Applications' usage in HPC2N

P. Ojeda, B. B. Brydsö, J. Eriksson

Why do we need HPC?

```
do j=1,N
      do i=1,N
        do k=1,N
           p(i,j) = p(i,j) + v1(i,k)* v2(k,j)
        enddo
      enddo
enddo
```

Heavy loops

MD, QM, CFD, …

```
!$omp parallel do private(i,j,k) shared(p,v1,v2) collapse(2)
     do j=1,N
       do i=1,N
         s=0.0d0
         do k=1,N
            s = s + v1(i,k)* v2(k,j)
         enddo
         p(i,j) = s
       enddo
     enddo
!$omp end parallel do
```

Parallelized loops

Processes communication

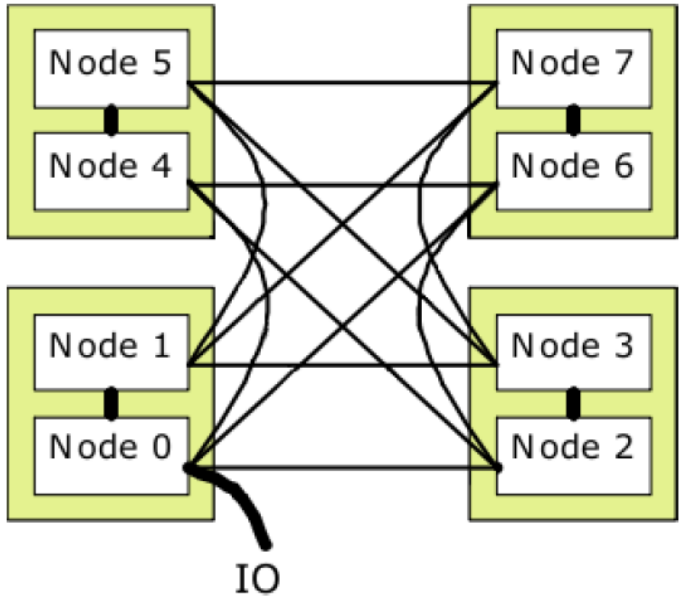


Figure : Nodes.

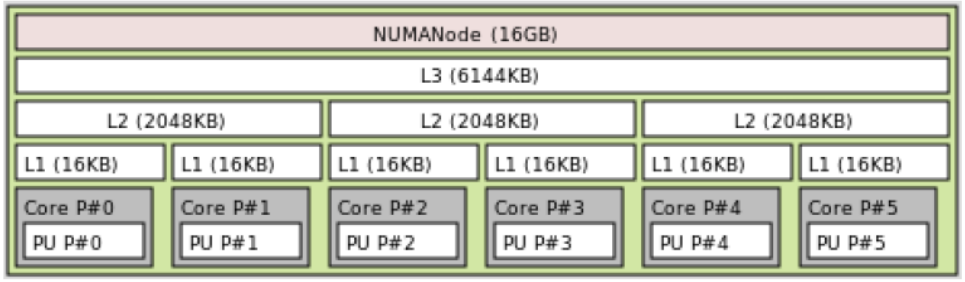


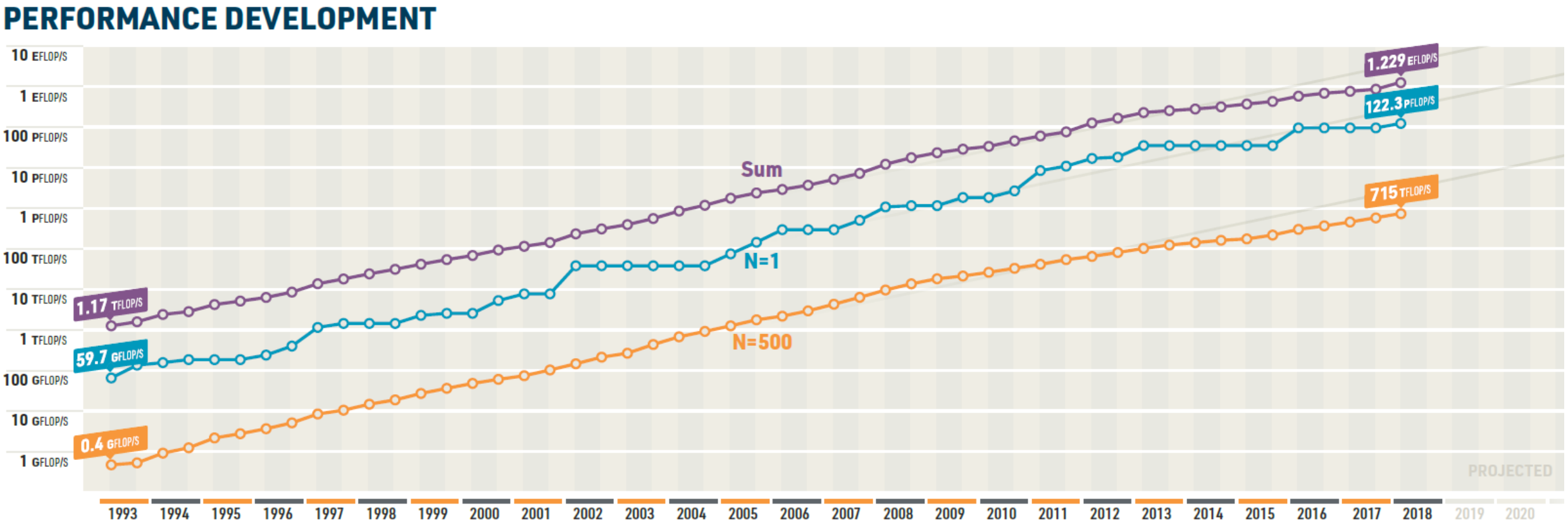
Figure : NUMA machine.

$$T_p = \frac{T_s}{P} + T_{comm}$$

Top 500 supercomputers

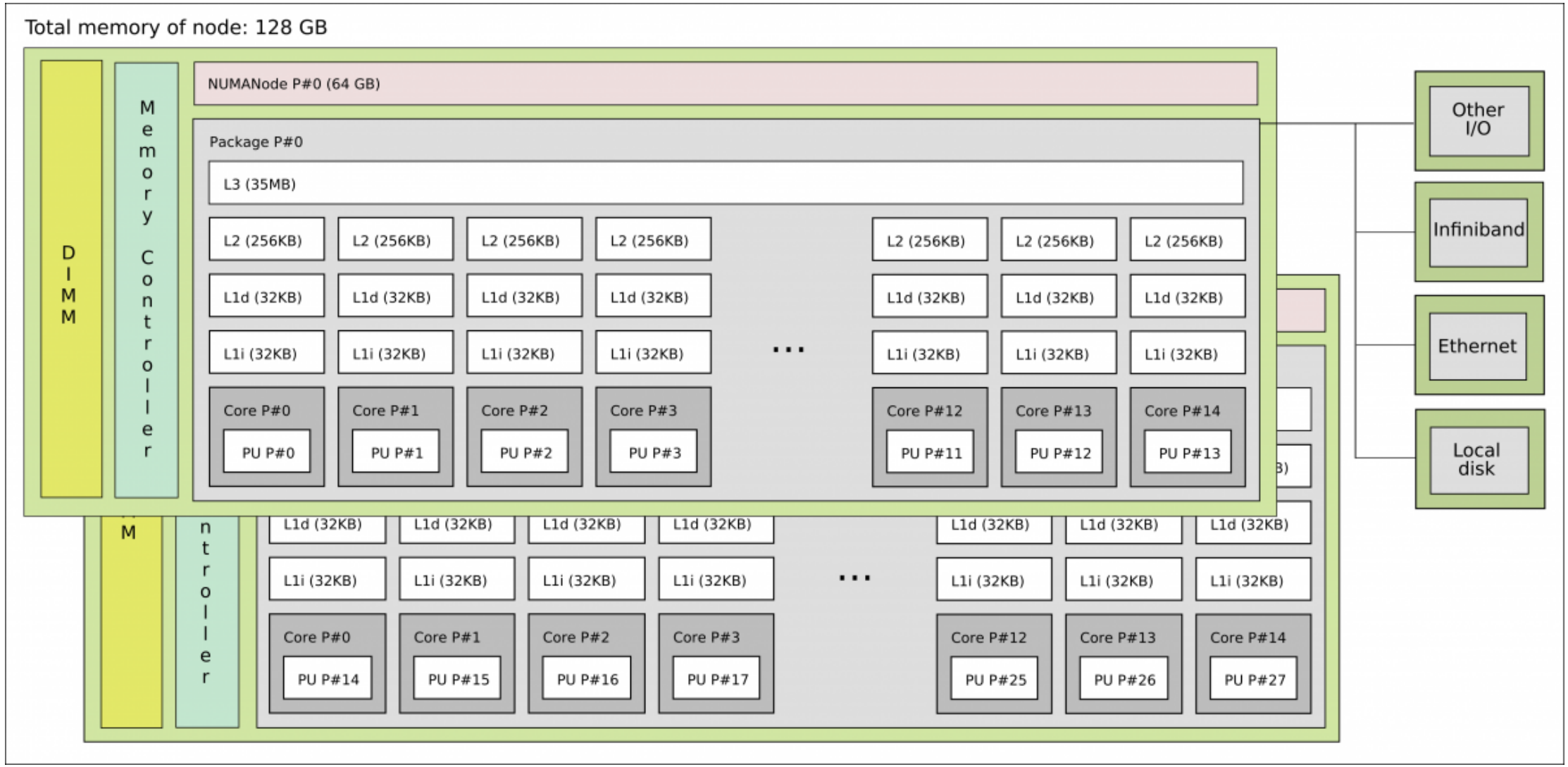
		SPECS	SITE	COUNTRY	CORES	RMAX PFLOP/S	POWER MW
1	**Summit**	IBM POWER9 (22C, 3.07GHz), NVIDIA Volta GV100 (80C), Dual-rail Mellanox EDR Infiniband	DOE/SC/ORNL	USA	2,282,544	122.3	10.8
2	**Sunway TaihuLight**	Shenwei SW26010 (260C 1.45 GHz) Custom interconnect	NSCC in Wuxi	China	10,649,600	93.0	15.4
3	**Sierra**	IBM POWER9 (22C, 3.1GHz), NVIDIA Tesla V100 (80C), Dual-rail Mellanox EDR Infiniband	DOE/NNSA/LLNL	USA	1,572,480	71.6	
4	**Tianhe-2A (Milkyway-2A)**	Intel Ivy Bridge (12C 2.2 GHz) & TH Express-2, Matrix-2000	NSCC Guangzhou	China	4,981,760	61.4	18.5
5	**AI Bridging Cloud Infrastructure**	PRIMERGY CX2550 M4, Xeon Gold 6148 (20C 2.4GHz), NVIDIA Tesla V100 (80C) SXM2, Infiniband EDR	AIST	Japan	391,680	19.9	

