Rainer Spurzem

Parallel Computing with NBODY6++ with and without GPU

Friday, Aug. 3, 2017, Uni Heidelberg, GPU Block Course

1. Compiling and Running NBODY6++

1.1 Compiling and running the code on kepler

```
cd worknb6
module unload cuda
module load cuda/5.0
make clean ; make mpich
ls -ltr
```

You find the executable file nbody6, move it to Run/nbody6.gpu :

```
mv nbody6 Run/nbody6.mpi

make clean ; make mpichgpu_kepler
ls -ltr
```

You should find the executable file nbody6.gpu

```
mv nbody6.gpu Run/

cd Run/
ls -lrt
```

Here is the batch job script gpu_script.sh . You should run it with nbody6 for nodes=1,2,4,6 (gres=gpu:0) and with nbody6.gpu for nodes=1,2,4,6 (gres=gpu:1). Always use the student queue #SBATCH -p Student_GPU . Larger jobs (e.g. more than 6 nodes) will only run in the GPU queue (#SBATCH -p GPU) and may take long or very long waiting time. Also set time=00:30:00 in the job script.

Furthermore you have to select In the job script (mpirun command) whether to use nbody6.mpi or nbody6.gpu and also what is the name of an input and output file, see below.

The job script uses input files like in5000.comment to read certain parameters. We only care about the parameters N (initial paricle number) and TCRIT (termination time in N-body units). Please make sure that we have N and TCRIT set to 5000 and 5.0, respectively. The files in5000.comment, in10k.comment and in32k.comment are used for N=5000, 10000, 32000. For the course exercise you may use either in5000.comment (for N=5000) or in10k.comment (here N=10000), you have the free choice.

The files like in10k.ktg.sev switch on stellar evolution. This will include astrophysical stellar evolution and produce data for a Hertzsprung Russell diagram in sev.83. This is an optional experiment, not required for the course.

The code produces a lot of different output files. Most of them are not interesting for us. We will look mainly at the output listing such as e.g. out5000... With

`grep ADJUST out5000` (or whatever name you chose for out...)

you can see whether the run has done well. To extract the timing data relevant for the course exercise, find the lines below ADJUST, headed by “PE N ttot...” . Also interesting: the stellar evolution output file sev.83 (if switched on) – you get in it lines with following columns:

time, index, name, stellar type, pos. in cluster, mass, log luminosity, log radius, log effective temperature (last three in solar units).

1.2 Some more information

Usage of .F files:

intgrt.F is pre-processed with C-Preprocessor; it evaluates so-called preprocessor directives in the source code; they start with # , for example:

#ifdef PARALLEL
...
#endif

Preprocessor directives are selected with a compiler option:
-D PARALLEL compiles code between #ifdef PARALLEL and #endif.
Without -D PARALLEL these code lines will not be used!
WARNING - never keep .f if you have .F - the preprocessor directives will fail.

2 Parallel Communication Schemes and Literature

NBODY6++ runs in the SPMD (Single Program Multiple Data) Scheme. It means when you start the parallel NBODY6++ run on n cores (by using the command (`mpirun -np n …`), n identical copies of the program will start. In parallel sections these copies of the code share their work and communicate data with each other through the MPI functions in the code.

NBODY6++ uses for communication a copy algorithm (all new information is copied immediately to all nodes); other algorithms are ring algorithm or (hyper)systolic algorithm, see Dorband, Hemsendorf, Merritt, 2003 (Journ. Comp. Phys.); Makino (2002). If the number of particles per node is large enough, all algorithms scale equally well. The similar but simpler phiGRAPE and phiGPU codes by Berczik and others (see e.g. Harfst et al. 2007, New Astronomy) use a mixed algorithm.

The copy algorithm in NBODY6++ is implemented manually with MPI_SENDRECV. Current modern implementations of MPI_BCAST will be equally efficient.

3 Hands-On Experiment on parallel computer

3.1 Profiling for NBODY6/6++

The code measures the wall clock time used for many things:

total, regular force, irregular force, adjust, regularised, prediction, overhead for parallelisation, communication time…

Your task: Do some experiment - run on 1,2,4,6… processors, with and without GPU usage, as explained above. Find, cut and paste the lines below the timing header (“PE N ttot..”). Use the last one in your job (after ADJUST TIME= 5.00).

