CHPC Student Cluster Competition 2022

SWIFT

SWIFT is an acronym for **S**mooth Particle Hydrodynamics **W**ith **F**ine-grained inter-dependent **T**asking. It is a gravity and SPH solver designed to run cosmological simulations, scaling well across up to peta-scale machines.

Obtain a copy of the SWIFT source from either GitLab or GitHub.

```
$ git clone https://github.cosma.dur.ac.uk/swift/swftsim.git
```

Or, alternatively

```
$ git clone https://github.com/SWIFTSIM/swiftsim
```

Configuration, building and installation instructions can be found in the `INSTALL.swift` file, which detail the following dependencies to be installed or made available in your environment:

- Your choice of **Compiler** and **MPI**
- **HDF5** library (*optionally* optimize large IO and file processing)
- **GSL** (GNU Scientific Library, required for cosmological time integration)
- **FFTW**
- **METIS** or **ParMETIS** (*optionally* optimize load / domain partitioning between MPI tasks)

```
$ ./autogen.sh
$ ./configure --with-metis \
              --with-tbbmalloc \
              --with-fftw=/path/to/fftw
$ make
```

If your compilation succeeds, the build process will generate two binaries

```
$ ls examples/
swift swift_mpi
```

Benchmark 1: IsolatedGalaxy (single node)

This is a relatively small benchmark of an isolated galaxy disk embedded in an external Hernquist dark matter halo. The disk galaxy simulation comprises approximately 160 thousand start particles and is used for testing galaxy formation of the subgrid model implementation.

Run the ISOLATEDGALAXY_POTENTIAL, with the following options, pertaining to the physics aspects of the simulation. You may **not** edit the `isolated_galaxy.yml` file.

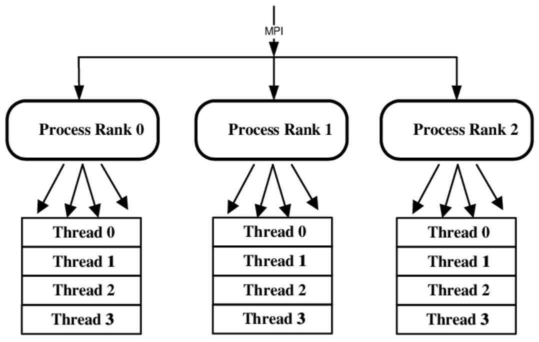
```
$ cd examples/IsolatedGalaxy/IsolatedGalaxy_potential
$ ./getIC.sh
$ ../../swift --external_gravity \
              --self_gravity \
              --stars \
              --threads=N \
              isolated_galaxy.yml 2>&1 | tee isolatedgalaxy.log
```

Copy your output files to your flash drive to the organizers for judging:

- `swift` binary,
- `run_swift.sh` submission generation script, and
- `isolatedgalaxy.log` error and output logfile.

Benchmark 2: Eagle_25 (Parallel Efficiency Investigation)

The second benchmark contains approximately 53 million dark matter, 50 million gas and 2 million start particles. The threading model used by SWIFT is **not** OPENMP. Parallel computing can be carried out in two ways. One is done on a single computer with multiple internal processors, known as a Shared Memory Multiprocessor. The other way is achieved through a series of computers interconnected by a network, known as a Distributed Memory Multicomputer. SWIFT uses MPI and POSIX THREADS (PTHREADS) to implement a hybrid parallelism scheme. This allows message passing between NUMA domains, with each domain containing multiple worker threads and utilizing shared memory.



You will need to explicitly tell SWIFT how many threads **N** to spawn per **MPI RANK** and conduct an experiment to determine the most efficient communication pattern. You will do so by populating a table similar to the one shown below:

Total MPI Ranks	Ranks per node	Threads/Rank	RAM usage per node	Run time
6	2	8		
12	4	4		
24	8	2		
…	…	…		

First fetch the initial conditions file.

```
$ cd examples/EAGLE_low_z/EAGLE_25
$ ./getICs.sh
```

NOTE: The `EAGLE_ICs_25.hdf5` file is approximately **3.6 GB**, but is only needs to be fetched once.

Configure a SLURM batch script or alternative, that executes an example MPI run on **<NUM NODES>**, with **<MPI RANKS>** per node and **<THREADS PER RANK>**, over **<TIME STEPS>**.

```
mpirun -np <Num Nodes><MPI Ranks> ../../swift_mpi \
       --cosmology \
       --hydro \
       --self-gravity \
       --stars \
       --threads=<Threads per Rank> \
       -n <Time Steps> eagle_25.yml 2>&1 | tee eagle25.log
```

NOTE: The simulation will run for approximately **<TIME STEPS> ~ 5000**. For your investigation, you may use fewer time steps, to determine the optimal communication pattern. This must however be consistent across your observations. For your final run, you may omit the **-n** switch.

Each SWFT run will produce a `timestamps_<MPI Ranks><Threads per Rank>.txt` file. The wall clock time of interest in seconds, can be obtained by:

```
$ awk `BEGIN{tot=0} {tot += $11} END {print tot/1000}` \
       < timesteps.txt
```

You are required to submit your (1) all binary and output files for your SMP runs, as well as your (2) table and (3) SLURM batch and (4) output files of your *most efficient* run, for judging

Hint: You can try to improve your computational throughput by disabling snapshot dumps and recompiling your binary executable files. Edit `src/engine.c` and `examples/main.c` to comment out calls to the function `engine_dump_snapshot()`.