Top 500 supercomputers

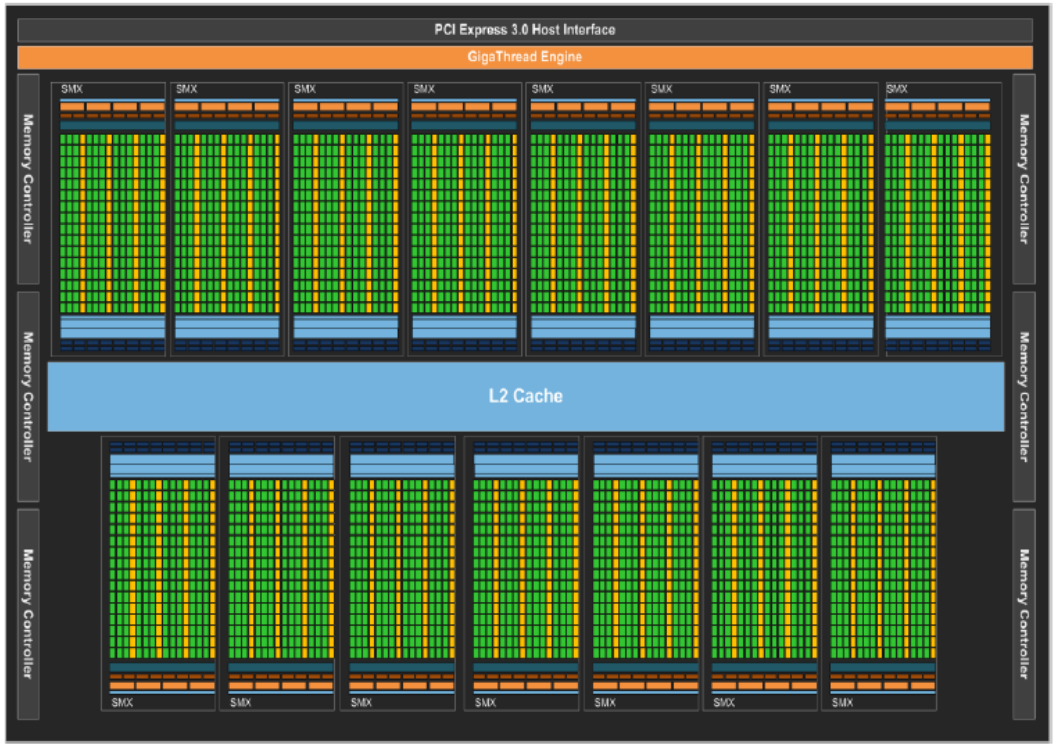


1GFLOPS = 10^9 FLOPS 1TFLOPS = 10^{12} FLOPS 1PFLOPS = 10^{15} FLOPS **1EFLOPS = 10^{18} FLOPS**

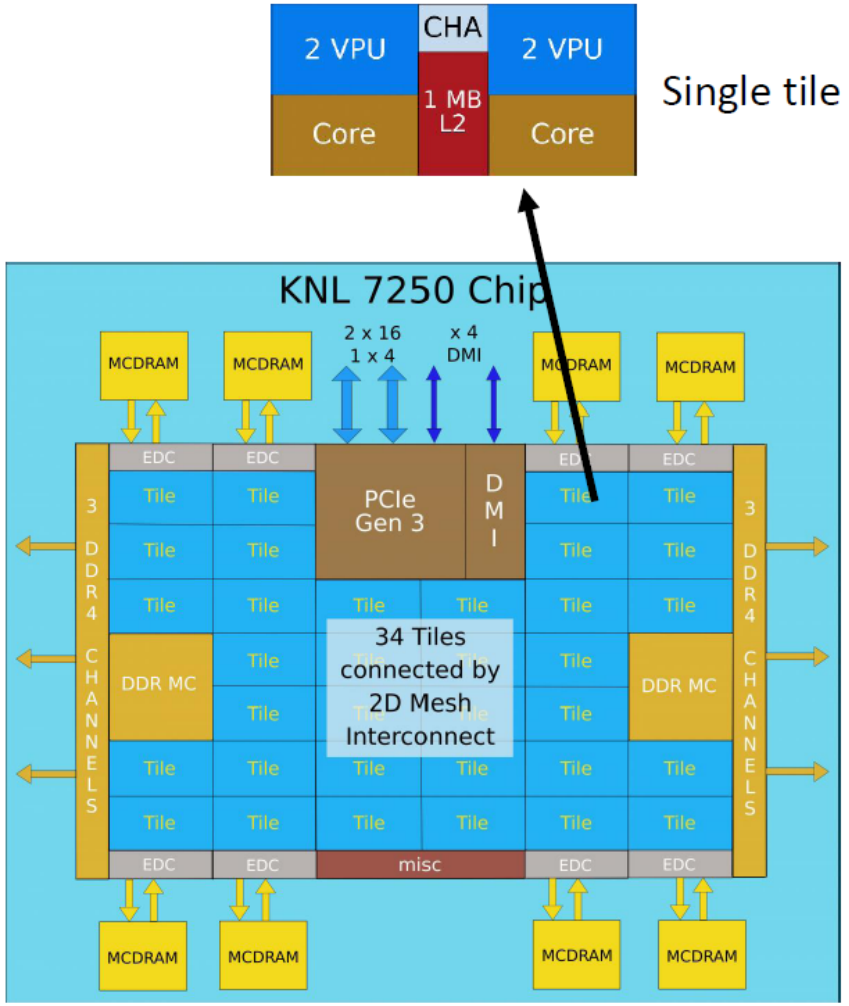
Broadwell node on Kebnekaise



Accelerators



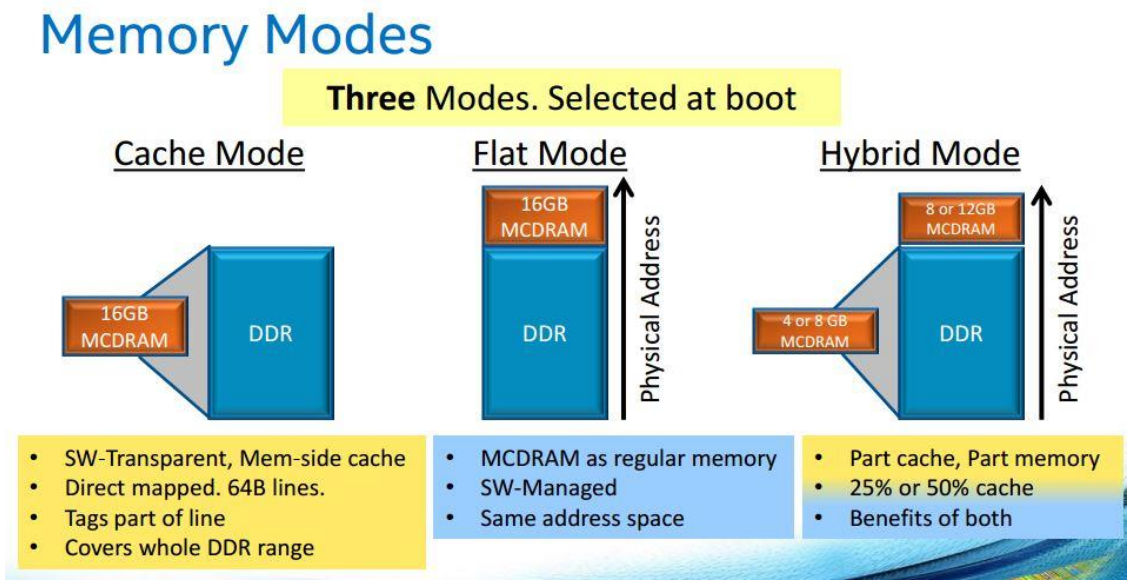
GPU showing the independent units Streaming Multiprocessors (SM).

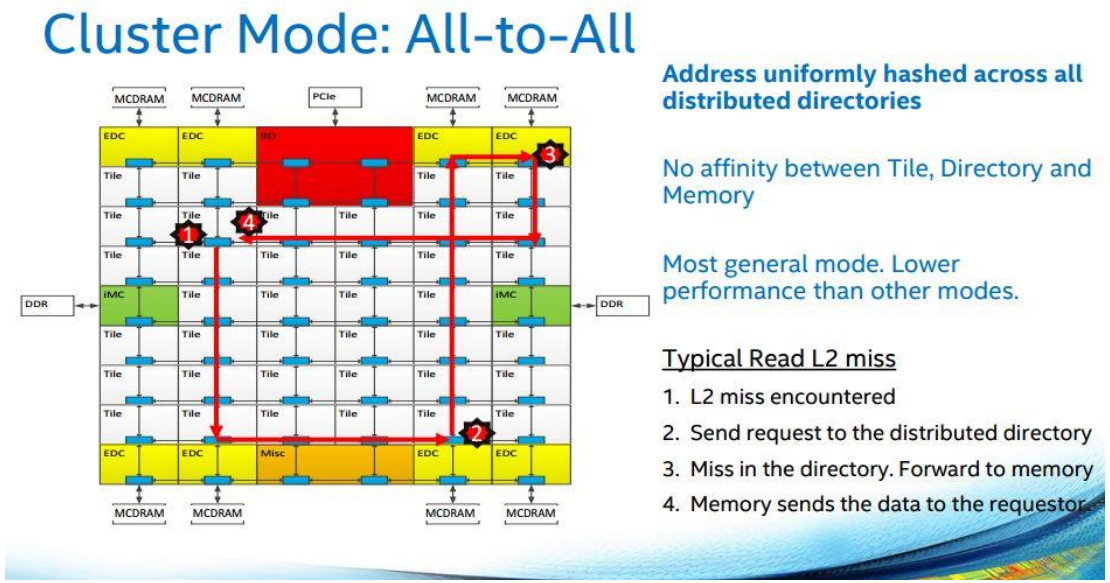


KNL, composed of several Tiles

$$Z(i) = A*X(i) + Y(i) \quad \text{(Vector Op.)}$$

KNL





#SBATCH --constraint=a2a,cache

#SBATCH --gres=hbm:4G

Credits: PRACE Best practice KNL (2017)

KNL Thread affinity

There are physical 68 cores with 4 hyperthreads on each.

Bind the threads by using OpenMP env. var.

```
export OMP_NUM_THREADS=4
export OMP_PROC_BIND=spread
export OMP_PLACES=cores
```

```
srun -n 68 -c 4 --cpu_bind=cores a_knl.out
```

Alternatively, use Intel var.

KNL Thread affinity

#export OMP_PROC_BIND=spread, close, etc.
#export OMP_PLACES=threads, cores, etc.
export OMP_NUM_THREADS=4
srun -n 16 -c 4 --cpu_bind=cores ./xthi

Hello from rank 0, thread 0, on b-cn1209.hpc2n.umu.se. (core affinity = 0)
Hello from rank 0, thread 1, on b-cn1209.hpc2n.umu.se. (core affinity = 68)
Hello from rank 0, thread 2, on b-cn1209.hpc2n.umu.se. (core affinity = 136)
Hello from rank 0, thread 3, on b-cn1209.hpc2n.umu.se. (core affinity = 204)
Hello from rank 1, thread 0, on b-cn1209.hpc2n.umu.se. (core affinity = 1)
Hello from rank 1, thread 1, on b-cn1209.hpc2n.umu.se. (core affinity = 69)
Hello from rank 1, thread 2, on b-cn1209.hpc2n.umu.se. (core affinity = 137)
Hello from rank 1, thread 3, on b-cn1209.hpc2n.umu.se. (core affinity = 205)

Tuning your own Applications

<no current project> - Intel VTune Amplifier@b-cn1203.hpc2n.umu.se

Welcome | profile

Basic Hotspots — Hotspots by CPU Usage viewpoint (change)

INTEL VTUNE AMPLIFIER XE 2017

Collection Log | Analysis Target | Analysis Type | Summary | Bottom-up | Caller/Callee | Top-down Tree | Platform

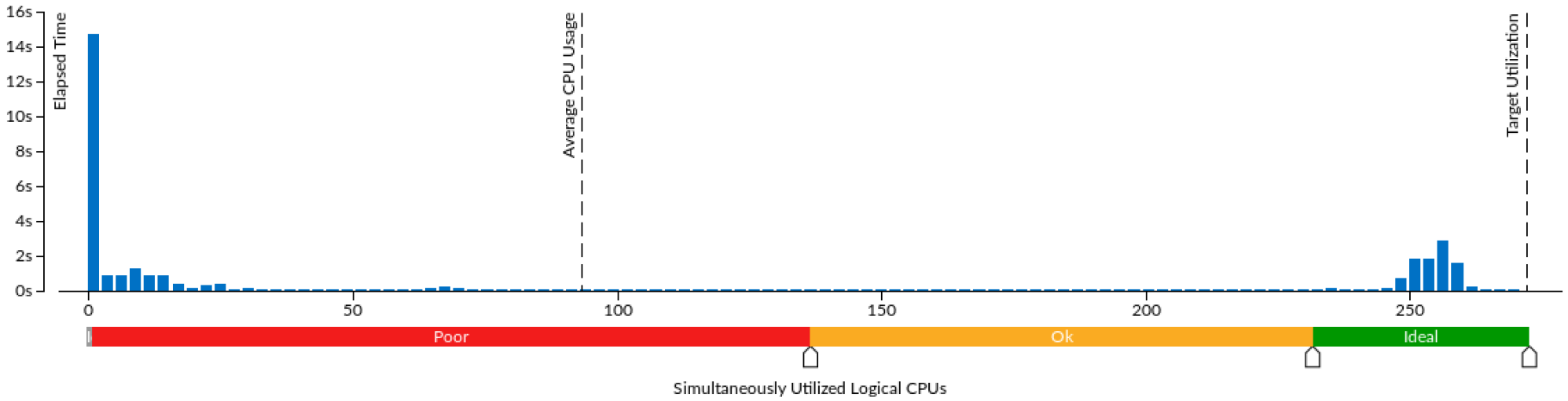
Top Hotspots

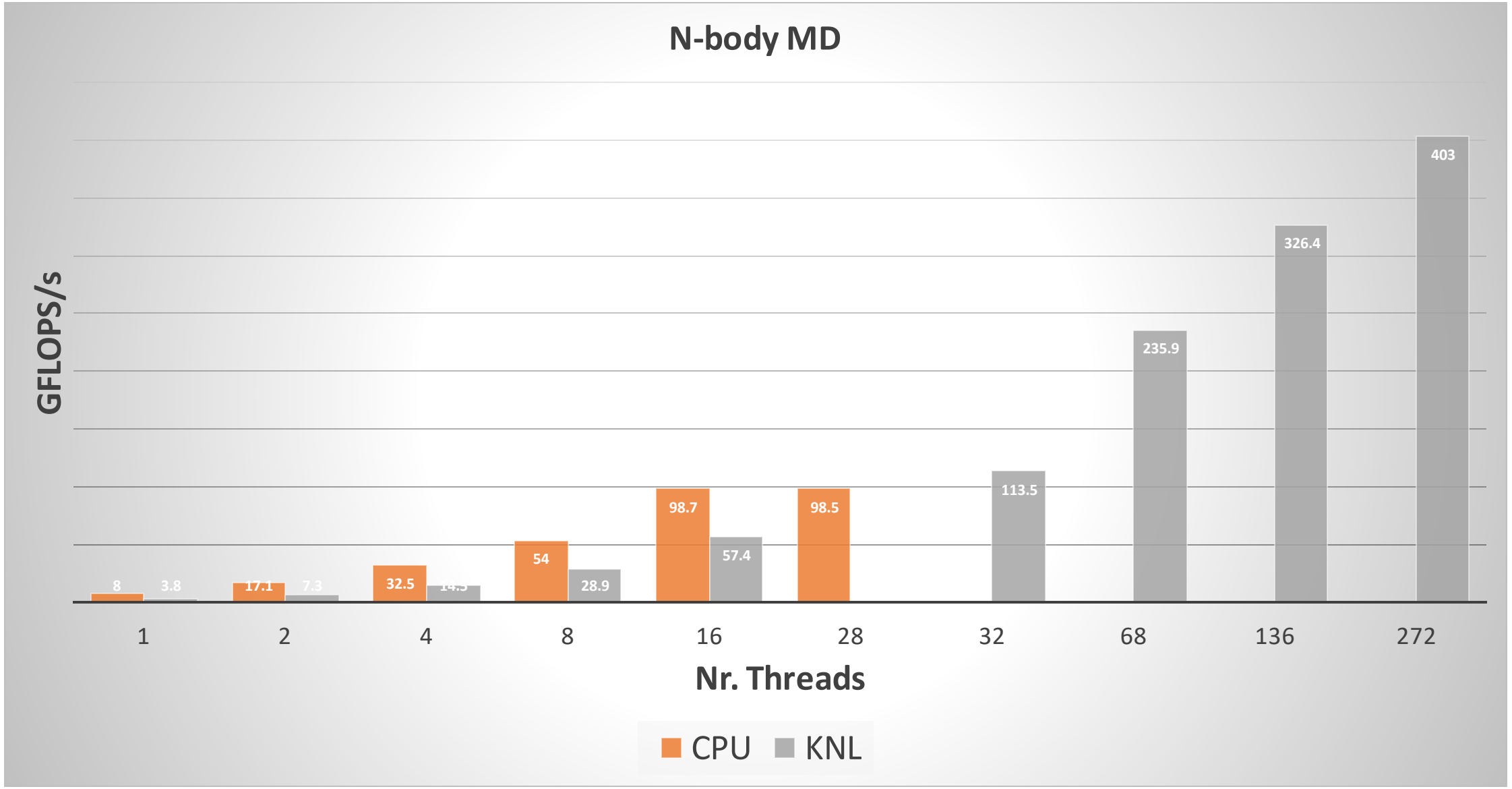
This section lists the most active functions in your application. Optimizing these hotspot functions typically results in improving overall application performance.

