Debugging strategies for NBODY6/GPU (Sverre Aarseth 2/2011)

Although the GPU code has been tested extensively, a variety of problems may still occur when facing new situations. It is hoped that the following suggestions for tracking down some common types of problems will be useful. Explicit procedures will be presented which do not require a thorough familiarity with the code. Debugging strategy can be divided into two separate parts: (i) identify the precise origin (or nature) of the complaint, and (ii) fix the problem. In general, the second stage assumes some understanding of the code structure and may well be beyond the scope of many users. However, even pinpointing the source of trouble can be useful when accompanied by relevant documentation. For the purpose of identifying such behaviour, the start of a new block-step provides useful information. Depending on the outcome, several strategies are available. In the following we consider a few characteristic problem types that cause loss of accuracy or premature termination.

1. Assert error in `gpunb.reduce.cu`

This type of error usually means a `NaN` in the data sent to GPU. The following simple steps may be helpful.

Restart from the last good `common6` save, copied from `fort.2` or obtained directly from a recent save on `fort.1`. The diagnostic output at label 22 of `intgrt.omp.f` (including `CALL FLUSH`) should be activated, replacing the irregular time-step counter `NSTEPI` by the regular step counter `NSTEPR`.

Restart again with `WRITE (6,22)` suppressed and a loop from `IFIRST,NTOT` placed just before `CALL GPUNB_SEND` as defined by the last diagnostic value of `NSTEPR`. A `NaN` may then be identified for a given index `J` or `NAME(J)` using the test `IF(X(1,J).NE.X(1,J))`, followed by `STOP`, or try `XDOT(1,J)` instead if no luck. Note that the actual `NaN` may have been inflicted at a slightly earlier stage. It would then be necessary to consider `X` or `X0` before the full predictor loop `CALL CXVPRED`.

Now a third restart should be made, tracking the critical particle `JCRIT` in the regular force corrector `gpucor.f`. Some diagnostics can be added just before label 110, using `IF(NAME(I).EQ.NAME(JCRIT))`, followed by printing some relevant variables, e.g. `NSTEPR, NBGAIN, NBLOSS, NNB, FIRR(1), FREG(1), STEPR(I)`. Inspection of this information may then provide a clue to the nature of the problem, which calls for further investigation.

2. Stop on small time-step

After enforced stop with small `STEP(I)`, a similar strategy as above may be tried. Alternatively, the block-step information would show the pre-history, including the size of the active particle block. For example, only two particles repeatedly assigned to the block-step would suggest the offending particles are unsuccessful candidates for new KS or chain treatment. Further tracking as above should clarify the situation.

3. Infinite looping without output

Restart with same diagnostics as for Problem 1, where printing the last value of `NSTEPR` may also be useful. One attempt would now be to make a further search similar

to the third procedure above. However, the problem might be associated with KS or chain regularization. This can be ruled out by bisecting the `CALL SUBINT` procedure; i.e. placing a write statement (with `FLUSH` both before and after this call, using the last known time-step counter of any type as a starting point. If this attempt fails, a similar one may be tried for `CALL NBINT` and (if relevant) even bracketing the parallel irregular force loop. This approach should eventually lead to identification of the crucial procedure through further internal bisecting once the relevant routine has been isolated.

4. Force discontinuity

This type of problem is harder to resolve. In fact, the reasons for some of the other problems are often due to this type of behaviour. If option #38=1, restart with the following diagnostics just before the regular corrector `DO 75` in `gpucor.f`:

```
IF(NBGAIN+NBLOSS.EQ.0) THEN
WRITE NAME(I),NNB,FIRR(1)-FI(1,I),FIRR
```

Alternatively, the general bug trap at label 560 of `gpucor.f` for #38=1 (suppressed, together with FXI at the beginning) would also produce some information on force discontinuities. Note that the relative size of the irregular force discontinuity, as well as the absolute value is relevant. Strictly speaking the irregular force difference should be zero on no neighbour change but this cannot be guaranteed when binaries are present, in which case a small error may be tolerated. However, even small systematic errors may cause shrinking time-steps. Otherwise if #38=0 or 2, this test may also work for small and decreasing regular time-steps when the neighbour list tends to be unchanged. The case of force discontinuity with no change of neighbours could also be due to the retention of an old member, where its contribution is evaluated on the host in different precision and when the force is large (relative size $\leq 1D-07$). Further study of the force divergence would involve a detailed comparison of the irregular forces as obtained by `nbint.f` and `gpucor.f` at the same value of `TIME`.

