OpenMP

Pedro Ojeda, B. Brydsö, J. Eriksson

HPC2N

pedro.ojeda-may at umu.se

<no current project> - Intel VTune Amplifier@b-an01.hpc2n.umu.se

Welcome | profile

Basic Hotspots Hotspots by CPU Usage viewpoint (change)

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

Elapsed Time: 0.017s

- CPU Time: 0.010s
- Total Thread Count: 1
- Paused Time: 0s

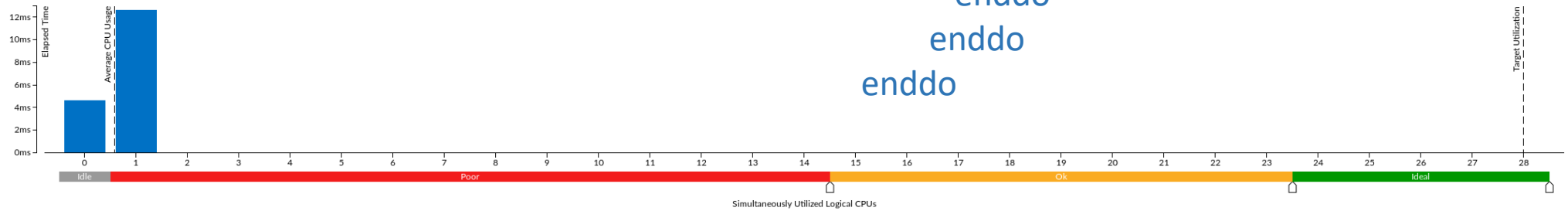
Top Hotspots

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
vec	a.out	0.010s

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.



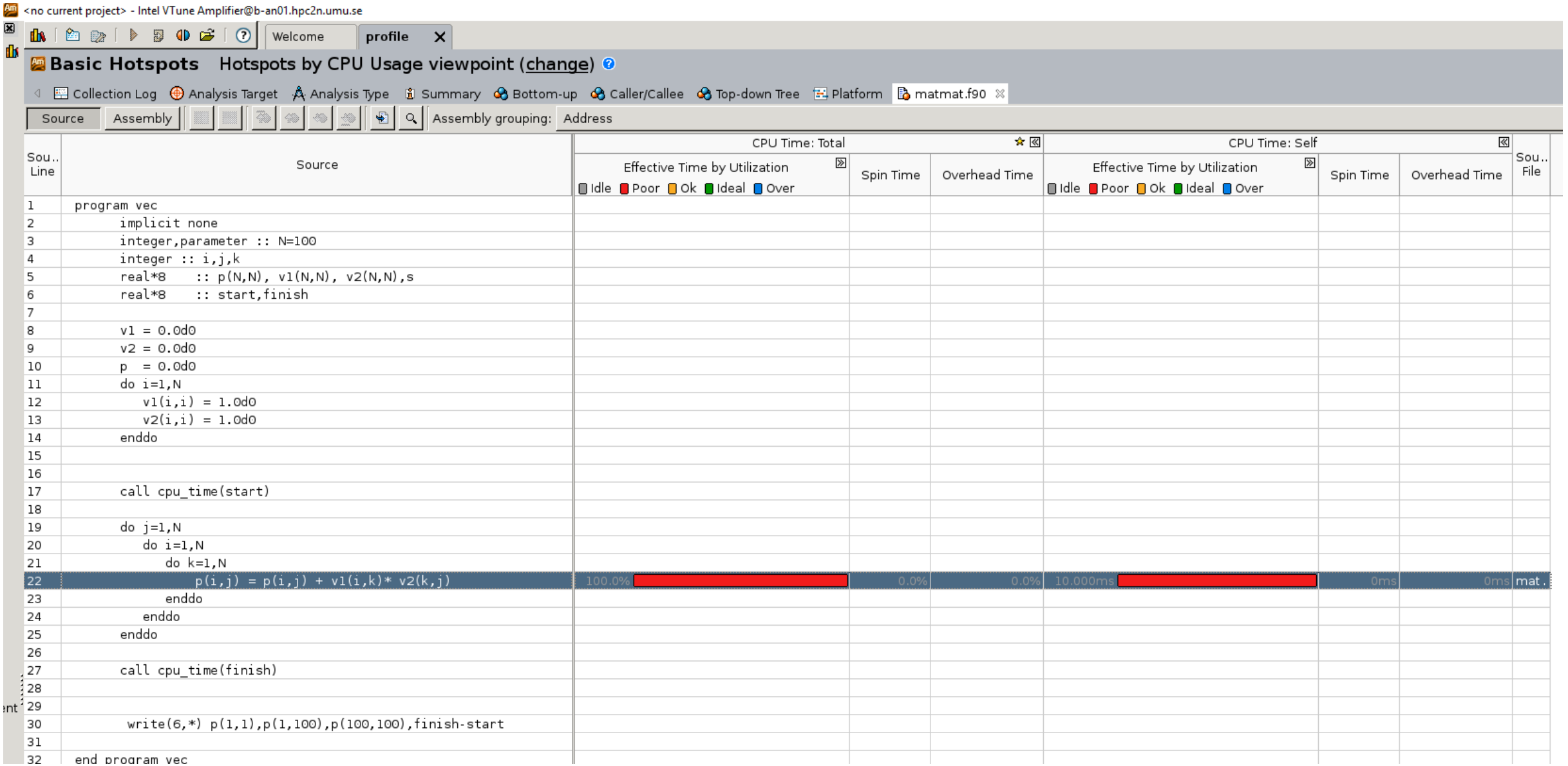
Collection and Platform Info

This section provides information about this collection, including result set size and collection platform data.

- Application Command Line: /home/p/pojedama/pfs/OFFLOADS/a.out
- Operating System: 4.4.0-101-generic NAME="Ubuntu" VERSION="16.04.3 LTS (Xenial Xerus)" ID=ubuntu ID_LIKE=debian PRETTY_NAME="Ubuntu 16.04.3 LTS" VERSION_ID="16.04" HOME_URL="http://www.ubuntu.com/" SUPPORT_URL="http://help.ubuntu.com/" BUG_REPORT_URL="http://bugs.launchpad.net/ubuntu/" VERSION_CODENAME=xenial UB
- Computer Name: b-cn0841.hpc2n.umu.se
- Result Size: 2 MB
- Collection start time: 12:43:07 05/12/2017 UTC
- Collection stop time: 12:43:08 05/12/2017 UTC
- CPU
  - Name: Intel(R) Xeon(R) Processor code named Broadwell-EP
  - Frequency: 2.6 GHz
  - Logical CPU Count: 28

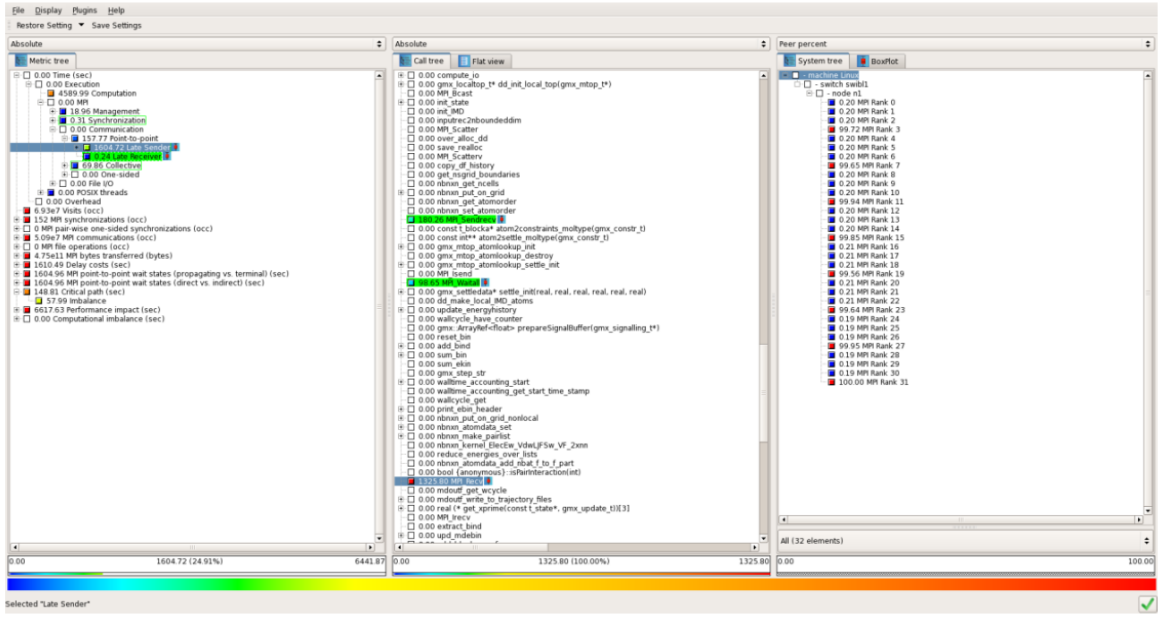
```
do j=1,N
      do i=1,N
        do k=1,N
           p(i,j) = p(i,j) + v1(i,k)* v2(k,j)
        enddo
      enddo
enddo
```

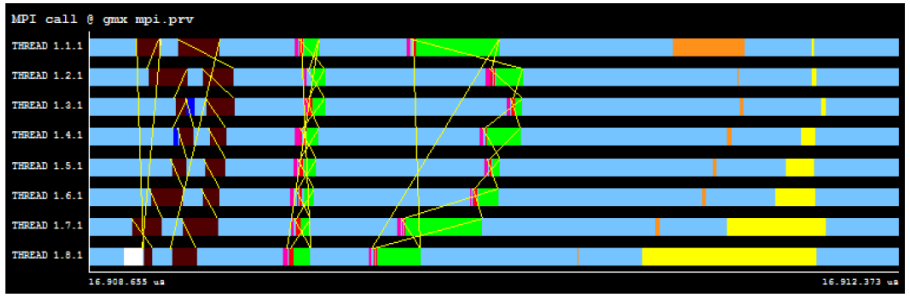
Analyze the serial version of the code


<no current project> - Intel VTune Amplifier@b-an01.hpc2n.umu.se
Welcome
profile
Basic Hotspots
Hotspots by CPU Usage viewpoint (change)
Collection Log
Analysis Target
Analysis Type
Summary
Bottom-up
Caller/Callee
Top-down Tree
Platform
matmat.f90
Source
Assembly
Assembly grouping:
Address
Sou.. Line
Source
CPU Time: Total
Effective Time by Utilization
Idle Poor Ok Ideal Over
Spin Time
Overhead Time
CPU Time: Self
Effective Time by Utilization
Idle Poor Ok Ideal Over
Spin Time
Overhead Time
Sou.. File
1 program vec
2 implicit none
3 integer,parameter :: N=100
4 integer :: i,j,k
5 real*8 :: p(N,N), v1(N,N), v2(N,N),s
6 real*8 :: start,finish
7
8 v1 = 0.0d0
9 v2 = 0.0d0
10 p = 0.0d0
11 do i=1,N
12 v1(i,i) = 1.0d0
13 v2(i,i) = 1.0d0
14 enddo
15
16
17 call cpu_time(start)
18
19 do j=1,N
20 do i=1,N
21 do k=1,N
22 p(i,j) = p(i,j) + v1(i,k)* v2(k,j)
100.0% 0.0% 0.0% 10.000ms 0ms 0ms mat.
23 enddo
24 enddo
25 enddo
26
27 call cpu_time(finish)
28
29
30 write(6,*) p(1,1),p(1,100),p(100,100),finish-start
31
32 end program vec

Profiling Tools at HPC2N

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



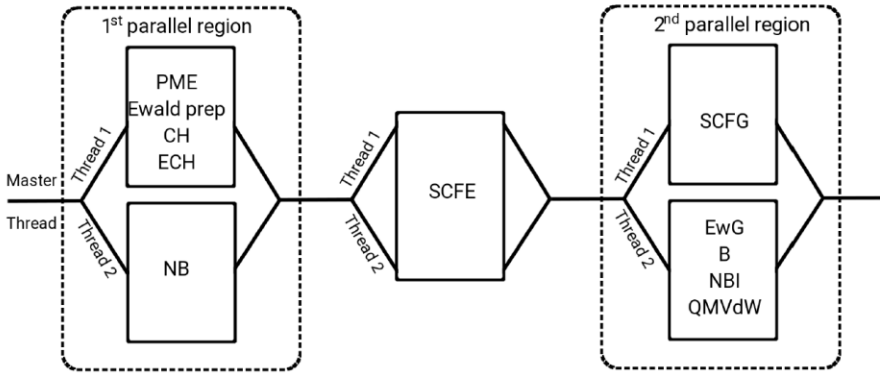


J. Eriksson, P. Ojeda, T. Ponweiser, T. Steinreicher (2017)

OpenMP

A portable fork-join parallel model for architectures with shared memory

- Portable, Fortran, C/C++ bindings
- Many implementations
- Fork-join model
- Shared memory
- Ease of use, significant improvement with 3 or 4 directives
- Task parallelism and loop parallelism



