MOLECULAR DYNAMICS ON GPUs

13.05.2014 – Kaiserslautern –

Dipl.–Biophys. Christian Mücksch

Technische Universität Kaiserslautern

Fachbereich Physik
AG Urbassek

OUTLINE

- Motivation
- Hardware
- Benchmarks
- Conclusions

MOTIVATION

A little history

- late 1980s and early 1990s more usage of graphical operating systems (e. g. Microsoft Windows) → use of display accelerator cards (hardware assisted bitmap operations)
- 1992: OpenGL library as platform independent method for 3D graphics applications
- mid–1990s: first–person computer games such as Doom, Duke Nukem 3D, and Quake accelerated the use of 3D graphics in consumer computing significantly → need for more realistic 3D environment
- 1999: release of NVIDIA‘s GeForce 256. First ‘real‘ GPU (Graphics Processing Unit) to perform transform and lighting processing on it‘s own processor → beginning of implementing more and more of the graphics pipeline directly on the graphics processor

MOTIVATION

A little history

- 2001: NVIDIA GeForce 3 series implemented Microsoft's then–new DirectX 8.0 standard and with it the possibility of programmable graphics operations (each pixel can be processed) → control over exact computations performed on the GPU

- 2006/2007: NVIDIA's **CUDA** (Compute Unified Device Architecture) platform with CUDA C compiler allowed general–purpose computations without the need to express/ mask arithmetic computations as graphical operations (shading languages)

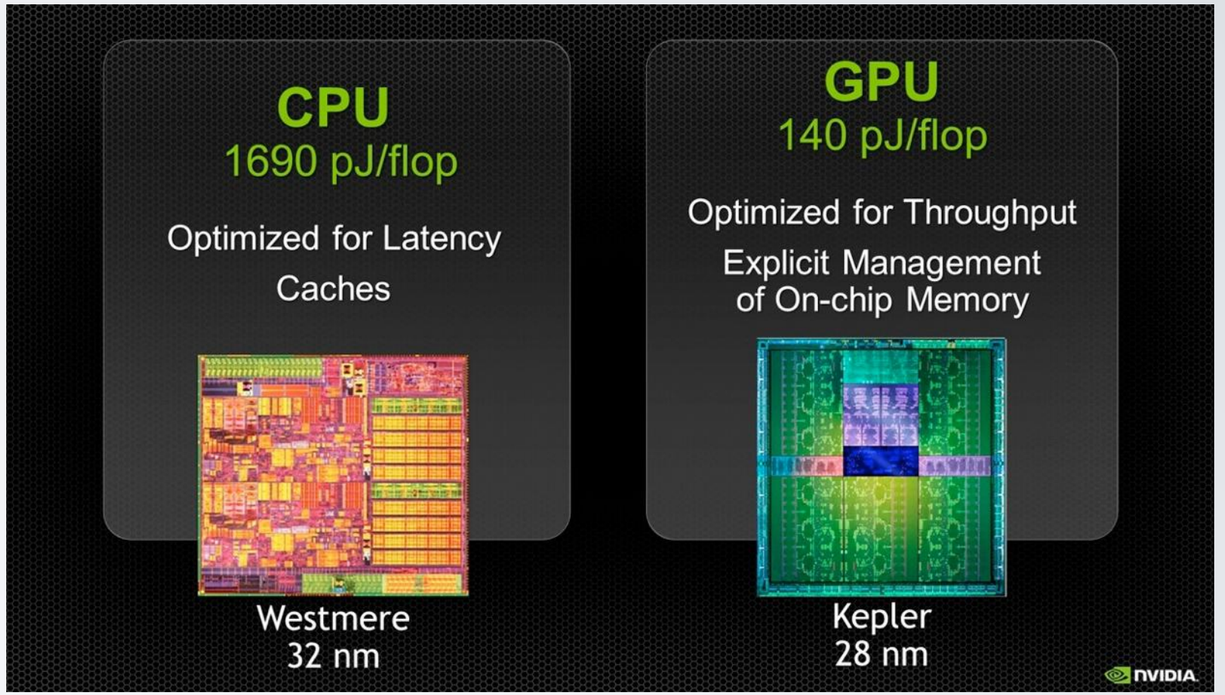
- to conclude: GPUs were initially used for memory–intensive work (**matrix and vector operations**) for 3D graphics. They evolved into highly–parallel multi–core systems (stream processors) which after introduction of APIs such as CUDA, OpenCL etc. became available for non–graphical computations.

