1 Compiling Dendro-GR GPU extension

The developed Dendro-GR GPU extension is available here. The following dependencies are required for the compilation. The core kernel generator is based on SymPyGR.

- C/C++ compilers with C++11 standards and OpenMP support
- CUDA compilation tools version 10.2
- Python3 SymPy, networkx for code generation.
- MPI implementation (e.g. openmpi, mvapich2)
- ZLib compression library
- BLAS , LAPACK, and GSL libraries.
- CMake 2.8 or higher version

To build the code use the following commands.

```
$cd <path to root source dir >
$ mkdir build
$ cd build
$ cmake ../. -DWITH_CUDA=ON
$ make bssnSolverCtx bssnSolverCUDA tpid -j4
```

- Note that that, -DWITH_CUDA=ON build code for both CPU and GPU, while -DWITH_CUDA=OFF compilation only happens for the CPU code.
- The above will build three targets in `<build dir>/BSSN_GR/` folder these corresponds to CPU BSSN Solver, GPU BSSN Solver and two punctures initial condition solver for the binary black hole problem.

1.1 Using Singularity

The singularity container definition file is provided in the repository under the folder `container`. The following command can be used to build the Dendro-GR container which installs all the required dependencies and compile the Dendro-GR code. Note that by default the singularity `.def` file is set to built for both CPU and GPU (i.e., CUDA) architectures.

```
sudo singularity build --sandbox dgr-cuda dgr.def
singularity run dgr-cuda dgr.def
```

2 Experimental setup

The CPU/GPU performance comparisons were performed in TACC's Lonestar6 Cluster. We performed our experiments on Frontera and Lonestar 6 at the Texas Advanced Computing Center (TACC). Frontera has 8K Intel Cascade Lake nodes and Lonestar 6 has 16 dual-NVIDIA A100 nodes. All GPU-CPU comparisons were done on a single NVIDIA A100 GPU with full CPU node (i.e., 128 cores, 64 cores per socket) with AMD EPYC 7763 CPU. Frontera is used for performing a large scale weak scaling study.

- Lonestar6 Cluster module environment used is given below.

```
1) intel/19.1.1   2) impi/19.0.9   3) python3/3.9.7   4) cmake/3.21.3
5) pmix/3.2.3   6) xalt/2.10.32   7) TACC   8) cuda/11.4 (g)   9) gsl/2.7

Where:
g:  built for GPU
```

- Frontera Cluster module environment used is given below.

```
Currently Loaded Modules:
1) intel/19.1.1   2) impi/19.0.9   3) git/2.24.1   4) autotools/1.2
5) python3/3.7.0   6) pmix/3.1.4   7) hwloc/1.11.12   8) xalt/2.10.34
9) TACC  10) gsl/2.6  11) cmake/3.20.3
```

3 Running experiments

3.1 Binary mergers and GWs

- The parameters for the applications has to be provided with `.json` file. Example parameter files for mass ratios 1, 2, and 4 can be found in `BSSN_GR/pars` folder.
  - `BSSN_GR/pars/q1.par.json` : q=1 binary black hole merger
  - `BSSN_GR/pars/q2.par.json` : q=2 binary black hole merger
  - `BSSN_GR/pars/q4.par.json` : q=4 binary black hole merger
- Create the following folders in the relative path to the BSSN executable.
  - cp - checkpoint folder where the checkpoints are stored in the frequency specified by the parameter file (i.e., `BSSN_CHECKPT_FREQ`).
  - vtu - VTU folder where the solution is written with parallel VTU file format, in the frequency specified by the parameter file (i.e., `BSSN_IO_OUTPUT_FREQ`).
  - dat - Data files, diagnostics data on the binary. Requested modes (i.e., `"BSSN_GW_L_MODES"`: [2,3,4]) of the gravitational waves are extracted by the observers specified by
    `"BSSN_GW_RADAII": [50,60,70,80,90,100]`

- First run the `tpid` solver with the chosen parameter file. The above will solve for the initial condition for the binary using Two puncture gauge.
- Once the `tpid` solver is finished, user can launch the BSSN solver `bssnSolverCUDA` or `bssnSolverCtx` for GPU and CPU versions respectively.

The following executables are used in the paper.

- BSSN_GR/tpid - two puncture initial condition solver.
- BSSN_GR/bssnSolverCUDA - GPU BSSN solver
- BSSN_GR/bssnSolverCtx - CPU BSSN solver

```
$ ./BSSN_GR/tpid q1.par.json  <number of threads to use>
$ mpirun -np <number of GPUs> ./BSSN_GR/bssnSolverCUDA q1.par.json 1
```

3.2 GPU and CPU experiments

Additional scripts files for conducted experiments are presented in the `<source dir>/BSSN_GR/experiment_scripts/ls6` folder.

- q1-ss : GPU/CPU strong scaling experimental scripts.
- q1-ws : GPU weak scaling experimental scripts.
- Weak scaling: `make bssnWSTestCUDA` and use `bssnWTestCUDA` for weak scaling on GPUs. Use `mpirun -np <number of GPUs> ./BSSN_GR/bssnWSTestCUDA q1_ws.par.json 1`
- Strong scaling: Use the parameter file in `BSSN_GR/pars/scaling/q1_r2.2.par.json`.
  - CPU : `mpirun -np <number of CPUs> ./BSSN_GR/bssnSolverCtx q1_r2.2.par.json 1`
  - GPU : `mpirun -np <number of GPUs> ./BSSN_GR/bssnSolverCUDA q1_r2.2.par.json 1`
- `octant to patch` and `patch to octant` Experiments: `make run_meshgpu_tests` to build the benchmark for padding zone computation. Note that this will built the executable in `<source dir>/build` folder. The parameters should be specified in the order and corresponds to
  - maximum allowed depth of the octree, tolerance value for refinement, partition tolerance (recommend to keep it at 0.1), order of interpolation for each octant (all the experiments used 6 order interpolations in the paper), flag 0 for CPU padding zone computations, flag 1 for GPU padding zone computations.
  - CPU tests `mpirun -np <number of CPUs> ./run_meshgpu_tests 8 1e-3 0.1 6 0`
  - GPU tests `mpirun -np <number of CPUs> ./run_meshgpu_tests 8 1e-3 0.1 6 1`

3.3 Running experiments with Singularity

If you built Dendro-GR with Singularity, the following command can be used to launch the main BSSN solver and other benchmarks.

```
singularity exec dgr-cuda \
sc22-dgr/build_gpu/BSSN_GR/./bssnSolverCtx sc22-dgr/build_gpu/q1.par.json 1
```

More details on the running of the solvers are described in the §3