Credits: JCTC, 13, 3525 (2017)

Brief history

- OpenMP was introduced in 1996
- Version 3.0 offered support for tasking
- Version 4.0 started to work with offloads
- Version 4.5-5.0 better support for tasking, for instance taskloop, task_reduction

Processes communication

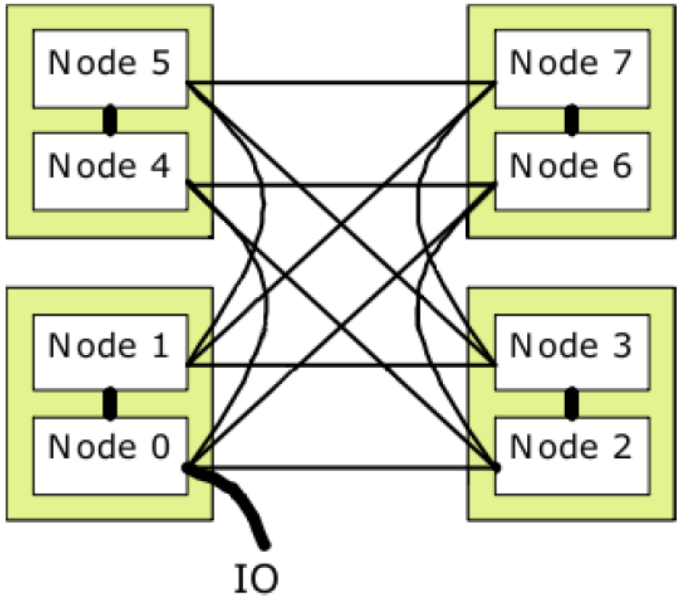


Figure : Nodes.

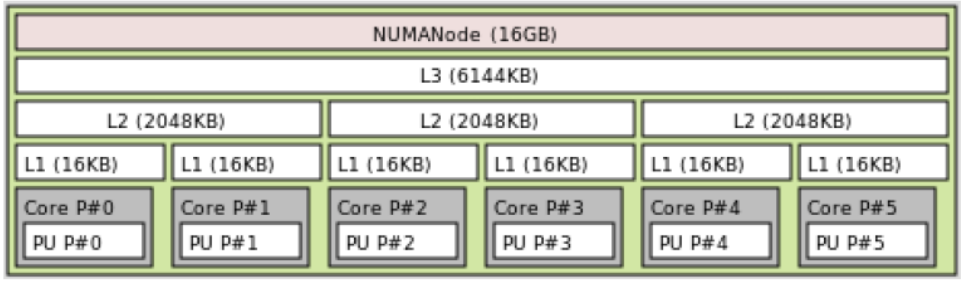


Figure : NUMA machine.

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

OpenMP directives

```
#pragma omp name [clause[[,] clause]...]
```

- Each directive begins with `#pragma omp`
- followed by the **name** of the directive
- and a possibly empty list of **clauses**.
- The directive must end with a new line.
- Long directives may be split into multiple source lines by appending a backslash to continued lines.

Example:

```
#pragma omp parallel
{
   // ..inside parallel construct..
   subroutine( );
}
```

Loop-level parallelism

```
#pragma omp for [clauses]
for( init-expr ; test-expr ; inc-expr )
{ // ..loop body..  }
```

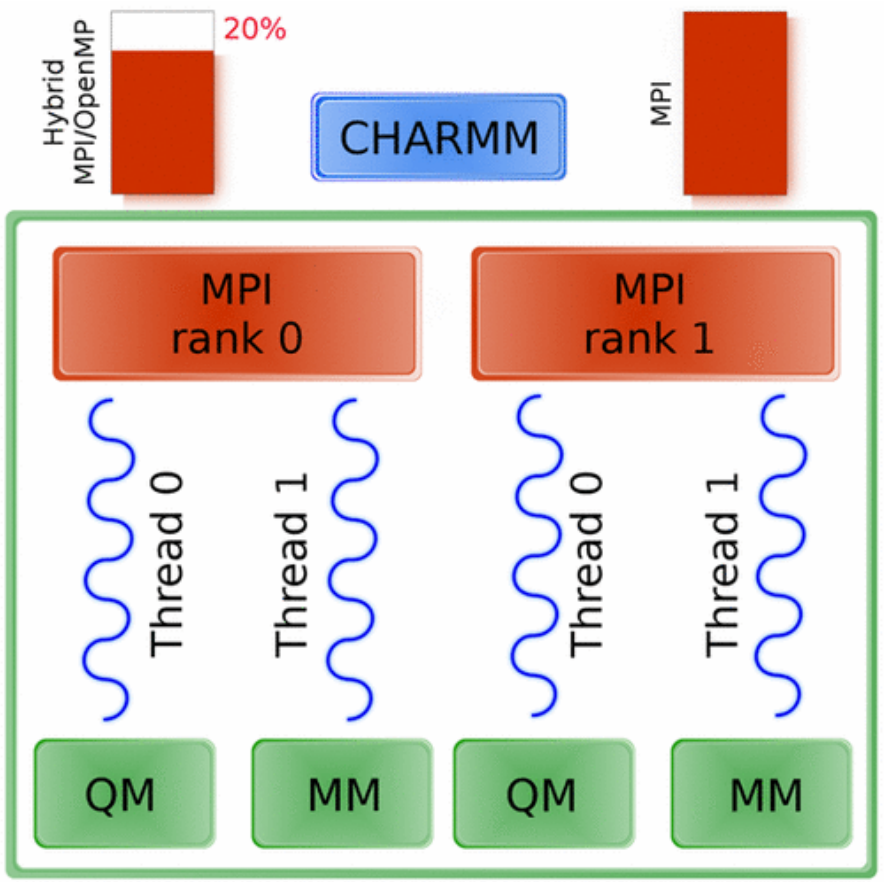
```
int k;
#pragma omp for
for( k = 0; k < 10; ++k )
{
  // ..k implied private by parallel for..
}
```

Task-level parallelism

Each thread can do an independent task for each section

Example (Fortran):

```
        PROGRAM HELLO
!$OMP SECTIONS clauses...
!$OMP SECTION
      ...task
!$OMP SECTION
      ...task
!$OMP SECTION
      ...task
!$OMP END SECTIONS end_clauses
        END
```

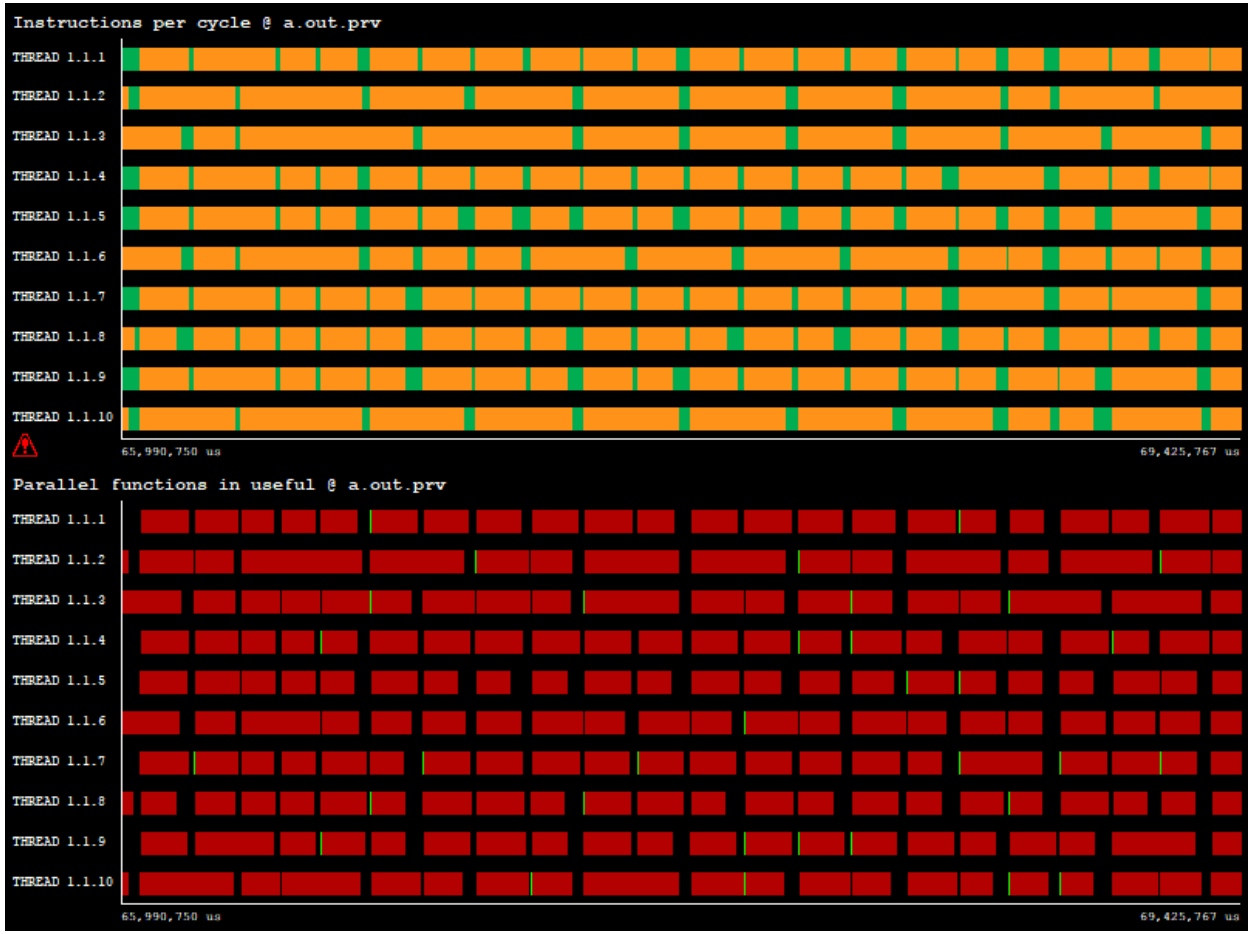


Credits: JCTC, 13, 3525 (2017)

Thread synchronization

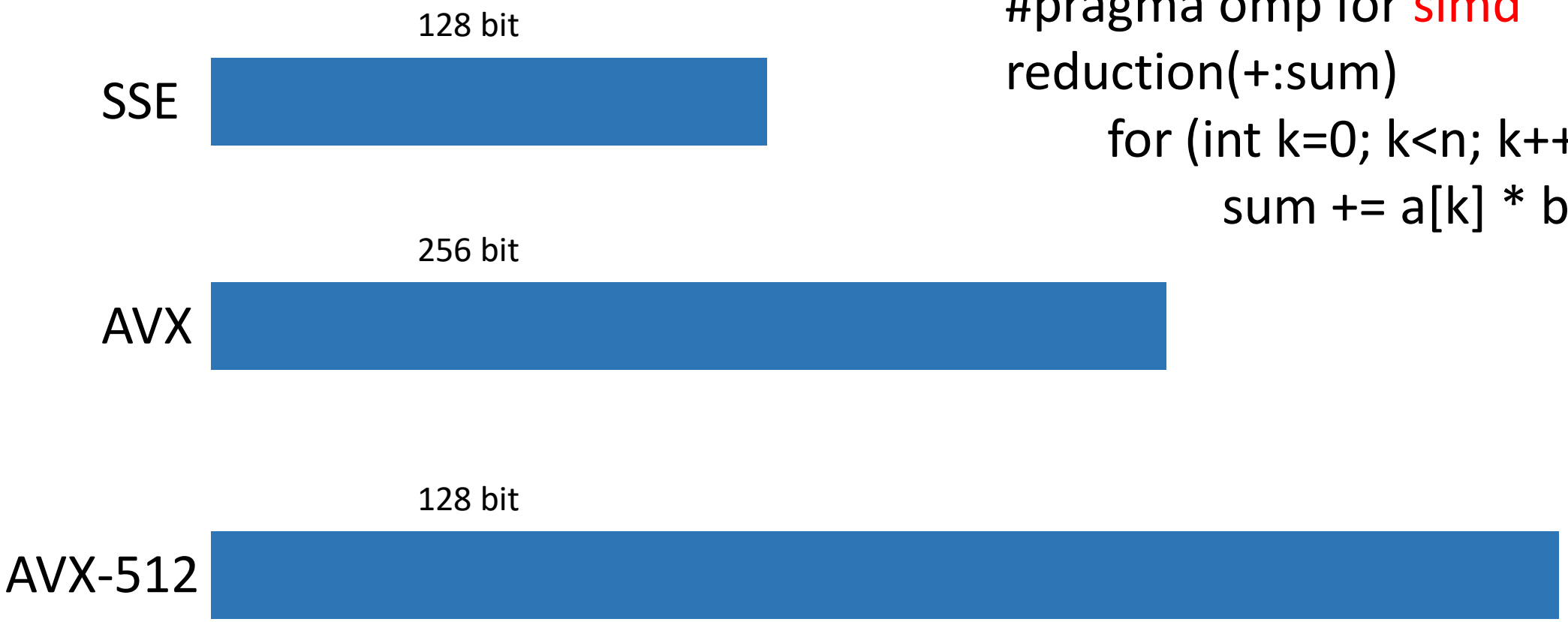
```
!$OMP BARRIER        (Fortran)
```

```
#pragma omp barrier   (C)
```



Barriers are expensive!