5. General problem of increased energy errors

In principle this could be due to a wide variety of processes. The cause can be narrowed down by looking for sudden shrinkage of time-steps, in particular the regular time-step is more sensitive to numerical problems, which in turn leads to shrinkage of the corresponding irregular step at end of `gpucor.f`. In the first instance, the procedure for identifying sudden shrinkage of the natural regular time-step `DT0` in the corrector `gpucor.f` may be activated. Now pick the most obvious candidate (if any) and include diagnostics for tracking its pre-history as for Problem 1 above. Other problems that are reproduced after a restart may be pin-pointed by successive sub-divisions (factor of 2) of the output interval and appropriate common saves, using the bisecting principle. Note that in some cases reproducibility may well be lost because of extra predictions.

The sudden onset of time-step shrinkage may be correlated with some event as for example KS termination of a binary or long-lived triple, or a particular stellar evolution process (e.g. binary coalescence with mass loss). During such investigations it may be helpful to activate options for providing more information, like #10=2 for KS, #15=2 for hierarchies or #30=2 or =3 for chain regularization.

6. Segmentation error

This fatal error is relatively rare in a well tested code. One approach is to recompile the whole code with the bug trap `-fbounds-check` replacing the standard option in `Makefile_gpu`. If applicable, detailed information about over-writing a specific array is supplied. Hence the typical cause is for an array index on the host to be outside the defined range.

7. Difficult cases

A general star cluster simulation has to deal with a wide range of events or different configurations, some of which may lead to numerical difficulties. Among the main ones are long-lived hierarchical systems and close stellar interactions involving the common-envelope process or collision where mass-loss is a characteristic feature. Hierarchical systems which fail the stability test may still be long-lived and such configurations introduce systematic energy errors which also slows down the advance. Use of option #15=2 is helpful in identifying any critical triples or quadruples, while information about higher-order systems is provided regardless. Note that the stability of hierarchies is assessed frequently (every apocentre passage) so that even a slow secular change will signal termination if the boundary is crossed.

In astrophysical simulations, the interaction of eccentric binary components may present particular problems. Characteristically, such events are often triggered by strong perturbations. Hence any corrections for mass loss may involve large terms and consequent loss of accuracy. Force polynomial initialisation of neighbours adds to the numerical problems if, as usual, the block time-step is small. Tidal circularization is also a complicated process where two optional formulations have been implemented.

One common source of systematic errors is due to massive binaries that are slightly too wide to be regularized. Thus direct integration of such systems may be responsible for any drift in total energy (cf. `DETOT`). Alternatively, binaries may require more perturbers than available in the neighbour list. This type of problem is more often associated with small-N simulations.

8. Reproducibility

It is highly desirable that the results of a simulation can be reproduced. This is also important for problem identification. However, the interplay of different methods on the host and GPU creates complications and to this we should add the tricky business of parallel procedures which introduce additional complications over sequential treatment. At the simplest level, reproducibility is usually achieved when setting one thread only (`setenv OMP_NUM_THREADS 1`). A relatively short interval since the last common save may also increase the chance of reconstructing the desired outcome. Note that episodes involving chain regularization are especially sensitive to obtaining the correct detailed time-step history. As far as parallel loops are concerned, the treatment of each particle should only depend on the *predicted* quantities of other particles. One way to improve reproducibility is to suppress the parallel corrector loop in `intgrt.omp.f`, (`CALL GPUCOR`), when using full threads, albeit with some reduction in performance. It can be seen that procedures used by `gpucor.f` are consistent with full parallel treatment. However, additional factors could be responsible for any lack of success.