Explanation of times in output (line below 'PE N'):

```
ttot: total  wallclock time
treg: regint, regular force (PAR)
tirr: nbint, neighbour force (PAR)
tadj: energy check (PAR)
tinit: computing of initial model (PAR)
tsub,tsub2: communication time using MPI_SENDRECV
xtsub1,xtsub2: number of bytes transferred
```

(PAR) means these routines are parallelised (contain shared work and MPI functions); there are more times listed, but we do not need them here. treg, tirr, tinit and tadj are required to determine X (see below); ttot, treg, tirr and tsub+tsub2 should be plotted as a function of number of nodes used (1,2,4,6), both for MPI and GPU jobs. What is the maximum speedup we get, without GPU, with GPU? What is the prediction of Amdahl's law? Note that the speed-up should be measured relative to the single node non-GPU case; you may try (for the GPU case) to use the single node GPU run as a basis for speedup computation, but remember that the GPU is already a parallel computing device (with p ~ 2500!), so using a single node with GPU is not really a sequential run.

3.2 Example Solution for NBODY6++ Tasks:

Here are my time measurements
(taken from output files out.....):

PE	N	ttot	treg	tirr	tpredtot	tint	tinit	tks	ttcomm	tadj	tmov	tprednb	tsub	tsub2	xtsub1	xtsub2	
1	5000	517.01170	447.73	43.23	0.00	504.89	4.20	0.22	0.00	7.83	1.46	6.85	0.00	0.00	0.00000D+00	0.00000D+00	
2	5000	273.74747	226.19	24.01	0.00	266.98	2.28	0.23	0.00	4.38	3.94	6.86	0.69	1.16	2.35923D+09	3.00958D+09	
4	5000	154.28701	110.07	14.23	0.00	150.33	1.29	0.24	0.00	2.56	8.72	6.82	1.72	2.39	3.53682D+09	4.51333D+09	
6	5000	110.29107	74.53	9.87	0.00	107.22	0.98	0.22	0.00	1.99	8.29	6.70	2.15	1.89	3.92945D+09	5.01460D+09	

Calculate Amdahl's Law:

Let X be the part of my program (in terms of computing time) which can be parallelised. The sequential computing time Tseq is normalized to unity (1), and can be expressed as:

Tseq = 1 = X + (1-X)

The parallel computing time Tpar under ideal conditions (ideal load balancing, ultrafast communication):

Tpar = X/p + (1-X) with number of processes (number of GPUs) p

Then the speed-up of the program S = Tseq / Tpar :