SIMD instructions




```
float sum = 0.0f
#pragma omp for simd
reduction(+:sum)
    for (int k=0; k<n; k++)
        sum += a[k] * b[k];
```

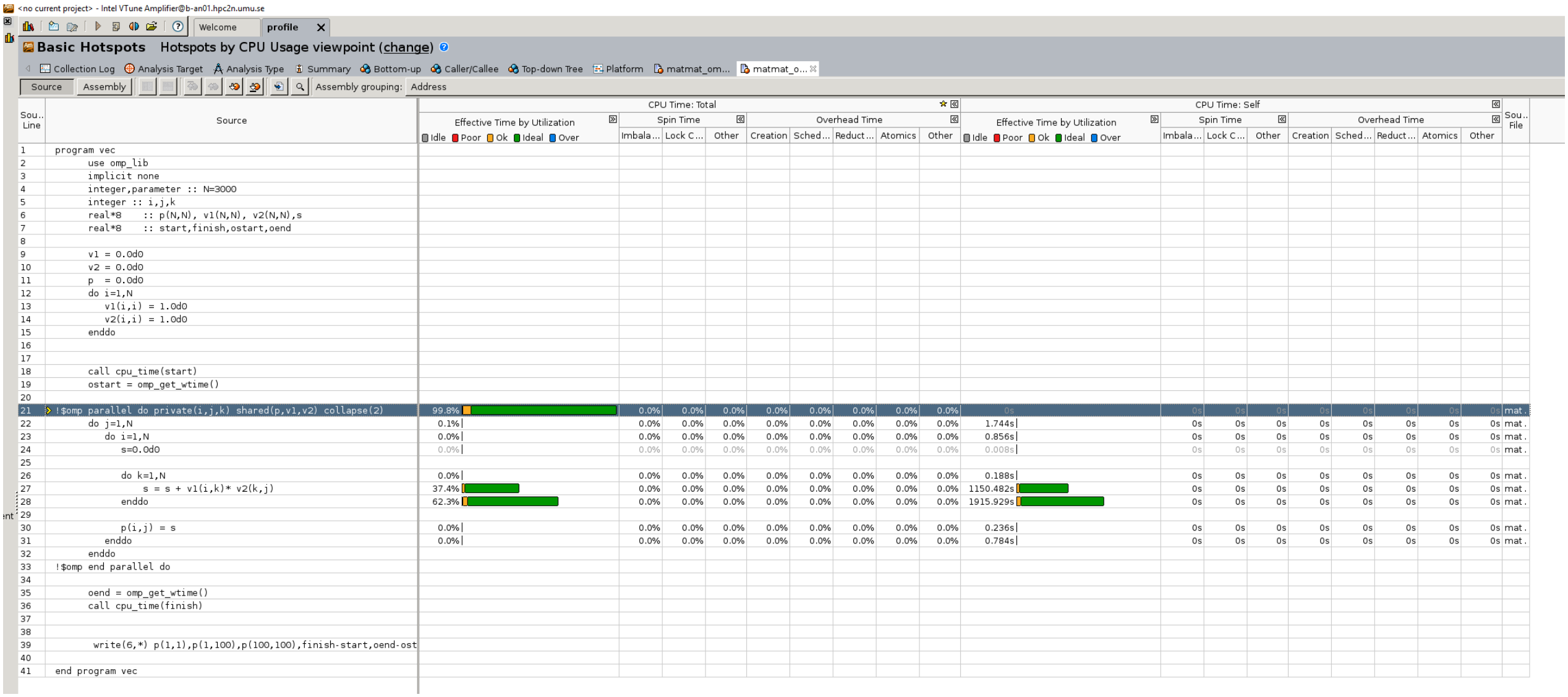
Very useful for KNL processors. Functions can be vectorized as well.
Check if your compiler is vectorising important loops (-qopt-report=5)

Matrix-matrix multiplication

```
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
```

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

gfortran -O3 -fopenmp -ffast-math


<no current project> - Intel VTune Amplifier@b-an01.hpc2n.umu.se
Welcome
profile
Basic Hotspots
Hotspots by CPU Usage viewpoint (change)
Collection Log
Analysis Target
Analysis Type
Summary
Bottom-up
Caller/Callee
Top-down Tree
Platform
matmat_om...
matmat_o...
Source
Assembly
Assembly grouping:
Address
Sou.. Line
Source
CPU Time: Total
Effective Time by Utilization
Idle Poor Ok Ideal Over
Spin Time
Imbala...
Lock C...
Other
Overhead Time
Creation
Sched...
Reduct...
Atomics
Other
CPU Time: Self
Effective Time by Utilization
Idle Poor Ok Ideal Over
Spin Time
Imbala...
Lock C...
Other
Overhead Time
Creation
Sched...
Reduct...
Atomics
Other
Sou.. File
1 program vec
2 use omp_lib
3 implicit none
4 integer,parameter :: N=3000
5 integer :: i,j,k
6 real*8 :: p(N,N), v1(N,N), v2(N,N),s
7 real*8 :: start,finish,ostart,oend
8
9 v1 = 0.0d0
10 v2 = 0.0d0
11 p = 0.0d0
12 do i=1,N
13 v1(i,i) = 1.0d0
14 v2(i,i) = 1.0d0
15 enddo
16
17
18 call cpu_time(start)
19 ostart = omp_get_wtime()
20
21 !$omp parallel do private(i,j,k) shared(p,v1,v2) collapse(2) 99.8% 0.0% 0.0% 0.0% 0.0% 0.0% 0.0% 0.0% 0.0% 0s 0s 0s 0s 0s 0s 0s 0s 0s mat..
22 do j=1,N 0.1% 0.0% 0.0% 0.0% 0.0% 0.0% 0.0% 0.0% 0.0% 1.744s 0s 0s 0s 0s 0s 0s 0s 0s mat..
23 do i=1,N 0.0% 0.0% 0.0% 0.0% 0.0% 0.0% 0.0% 0.0% 0.0% 0.856s 0s 0s 0s 0s 0s 0s 0s 0s mat..
24 s=0.0d0 0.0% 0.0% 0.0% 0.0% 0.0% 0.0% 0.0% 0.0% 0.0% 0.008s 0s 0s 0s 0s 0s 0s 0s 0s mat..
25
26 do k=1,N 0.0% 0.0% 0.0% 0.0% 0.0% 0.0% 0.0% 0.0% 0.0% 0.188s 0s 0s 0s 0s 0s 0s 0s 0s mat..
27 s = s + v1(i,k)* v2(k,j) 37.4% 0.0% 0.0% 0.0% 0.0% 0.0% 0.0% 0.0% 0.0% 1150.482s 0s 0s 0s 0s 0s 0s 0s 0s mat..
28 enddo 62.3% 0.0% 0.0% 0.0% 0.0% 0.0% 0.0% 0.0% 0.0% 1915.929s 0s 0s 0s 0s 0s 0s 0s 0s mat..
29
30 p(i,j) = s 0.0% 0.0% 0.0% 0.0% 0.0% 0.0% 0.0% 0.0% 0.0% 0.236s 0s 0s 0s 0s 0s 0s 0s 0s mat..
31 enddo 0.0% 0.0% 0.0% 0.0% 0.0% 0.0% 0.0% 0.0% 0.0% 0.784s 0s 0s 0s 0s 0s 0s 0s 0s mat..
32 enddo
33 !$omp end parallel do
34
35 oend = omp_get_wtime()
36 call cpu_time(finish)
37
38
39 write(6,*) p(1,1),p(1,100),p(100,100),finish-start,oend-ost
40
41 end program vec

Elapsed Time: 117.487s

- CPU Time: 3076.825s
- Total Thread Count: 28
- Paused Time: 0s

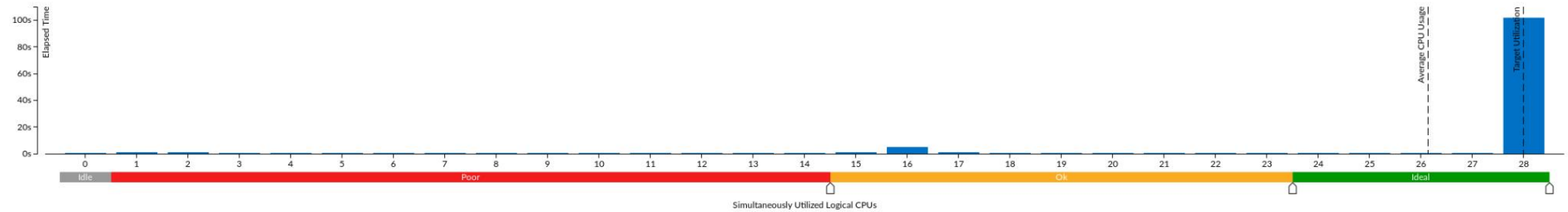
Top Hotspots

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
MAIN__ompparallel_for@21	a.out	3070.227s
__kmp_fork_barrier	libiomp5.so	4.974s
__kmp_join_barrier	libiomp5.so	0.746s
__kmp_join_call	libiomp5.so	0.368s
vec	a.out	0.310s
[Others]		0.200s

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.



Collection and Platform Info

This section provides information about this collection, including result set size and collection platform data.

- Application Command Line: /home/p/pojedama/pfs/OFFLOADS/a.out
- Operating System: 4.4.0-101-generic NAME="Ubuntu" VERSION="16.04.3 LTS (Xenial Xerus)" ID=ubuntu ID_LIKE=debian PRETTY_NAME="Ubuntu 16.04.3 LTS" VERSION_ID="16.04" HOME_URL="http://www.ubuntu.com/" SUPPORT_URL="http://help.ubuntu.com/" BUG_REPORT_URL="http://bugs.launchpad.net/ubuntu/" VERSION_CODENAME=xenial UBUNTU_CODENAME=xenial
- Computer Name: b-cn0603.hpc2n.umu.se
- Result Size: 57 MB
- Collection start time: 13:55:12 05/12/2017 UTC
- Collection stop time: 13:57:10 05/12/2017 UTC
- CPU
  - Name: Intel(R) Xeon(R) Processor code named Broadwell-EP
  - Frequency: 2.6 GHz
  - Logical CPU Count: 28

How to specify the number of threads

- pragma omp parallel num_threads(N)
- omp_set_num_threads(N) [function]
- export OMP_NUM_THREADS=N [env. var.]

OpenMP code is not scaling?