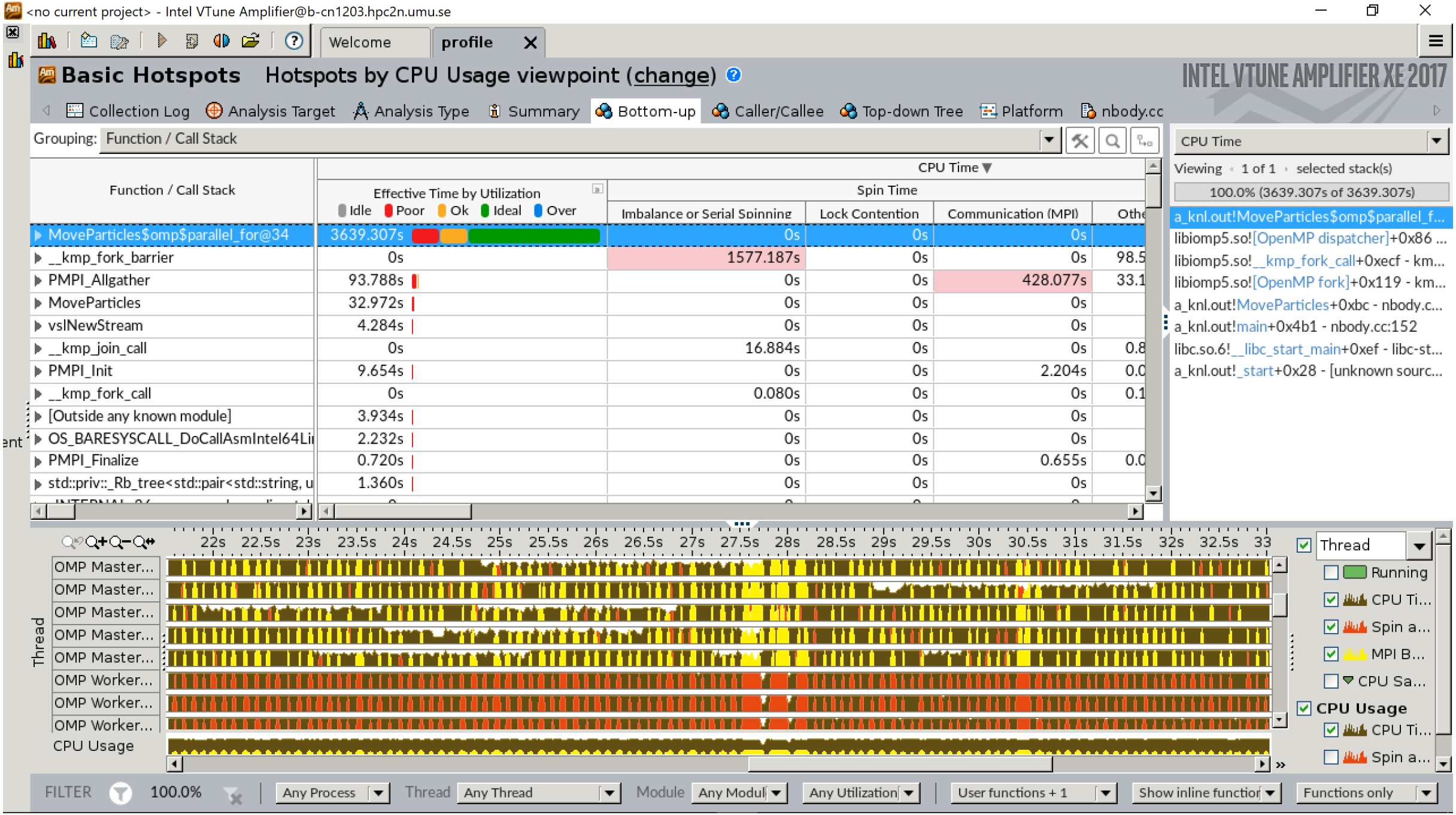
Function	Module	CPU Time
MoveParticlesompparallel_for@34	a_knl.out	3639.307s
__kmp_fork_barrier	libiomp5.so	1675.892s
PMPI_Allgather	libmpi.so.12	555.005s
MoveParticles	a_knl.out	32.972s
vslNewStream	libmkl_intel_lp64.so	24.794s
[Others]		57.713s

CPU Usage Histogram

This histogram displays a percentage of the wall time the specific number of CPUs were running simultaneously. Spin and Overhead time adds to the Idle CPU usage value.

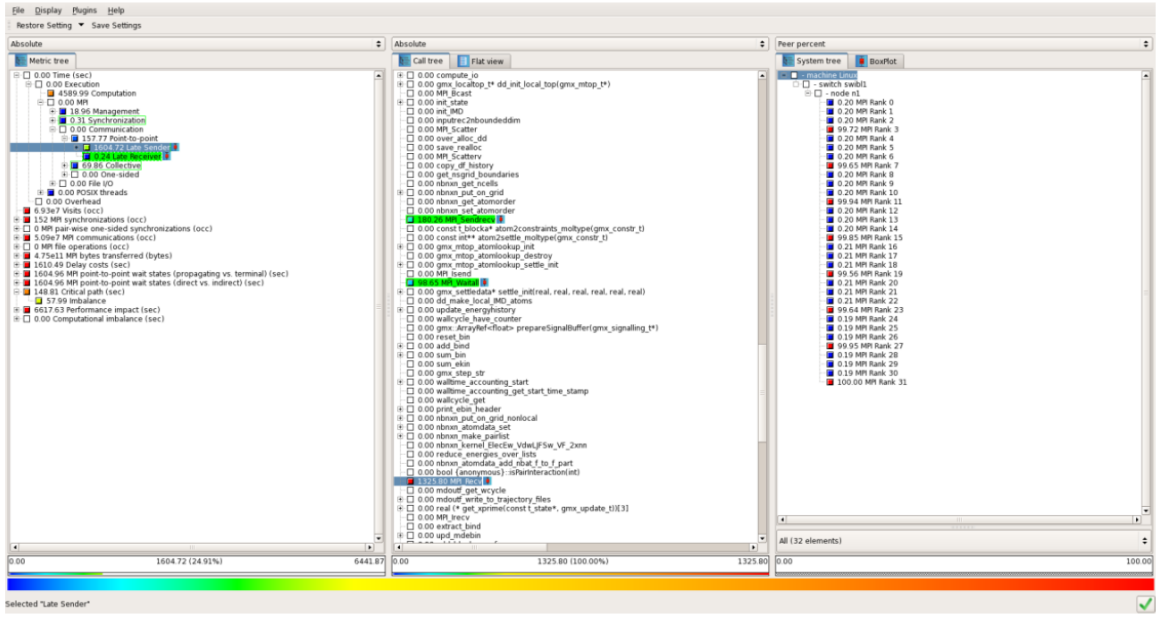


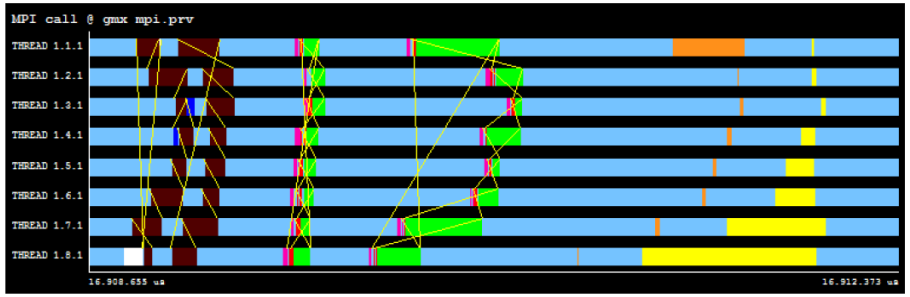

N-body MD
GFLOPS/s
8
3.8
17.1
7.3
32.5
14.5
54
28.9
98.7
57.4
98.5
113.5
235.9
326.4
403
1
2
4
8
16
28
32
68
136
272
Nr. Threads
CPU
KNL


<no current project> - Intel VTune Amplifier@b-cn1203.hpc2n.umu.se
Welcome
profile
Basic Hotspots
Hotspots by CPU Usage viewpoint (change)
INTEL VTUNE AMPLIFIER XE 2017
Collection Log
Analysis Target
Analysis Type
Summary
Bottom-up
Caller/Callee
Top-down Tree
Platform
nbody.cc
Grouping: Function / Call Stack
CPU Time
Viewing 1 of 1 selected stack(s)
100.0% (3639.307s of 3639.307s)
Function / Call Stack
CPU Time
Effective Time by Utilization
Idle Poor Ok Ideal Over
Spin Time
Imbalance or Serial Spinning
Lock Contention
Communication (MPI)
MoveParticlesompparallel_for@34 3639.307s 0s 0s 0s
__kmp_fork_barrier 0s 1577.187s 0s 0s
PMPI_Allgather 93.788s 0s 0s 428.077s
MoveParticles 32.972s 0s 0s 0s
vslNewStream 4.284s 0s 0s 0s
__kmp_join_call 0s 16.884s 0s 0s
PMPI_Init 9.654s 0s 0s 2.204s
__kmp_fork_call 0s 0.080s 0s 0s
[Outside any known module] 3.934s 0s 0s 0s
OS_BARESYSCALL_DoCallAsmIntel64Li 2.232s 0s 0s 0s
PMPI_Finalize 0.720s 0s 0s 0.655s
std::priv::_Rb_tree<std::pair<std::string, u 1.360s 0s 0s 0s
a_knl.out!MoveParticlesompparallel_f...
libiomp5.so![OpenMP dispatcher]+0x86 ...
libiomp5.so!__kmp_fork_call+0xecf - km...
libiomp5.so![OpenMP fork]+0x119 - km...
a_knl.out!MoveParticles+0xbc - nbody.c...
a_knl.out!main+0x4b1 - nbody.cc:152
libc.so.6!__libc_start_main+0xef - libc-st...
a_knl.out!_start+0x28 - [unknown sourc...
22s 22.5s 23s 23.5s 24s 24.5s 25s 25.5s 26s 26.5s 27s 27.5s 28s 28.5s 29s 29.5s 30s 30.5s 31s 31.5s 32s 32.5s 33
Thread
OMP Master...
OMP Master...
OMP Master...
OMP Master...
OMP Master...
OMP Worker...
OMP Worker...
OMP Worker...
CPU Usage
Thread
Running
CPU Ti...
Spin a...
MPI B...
CPU Sa...
CPU Usage
CPU Ti...
Spin a...
FILTER 100.0%
Any Process
Thread Any Thread
Module Any Modul
Any Utilization
User functions + 1
Show inline functior
Functions only

Profiling Tools at HPC2N

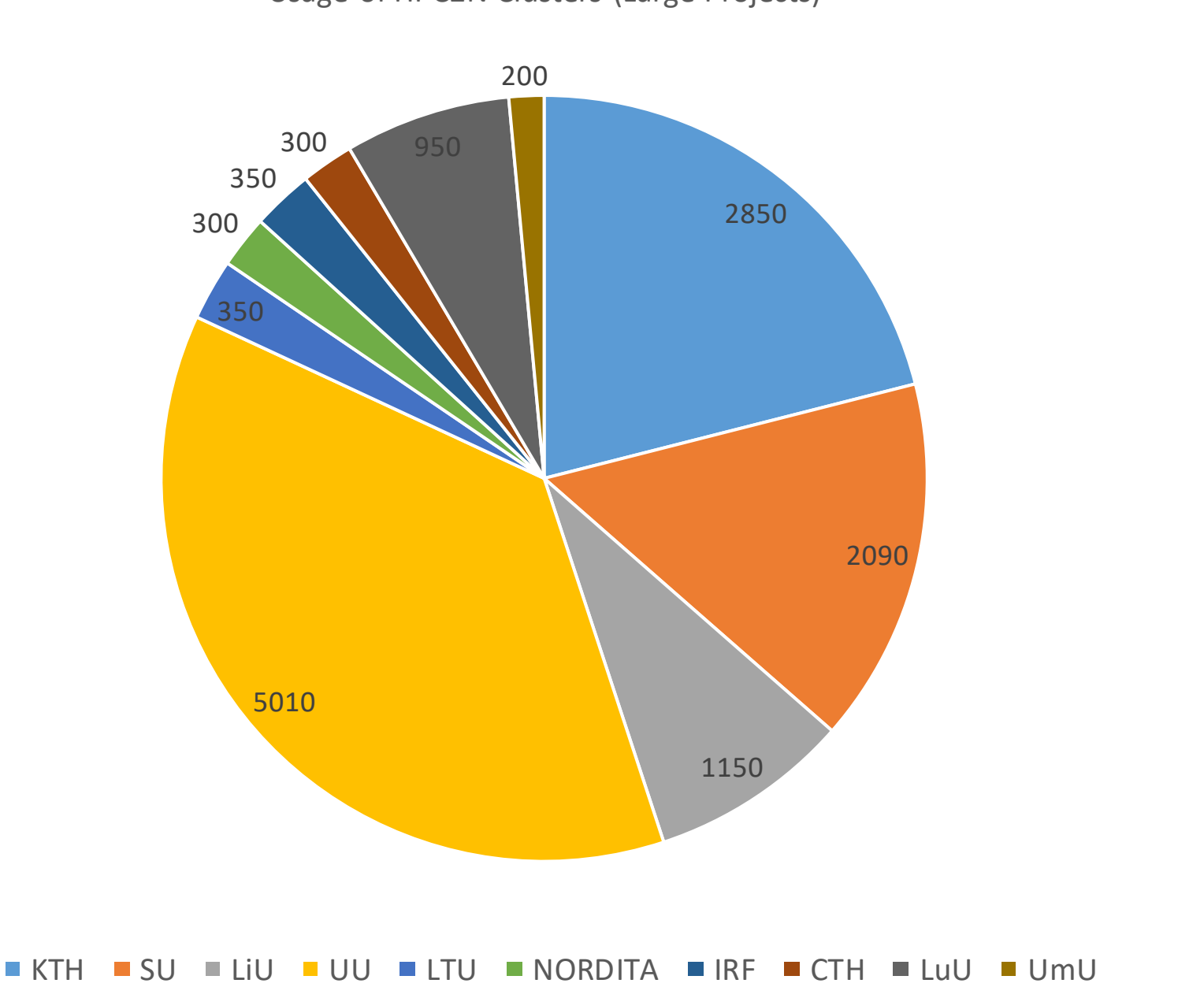
- Intel Vtune, Intel Inspector
- SCALASCA, Score-P, Cube
- Extrae/Paraver
- Allinea DDT



