Rainer Spurzem

Parallel Computing with NBODY6++ with and without GPU

Thursday, July 28, 2016, Uni Heidelberg, GPU Block Course

1. Compiling and Running NBODY6++

1.1 Copy and run the code on kepler

```
rsync -av ~spurzem/worknb6/ worknb6/
cd worknb6
module load cuda/5.0
module load openmpi-x86_64
make clean ; make mpich
ls -ltr
```

You should find the executable file nbody6.

```
mv nbody6 Run/
```

```
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 (gres=gpu:0) and with nbody6.gpu for nodes=1,2,4 (gres=gpu:1). Always use the student queue #SBATCH -p Student_GPU . Larger jobs (e.g. with nodes 6,8) 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.

The standard input file in5000.comment should be used normally. You can also try some of the other input files, e.g. in10k.ktg.sev . This will include astrophysical stellar evolution and produce data for a Hertzsprung Russell diagram in sev.83.

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,... processors, with and without GPU usage, as explained above

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)
tpred, tprednb: prediction
tmov: overhead for data move
tsub,tsub2: communication time using MPI_SENDRECV
xtsub1,xtsub2: number of bytes transferred
```

(PAR) means these routines are parallelised (contain shared work and MPI function).

Plot results for ttot, treg, tirr and tsub+tsub2 as a function of number of nodes used (1,2,4). What is the maximum speedup we get, without GPU, with GPU? What is the prediction of Amdahl's law? Compare. Normalize the speedup for all runs (with and without GPU) always to the one node run without GPU.

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
 12  5000     85.25265    40.48     9.90     0.00     82.62     0.69      0.22     0.00     1.83    14.99     6.74     4.48     3.72  4.32197D+09  5.51592D+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

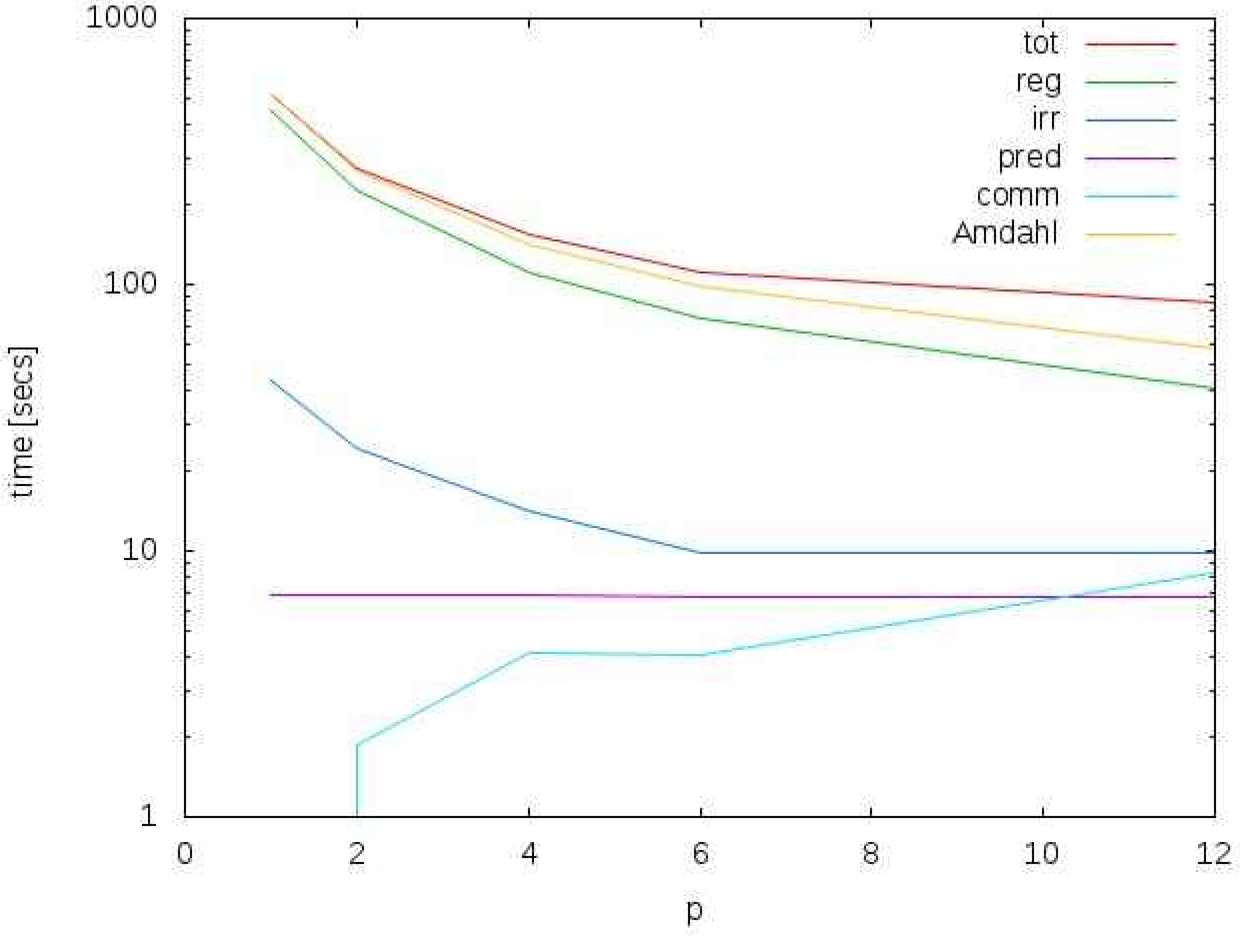
Then the speed-up of the program S = Tseq / Tpar :

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...)

Use gnuplot:

```
set logscale y
plot 'time' u 1:3 w l t'tot', '' u 1:4 w l t 'reg', '' u 1:5 w l t'irr', '' u 1:13 w l t'pred', \
                        '' u 1:($14+$15) w l t'comm', '' u 1:(517.*(0.03+0.97/$1)) w l lt 9 t'Amdahl'
```


1000
100
10
1
0
2
4
6
8
10
12
p
time [secs]
tot
reg
irr
pred
comm
Amdahl