- Loops are not sufficiently large
- Inner loops not being vectorized
- Outermost loops are better suited for parallelization
- Maybe a task-based approached is needed
- Couple multiple parallel regions into a single one
- False sharing

False sharing

```
#pragma omp parallel num_threads(NUM_THREADS)
{
 int thread = omp_get_thread_num();
 D[thread] = 0;
 #pragma omp for schedule(static)
 for (i=0; i<=n; i++)
    D[thread] += i%10;
#pragma omp atomic
sum+=D[thread];
}
```

```
#pragma omp parallel num_threads(NUM_THREADS)
{
 int thread = omp_get_thread_num();
 D[thread][0] = 0;
 #pragma omp for schedule(static)
 for (i=0; i<=n; i++)
    D[thread][0] += i%10;
#pragma omp atomic
sum+=D[thread][0];
}
```

Specifications - OpenMP

www.openmp.org/specifications/

www.openmp.org/specifications/

OpenMP
Enabling HPC since 1997

The OpenMP API specification for parallel programming

Home | Specifications | Blog | Community | Resources | News & Events | About

Specifications

Home > Specifications

OpenMP 4.5 Specifications

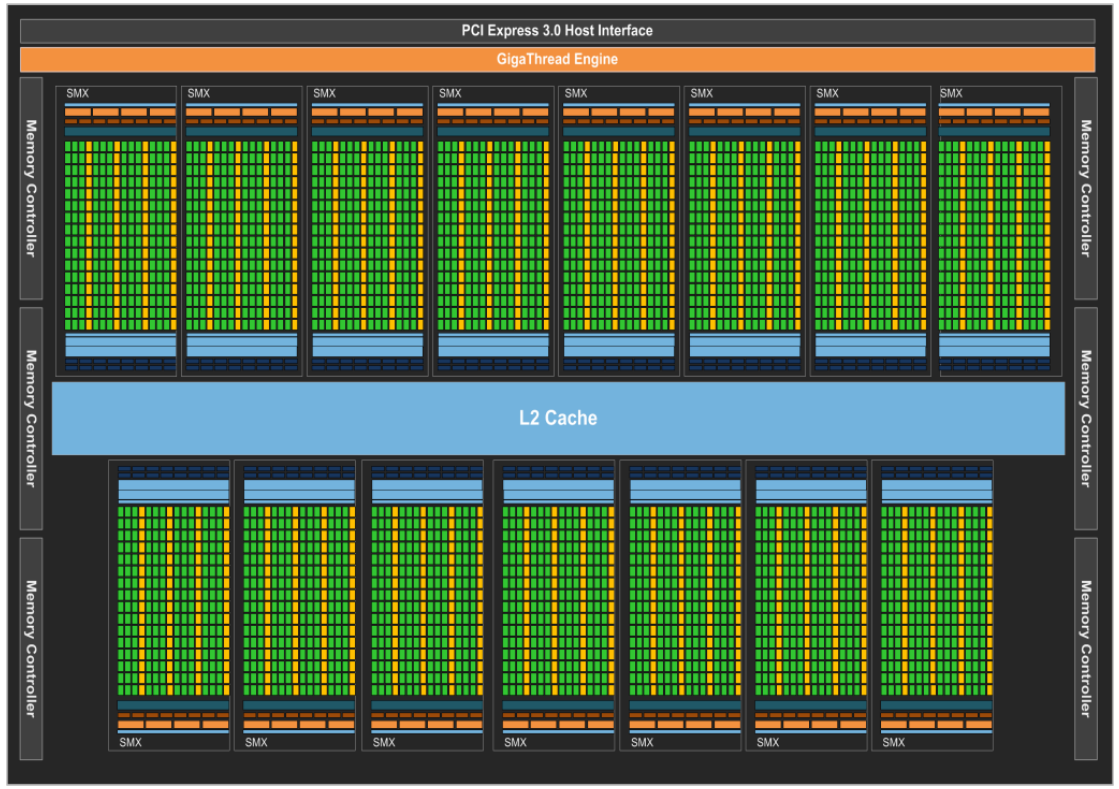
- OpenMP 4.5 Complete Specifications (Nov 2015) *pdf*
  - OpenMP 4.5 Discussion Forum
- OpenMP 4.5 Summary Card – C/C++ (Nov 2015) *pdf*
- OpenMP 4.5 Summary Card – Fortran (Nov 2015) *pdf*
- OpenMP 4.5 Examples (Nov 2016) *pdf*
  - OpenMP 4.5 Examples Discussion Forum

OpenMP 4.0 Specifications

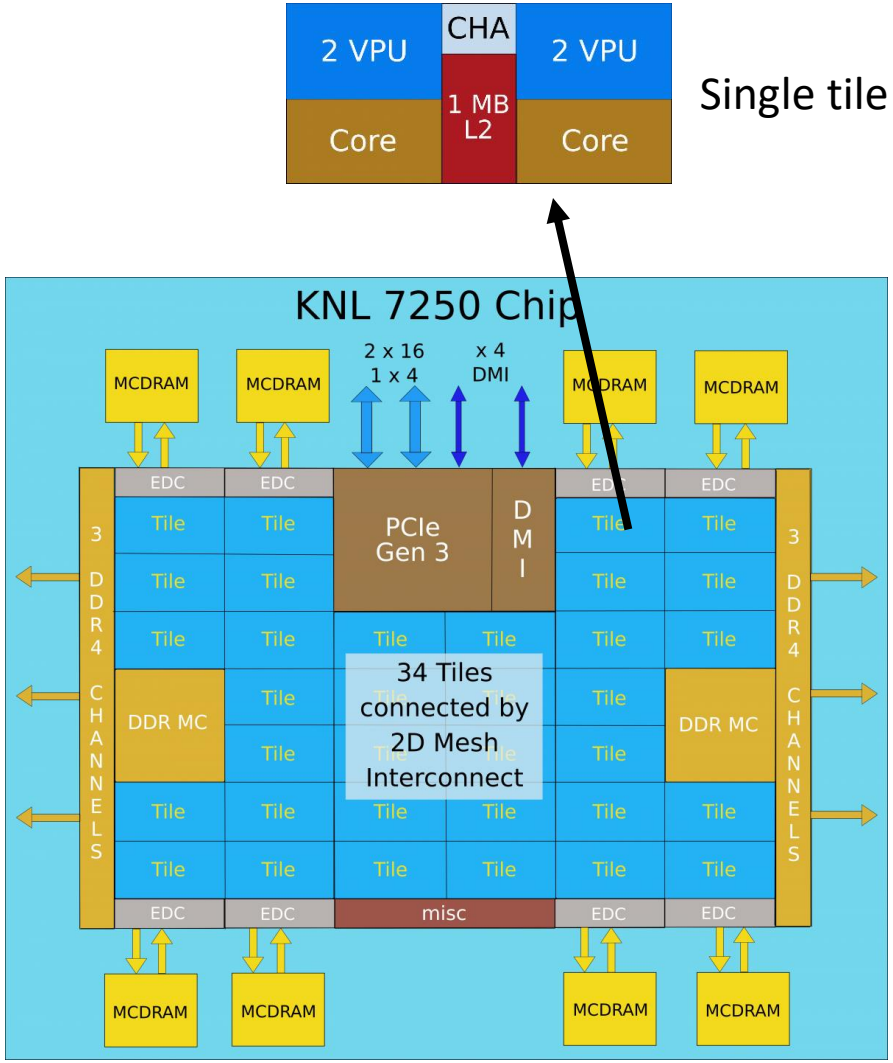
- OpenMP 4.0 Complete Specifications (July 2013) (PDF)
- OpenMP 4.0 Discussion Forum
- OpenMP 4.0 Summary Card – C/C++ (October 2013 PDF)
- OpenMP 4.0 Summary Card – Fortran (October 2013 PDF)
- OpenMP Examples 4.0.2 (March 2015 PDF)
- OpenMP 4.0.1 Examples (February 2014 PDF)

OpenMP+Accelerators

Accelerators

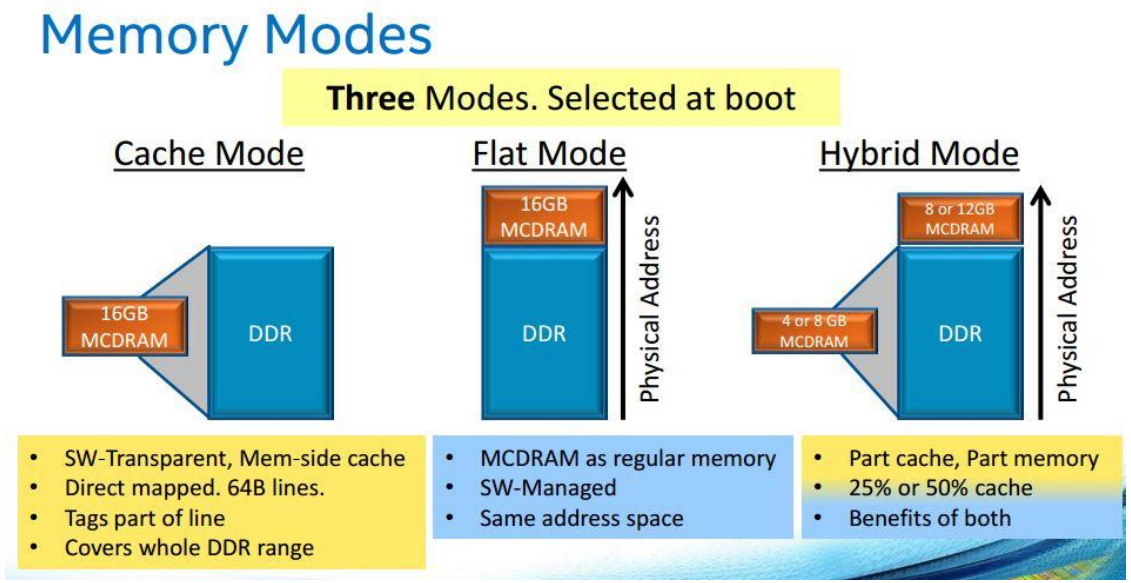


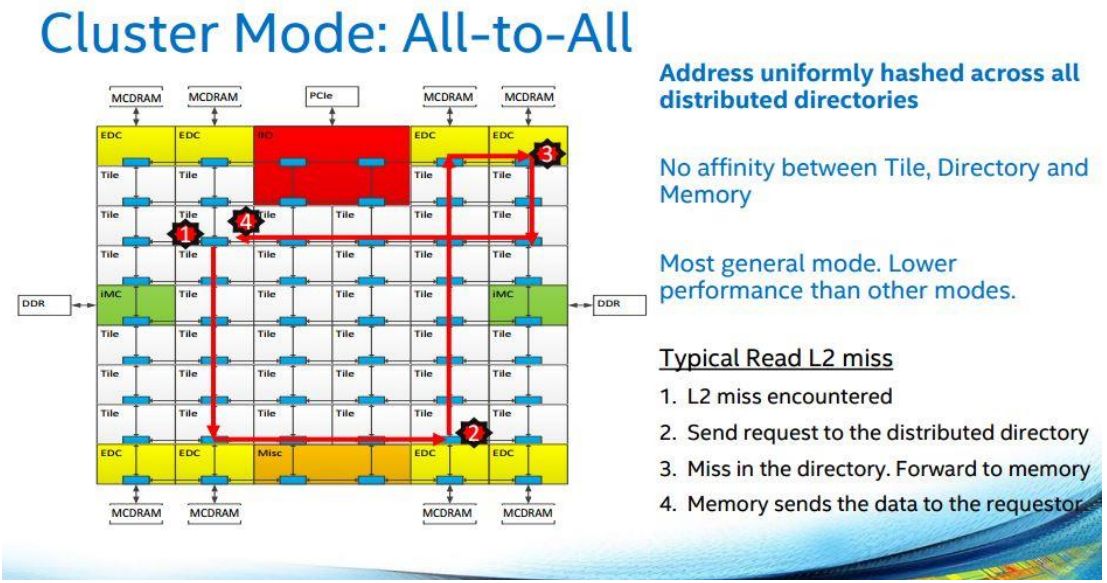
GPU showing the independent units Streaming Multiprocessors (SM).