MOTIVATION

Parallel computing

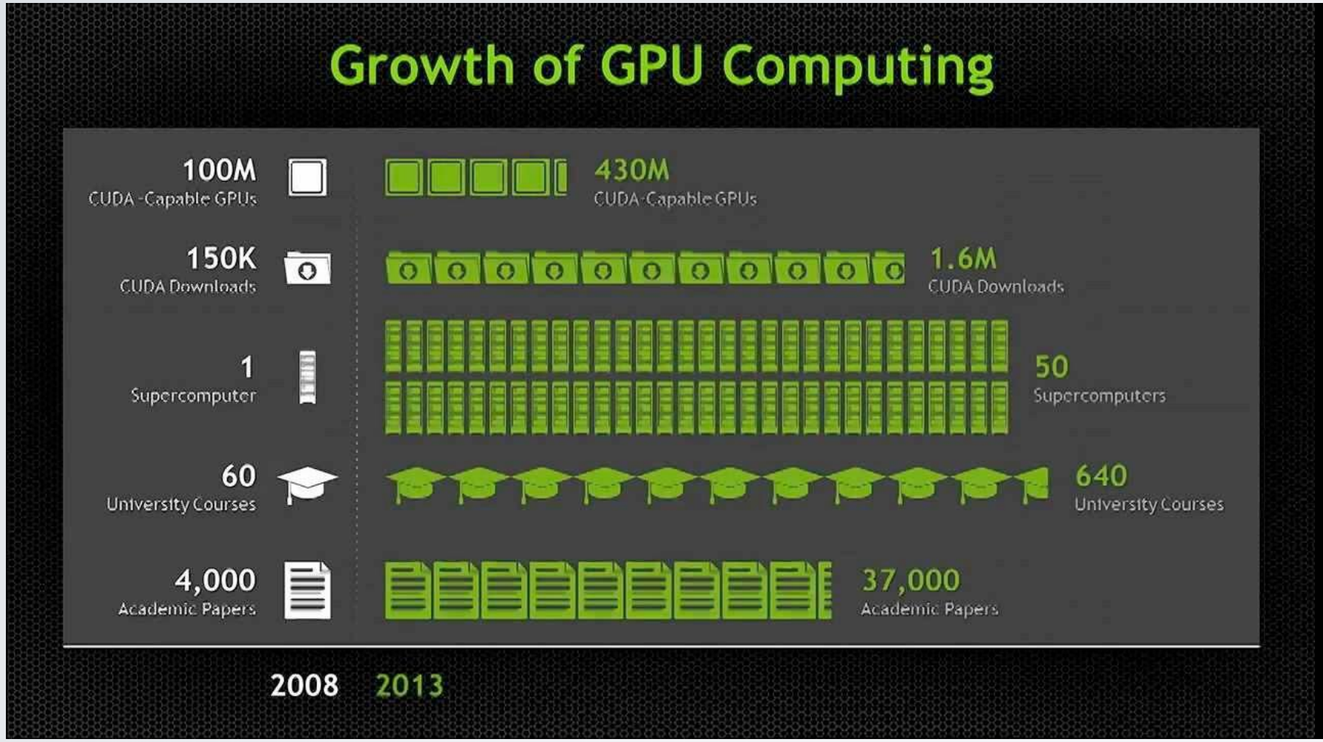
- Limitations of CPU fabrication: no increase in clock speed any more due to power and heat restrictions as well as physical limit of transistor size
  → supercomputers for decades increased performance by increasing number of processors

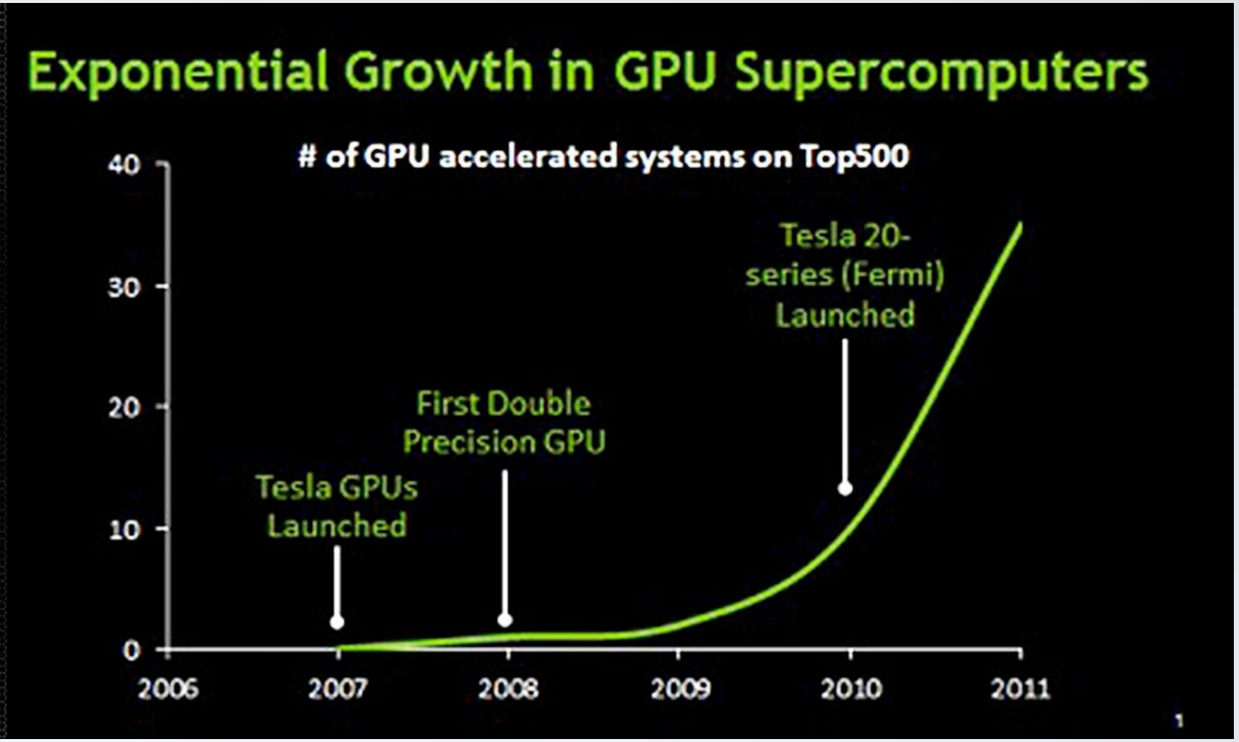
- GPUs more efficient in terms of energy consumption (FLOPS/Watt) and economy (FLOPS/Price) than CPUs
  → towards **exascale computing** power efficiency will be one major point



