miniFE

Summary Version

2

Purpose of Benchmark

miniFE is a Finite Element mini-application which implements a
couple of kernels representative of implicit finite-element applications.

Characteristics of Benchmark

The method of solution is as follows.

It assembles a sparse linear-system from the steady-state conduction
equation on a brick-shaped problem domain of linear 8-node hex elements.
It then solves the linear-system using a simple un-preconditioned
conjugate-gradient algorithm.

Thus the kernels that it contains are:
- computation of element-operators (diffusion matrix, source vector)
- assembly (scattering element-operators into sparse matrix and vector)
- sparse matrix-vector product (during CG solve)
- vector operations (level-1 blas: axpy, dot, norm)

Finally, miniFE compares the computed solution against an analytic solution
for steady-state temperature in a cube.

Mechanics of Building Benchmark

Two README files are included: README_version2 and README.runrules.
These are the steps for building miniFE.x:

```
$ tar xzf miniFE_openmp-2.0-rc3.tar
$ cd miniFE_openmp-2.0-rc3
$ cd src
```

Edit makefile for your environment's MPI wrappers, compilation flags.
For Sequoia class problems you might need

```
MINIFE_TYPES =  \
      -DMINIFE_SCALAR=double   \
      -DMINIFE_LOCAL_ORDINAL=int     \
      -DMINIFE_GLOBAL_ORDINAL="long long int"

$ make miniFE.x
```

If successful, that will create an executable called miniFE.x.

Special builds can be done (though not tested on LLNL's BG/Q) for things like:
- gnu compilers (g++, gcc), no MPI
  type 'make -f makefile.gnu.serial'

- enable Intel TBB support.
  In the miniFE_nodeAPI version:
  Edit makefile.tbb to set the path to your installation of Intel TBB, then
  type 'make -f makefile.tbb'
- enable NVidia/CUDA support.
  In the miniFE_nodeAPI version:
  Edit makefile.cuda to set the path to your installation of NVidia/CUDA, then
  type 'make -f makefile.cuda'

- enable TPI (ThreadPool) support.
  In the miniFE_nodeAPI version:
  Edit makefile.tpi to set the path to your installation of ThreadPool, then
  type 'make -f makefile.tpi'

- enable the use of the Sierra Toolkit mesh database stk::mesh, along with
  TPI threading.
  In the miniFE_stkmesh version:
  (At this time TPI threading is only used in the kernels
  used by the CG solve, not in any stk::mesh operations.)
  type 'make -f makefile.stk.tpi'

If the default makefile isn't correct for your system, copy and edit and
use a 'make -f' command similar to the above cases.

Note 1:
  'make' calls the script 'generate_info_header' to generate miniFE_info.hpp
  with platform and build information. If that script fails for some reason and
  doesn't produce a valid miniFE_info.hpp header, you can still build miniFE by
  typing
  'make MINIFE_INFO=0'.

Note 2:
  miniFE contains support for using the 'long long' integer type. If your
  compiler doesn't support that type, add this macro to the CPPFLAGS in
  the makefile: -DMINIFE_NO_LONG_LONG

About the code:
The main program is located in main.cpp. It calls miniFE::driver to do most
 of the assembly/solve work. If desired, change main.cpp to adjust the
 template-parameters given to miniFE::driver, to switch from double to float,
 or from int to long long, etc.

If the code fails to compile after changing these template parameters, it may be because a new traits type specialization needs to be added to TypeTraits.hpp.

General comments describing what the code is doing are located in main.cpp and driver.hpp.

Mechanics of Running Benchmark

The rules for running the benchmark are included in a README.runrules file. Four example command-line parameters or inputs for miniFE are shown below using SLURM notation. The parameter "n" indicates the number of MPI tasks and the parameter "N" indicates the number of nodes. Arguments nx, ny and nz are chosen to vary the size of the problem.

1. Small problem: single node
   srun -N 1 -n 16 miniFE.x -nx 264 -ny 256 -nz 256 # 16MPI tasks on 1 node

2. Medium problem:(<1K node)
   srun -N 512 -n 8192 miniFE.x -nx 2112 -ny 2048 -nz 2048

3. Large CORAL reference (for FOM baseline measurement on IBM BG/Q) problem:
   srun -N 4096 -n 65536 miniFE.x -nx 4224 -ny 4096 -nz 4096

4. CORAL class problem (2X the size of CORAL reference problem):
   srun -N 8192 -n 131072 miniFE.x -nx 4224 -ny 8192 -nz 4096

miniFE memory footprint calculation guidance:

A 264 x 256 x 256 problem fits on a BG/Q node with 16 GB of memory. Four racks can handle a problem that is 4096 times bigger. Distributed over three dimensions that implies each dimension is 16 times bigger, i.e. a 4-rack baseline solution (1/24th the full size of the 96-rack system) can execute a problem of size 4192 x 4096 x 4096 that fits within 64TB of memory.

Details of executing miniFE (arguments/options) are provided in the README file in that tarball.

Verification of Results

How are the benchmark results verified for correct answers?

If 'verify_solution=1' is added to the "mpirun"command line, as shown,

srun –N 1 –n 16 ./miniFE.x nx=264 ny=256 nz=256 verify_solution=1
then the computed solution will be compared against an analytic solution. verify_solution defaults to 0 which means don't verify the solution.

What FOM should be reported?

miniFE writes results to a YAML file which should be saved for reporting the figure of merit (FOM). From the YAML files, report "Total CG Mflops" as the FOM for miniFE.