SSE and GPU2 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 GPU2 version, type `make gpu` in dir `GPU2`.

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 `GPU2` has several extra Fortran routines which contain the new procedures, as well as some modified standard routines. The subdirectory `lib` holds the GPU library for regular force. To obtain the GPU2 version `nbody6.gpu`, type `make gpu` in the directory `GPU2` while `make sse` produces `nbody6.sse`. In both versions we use the OpenMP directives in some routines. The executable is sent to the run subdirectory `GPU2/run` which also contains simple input templates (or see dir `Docs`). Input for different simulations remains as before, with most options having the usual meaning.

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 in the output is larger than the wall-clock time in the ADJUST line. 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 GPU2 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).

repair.f: modification of array LISTQ after new or terminated KS.

jpred.f: standard prediction of & XDOT and resolved KS components (J > N).

cxvpred.cpp: full X & XDOT prediction in C++ except for TPRED(J)=TIME.

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

cmfirr.f: irregular force on perturbed c.m.'s.

cmfirr2.f: irregular force on singles from c.m.'s at regular force times.

kspert.f: KS perturbation force loops done in C++ by cnbint.cpp.

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

nbint.f: fast neighbour force (< NPMAX), corrector & decision-making.

cnbint.cpp: neighbour force loop (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 results in the last block members (< NIMAX) being recalculated. 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. 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.

Overflows:

Overflows may occur in two different ways. There are 32 blocks for use on the GPU. So if the maximum neighbour number is 400 and the indices are distributed evenly, there would be 400/32 members in each block. Thus a random distribution would be within the permitted 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 later. The former effect is alleviated by initial random swapping of particle indices (but *not* names) after data allocation. Hence neighbour lists are sequential with masses randomized.

Options:

In order to save time on the host (large N only), #38 = 2 restricts the neighbour force derivative corrections to 1 % 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).

GPUIRR

The new irregular library deals with prediction of active particles and evaluation of neighbour forces. In the case of no regular force calculation, the predictions are done by GPUIRR_PRED_ACT or PRED_ALL. Here the parameter NPACT is used to decide between a scalar and vector version, respectively. Irregular forces for active particles are obtained by the SSE vector procedure GPUIRR_FIRR_VEC. Finally corrected quantities are sent to this library by GPUIRR_SET_JP at the end of the cycle.