→ Top10 of Green500 supercomputers (Nov. 2013) are all heterogenous systems comprised of CPUs and GPUs

MOTIVATION


Growth of GPU Computing
100M
CUDA -Capable GPUs
430M
CUDA-Capable GPUs
150K
CUDA Downloads
1.6M
CUDA Downloads
1
Supercomputer
50
Supercomputers
60
University Courses
640
University Courses
4,000
Academic Papers
37,000
Academic Papers
2008
2013


Exponential Growth in GPU Supercomputers
# of GPU accelerated systems on Top500
40
30
20
10
0
2006
2007
2008
2009
2010
2011
Tesla GPUs Launched
First Double Precision GPU
Tesla 20-series (Fermi) Launched

HARDWARE

Our GPU node:

- 2x Intel® Xeon® Ivy Bridge Processors E5–2620v2 (6 cores)
  → 24 cores with hyperthreading
- 2x NVIDIA Tesla K20Xm
  - Kepler Architecture: see http://www.nvidia.com/object/nvidia-kepler.html
  - 2688 CUDA cores
  - 6 GB GDDR5
  - 250 GB/s Memory Bandwith
  - 3.95 TFlops peak performance in single precision and 1.31 TFlops with double precision
- 32 GB DDR3 RAM
- total price of node: 8840 €
- available via new login4 on our cluster: Port 42314

BENCHMARKS

The following GPU implementations into MD–code were tested:

- GROMACS
- NAMD
- LAMMPS

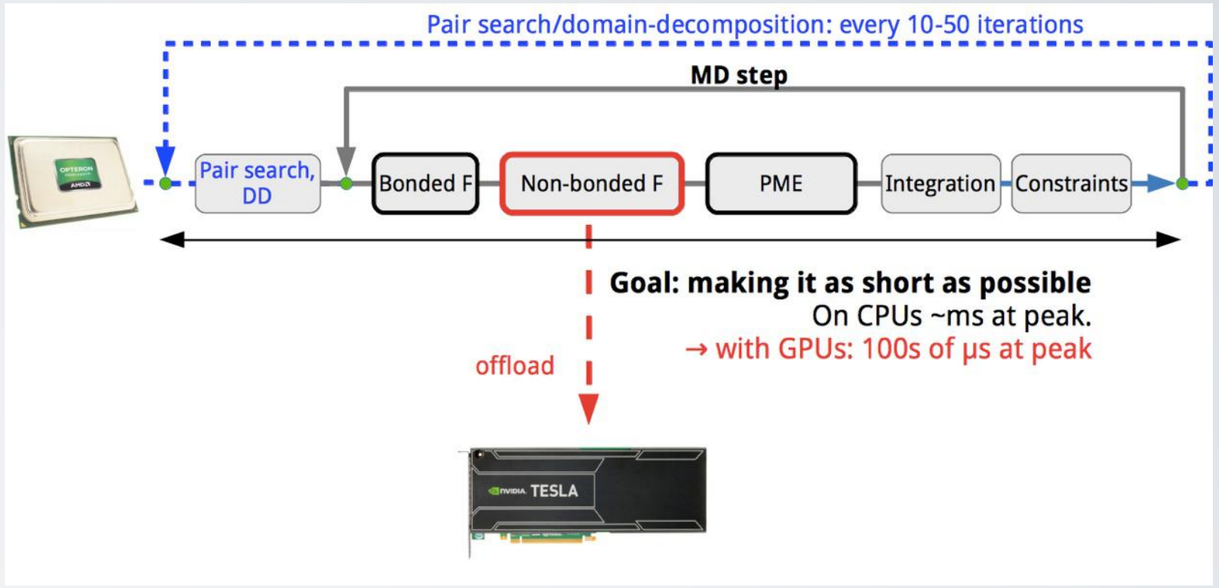
BENCHMARKS

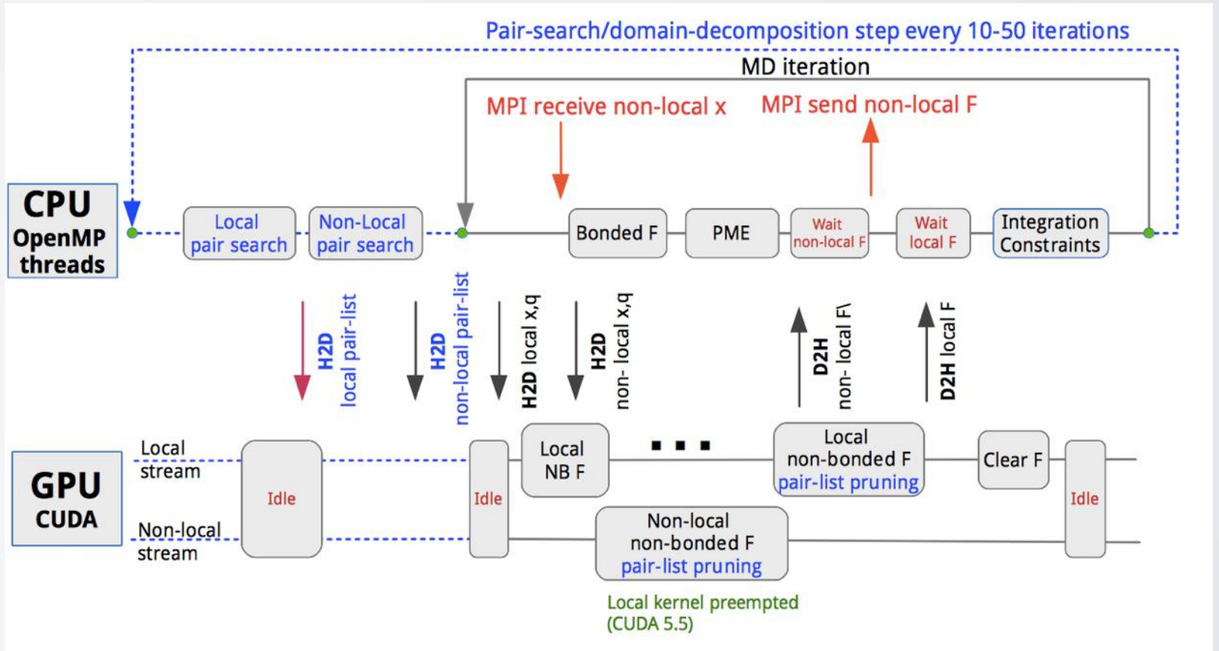
GROMACS:

- GROMACS since version 4.6 (Jan. 2013) has a new, native GPU acceleration developed through funding from a grant of the European Research Council (#209825)
  → **algorithmic redesign of pair search**

- speedup: the most compute–intensive part, the **non–bonding interactions**, are offloaded to the GPU, while the CPU does bonded forces and lattice summation (PME) in the mean time
  → good ratio of CPUs/GPU for a job increases performance

- to use the GPU acceleration: just add `cutoff-scheme = verlet` to your script and compile GPU version





BENCHMARKS

NAMD:

- NAMD started GPU support very early in 2007
- similar to GROMACS the nonbonded–force evaluations (most computationally expensive) are offloaded to the GPU
- at least one thread per GPU
  → multiple threads can share a single GPU which increases performance as well
- no changes to the runscript necessary, only pre–built GPU binary or compiled GPU version of NAMD needed

WILEY InterScience® DISCOVER SOMETHING GREAT

Accelerating Molecular Modeling Applications with Graphics Processors

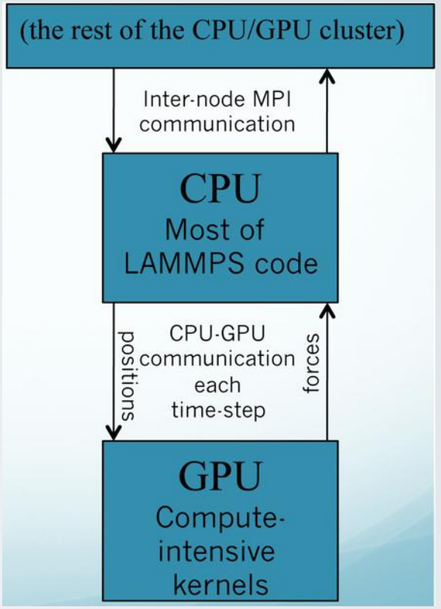
JOHN E. STONE,[1*] JAMES C. PHILLIPS,[1*] PETER L. FREDDOLINO,[1,2*] DAVID J. HARDY,[1*] LEONARDO G. TRABUCO,[1,2] KLAUS SCHULTEN[1,2,3]

[1]*Beckman Institute, University of Illinois at Urbana-Champaign, Urbana, Illinois, 61801*
[2]*Center for Biophysics and Computational Biology, University of Illinois at Urbana-Champaign, Urbana, Illinois, 61801*
[3]*Department of Physics, University of Illinois at Urbana-Champaign, Urbana, Illinois, 61801*

Received 5 April 2007; Revised 27 June 2007; Accepted 30 July 2007
DOI 10.1002/jcc.20829
Published online 25 September 2007 in Wiley InterScience (www.interscience.wiley.com).

BENCHMARKS

LAMMPS – 2 Implementations:



- *gpu* package implemented by W. Michael Brown and collaborators in 2010 (Oak Ridge National Laboratories)
  - mutliple CPUs can share a single GPU
  - data is moved between GPU and CPU every time step
  - only accelerates pair force, neighbor list, and PPPM calculations
  - check: https://www.olcf.ornl.gov/wp-content/uploads/2013/02/Porting-LAMMPS-to-Titan.pdf

- *USER–CUDA* package implemented by Christian Trott in 2011 (PhD at TU Ilmenau, since 2012: PostDoc at Sandia National Laboratories)
  - calculation is run for many timesteps (until a CPU calculation is required) entirely on the GPU → one CPU per GPU
  - supports a wider range of pair styles and can also accelerate many fix styles and some compute styles, as well as neighbor list and PPPM calculations

BENCHMARKS

LAMMPS *gpu*:

- compile LAMMPS with package *gpu*
- two lines in runscript necessary: `package gpu force/neigh 0 1 1` and `newton off`; run job with `-suffix gpu`
- available potentials (February 2014):
  - beck
  - born_coul_long
  - born_coul_wolf
  - born
  - buck_coul_cut
  - buck_coul_long
  - buck
  - coul_dsf
  - coul_long
  - eam_alloy
  - eam_fs
  - eam
  - gauss
  - lj96_cut
  - lj_charmm_coul_long
  - lj_cut_coul_cut
  - lj_cut_coul_debye
  - lj_cut_coul_dsf
  - lj_cut_coul_long
  - lj_cut_coul_msm
  - lj_cut
  - lj_expand
  - lj_gromacs
  - mie_cut
  - morse
  - soft
  - sw
  - table
  - yukawa
  - pppm
- downsides (for this moment):
  - uses more GPU memory than the USER–CUDA package
  - much slower (in all my tests)

BENCHMARKS

LAMMPS *USER–CUDA*:

- compile LAMMPS with package *USER–CUDA*
- one line in runscript necessary: `package cuda gpu/node 2` (= 2 GPUs) and run job with `-suffix cuda`
- available potentials (February 2014):
  - born_coul_long
  - buck_coul_cut
  - buck_coul_long
  - buck
  - eam_alloy
  - eam
  - eam_fs
  - gran_hooke
  - lj96_cut
  - lj_charmm_coul_charmm
  - lj_charmm_coul_charmm_implicit
  - lj_charmm_coul_long
  - lj_class2_coul_cut
  - lj_class2_coul_long
  - lj_class2
  - lj_cut_coul_cut
  - lj_cut_coul_debye
  - lj_cut_coul_long
  - lj_cut
  - lj_cut_experimental
  - lj_expand
  - lj_gromacs_coul_gromacs
  - lj_gromacs
  - lj_sdk_coul_long
  - lj_sdk
  - lj_smooth
  - morse
  - sw
  - tersoff
  - tersoff_zbl
- downsides (for this moment):
  - no acceleration for minimization
  - no hybrid pair styles

BENCHMARKS

LAMMPS *USER–CUDA*:

example runscript on our cluster:

```
#!/bin/bash
#PBS -l nodes=1:ppn=2:gpus=2
#PBS -l walltime=1024:00:00
#PBS -j oe
#PBS -o anyoutput
#PBS -N GPU_bench

cd $PBS_O_WORKDIR

(
NP=`wc -l $PBS_NODEFILE|awk '{print$1}'`

mpirun -np $NP lmp_openmpi_cuda < thinfilm500K.eam -suffix cuda -cuda on

)>> output 2>&1
```

BENCHMARKS

LAMMPS *USER–CUDA*:

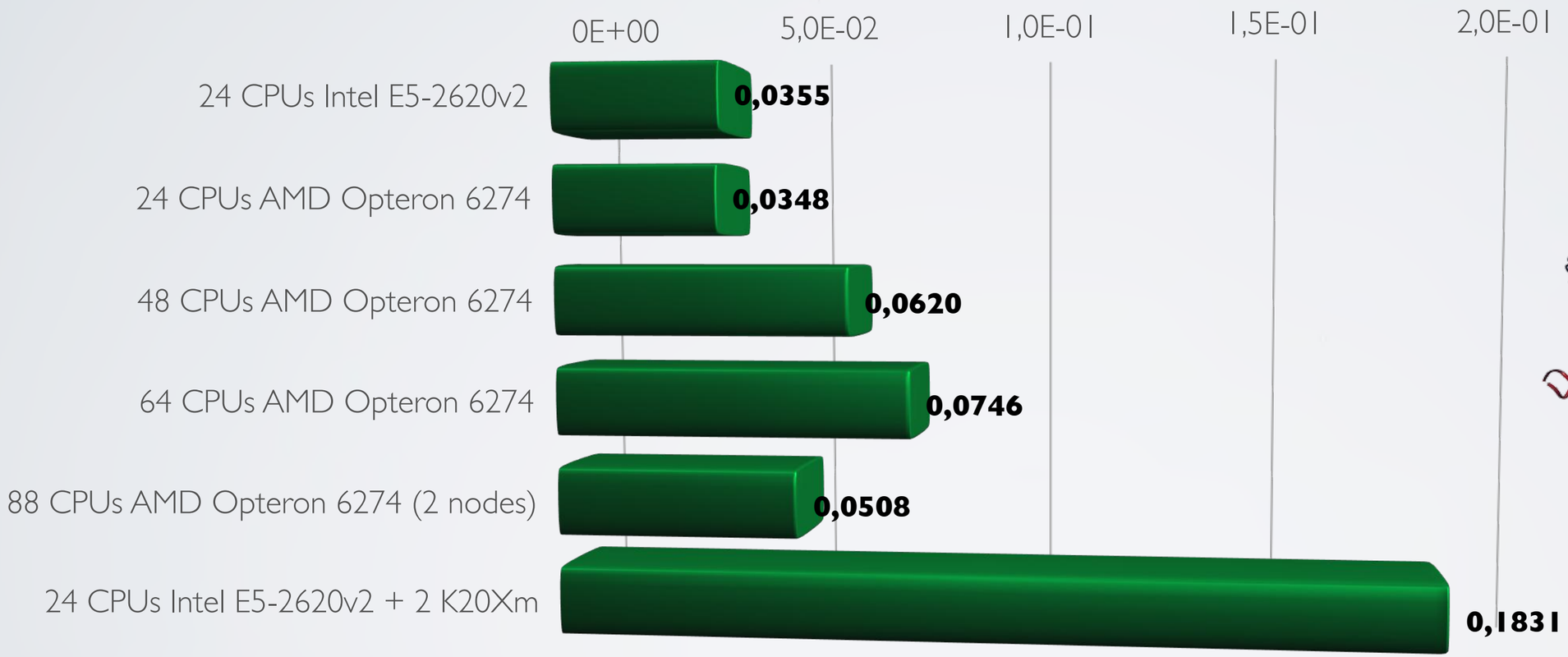
example runscript (by Emilia):