KNL, composed of several Tiles

KNL





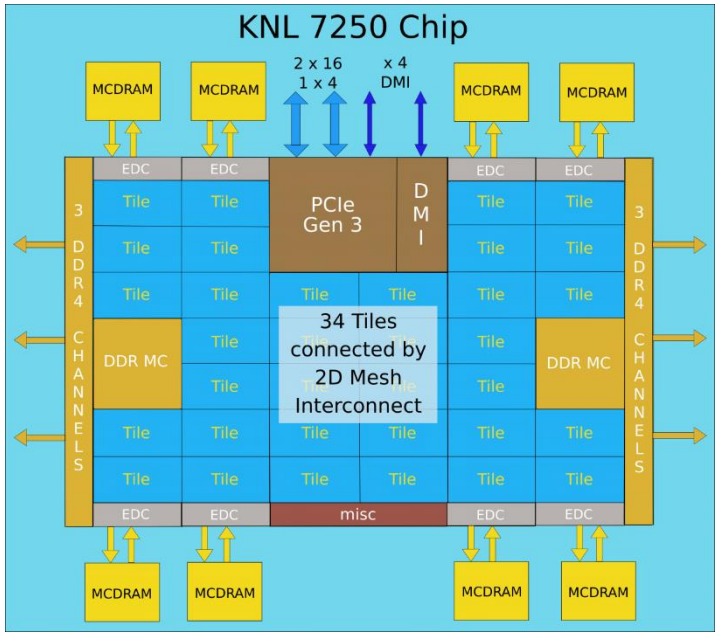
#SBATCH --constraint=a2a,cache

#SBATCH --gres=hbm:4G

Credits: PRACE Best practice KNL (2017)

KNL Thread affinity

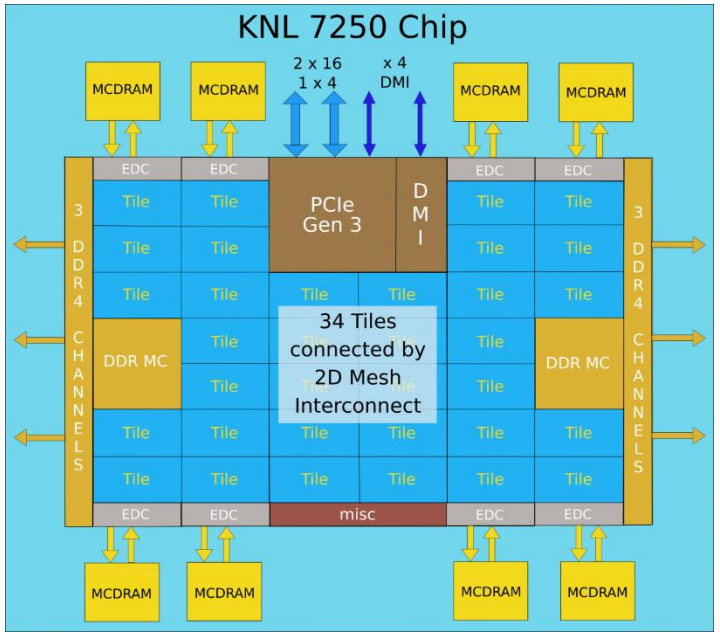
There are physical 68 cores with 4 hyperthreads on each.



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

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

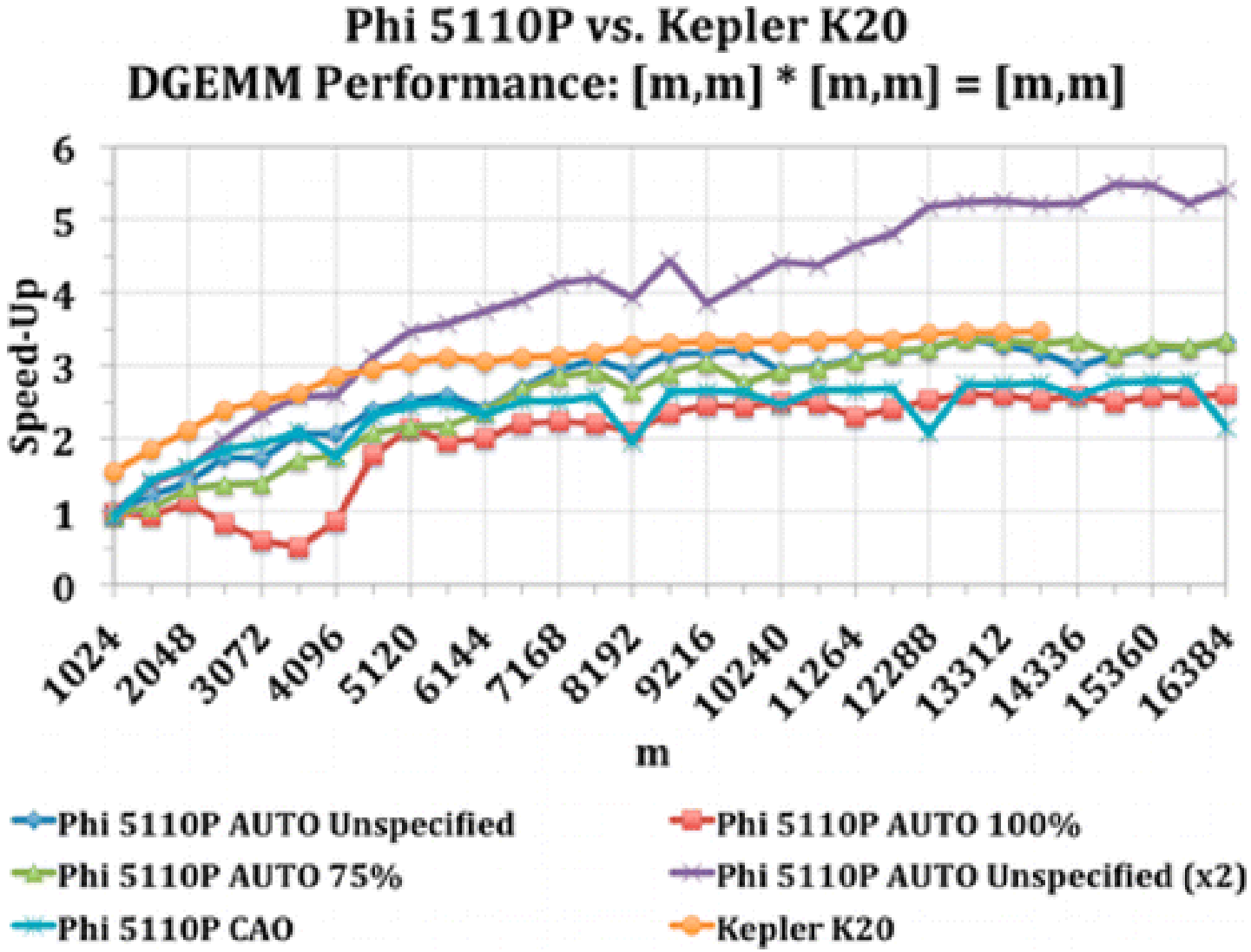
Alternatively, use Intel var.

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)
```

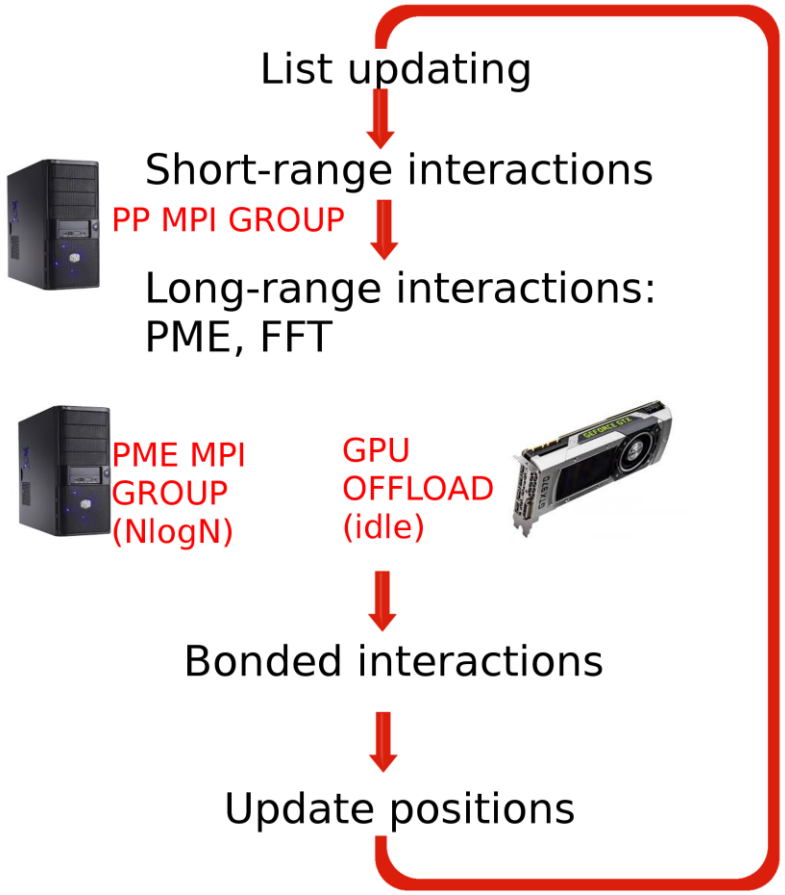
Intel Xeon Phi Coprocessor vs. GPU



JCTC, 10, 908 (2014)

OpenMP Offloads

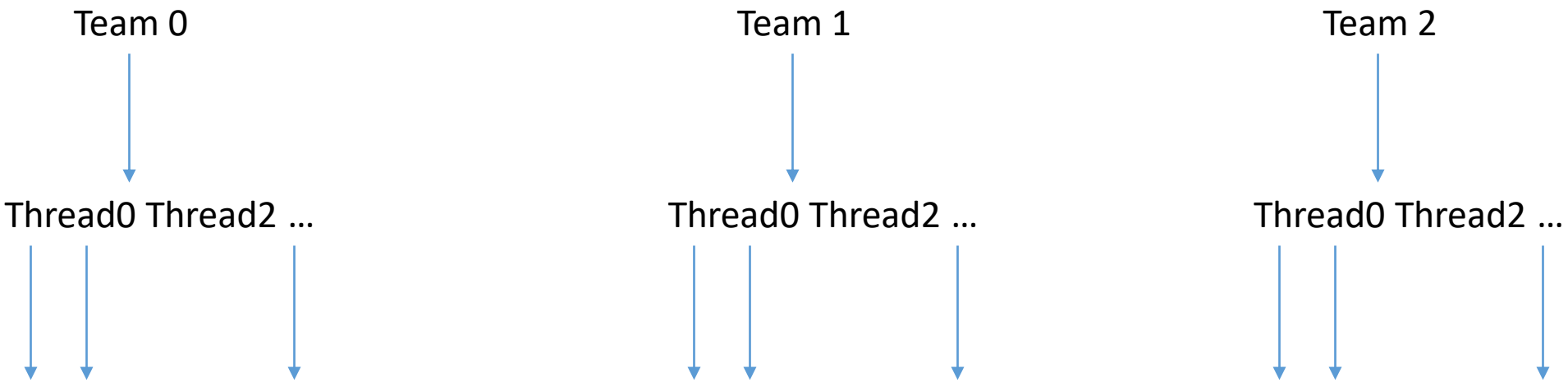
- Starting from version 4.0 OpenMP offers support for offloading tasks to accelerators



$$U = \sum_{\text{bonds}} \frac{1}{2} k_{\text{bonds}} (r - r_0)^2 + \sum_{\text{angles}} \frac{1}{2} k_{\text{angle}} (\theta - \theta_0)^2$$
$$+ \sum_{\text{torsions}} \sum_{j} V_j (1 + \cos j\phi)$$
$$+ \sum_{\text{Coulomb}}^{i<j} \frac{q_i q_j}{r_{ij}} + \sum_{\text{VdW}}^{i<j} \left\{ 4\epsilon_{ij} \left[\left(\frac{\sigma_{ij}}{r_{ij}} \right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}} \right)^{6} \right] \right\}$$

OpenMP Offloads

- C/C++ #pragma omp target
- Fortran !$omp target
- Team concept (useful for GPUs):



Compiling code

- OpenMP offloads can be done on KNLs using common compiler chains:

  GNU: gcc/gfortran -march=native -mtune=native <source>

  Intel: icc/ifort –O3 –xHost <source> (KNL build node)

  icc/ifort –O3 -xMIC-AVX512 <source> (login node)

- OpenMP offloads to GPU:
- GNU: gcc/gfortran –fopenmp <source>

OpenMP Offloads common issues

GPUs:

- Size of the loop and matrices
- Data transfer from and to GPUs
- Amount of FLOP

KNLs:

- Size of loops and matrices
- Binding of threads
- Vectorization
- For Hybrid MPI/OpenMP codes, number of MPI processes

Practical example GPUs