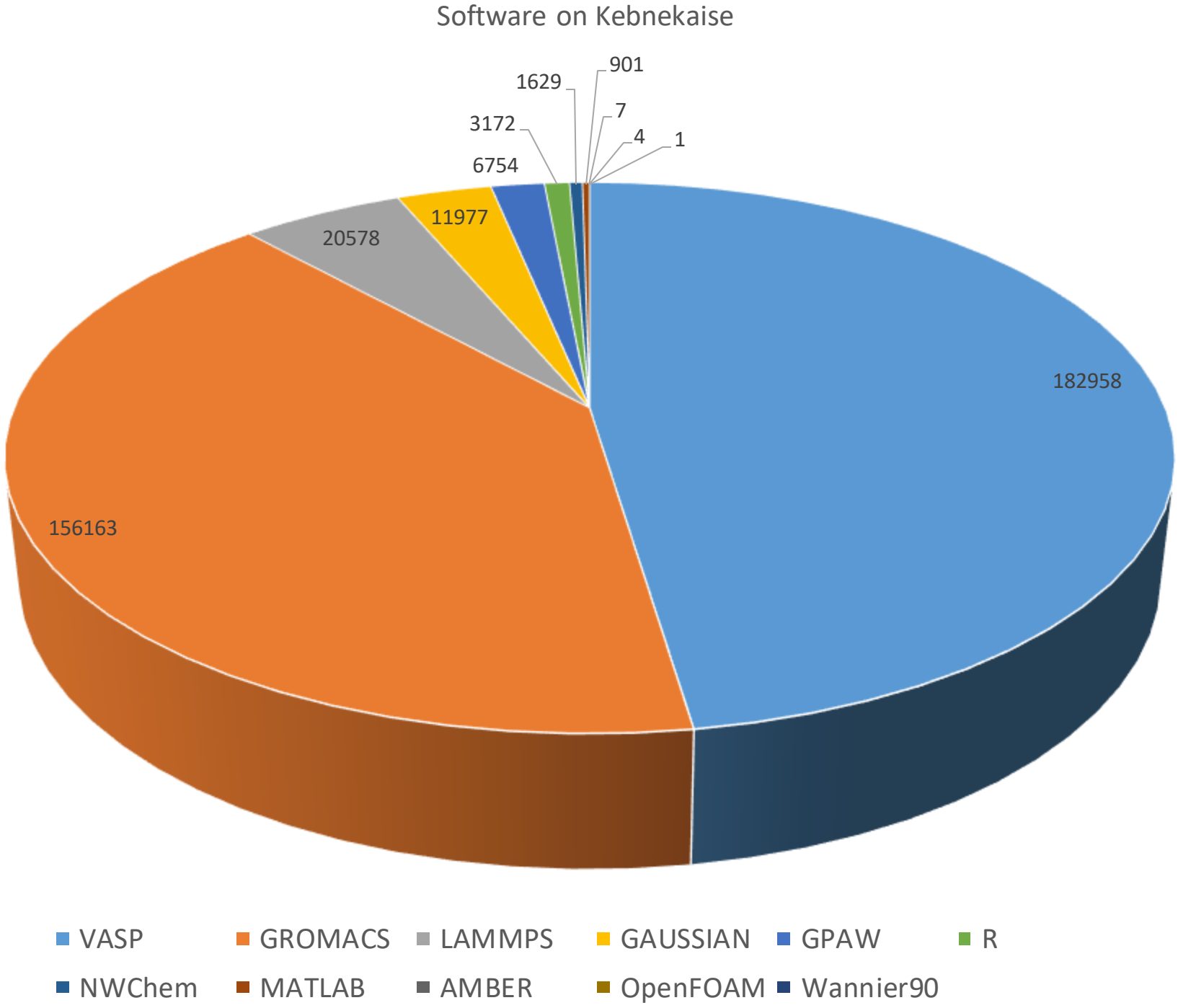


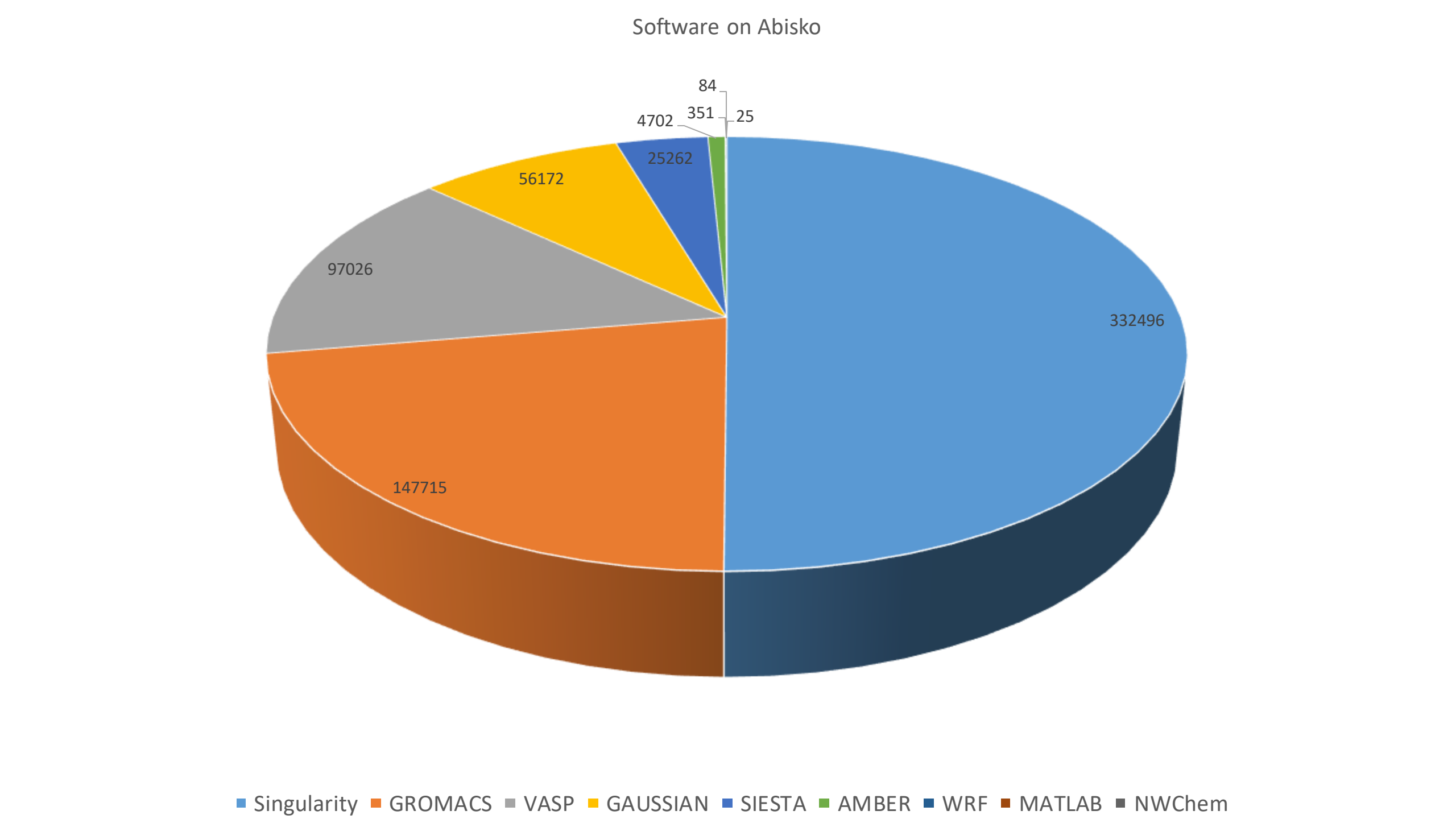
J. Eriksson, P. Ojeda, T. Ponweiser, T. Steinreicher (2017)

Applications



x1000 core-hour/month. Source: www.snic.se

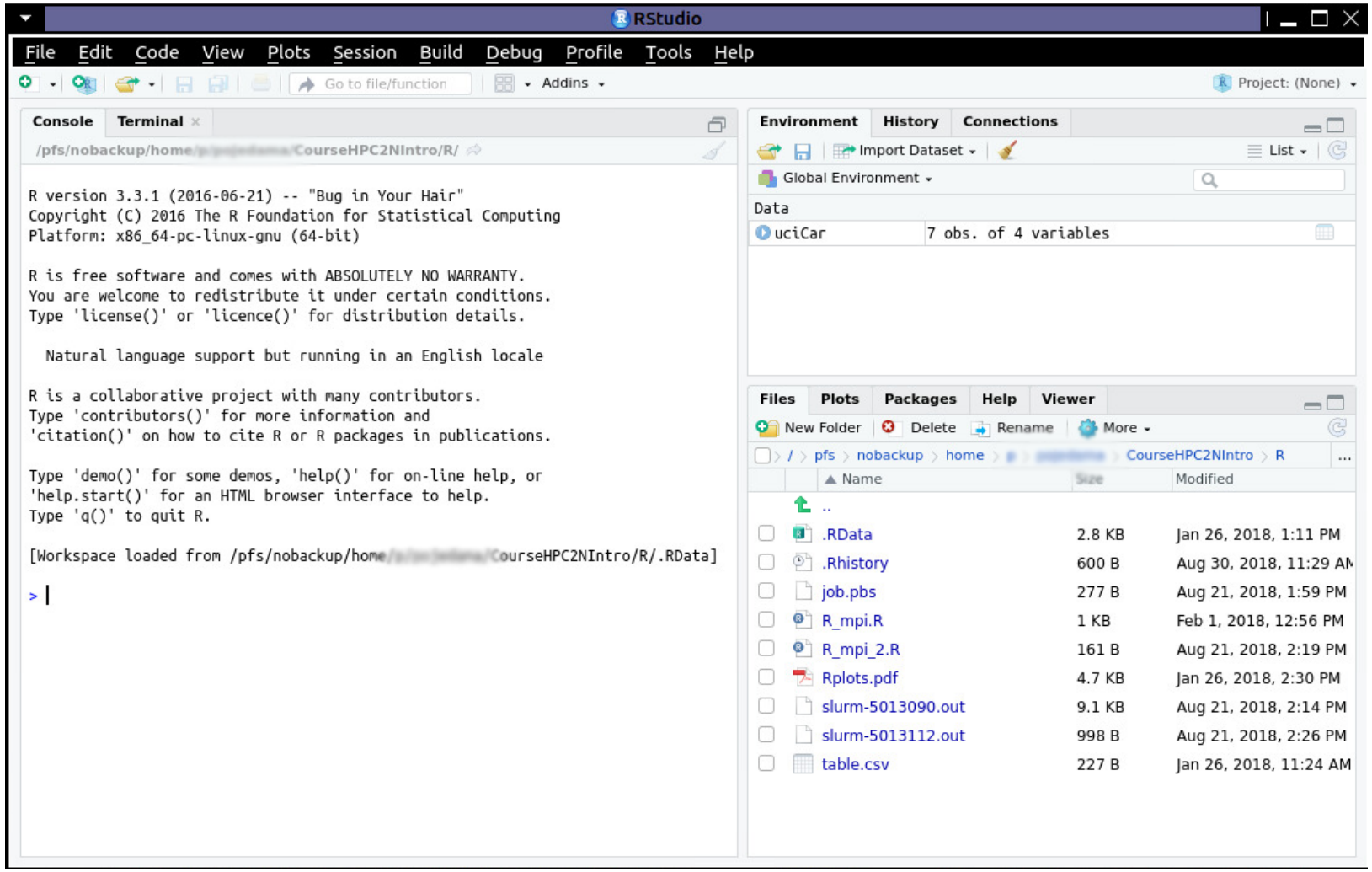

Software on Kebnekaise
182958
156163
20578
11977
6754
3172
1629
901
7
4
1
VASP
GROMACS
LAMMPS
GAUSSIAN
GPAW
R
NWChem
MATLAB
AMBER
OpenFOAM
Wannier90


Software on Abisko
84
351
25
4702
25262
56172
97026
332496
147715
Singularity
GROMACS
VASP
GAUSSIAN
SIESTA
AMBER
WRF
MATLAB
NWChem

Rstudio

$ml icc/2017.1.132-GCC-6.3.0-2.27 impi/2017.1.132 ifort/2017.1.132-GCC-6.3.0-2.27

$ml R/3.3.1
$rstudio



R

```
library("Rmpi")

rk <- mpi.comm.rank(0)
sz <- mpi.comm.size(0)

name <- mpi.get.processor.name()
cat("Hello, rank",rk,"size",sz,"on",name,"\n")

mpi.quit()
```