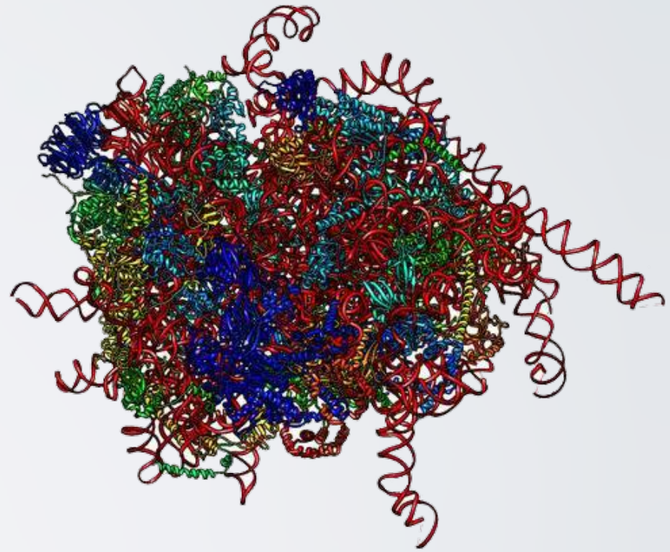
```
package          cuda gpu/node 2

units            metal
atom_style       atomic

lattice          bcc 2.87 orient x 0 -1 -1 orient y 0 1 -1 orient z 1 0 0
region           box block 0 49 0 49 0 31
boundary         p p s

create_box       1 box
create_atoms     1 region box

pair_style       eam/alloy
pair_coeff       * * Meyer.setfl Fe
mass             * 55.845

velocity         all create 800 4928459 mom yes rot yes dist gaussian

thermo           100

fix              1 all npt temp 500 500 2.0 x 0 0 20.0 y 0 0 20.0 drag 10.0
run              50000
```

BENCHMARKS – RESULTS

NAMD:

- Ribosomal protein in water: ~2.5 million atoms

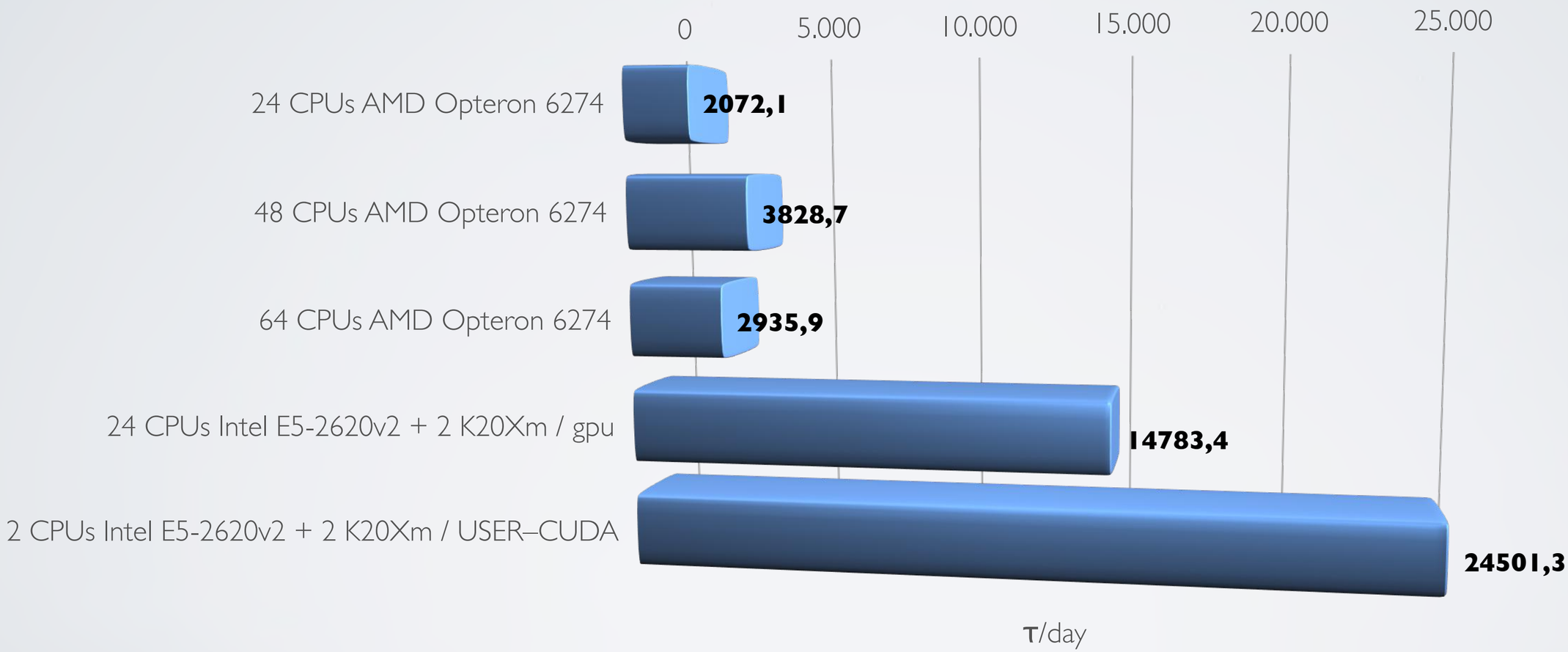




BENCHMARKS – RESULTS

LAMMPS:

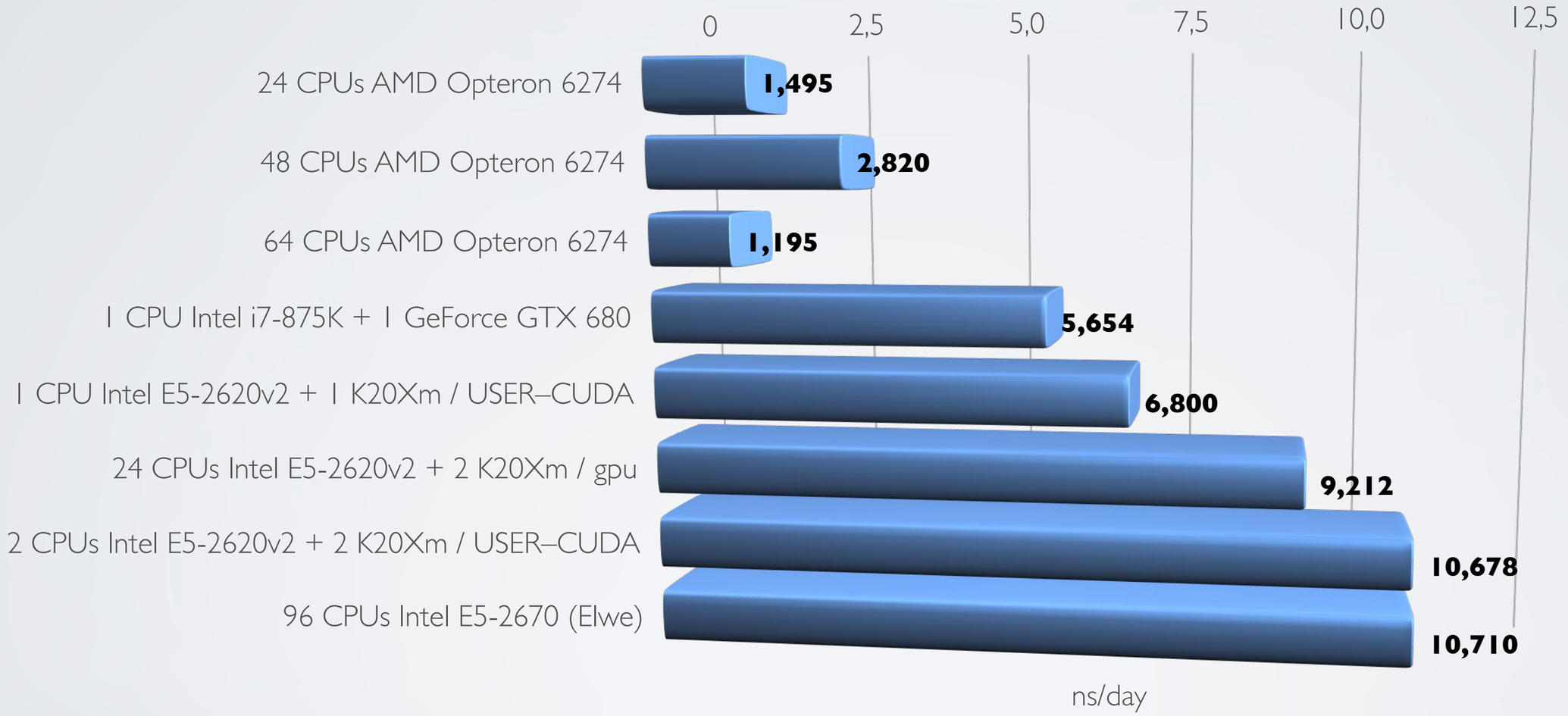
- Lennard–Jones melt: ~2 million atoms



BENCHMARKS – RESULTS

LAMMPS:

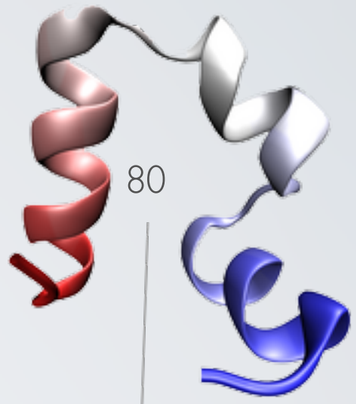
Iron thin film with EAM (by Emilia): ~300k atoms