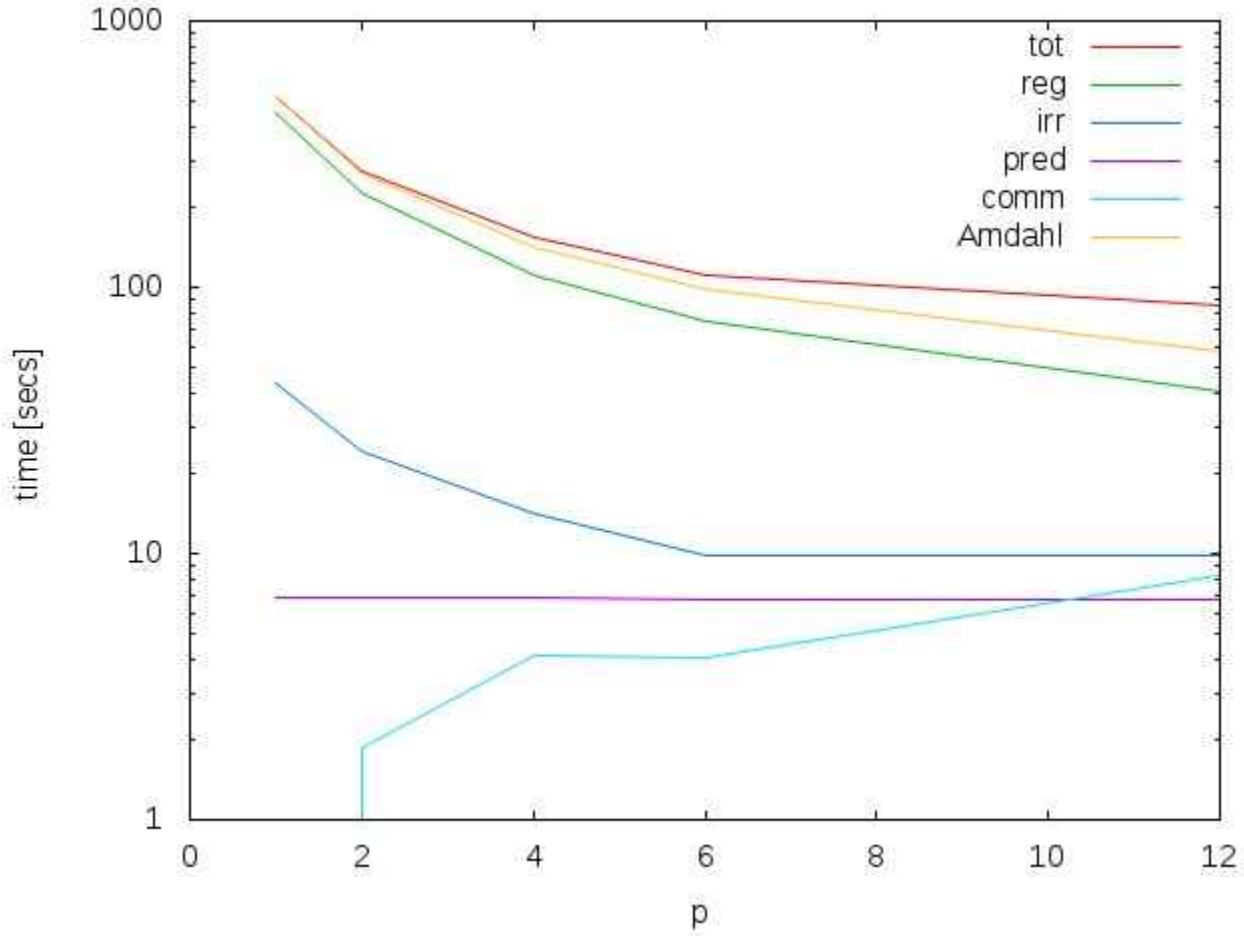
S = 1 / (1-X+X/p)

Note the limit if p is very large: S = 1/(1-X). We find from our measurements with NBODY6++ given above: X = (treg + tirr + tinit + tadj) / ttot = 503 / 517 = 0.97 . Hence S = 1/(0.03 + 0.97/p), for large p max speed-up: S = 1/0.03 = 33.3333 (Note: this is only for 5000 Particles - for larger N we get MUCH higher X...) So, if we know X and p we can compute a predicted speedup S, and with that predict a parallel computing time by using
Tpar,p = Tseq/S
The times Tpar,p are used to make the plotted line for Amdahl's law below..

Use gnuplot:

```
set logscale y
plot 'time' u 1:3 w l t'tot', '' u 1:6 w l t 'reg', '' u 1:7 w l t'irr', \
                        '' u 1:($13+$14) w l t'comm', '' u 1:(517.*(0.03+0.97/$1)) w l lt 9 t'Amdahl'
```

See an example result below. Your results may deviate, look less nice, this is usually due to the high load of the kepler computer during the course. If you have successfully finished the 8 runs and collected your data, no matter whether they look good or bad, it is ok to pass the course.



Summary of your tasks to pass the course; please turn in the following results to spurzem@ari.uni-heidelberg.de (no deadline given)

1.) Two plots showing the times ttot, treg, tirr and tsub+tsub2 (communication time) as a function of p – one for MPI, one for GPU jobs. Plot also the predicted times obtained from Amdahl's law.
2.) A data file containing the data you have used for 1; or a notice where I can find the file on kepler. A notice where I can find the output file of all your 8 runs.
3.) A few (one, two, three...) sentences for interpretation: How good is Amdahl's law working? How good works GPU acceleration? Did you get outliers / bad results which do not match expectations? Anything else you like to mention. And please also questions if there are.