Batch script

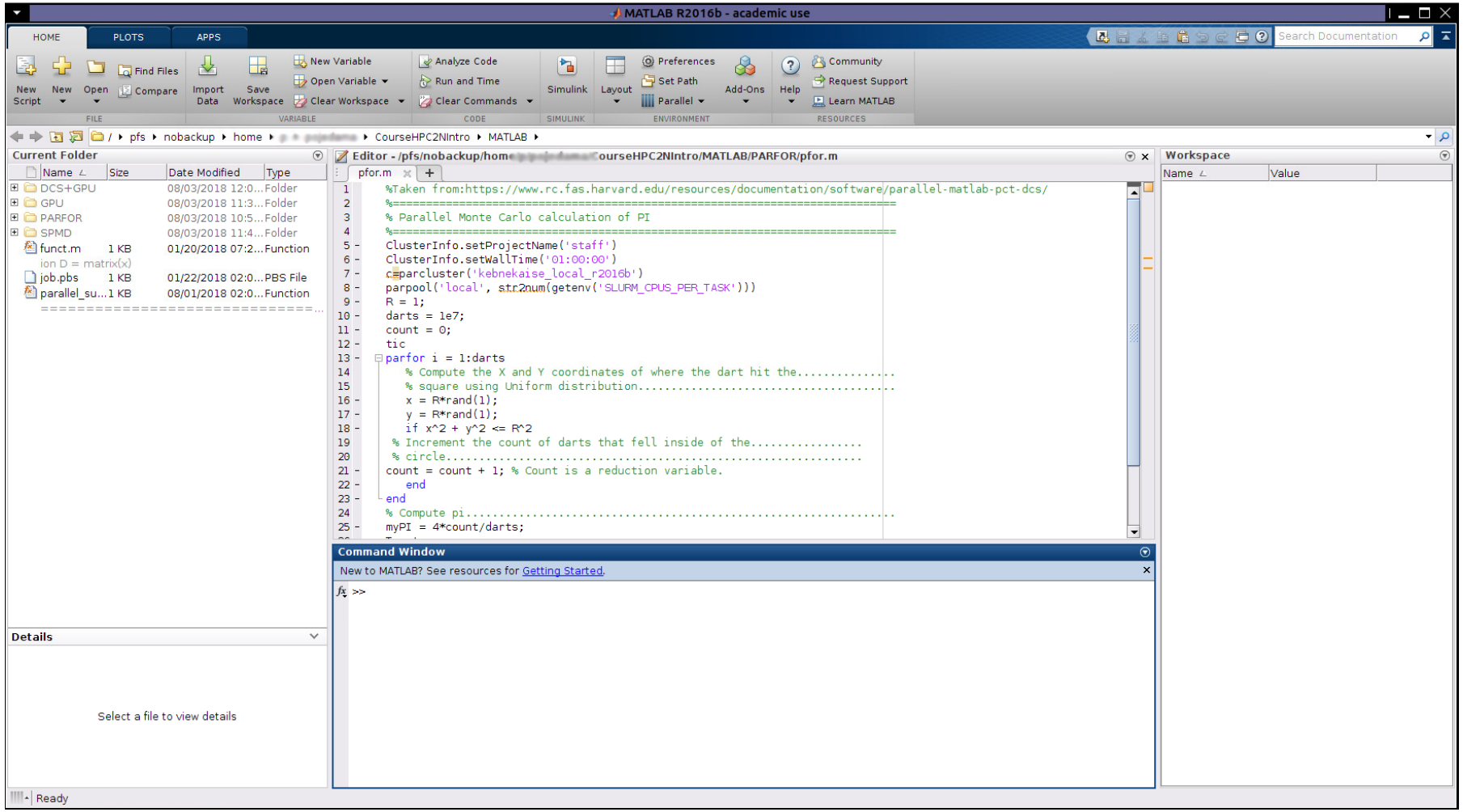
```
#SBATCH -N 1
#Ask for 28 processes
#SBATCH -n 28
#SBATCH --exclusive

ml icc/2017.1.132-GCC-6.3.0-2.27
impi/2017.1.132 ifort/2017.1.132-GCC-6.3.0-2.27
ml R/3.3.1

mpirun R -q --slave -f R_mpi_2.R
```

Further information:
https://www.hpc2n.umu.se/resources/software/r

MATLAB



$ml MATLAB/2016b

Further information: https://www.hpc2n.umu.se/resources/software/matlab

MATLAB

- Parallel Computing Toolbox (PCT)
  - Parfor loops
  - Single Program Multiple Data, Labs are used to solve a task

- Distributed Computing Server (DCS)

Parfor

```
parpool('local', str2num(getenv('SLURM_CPUS_PER_TASK')))
parfor i = 1:darts
x = rand(1);
y = rand(1);
   if x^2 + y^2 <= R^2
    count = count + 1; % reduction variable.
   end
end
```

Batch script:

```
#SBATCH -N 1
#SBATCH -c 4
#SBATCH --exclusive
#SBATCH -p batch
#Load modules necessary for running MATLAB
ml MATLAB/2016b

srun -n 1 -c 4 matlab -nosplash -nodesktop -r "pfor"
```

Source: https://www.rc.fas.harvard.edu/resources/documentation/software/parallel-matlab-pct-dcs/

MATLAB+GPU

```
c = parcluster;
ClusterInfo.setProjectName('staff');
ClusterInfo.setWallTime('00:10:00');
% Tell the scheduler that you want to use the GPUs
ClusterInfo.setUseGpu(true)
% number of GPUs per node to use
ClusterInfo.setGpusPerNode(2)
ClusterInfo.setUserDefinedOptions('--gres=gpu:k80:2,mps')
gpuDevice

j = c.batch(@test_parallel, 1, {1}, 'pool', 2);
```

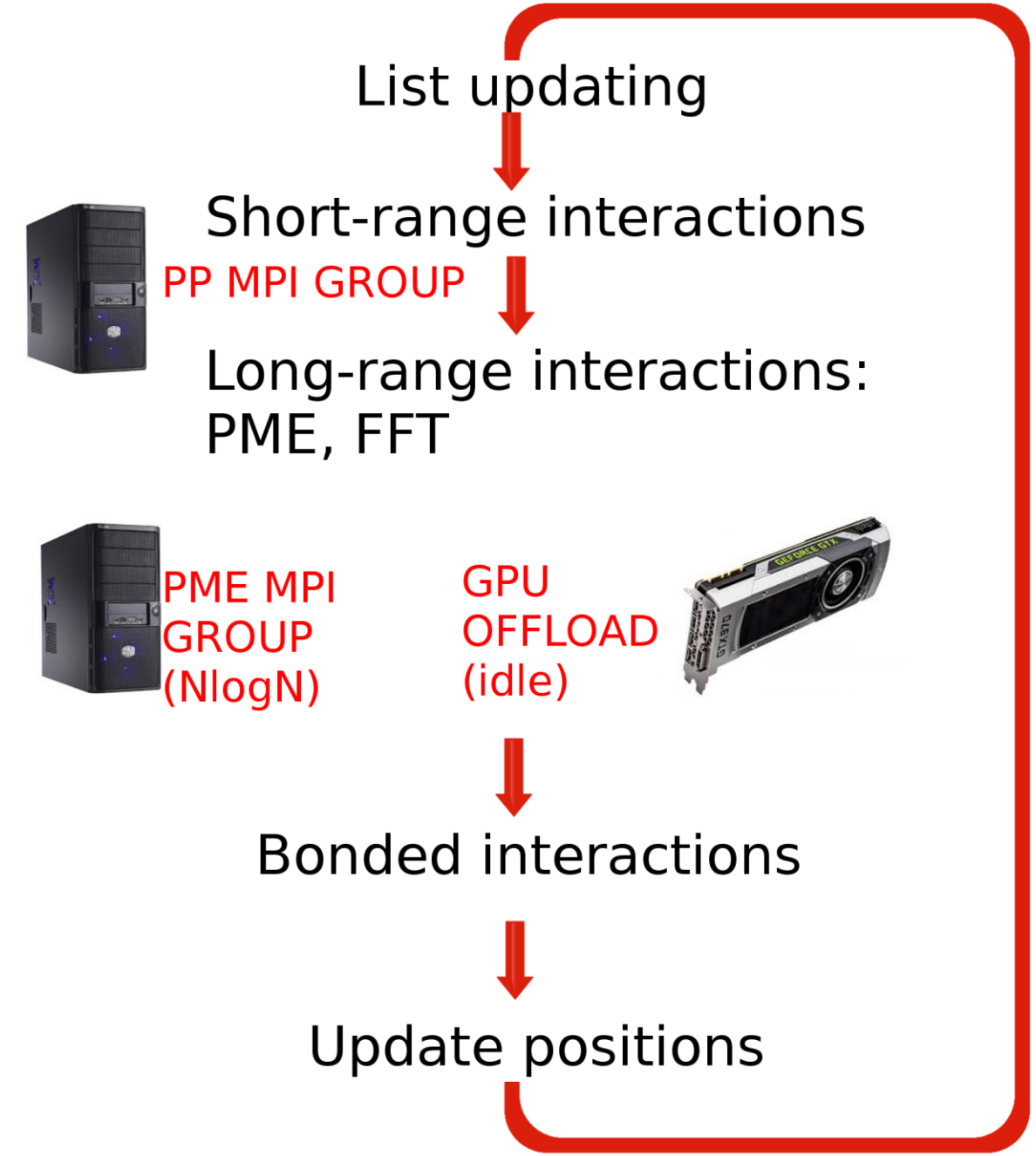
```
function S = parallel(x)
format long
A1 = rand(10000,10000);
tic;
B1 = fft(A1);
time1 = toc;


A2 = gpuArray(A1);
tic;
B2 = fft(A2);
time2 = toc;

speedUp1 = time1/time2;
end
```

srun matlab -nodesktop -nodisplay -nosplash -singleCompThread -nojvm –r “script_name”

MD Codes

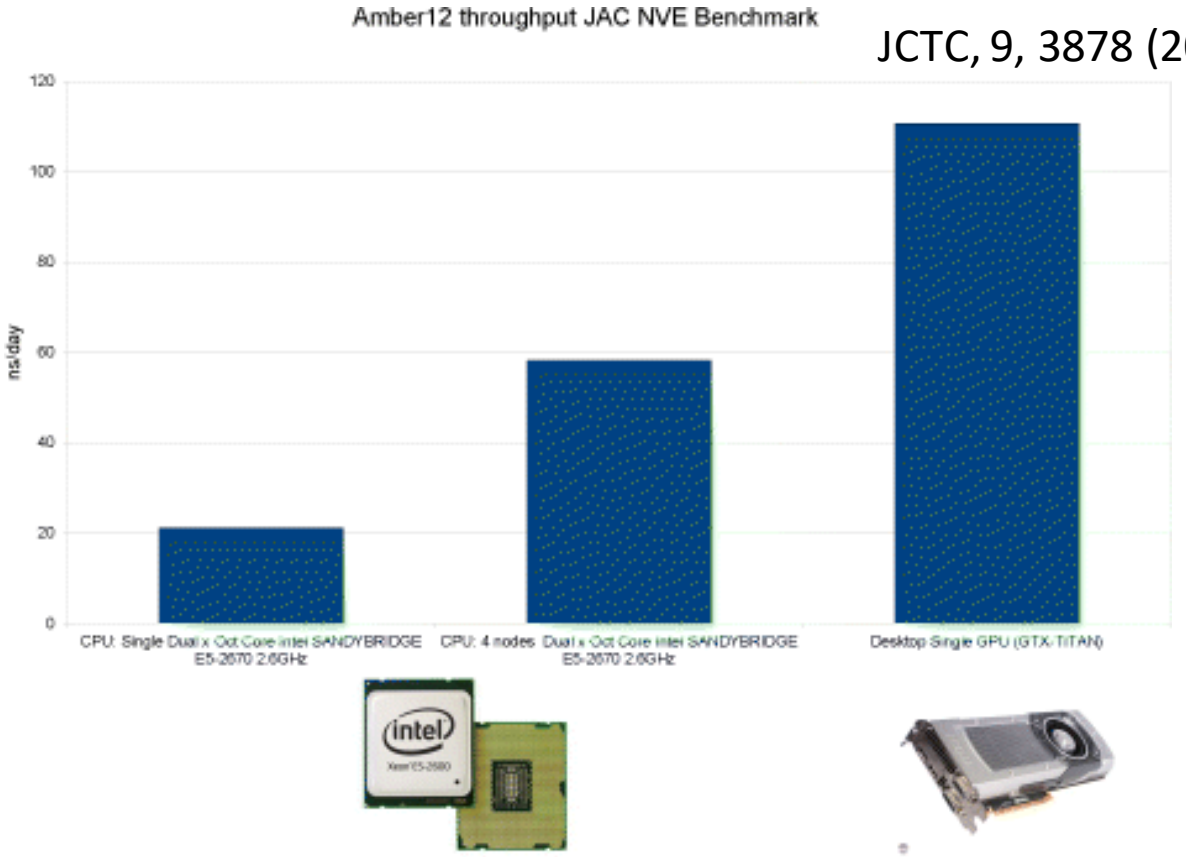

List updating
Short-range interactions
PP MPI GROUP
Long-range interactions:
PME, FFT
PME MPI
GROUP
(NlogN)
GPU
OFFLOAD
(idle)
Bonded interactions
Update positions

Benchmark

- Solvated protein
- 158945 atoms
- 1.2 nm cutoff radius
- 1 fs time step
- PME for electrostatics

Amber

- Collection of independent routines
- It uses Sander/PMEMD for solving the Newton's equations
- It offers a robust set of analysis tools



JCTC, 9, 3878 (2013)


```
module load icc/2017.1.132-GCC-5.4.0-2.26  CUDA/8.0.44
impi/2017.1.132
module load ifort/2017.1.132-GCC-5.4.0-2.26  CUDA/8.0.44
ml Amber/16-AmberTools-16-patchlevel-20-7-hpc2n
srun pmemd.MPI -O -i  input.mdin
srun pmemd.cuda.MPI -O –I input.mdin
```

