of the largest to smallest value in the set (npx, npy, npz) is not greater than 5, unless restricted by unresolvable hardware limitations. In cases where the value of this ratio exceeds 5 it must be shown that no lower ratio could be achieved. For example (npx = 20, npy = 15, npz = 7) requires no justification, while (npx = 79, npy = 17, npz = 13) should be justified by hardware limitations.
    - The process distribution (npx, npy, npz) may be set manually, provided it meets the criterion above, by specifying npx, npy, and npz values in the data.rea input. The benchmark will generate a distribution with a minimal ratio if the value of npx*npy*npz does not equal the number of MPI ranks.
    - The local element distribution (mx, my, mz) may be set manually to any value by specifying mx, my, and mz values in the data.rea input. The benchmark will generate a distribution with a minimal ratio if the value of mx*my*mz does not equal the value set for the local element count.
    - The values in the data.rea file that determine the polynomial orders (nx0, nxN, and nxD) are to be set to 9, 12, and 3 respectively.
    - Compile as described above with MPI enabled and altering *lp* and *letl* in SIZE as required
    - The benchmark must be compiled to use 8 byte floating point values and reported FLOP counts must be for 8 byte floating point operations.

- run nekbone using any desired combination of processes and threads that meet the constraints specified above and the general CORAL requirement that each process must have at least 1 GB of useable memory available to it.

Verification of Results

The FOM to be reported is the average aggregate MFlop rate calculated and reported in the nekbone output as "Av MFlops". Benchmark results are considered correct if the reported rnorm is small, generally less than $1x10^{-8}$, after 100 conjugate gradient iterations.