```
!$omp parallel do collapse(2)
private(j)
      do i=1,N
         p(i) = a* v2(i)+p(i)
enddo
!$omp end parallel do
```

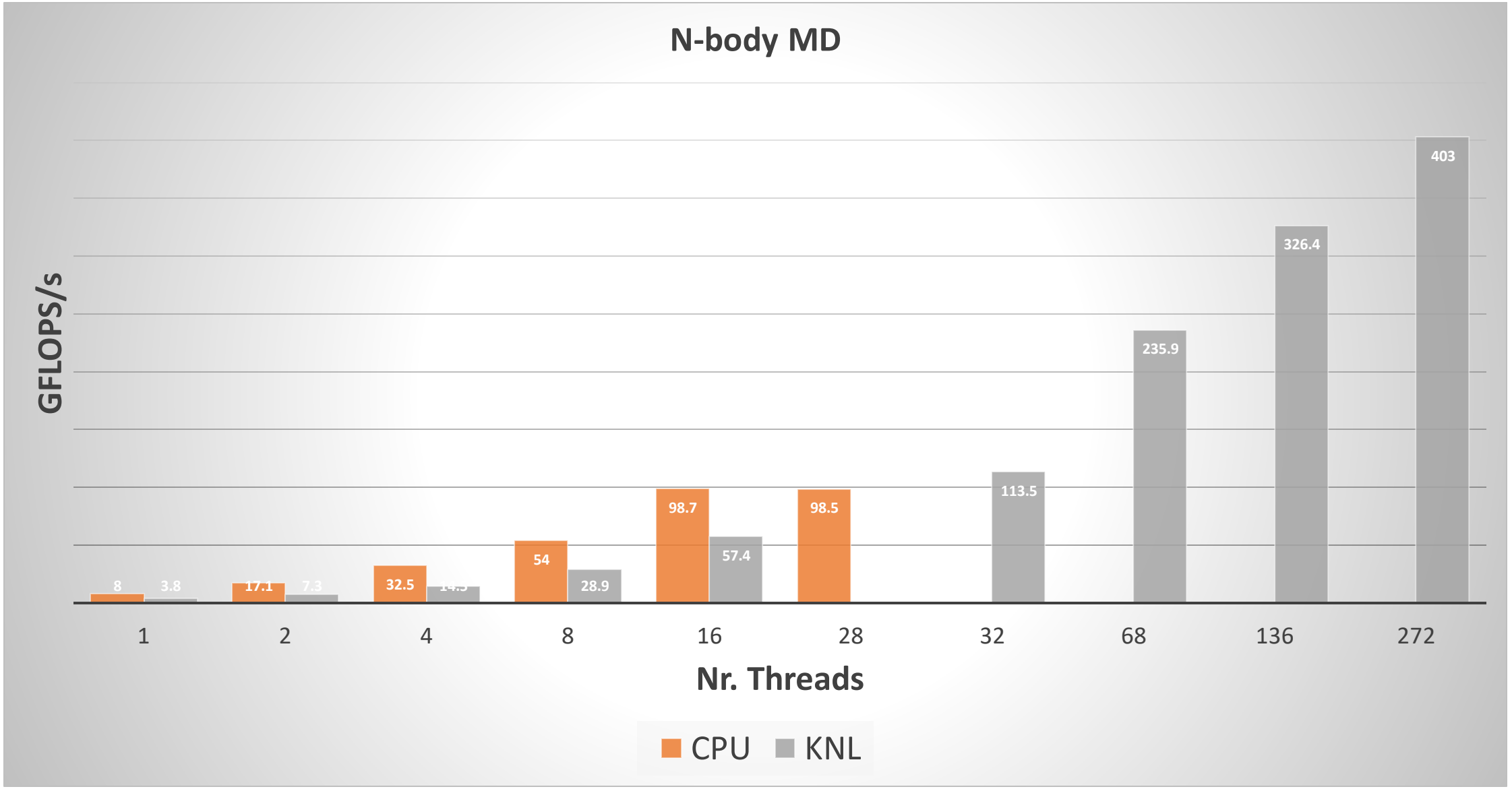
```
!$omp target teams distribute parallel do collapse(2)
private(i,j)
      do i=1,N
          p(i) = a* v2(i)+p(i)
       enddo
!$omp end target teams distribute parallel do
```

Practical example GPUs

```
!$omp parallel do collapse(2)
private(j)
     do i=1,N
        p(i) = a* v2(i)+p(i)
enddo
!$omp end parallel do
```