AMBER Tools

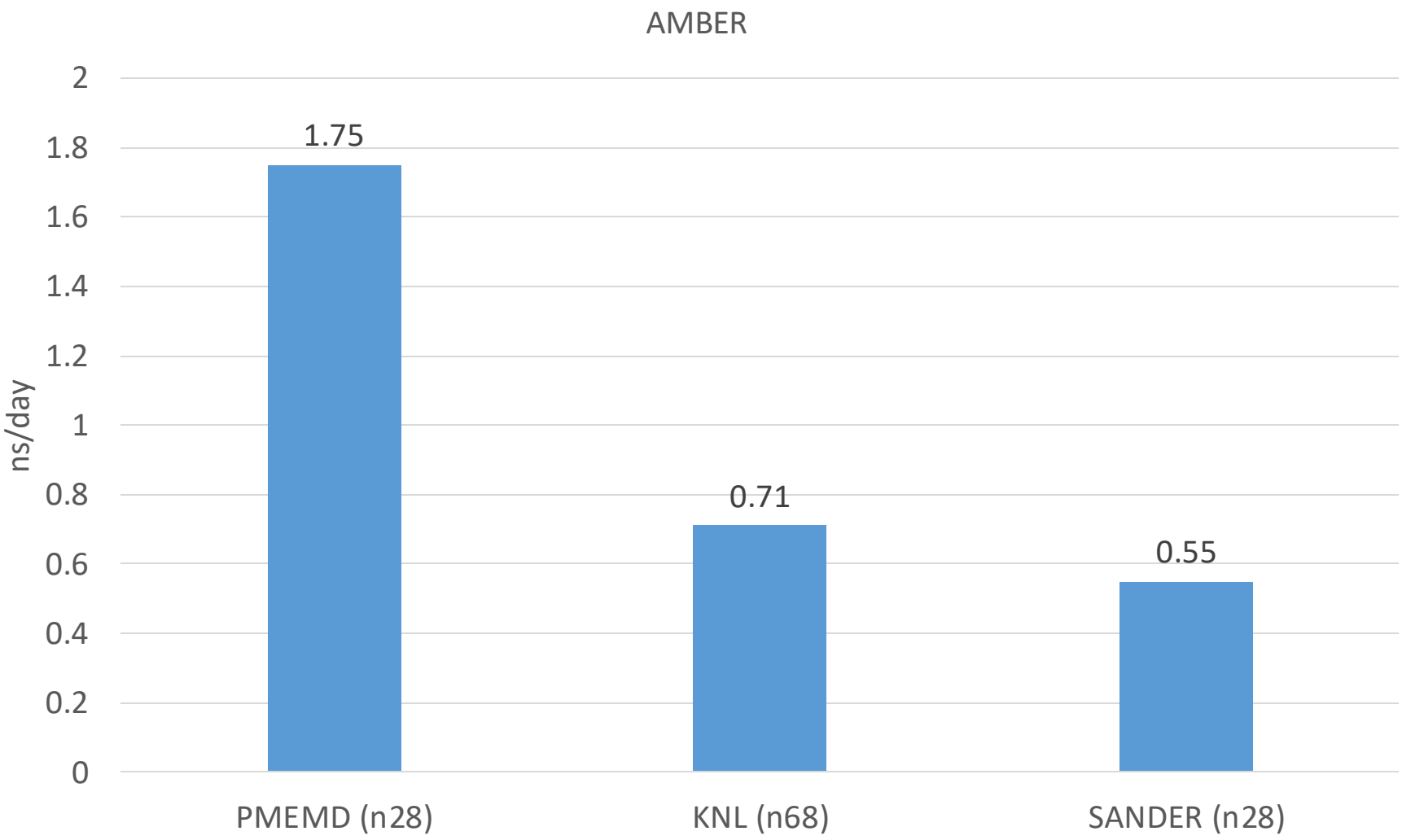
- For setting up a simulation (initial structure, solvation, ions, …):

Load the modules

$ml icc/2017.4.196-GCC-6.4.0-2.28 ifort/2017.4.196-GCC-6.4.0-2.28 impi/2017.3.196
$ml Amber/16-AmberTools-17-patchlevel-8-12

$tleap, antechamber, cpptraj, …

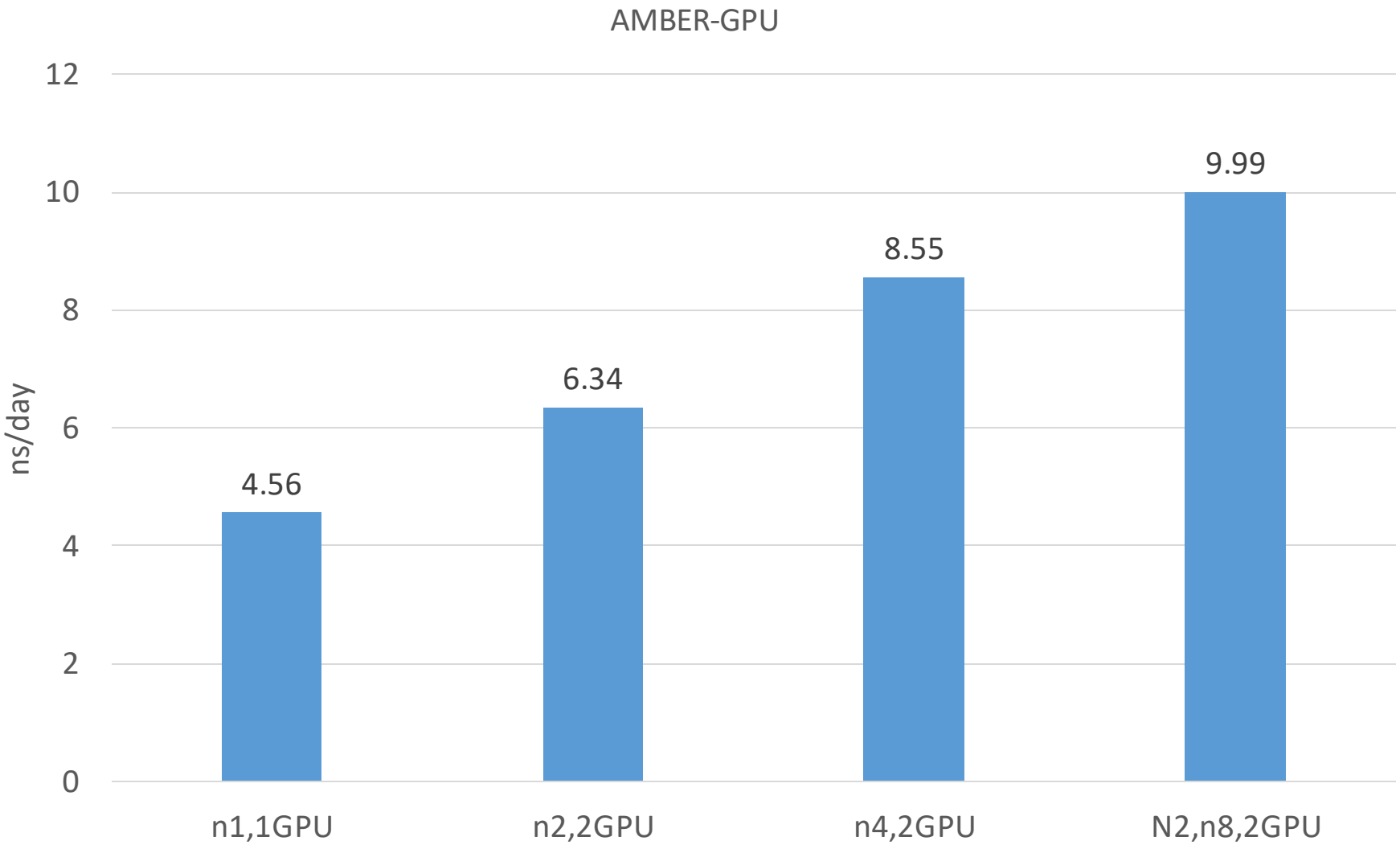
AMBER


AMBER
ns/day
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
1.75
0.71
0.55
PMEMD (n28)
KNL (n68)
SANDER (n28)

AMBER

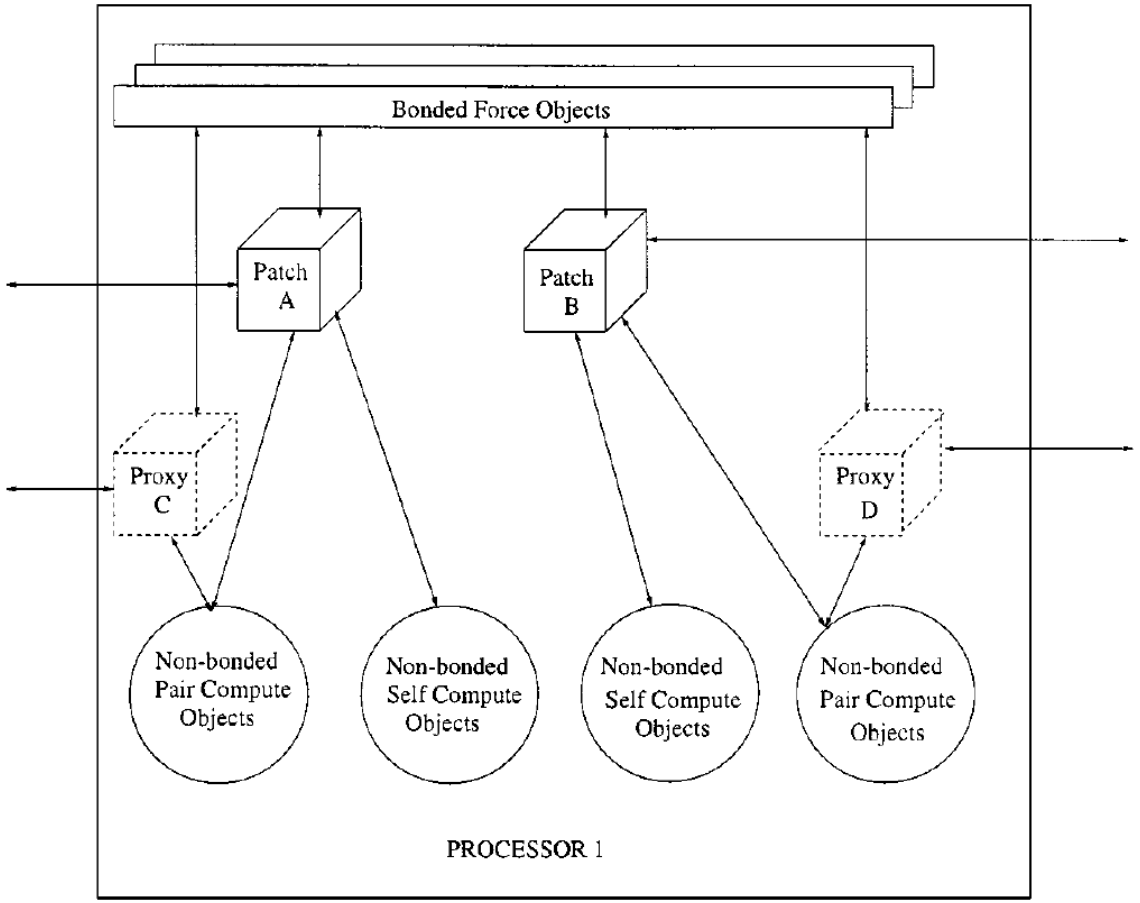
On Kebnekaise, best performance is achieved with 4 MPIs/Node and using 2 GPU cards

Notice that, the remaining CPUs are not used.



NAMD

- Based on charm++ communication protocol
- It is object-oriented
- Versions: single node, multi-node, GPU, and KNL
- Highly scalable



JCC, 151, 283 (1999)

NAMD (SMP)

```
#!/bin/bash
#SBATCH -A staff
#Asking for 10 min.
#SBATCH -t 00:50:00
#Number of nodes
#SBATCH -N 1
#Ask for 28 processes
#SBATCH -n 28
#SBATCH --exclusive
#Load modules necessary for running NAMD
module add icc/2017.1.132-GCC-6.3.0-2.27  impi/2017.1.132
module add NAMD/2.12-nompi
#Execute NAMD
namd2 +p 28 +setcpuaffinity step4_equilibration.inp > output_smp.dat
```

NAMD (GPU) (single CPU)

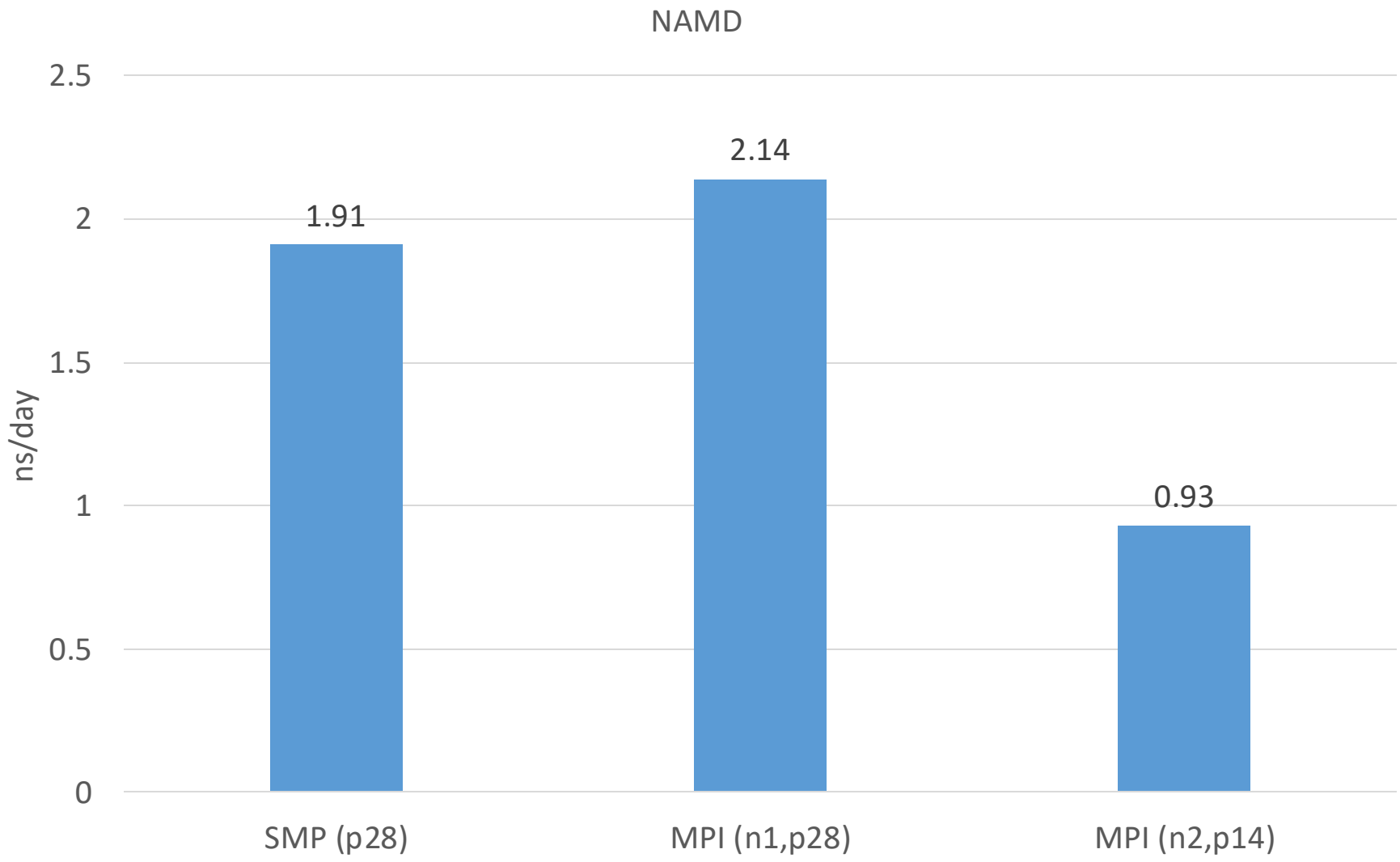
```
#!/bin/bash
#SBATCH -A staff
#SBATCH -t 00:50:00
#SBATCH -N 1
#SBATCH -n 28
#SBATCH –exclusive
#Ask for 2 GPU cards
#SBATCH --gres=gpu:k80:2
#Load modules necessary for running NAMD
module add GCC/5.4.0-2.26  CUDA/8.0.61_375.26  OpenMPI/2.0.2
module add NAMD/2.12-nompi
#Execute NAMD
namd2 +p 28 +setcpuaffinity step4_equilibration.inp > output_smp.dat
```

NAMD

- Colvars module for free energy calculations can be run on GPUs.

