MiniPIC for Crossroads

2018-11-07

Contents

1 Introduction

This documentation is an addition to the initial `README` file that came with MiniPIC for the ATS-3 procurement. That file's contents have been included and updated within this document.

MiniPIC solves the discrete Boltzman equation in an electrostatic field in an arbitrary domain with reflective walls. The code has potentially difficult dynamic memory requirements, localized work (potentially with imbalances) and stochastic processes. The MiniPIC benchmark uses an unstructured hex- or tet-based mesh with a static partition used for a particle mesh. Particles are tracked to every cell crossing, packed and passed off to adjacent processors using MPI. The main code base uses Tpetra objects from the Trilinos mathematics library for matrix/vector operations. Kokkos kernels are used to provide performance portability across architectures.

MiniPIC implements parallelism using the Kokkos performance portable programming model developed by Sandia National Laboratories. Typically it is configured to utilize OpenMP or CUDA backends depending on the target host. For the purposes of this benchmarking exercise, vendors are permitted to use any backend which is appropriate for their hardware (since Kokkos abstracts this for the application).

2 Building MiniPIC

The MiniPIC benchmarking used for the APEX procurement is based on Git revision 51e22de99774c94bd539f6a88af2cfa442a7a409 from Sandia's internal repository. This code was then modified to be able to utilize modern Trilinos; the commit used from the `master` branch of Trilinos (from GitHub.com/trilinos/Trilinos) for the testing on Trinity is described below.

```
commit 69f8850c25bdead51b38866ae34d4fa40947b67f
Merge: 8497d6816f ef22fd0b40
Author: Mark Hoemmen <mhoemmen@users.noreply.github.com>
Date:   Tue Jun 19 19:11:38 2018 -0600

    Merge pull request #2980 from trilinos/Fix-2968

    Tpetra: Fix #2968
```

The steps below are for a Cray XC40 environment circa July 2018. Please make appropriate modifications. The following modules were loaded.

- Intel version 18.0.2 Programming Environment
  - `module load PrgEnv-intel ; module swap intel intel/18.0.2`
- GCC 7.3.0 loaded for C++ header files for better C++11 support
  - `module load gcc/7.3.0`

The following 2 scripts are within the `extra` directory:

- `do-configure-trilinos-crayxc40.sh`
  - This will compile and build Trilinos VOTD (circa 2018-07).
- `do-configure-minipic.sh`
  - This will compile and build MiniPIC.

The following steps will build Trilinos:

1. Make a directory called `build-trilinos`.
   - e.g., `mkdir build-trilinos`
2. Run `do-configure-trilinos-crayxc40.sh` from within `build-trilinos`.
   - e.g., `pushd build-trilinos ; ../extra/do-configure-trilinos-crayxc40.sh`
3. Begin the build if CMake's configuration did not generate errors.
   - e.g., `make -j 16`
4. Install Trilinos if the build was successful.

- e.g., `make install`

5. Go back to base directory.
   - e.g., `popd`

If Trilinos was built successfully, then MiniPIC can be built with the following steps:

1. Make a directory called `build-minipic`.
   - `mkdir build-minipic`
2. Run `do-configure-minipic.sh` from within `build-minipic`
   - Change `Trilinos_PREFIX` to wherever Trilinos was installed earlier.
   - e.g., `pushd build-minipic ; ../extra/do-configure-minipic.sh`
3. Begin the build if CMake's configuration did not generate errors.
   - e.g., `make -j 16`
4. There should now be a binary named `main` in the `src` directory.
   - e.g., `ls -lh src/main`

3 Running MiniPIC

In the `run` directory is an example `brick.txt` file. This file defines the problem size in Nx, Ny, Nz being solved by MiniPIC and is read in during the start of computation. The problem state should be ideally kept with the dimensions equal.

The following is a list of relevant Global Problem Sizes:

- Small: 10x10x10
  - e.g., `echo "10 10 10" > brick.txt`
  - This was also with `dt=0.7` and `tfinal=7.0`.
  - This fits on a single XC40 Haswell node.
- Initial ATS-3 Large: 70x70x70
  - e.g., `echo "70 70 70" > brick.txt`
  - This was also with `dt=0.1` and `tfinal=1.0`.
  - This was targeted at 1,024 XC40 Haswell nodes.
- Target ATS-3 Problem: 88x88x88
  - e.g., `echo "88 88 88" > brick.txt`
  - This was also with `dt=0.07955` and `tfinal=0.7955`.
  - This is a case that approximately weak scales from the aforementioned 70x70x70 case to run on 2,048 XC40 Haswell nodes.

Mapping of MPI ranks to nodes or global mesh decomposition over nodes can be modifed by the user as required but the final mesh must meet the minimum sizes outlined.

3.1 Scaling Considerations

In order to maintain the same time stepping as the original algorithm, the user may choose to modify the `dt` (time stepping value) and `tfinal` (the simulated stop time). For the initial APEX reference problem, the benchmark default `dt` is 0.1 and default `tfinal` is 1.0. To provide an approximate weak scaling approach, the `dt` and `tfinal` ratio should be maintained. To scale these values, by example, if the problem size increases by 2X in each dimension, then a `dt` value of `old_dt/2 = 0.1/2 = 0.05` should be used. This can be specified by using the `--dt=0.05` option to MiniPIC when running the (2X * 2X * 2X = 8X) larger problem definition. Similarly, `tfinal` should be set to 0.5 to maintain the ratio.

