SSE and GPU with NBODY6

Keigo Nitadori and Sverre Aarseth

We provide some notes on the use of the SSE (Streaming SIMD Extension) and GPU (Graphics Processing Unit) versions for running `NBODY6`, first developed at the Institute of Astronomy in April 2008. As before, the standard code is compiled by typing `make nbody6` in the directory `Ncode` and there are no new routines inside the `Makefile`. For the GPU version, first type `make -f Makefile_gpu gpu` and ignore the error message (also `make nbody6` works). With single CPU use please check the array sizes defined in `params.h` to avoid saving excessive common blocks (256 Mb for NMAX = 100 K).

Hardware requirements for the SSE/GPU versions are either a multi-core CPU of the type x86 or x86_64 processors with SSE/SSE2 support and/or a GPU with CUDA support. Core2 Quad with 64-bit OS is recommended for high performance calculations with SSE. GeForce 8800 GTS/512 or GeForce 9800 GTX is adequate for single GPU.

Software requirement: GCC officially supports OpenMP (option `-fopenmp`) from version 4.2. However, CUDA 1.1 does not work with GCC 4.2. Hence in some distributions, GCC 4.2 is needed for host compilation and 4.1 for the GPU code. Since Fedora with GCC 4.1 supports OpenMP unoficially, it can be used for both compilations.

The directory `GPU` has a few extra Fortran routines which contain the new procedures, as well as some modified standard routines. The subdirectory `lib` holds the library. To obtain the GPU version `nbody6.gpu`, type `make gpu` in the directory `GPU` while `make sse` produces `nbody6.sse`. In both versions we use the OpenMP directives in some routines. The executables are sent to the run directory `Runs/Nbody6/GPU` but this can readily be changed as desired. Input for different simulations remain the same as before, with most options having the usual meaning (but see below).

Users can specify the number of threads per process by setting the environment variable `OMP_NUM_THREADS`. Since we use multiple cores, the CPU time given in the output may be larger than the wallclock time. The actual time for data send and gravity calculation is given on the screen at the end, together with the corresponding Gflops.

Some comments on the extra routines in dir GPU or `lib`.

gpunb.gpu.cu: main routine for GPU library in CUDA (dir `lib`).

intgrt.omp.f: integration flow control for GPU or SSE (also parallelized).

gpucor.f: regular force corrector and irregular force loop.

cmfirr2.f: irregular force on c.m. and regular force from c.m.'s.

cmfreg2.f: simplified version of regular c.m. force (no velocity criterion).

regint2.f: simplified regular force & corrector (rare cases of overflow on GPU).

kspert.f: KS perturbation force loop done by cnbint.cpp.

nbintp.f: parallel irregular force corrector with fast neighbour force.

nbint.f: irregular force corrector with fast neighbour force (blocks ¡ NPMAX).

cnbint.cpp: neighbour force loop (written in C++ with SSE).

adjust.f: standard energy check routine but calls energy2.f.

gpupot.cu: fast evaluation of all potentials on GPU (in dir `lib`).

energy2.f: summation of individual potentials after differential correction.

phicor.f: differential potential corrections due to binary interactions.

swap.f: randomized particle swapping at T = 0 (reduces crowding).

Optimization:

Optimized performance is achieved by minimizing the number of overflows which result in force evaluation on the host. However, small average neighbour numbers (also in #9 output line) may affect the accuracy. Based on preliminary tests, a relatively large value of NNBMAX and option #40 = 2 (or 3 for decreasing NNBMAX after escape) appears to be a good strategy. There is also an important parameter in `gpunb.gpu.cu` called NBMAX which is set to a maximum of 128. Smaller values (32 or 64) would be more efficient for short runs or equal-mass systems.

The fast routine cnbint.cpp is used by gpucor.f, nbint.f, nbintp.f and kspert.f. Note that all arguments are offset by -1 in cnbint.cpp for consistency with the C++ convention. Moreover, NNB in the force loops represents the neighbour number + 1 for historical reasons (and is also offset by -1 in cnbint).

Overflows:

Overflows may occur in two different ways. There are 16 blocks for use on the GPU. Thus if the maximum neighbour number is 400 and the indices are distributed evenly, there would be 400/16 members in each block. With a maximum block membership of NBMAX = 128, a random distribution would be within this range nearly all the time. However, crowding in the first bin due to mass segregation and terminated KS components with small time-steps may occur at later times. The former effect is alleviated by random swapping of particle indices (but *not* names) after data allocation.

Options:

In order to save time on the host (large N only), #38 = 2 restricts the neighbour force derivative corrections to 10 % regular force change, while for #38 = 1 all corrections are done. The CPU time may also be reduced by saving the common blocks on fort.1 only at every main output as a backup for rare restarts (#1 = 2 and #2 = 0). Option #40 >= 2 stabilizes the average neighbour number (in adjust.f) on a fraction of the maximum (e.g. NNBMAX/5) for small overflow numbers. The overflow counters (#9 OVERFLOWS; current and accumulated) at main output provide useful diagnostics of the behaviour (#33 >= 2). To be consistent with decreasing particle numbers, the maximum membership is reduced by a square root relation scaled by the initial value (#40 = 3).