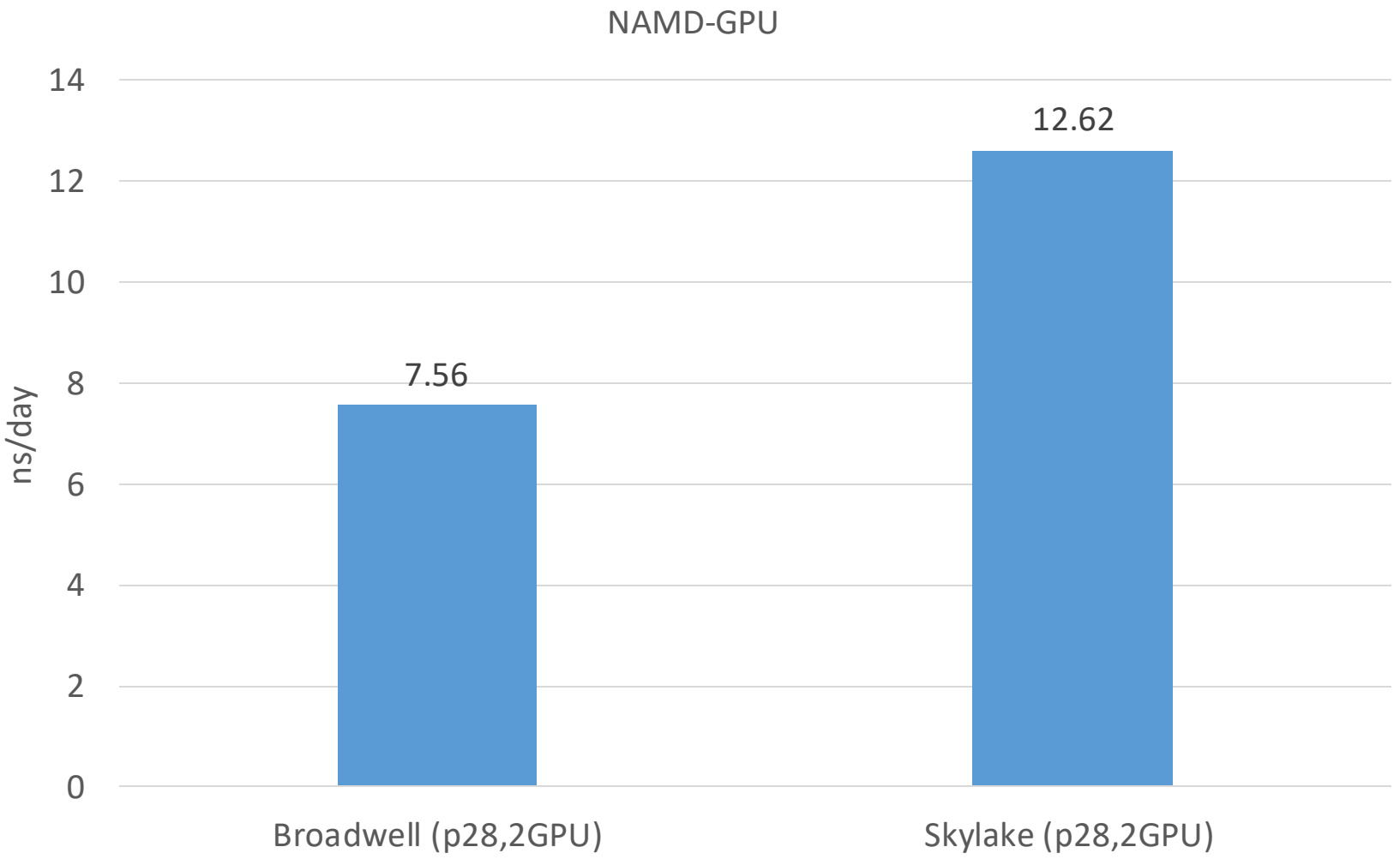
NAMD

Use the +setcpuaffinity flag for a 10% speedup



NAMD

With the new Volta GPU cards one can speed the simulation by 1.66x



GROMACS

```
#SBATCH -n 4
#SBATCH -c 7
# Asking for 2 GPUs
#SBATCH --gres=gpu:k80:2
#SBATCH -p batch

ml GCC/5.4.0-2.26
ml CUDA/8.0.44
ml OpenMPI/2.0.1
ml GROMACS/2016-hybrid

if [ -n "$SLURM_CPUS_PER_TASK" ]; then
   mdargs="-ntomp $SLURM_CPUS_PER_TASK"
else
   mdargs="-ntomp 1"
fi

srun -n $SLURM_NTASKS gmx_mpi mdrun $mdargs -npme 0 -dlb yes  -v -deffnm step4.1_equilibration
```

GROMACS recognizes the number of available GPU cards

gmx tune_pme

Individual timings for input file 0 (npt_bench00.tpr):

PME ranks	Gcycles	ns/day	PME/f	Remark
0	4355.019	57.776	-	OK.
0	4547.105	55.335	-	OK.
0	4289.420	58.659	-	OK.
-1(0)	4455.791	56.469	-	OK.
-1(0)	4440.157	56.668	-	OK.
-1(0)	4275.551	58.850	-	OK.

Tuning took 7.7 minutes.

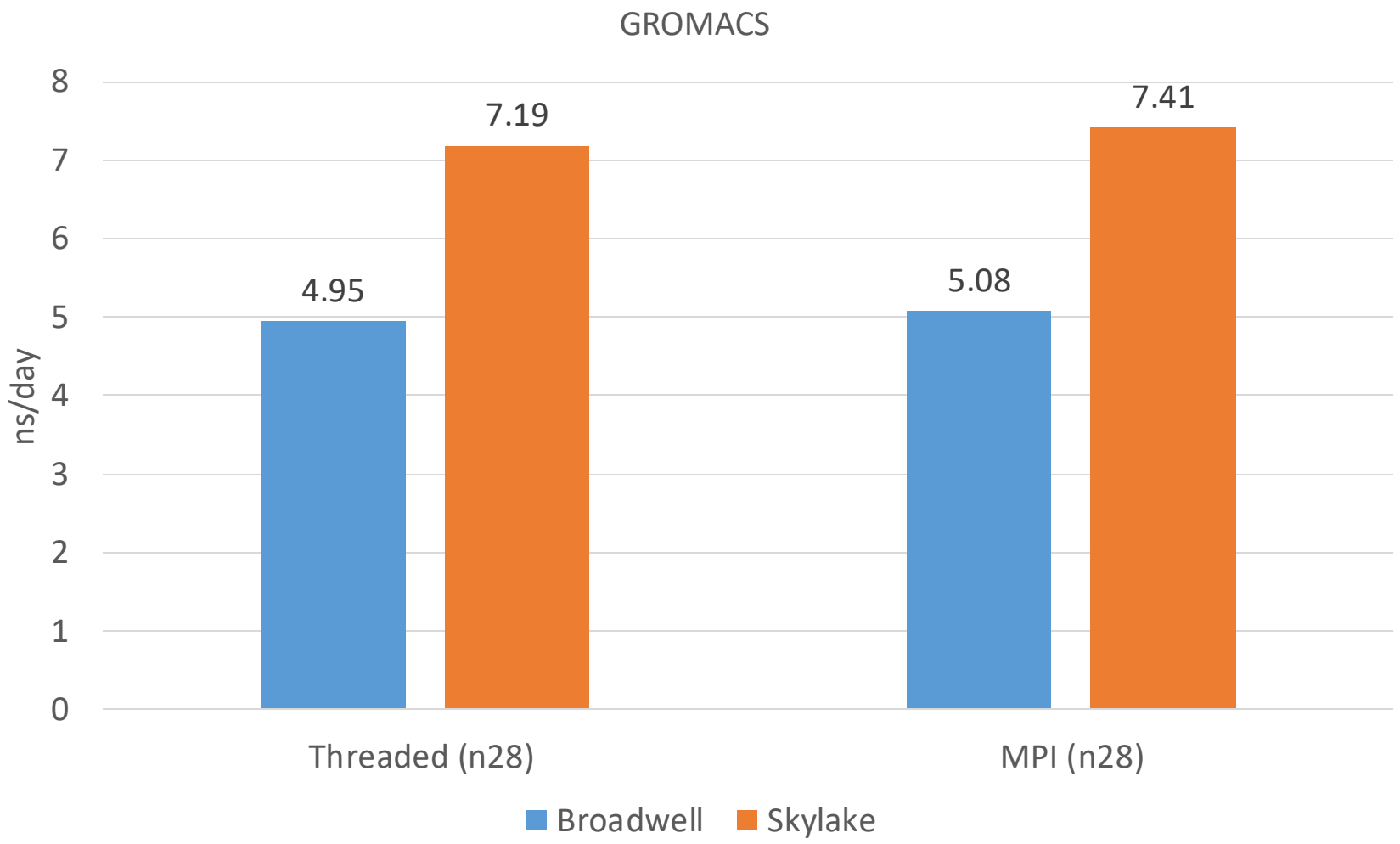
Summary of successful runs:

Line	tpr	PME ranks	Gcycles Av.	Std.dev.	ns/day	PME/f	DD grid
0	0	0	4397.181	133.917	57.257	-	4 1 1
1	0	-1(0)	4390.500	99.855	57.329	-	4 1 1

Best performance was achieved with the automatic number of PME ranks (see line 1)
Please use this command line to launch the simulation:

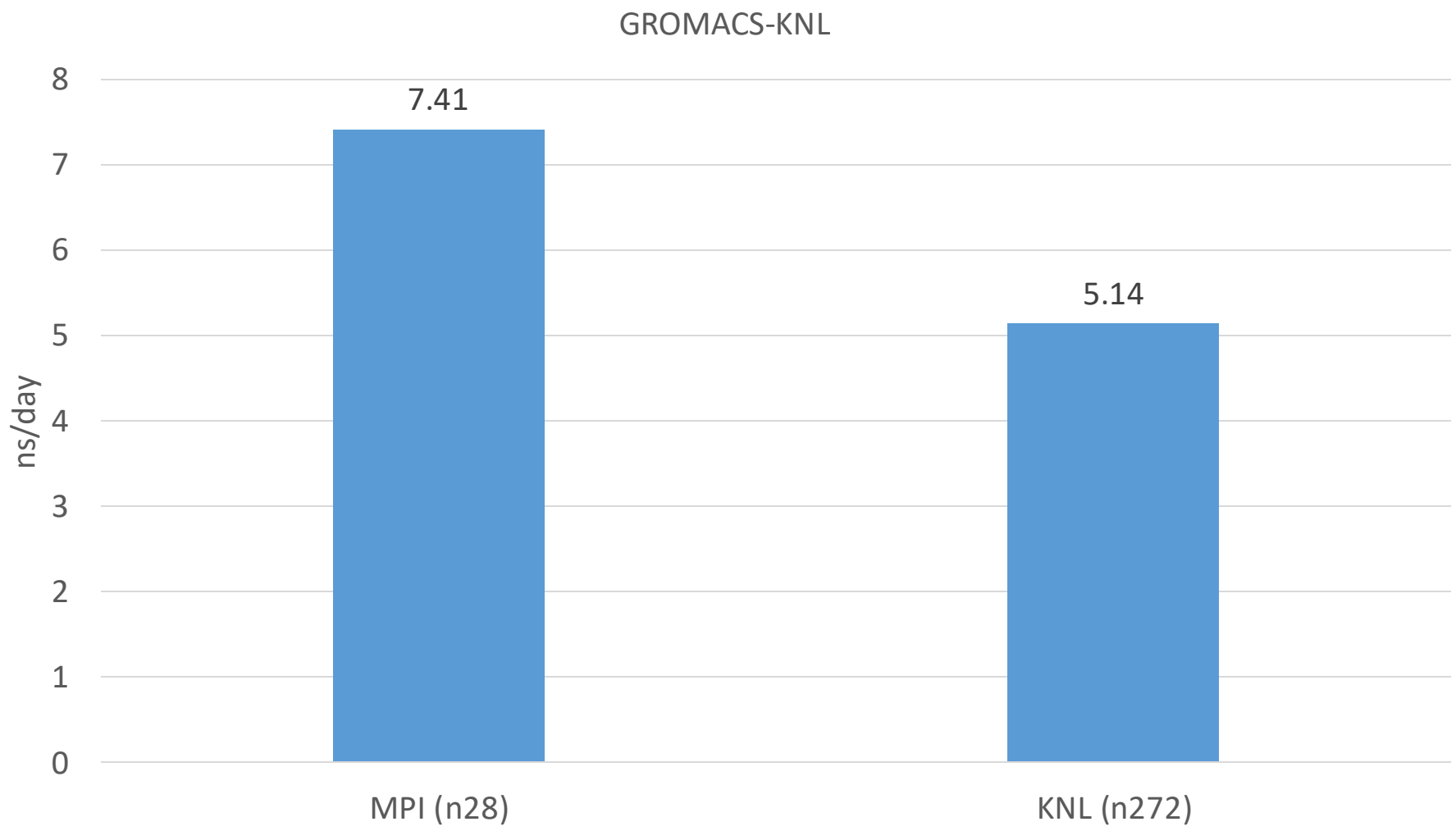
mpirun -np 4 gmx_mpi mdrun -npme -1 -s npt.tpr -ntomp 7 -dlb yes