```
!$omp target map(tofrom:p) map(to:v2,i,a)
!$omp teams distribute parallel do collapse(2)
private(i,j)
     do i=1,N
         p(i) = a* v2(i)+p(i)
      enddo
!$omp end teams distribute parallel do
!$omp end target
```

It is more efficient to copy the date once at the beginning


N-body MD
GFLOPS/s
8
3.8
17.1
7.3
32.5
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

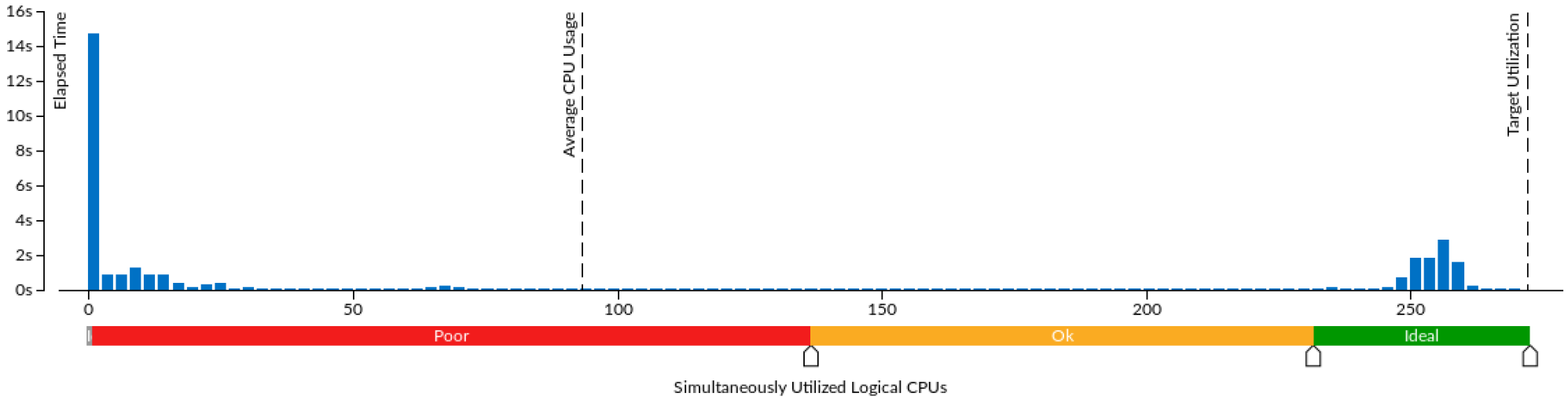
Top Hotspots

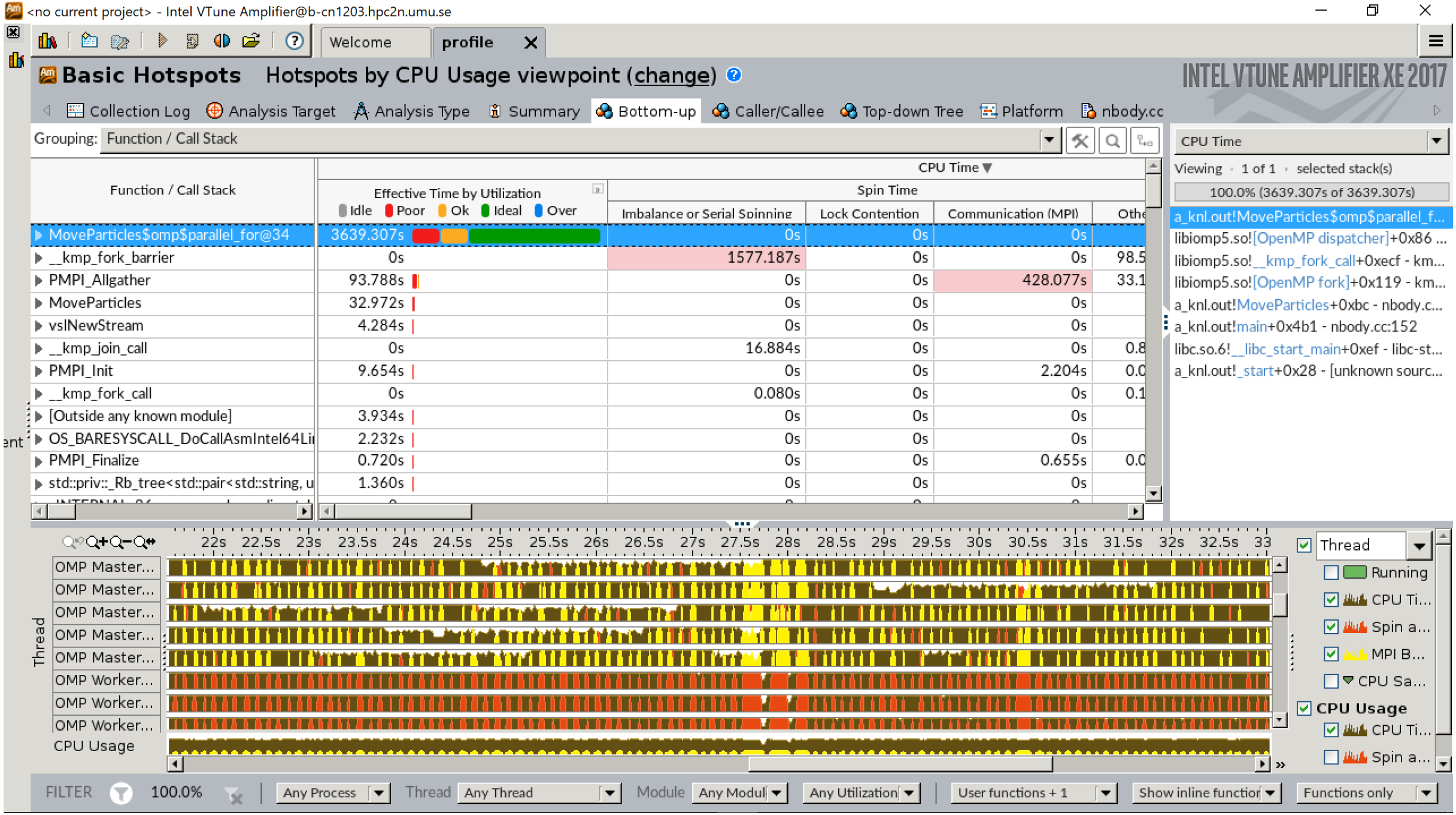
This section lists the most active functions in your application. Optimizing these hotspot functions typically results in improving overall application performance.

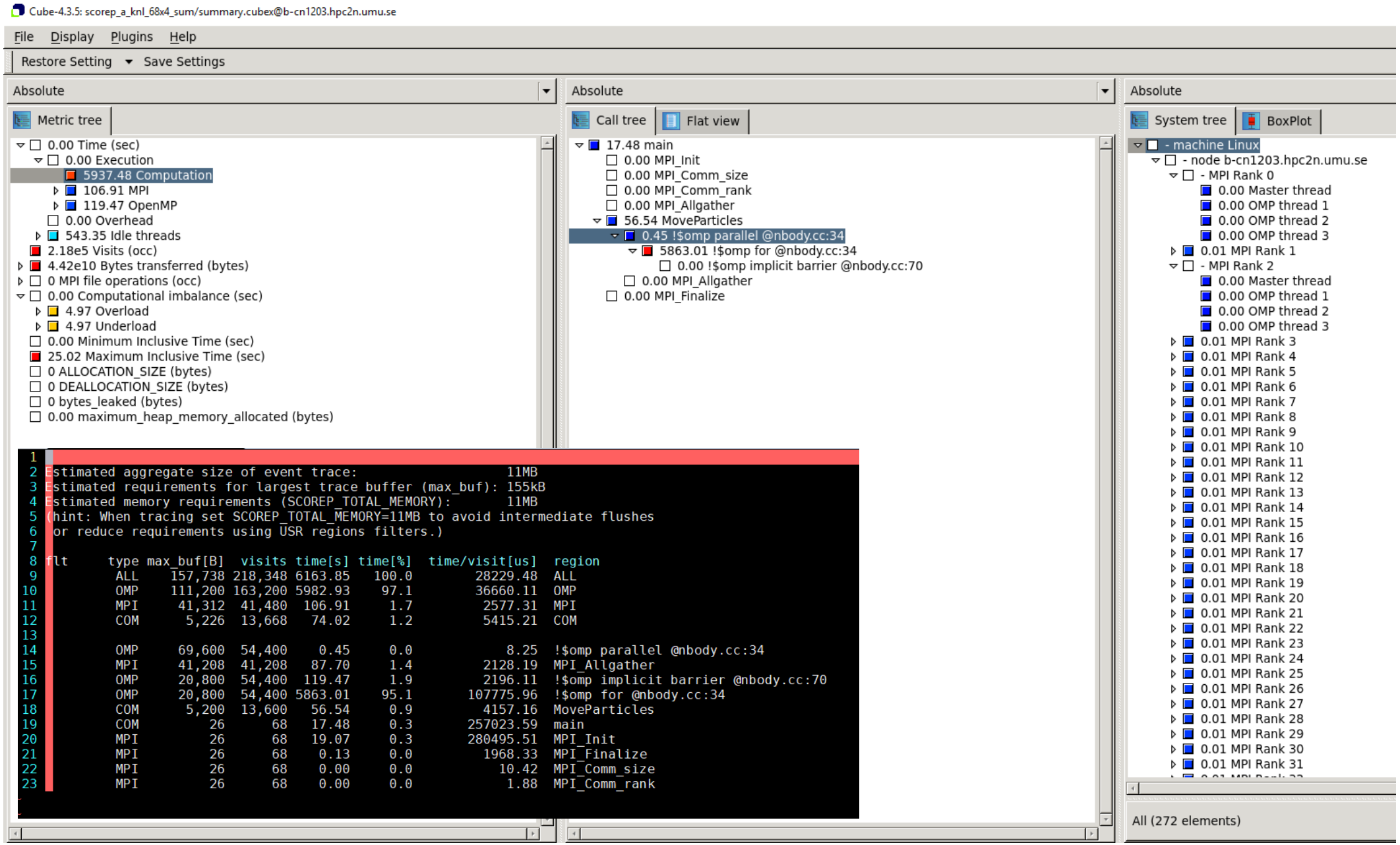
Function	Module	CPU Time
MoveParticlesompparallel_for@34	a_knl.out	3639.307s
__kmp_fork_barrier	libiomp5.so	1675.892s
PMPI_Allgather	libmpi.so.12	555.005s
MoveParticles	a_knl.out	32.972s
vslNewStream	libmkl_intel_lp64.so	24.794s
[Others]		57.713s

CPU Usage Histogram

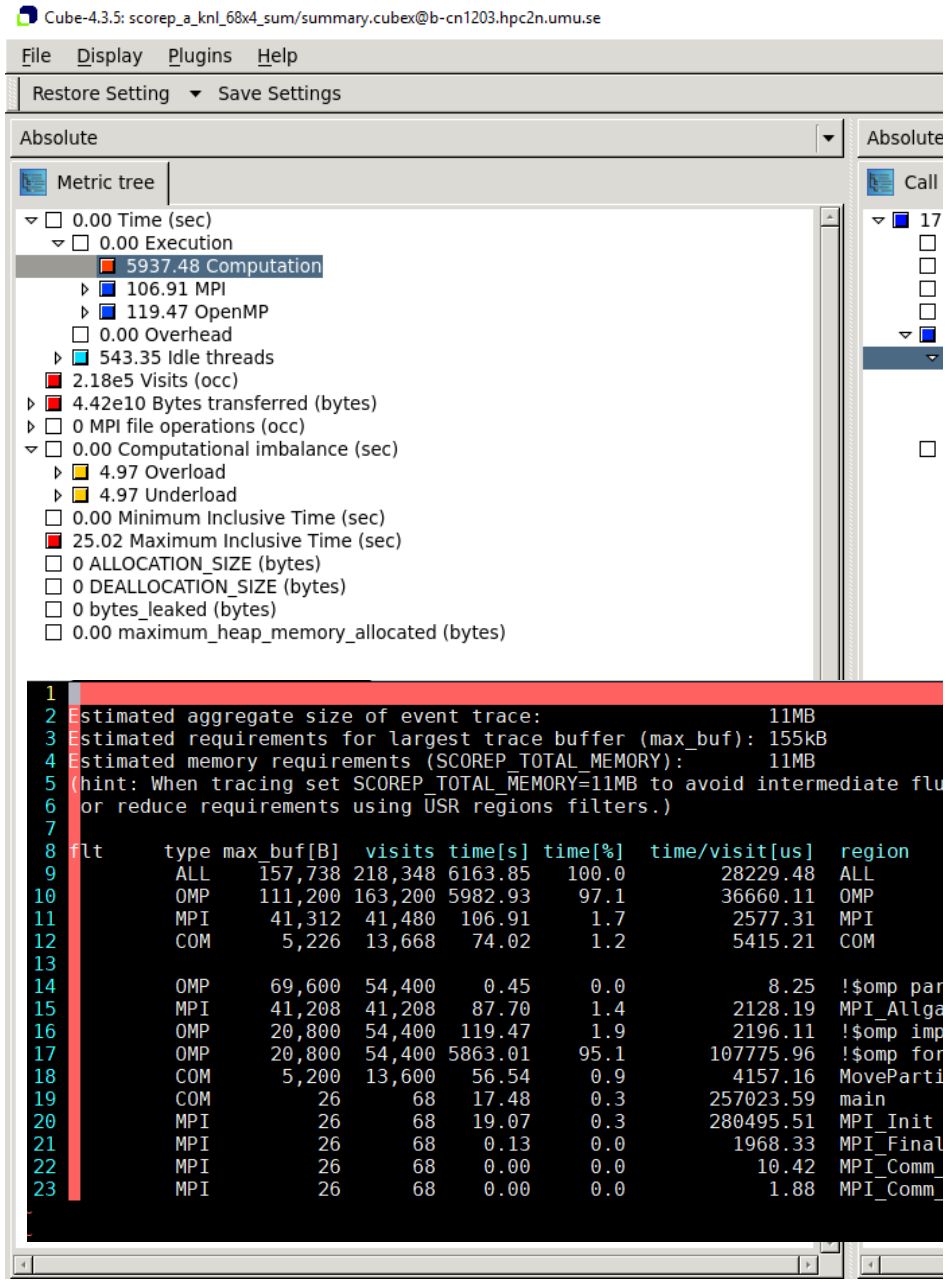
This histogram displays a percentage of the wall time the specific number of CPUs were running simultaneously. Spin and Overhead time adds to the Idle CPU usage value.




<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)
MovePartilesompparallel_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



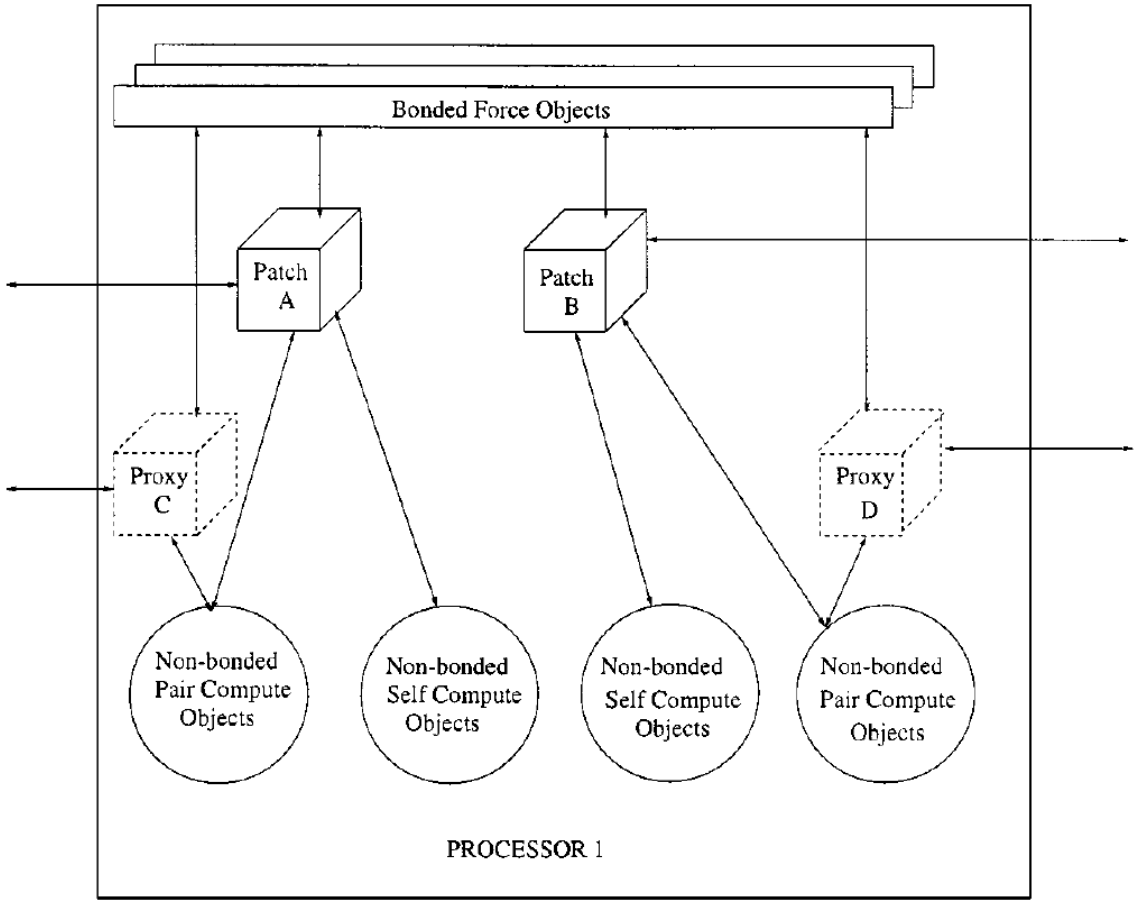

```
Estimated aggregate size of event trace:                   11MB
Estimated requirements for largest trace buffer (max_buf): 155kB
Estimated memory requirements (SCOREP_TOTAL_MEMORY):       11MB
(hint: When tracing set SCOREP_TOTAL_MEMORY=11MB to avoid intermediate flushes
 or reduce requirements using USR regions filters.)

flt     type max_buf[B]  visits time[s] time[%]  time/visit[us]  region
         ALL    157,738 218,348 6163.85   100.0        28229.48  ALL
         OMP    111,200 163,200 5982.93    97.1        36660.11  OMP
         MPI     41,312  41,480  106.91     1.7         2577.31  MPI
         COM      5,226  13,668   74.02     1.2         5415.21  COM

         OMP     69,600  54,400    0.45     0.0            8.25  !$omp parallel @nbody.cc:34
         MPI     41,208  41,208   87.70     1.4         2128.19  MPI_Allgather
         OMP     20,800  54,400  119.47     1.9         2196.11  !$omp implicit barrier @nbody.cc:70
         OMP     20,800  54,400 5863.01    95.1       107775.96  !$omp for @nbody.cc:34
         COM      5,200  13,600   56.54     0.9         4157.16  MoveParticles
         COM         26      68   17.48     0.3       257023.59  main
         MPI         26      68   19.07     0.3       280495.51  MPI_Init
         MPI         26      68    0.13     0.0         1968.33  MPI_Finalize
         MPI         26      68    0.00     0.0           10.42  MPI_Comm_size
         MPI         26      68    0.00     0.0            1.88  MPI_Comm_rank
```




