Rainer Spurzem

Parallel Computing with NBODY6++
with and without GPU

Thursday, February 23, 2023, Uni Heidelberg, GPU Block Course

1. Compiling and Running NBODY6++

1.1 Compiling and running the code on kepler

```
tar xvfz gworknb6.tar.gz
cd gworknb6
module purge
module load cuda/7.5
make clean ; make -j mpich
ls -ltr
```

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

```
mv -i nbody6.mpi Run/

make clean ; make -j mpichgpu_kepler
ls -ltr
```

You should find the executable file nbody6.gpu

```
mv -i nbody6.gpu Run/

cd Run/
ls -lrt
```

Here is the batch job script mpi_script.sh and gpu_script.sh . You should run it with nbody6.mpi for nodes=1,2,3,4 (gres=gpu:0) and with nbody6.gpu for nodes=1,2,3,4 (gres=gpu:1). Always use the student queues #SBATCH -p Student_CPU and Student_GPU .

The standard job prepared for you uses 16000 particles, and simulation time of 5 model units (check output how many years). The input file containing some control data is in16k.comment.

For our course you will need outputs of 8 jobs (mpi and gpu, 1,2,3,4 nodes), 16000 particles, 5 time units. Using more particles is also ok, but be considerate to all of us, the runs take some time, we have many runs, and the machine will be full.

Additional voluntary study (not required):

Our runs contain astrophysical stellar evolution, with stars ranging from initially 0.08 to 100 solar masses. Data for a Hertzsprung Russell diagram (HRD, temperature/luminosity) are in sev.83_n (here n stands for the time, like 0,1,2,3,4,5 time units). In this file you get 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).

Astronomers like to plot log(luminosity) as a function of effective temperature (high T left, low T right on the x-axis), that is called an HRD.

The code also produces a lot of other output files. Most of them are not interesting for us. We will look mainly at the output listing such as e.g. out16k.xxx.nnnnn ... Here xxx is gpu or mpi, and nnnnn is a job id number.

```
grep ADJUST out16k...xxx.nnnnn
```

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...” .

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 nodes (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 a software package called MPI (message passing

interface). Every node is using its GPU.

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

3 Hands-On Experiment on parallel computer

3.1 Profiling for NBODY6++

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 1,2,3,4... processes, 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)
tprednb: prediction for Hermite scheme
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, tprednb, and tcomm = 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 result comes from Amdahl's law? Note that the speed-up should be measured relative to the single node case;

3.2 Example Solution for NBODY6++ Tasks:

Here are some example time measurements (taken from an older 5000 body simulation output files...):

PE	N	ttot	treg	tirr	tprednb	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:

$T_{seq} = 1 = X + (1-X)$

The parallel computing time T_{par} under ideal conditions (ideal load balancing, ultrafast communication):

$T_{par} = X/p + (1-X)$ with number of nodes (number of GPUs) p

Then the speed-up of the program $S = T_{seq} / T_{par}$:

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

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

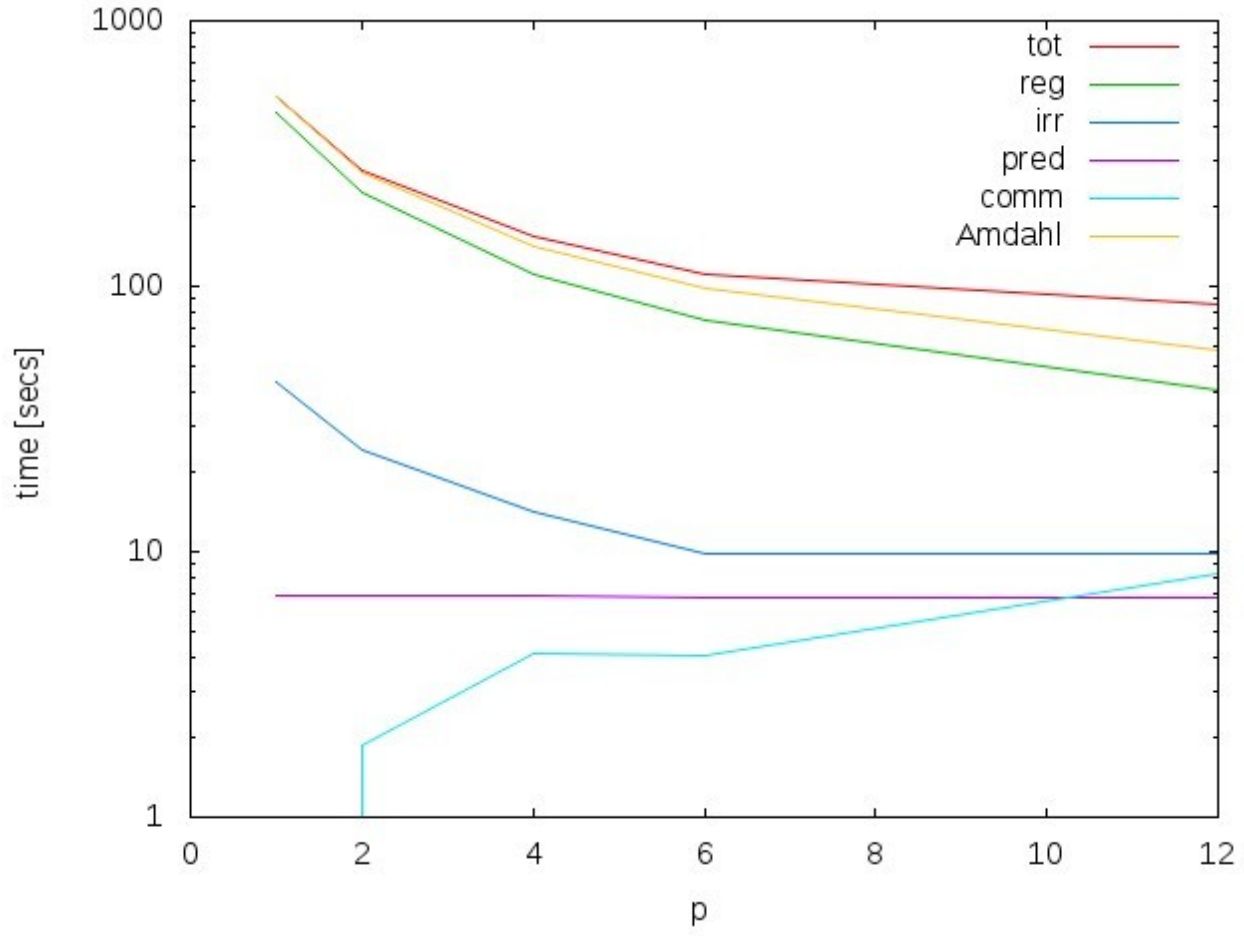
$T_{par,p} = T_{seq}/S$

The times $T_{par,p}$ are used to make the plotted line for Amdahl's law below..

Use gnuplot (for example):

```
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 (especially because you can only run 1,2,3,4), 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 your tutors:

1.) Two plots showing the times ttot, treg, tirr, tprednb, and tcomm = 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 we can find the file on kepler. **Please do not delete the output files of your 8 runs, because they are proof that you did the experiment.**
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.