BENCHMARKS – RESULTS

NAMD vs. GROMACS vs. LAMMPS: Protein folding simulation on the GPU

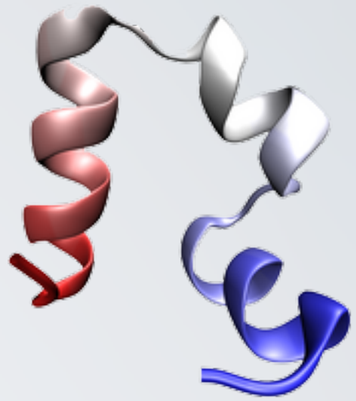
Villin headpiece protein using CHARMM and TIP3P water: ~30k atoms



BENCHMARKS – RESULTS

NAMD vs. GROMACS vs. LAMMPS: Protein folding simulation on the GPU

Villin headpiece protein using CHARMM and TIP3P water: ~30k atoms



- GROMACS most optimized GPU code → 1 μs folding simulation performed!
- LAMMPS code not specifically optimized for protein simulation
- Intel Xeon Phi not suited for MD simulation (LAMMPS code) at this moment → slower than CPU

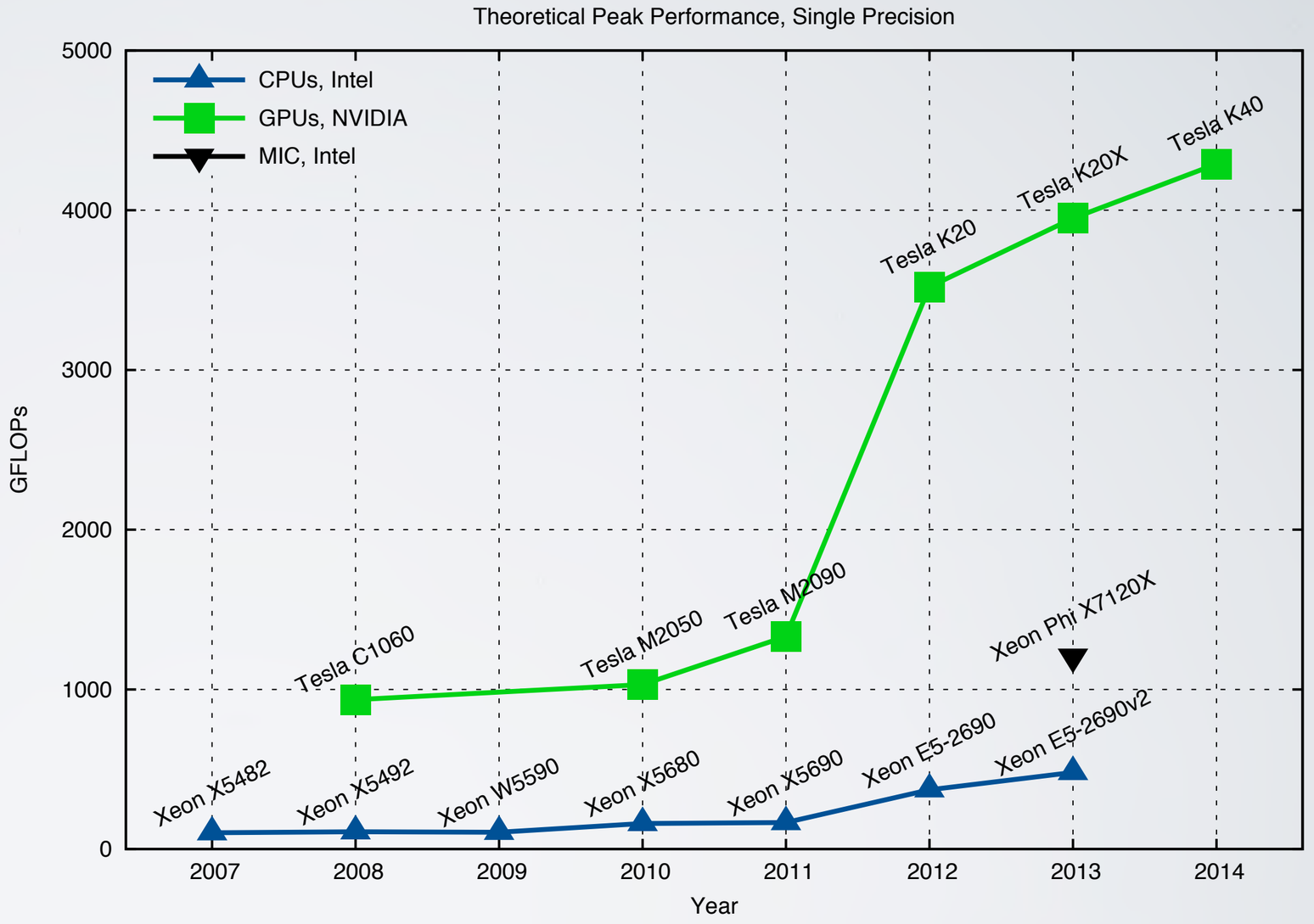
CONCLUSIONS

- GPU has a bright future in high performance computing and is well suited for MD simulations
  → GPU performance still increases:
  → MD–codes will improve GPU–support further

- for LAMMPS package USER–CUDA recommended

- one GPU node in our cluster faster than the whole cluster
  → try it out!

- CUDA courses offered by AHRP
  → useful for accelerating your own programs



“You simply can‘t get these levels of performance, power– and cost–efficiency with conventional CPU–based architectures. Accelerated computing is the best and most realistic approach to enable exascale performance levels within the next decade.“
— Steve Scott, NVIDIA chief technology officer

THANK YOU FOR YOUR ATTENTION!