3.2 Examples

This sub-section contains example invocation commands and their resultant output

3.2.1 Small: 10x10x10, 1 node

This is a small test case designed for a single XC40 Haswell node. An example minimal run script utilizing SLURM is provided below.

```
#!/usr/bin/env bash
export MINIPIC_HSW="/path/to/miniPIC-apex/build-minipic/src/main"

# setup environment
export brick_size=10
export mydt=0.7
export mytfinal=7.0
export nodes_per_job=$SLURM_JOB_NUM_NODES   # Number of nodes
export cores_per_node=32                    # Number MPI processes to run on each node (a.k.a. PPN)
export threads_per_core=1
export threads_per_rank=4
export ranks_per_node=$(($cores_per_node*$threads_per_core/$threads_per_rank))
export ranks_per_job=$(($ranks_per_node*$nodes_per_job))
export FILE_LOG="run-miniPIC.log"
export OMP_NUM_THREADS=$threads_per_rank
export OMP_PROC_BIND=close
export OMP_PLACES=cores
export MKL_NUM_THREADS=$threads_per_rank
echo "${brick_size} ${brick_size} ${brick_size}" > brick.txt
srun \
    --ntasks-per-node $ranks_per_node \
    --ntasks $ranks_per_job \
    --hint=nomultithread \
    "${MINIPIC_HSW}" \
        --kokkos-threads=${threads_per_rank} \
        --dt=${mydt} \
        --tfinal=${mytfinal} &>> "${FILE_LOG}"
grep -i update "${FILE_LOG}"
```

The output from this job is provided below (i.e., from the `grep` command).

```
Move time 94.95121 for 100,000,000 parts, or  5.30E6 updates/second
```

3.2.2 Initial ATS-3 Large: 70x70x70, 1,024 nodes

This is a large test case designed for 1,024 XC40 Haswell nodes. An example minimal run script utilizing SLURM is provided below.

```
#!/usr/bin/env bash
export MINIPIC_HSW="/path/to/miniPIC-apex/build-minipic/src/main"

# setup environment
export brick_size=70
export mydt=0.1
export mytfinal=1.0
export nodes_per_job=$SLURM_JOB_NUM_NODES   # Number of nodes
export cores_per_node=32                    # Number MPI processes to run on each node (a.k.a. PPN)
export threads_per_core=1
export threads_per_rank=2
export ranks_per_node=$(($cores_per_node*$threads_per_core/$threads_per_rank))
export ranks_per_job=$(($ranks_per_node*$nodes_per_job))
export FILE_LOG="run-miniPIC.log"
export OMP_NUM_THREADS=$threads_per_rank
export OMP_PROC_BIND=close
export OMP_PLACES=cores
export MKL_NUM_THREADS=$threads_per_rank
echo "${brick_size} ${brick_size} ${brick_size}" > brick.txt
srun \
    --ntasks-per-node $ranks_per_node \
    --ntasks $ranks_per_job \
    --hint=nomultithread \
    "${MINIPIC_HSW}" \
        --kokkos-threads=${threads_per_rank} \
        --dt=${mydt} \
        --tfinal=${mytfinal} &>> "${FILE_LOG}"
grep -i update "${FILE_LOG}"
```

The output from this job is provided below (i.e., from the `grep` command).

```
Move time 201.95635 for -59738368 parts, or  8.49E8 updates/second
```

3.2.3 Target ATS-3 Problem: 88x88x88, 2,048 nodes

This is a large test case designed for 2,048 XC40 Haswell nodes. An example minimal run script utilizing SLURM is provided below.

```
#!/usr/bin/env bash
export MINIPIC_HSW="/path/to/miniPIC-apex/build-minipic/src/main"

# setup environment
export brick_size=88
export mydt=0.07955
export mytfinal=0.7955
export nodes_per_job=$SLURM_JOB_NUM_NODES   # Number of nodes
export cores_per_node=32                    # Number MPI processes to run on each node (a.k.a. PPN)
export threads_per_core=1
export threads_per_rank=2
export ranks_per_node=$(($cores_per_node*$threads_per_core/$threads_per_rank))
export ranks_per_job=$(($ranks_per_node*$nodes_per_job))
export FILE_LOG="run-miniPIC.log"
export OMP_NUM_THREADS=$threads_per_rank
export OMP_PROC_BIND=close
export OMP_PLACES=cores
export MKL_NUM_THREADS=$threads_per_rank
echo "${brick_size} ${brick_size} ${brick_size}" > brick.txt
srun \
    --ntasks-per-node $ranks_per_node \
    --ntasks $ranks_per_job \
    --hint=nomultithread \
    "${MINIPIC_HSW}" \
```

```
        --kokkos-threads=${threads_per_rank} \
        --dt=${mydt} \
        --tfinal=${mytfinal} &>> "${FILE_LOG}"
grep -i update "${FILE_LOG}"
```

The output from this job is provided below (i.e., from the `grep` command).

```
Move time 180.222316 for -572276736 parts, or  1.8906E9 updates/second
```

4 MiniPIC Figure of Merit

MiniPIC will print performance information to the standard output. The FOM for the benchmark is the updates/second value printed at the end of the run. This figure should be provided in the Offeror response spreadsheet. An example of this line for the target ATS-3 problem is provided below.

```
Move time 180.222316 for -572276736 parts, or  1.8906E9 updates/second
```

5 MiniPIC Run Rules

The following enumerated list contains the overall run rules.

1. **Baseline Run** - The Offeror may choose to make modifications such as aggressive compiler and OpenMP directive optimizations as long as the compiler and optimizations are generally available and fully supported on the proposed platform. Source code changes used for the benchmark FOM must be reported with the Offeror response. The benchmark must verify correct operation.

2. Benchmark runs must be performed with a brick problem size no smaller than 88x88x88.