GROMACS

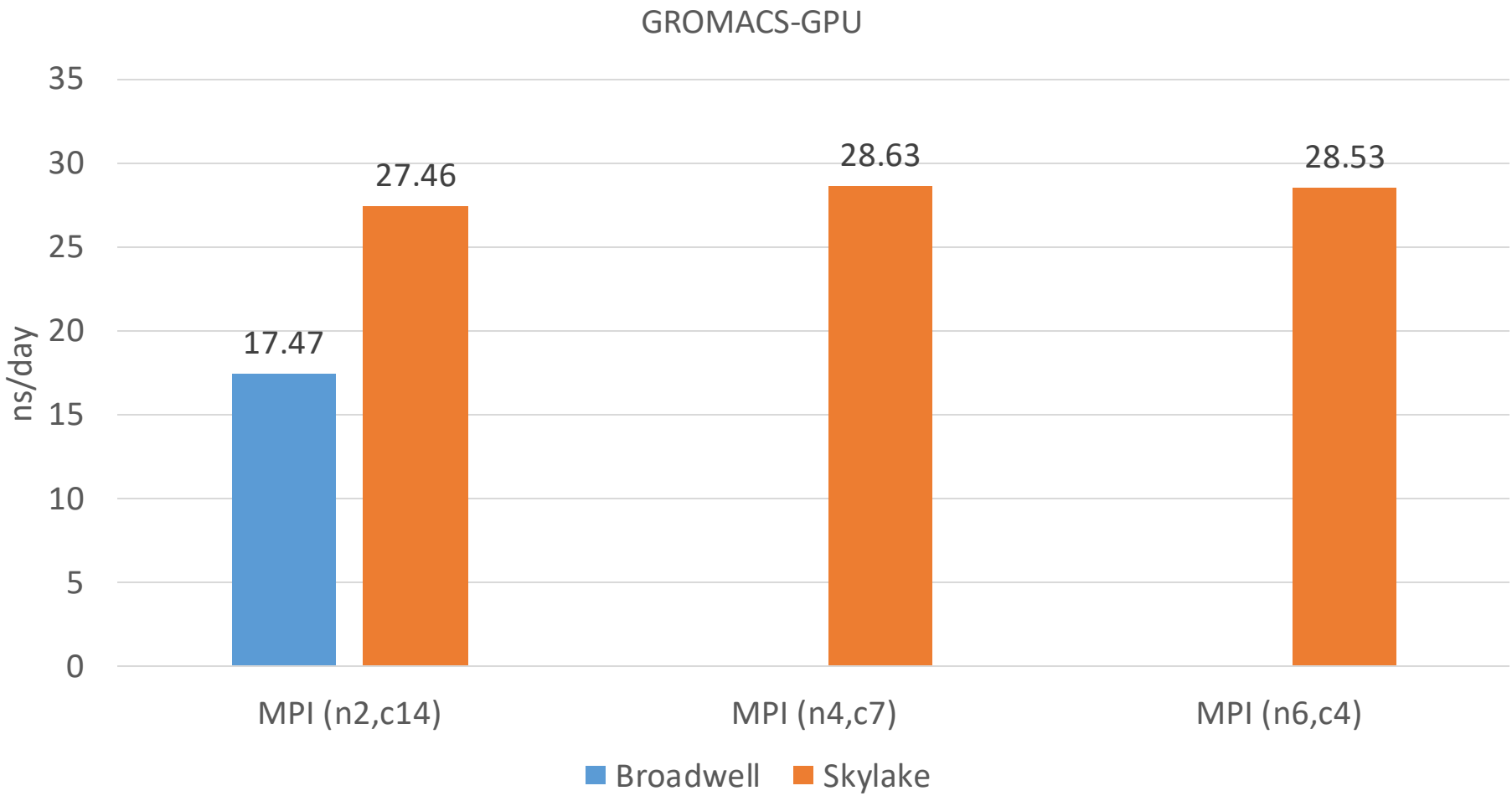

GROMACS
ns/day
0
1
2
3
4
5
6
7
8
4.95
7.19
5.08
7.41
Threaded (n28)
MPI (n28)
Broadwell
Skylake

GROMACS KNL

- Login to kebnekaise-knl.hpc2n.umu.se and submit your script from there
- KNL queue is most of the time available compared to GPU one



GROMACS


GROMACS-GPU
35
30
25
20
15
10
5
0
ns/day
17.47
27.46
28.63
28.53
MPI (n2,c14)
MPI (n4,c7)
MPI (n6,c4)
Broadwell
Skylake

GROMACS

- PLUMED, threaded MPI version is not supported.

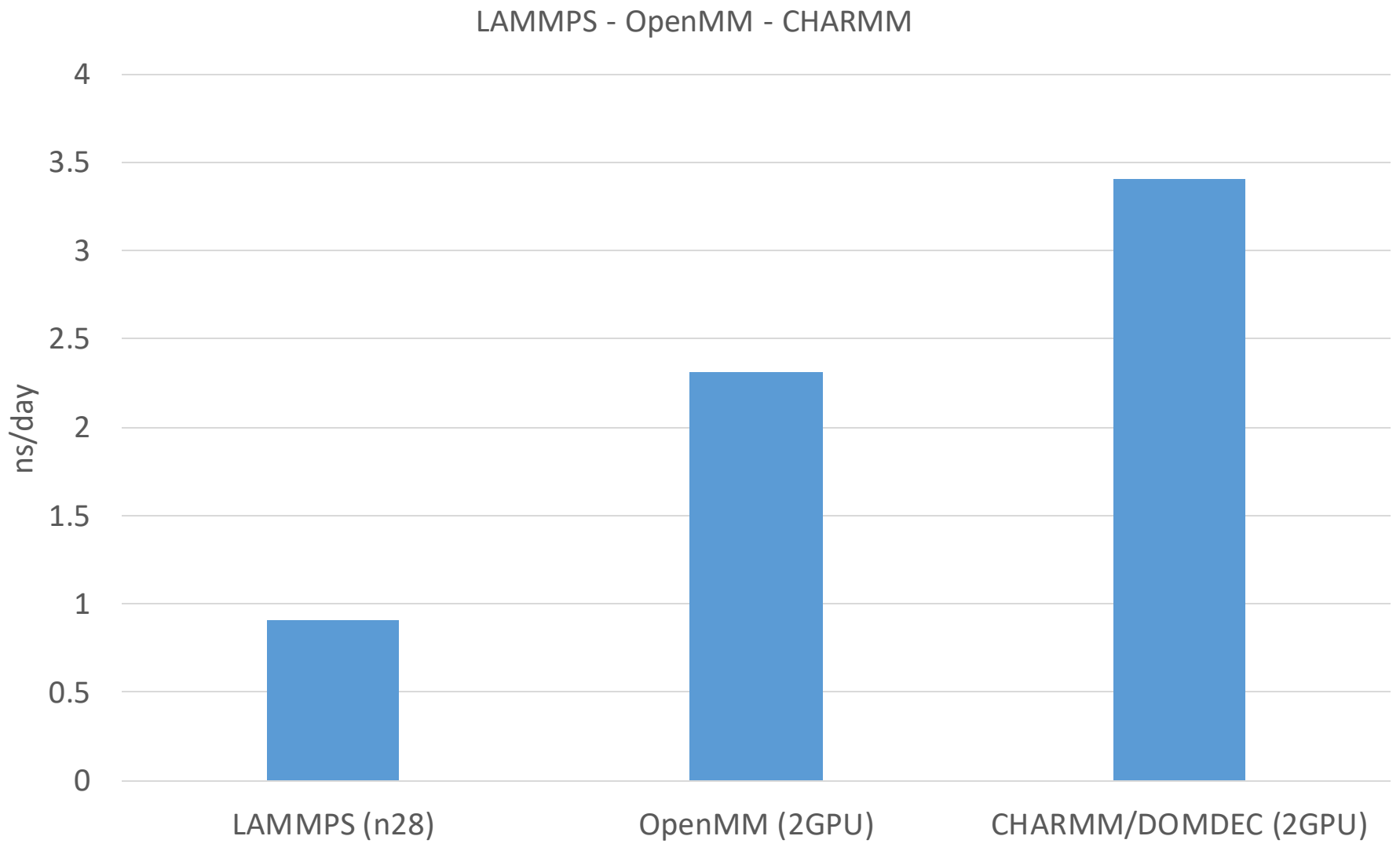
Instead of:

gmx mdrun –ntmpi X

Use:

mpirun gmx_mpi …

LAMMPS - OpenMM - CHARMM



Scratch directory

- If the program writes frequently data, you can get an additional speed up by using the local scratch directory:

```
rm -rf /scratch/test
export parent=/pfs/nobackup/home/u/username/benchmarks/
mkdir /scratch/test
rsync -avzh $parent/ /scratch/test/

cd /scratch/test

namd2 +p 28 +setcpuaffinity input.inp > output.dat

cd $parent
rsync -avzh /scratch/test/ $parent/

rm -rf /scratch/test
```

~10-15% speedup

Abisko (all nodes): 352 GB
Kebnekaise, standard nodes: 171 GB
Kebnekaise, GPU nodes: 171 GB
Kebnekaise, Largemem nodes: 352 GB
(a few have 391 GB)

Tensorflow

```
#!/usr/share/python

import tensorflow as tf
hello = tf.constant('Hello, TensorFlow!')
sess = tf.Session()
print(sess.run(hello))
```

```
#!/bin/bash
#SBATCH -A staff
#SBATCH -t 00:50:00
#SBATCH -N 1
#SBATCH --exclusive
#SBATCH --gres=gpu:k80:2

ml icc/2017.1.132-GCC-5.4.0-2.26
ml ifort/2017.1.132-GCC-5.4.0-2.26
mlCUDA/8.0.44  impi/2017.1.132
ml Python/3.6.1
ml Tensorflow/1.3.0-Python-3.6.1
python tensor.py
```

```
name: Tesla K80
major: 3 minor: 7 memoryClockRate (GHz) 0.8235
pciBusID 0000:0f:00.0
Total memory: 11.17GiB
Free memory: 11.10GiB
2017-12-05 17:13:03.555397: W
Found device 1 with properties:
name: Tesla K80
>>> print(sess.run(hello))
b'Hello, TensorFlow!'
```

Tensorflow

```
#!/usr/share/python
import tensorflow as tf

#Parameters
W = tf.Variable([.3],tf.float32)
b = tf.Variable([-.3],tf.float32)
#Input and output
x = tf.placeholder(tf.float32)
linear_model = W*x+b
y = tf.placeholder(tf.float32)

#Loss
square_delta = tf.square(linear_model-y)
loss = tf.reduce_sum(square_delta)
#Optimize
optimizer = tf.train.GradientDescentOptimizer(0.01)
train = optimizer.minimize(loss)

init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)

for i in range(1000):
   sess.run(train,{x:[1,2,3,4],y:[0,-1,-2,-3]})

print(sess.run([W,b]))
```

Computing the loss

```
name: Quadro K6000
major: 3 minor: 5 memoryClockRate (GHz) 0.9015
pciBusID 0000:06:00.0
Total memory: 11.92GiB
Free memory: 11.82GiB
Creating TensorFlow device (/gpu:0) -> (device: 0, name: Quadro K6000, pci bus id:
0000:06:00.0)

[array([-0.9999969], dtype=float32), array([ 0.99999082], dtype=float32)]
```

GAUSSIAN

Initial input file:

```
$more input_bk.com
%chk=geom_optim.chk
%mem=16GB
#UB3LYP/6-31+G(d) OPT=(ModRedun) SCF=(MaxCycle=256) pop=none NoSymm

45 atoms structure, RESP

+5 15
O       3.744336        -1.126487       6.111505
P       2.893853        -1.251776       4.246949
O       4.150424        -3.154051       4.078061
```

Corrected input file:

```
$more input.com
%cpu=0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27
%gpucpu=0-3=0,7,14,21
%chk=geom_optim.chk
%mem=16GB
#UB3LYP/6-31+G(d) OPT=(ModRedun) SCF=(MaxCycle=256) pop=none NoSymm
```

```
#!/bin/bash
#SBATCH -A staff
#SBATCH -N 1
#SBATCH -c 28
#SBATCH --exclusive
#SBATCH --gres=gpu:k80:2
#SBATCH --time=00:10:00

module add gaussian/16.A.03-AVX2
# Assume that the job file are located
g16.set-cpu+gpu-list input.com
time g16 input
```

Integral=(FineGrid,Acc2E=10) Constants=2006

VASP

Batch script:

#SBATCH -c 10 (number of cores per task)
#SBATCH -n 1 (number of tasks)

ml ifort/2017.1.132-GCC-6.3.0-2.27 OpenMPI/2.0.2
ml NWChem/6.6.revision27746-2015-10-20-Python-2.7.12

mpirun -n 10 nwchem input.nw (this script will fail)

VASP

Batch script:

#SBATCH -c 1 (number of cores per task)
#SBATCH -n 10 (number of tasks)

ml ifort/2017.1.132-GCC-6.3.0-2.27 OpenMPI/2.0.2
ml NWChem/6.6.revision27746-2015-10-20-Python-2.7.12

mpirun -n 10 nwchem input.nw (this script is correct!)