```
34 #pragma omp parallel for schedule(guided)
35   for (int ii = startParticle; ii < endParticle; ii += tileSize) {
36
37     // Components of the gravity force on particles ii through ii+tileSize
38     float Fx[tileSize], Fy[tileSize], Fz[tileSize];
39     Fx[:] = Fy[:] = Fz[:] = 0.0f;
40
41     // Avoid singularity and interaction with self
42     const float softening = 1e-20f;
43
44     // Loop over particles that exert force
45 #pragma unroll(tileSize)
46     for (int j = 0; j < nParticles; j++) {
47
48       // Loop within tile over particles that experience force
49 #pragma unroll(tileSize)
50       for (int i = ii; i < ii + tileSize; i++) {
51
52         // Newton's law of universal gravity
53         const float dx = particle.x[j] - particle.x[i];
54         const float dy = particle.y[j] - particle.y[i];
55         const float dz = particle.z[j] - particle.z[i];
56         const float rr = 1.0f/sqrtf(dx*dx + dy*dy + dz*dz + softening);
57         const float drPowerN32  = rr*rr*rr;
58
59         // Calculate the net force
60         Fx[i-ii] += dx * drPowerN32;
61         Fy[i-ii] += dy * drPowerN32;
62         Fz[i-ii] += dz * drPowerN32;
63       }
64     }
```

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)

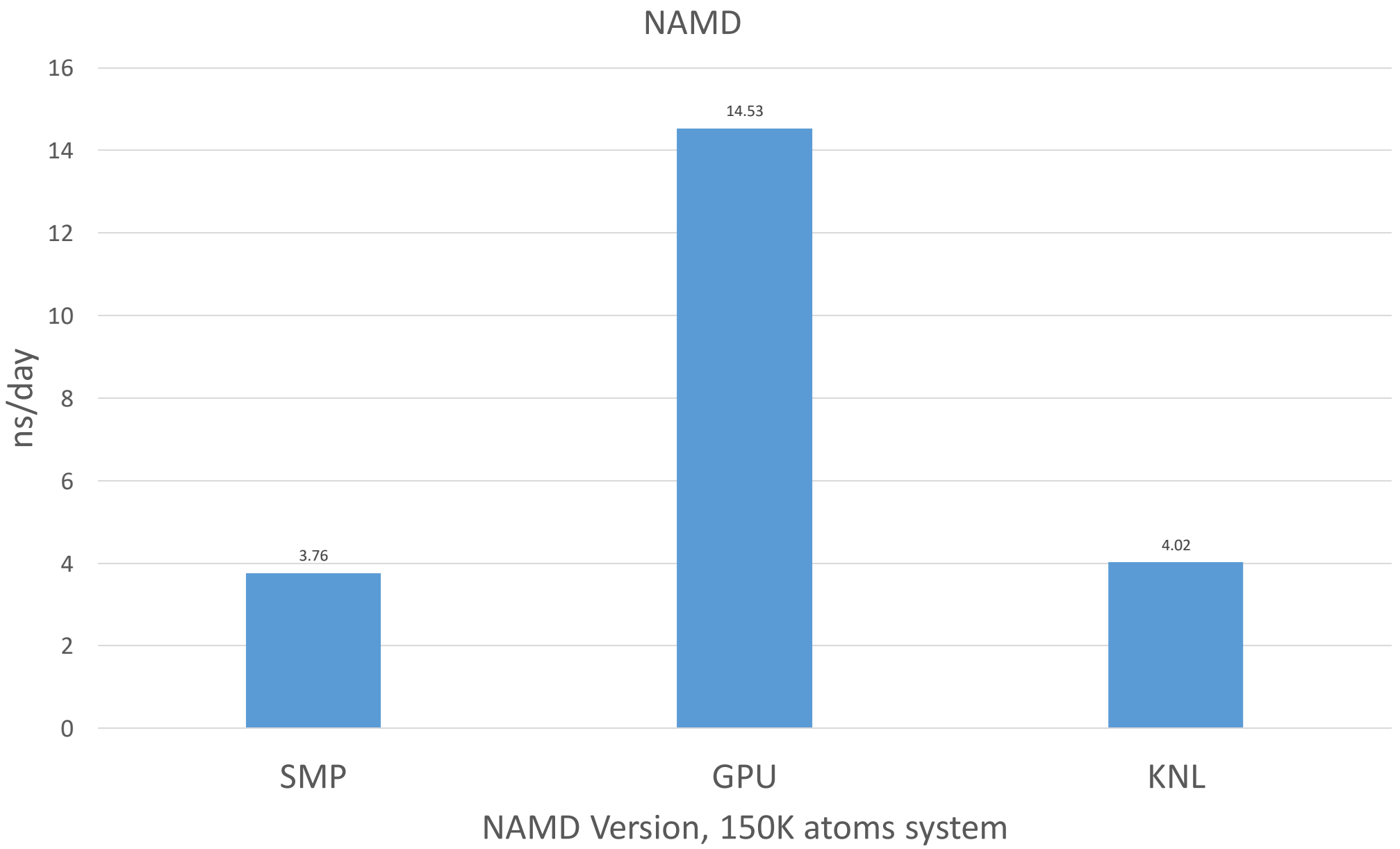
```
#!/bin/bash
#SBATCH -A staff
#Asking for 10 min.
#SBATCH -t 00:50:00
#Number of nodes
#SBATCH -N 1
#Ask for 28 processes
#SBATCH -n 28
#SBATCH --exclusive
#Load modules necessary for running NAMD
module add icc/2017.1.132-GCC-6.3.0-2.27 impi/2017.1.132
module add NAMD/2.12-nompi
#Execute NAMD
namd2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

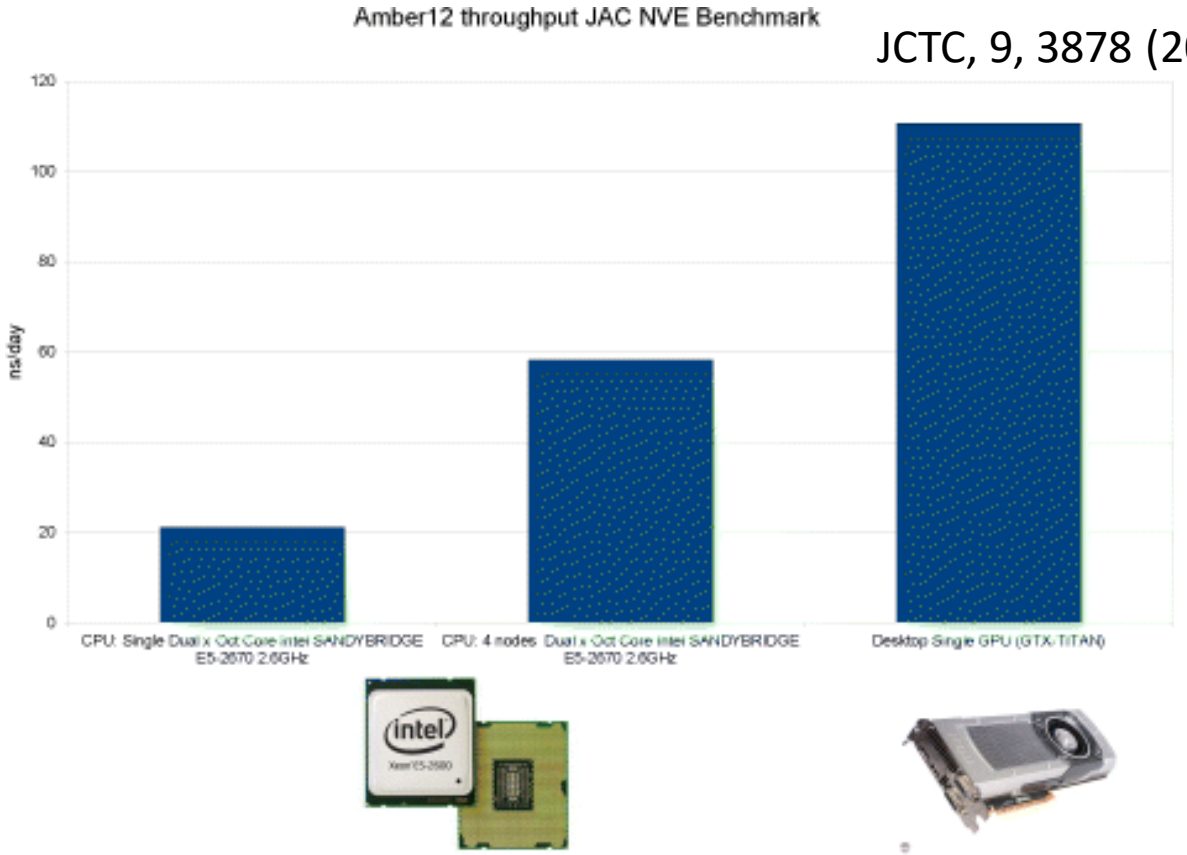
NAMD (KNL)

```
#!/bin/bash
##SBATCH -A SNIC2017-1-56
#SBATCH -A staff
#SBATCH -J namd
#SBATCH -t 00:50:00
#SBATCH -N 1
#SBATCH -n 68
#SBATCH -p knl
#SBATCH --threads-per-core=4
#SBATCH --constraint=hemi,flat

numactl -m 1 /Linux-KNL-icc/namd2 +setcpuaffinity +ppn 126 input.inp
```


NAMD
16
14
12
10
8
6
4
2
0
ns/day
14.53
3.76
4.02
SMP
GPU
KNL
NAMD Version, 150K atoms system

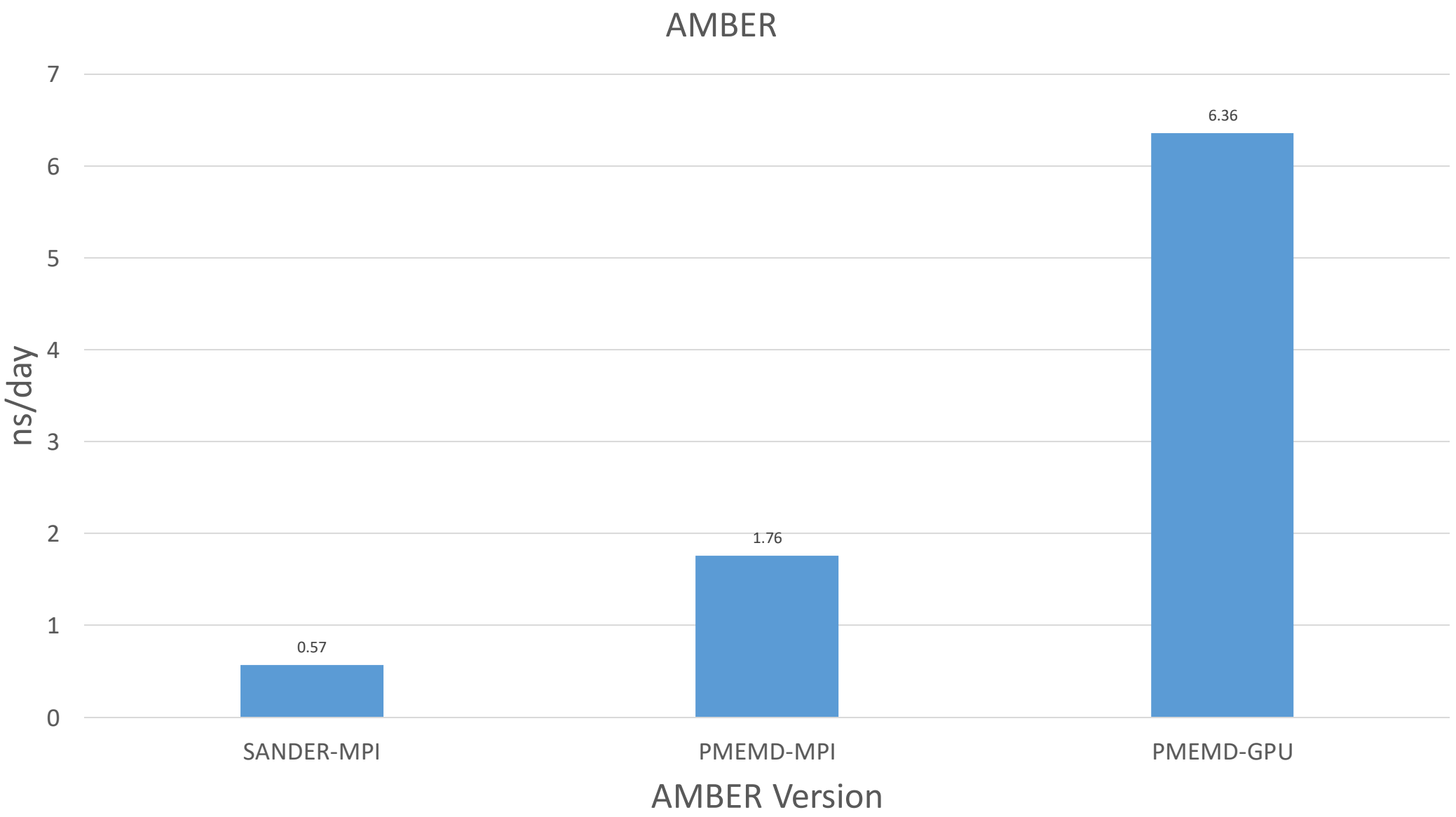
Amber

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



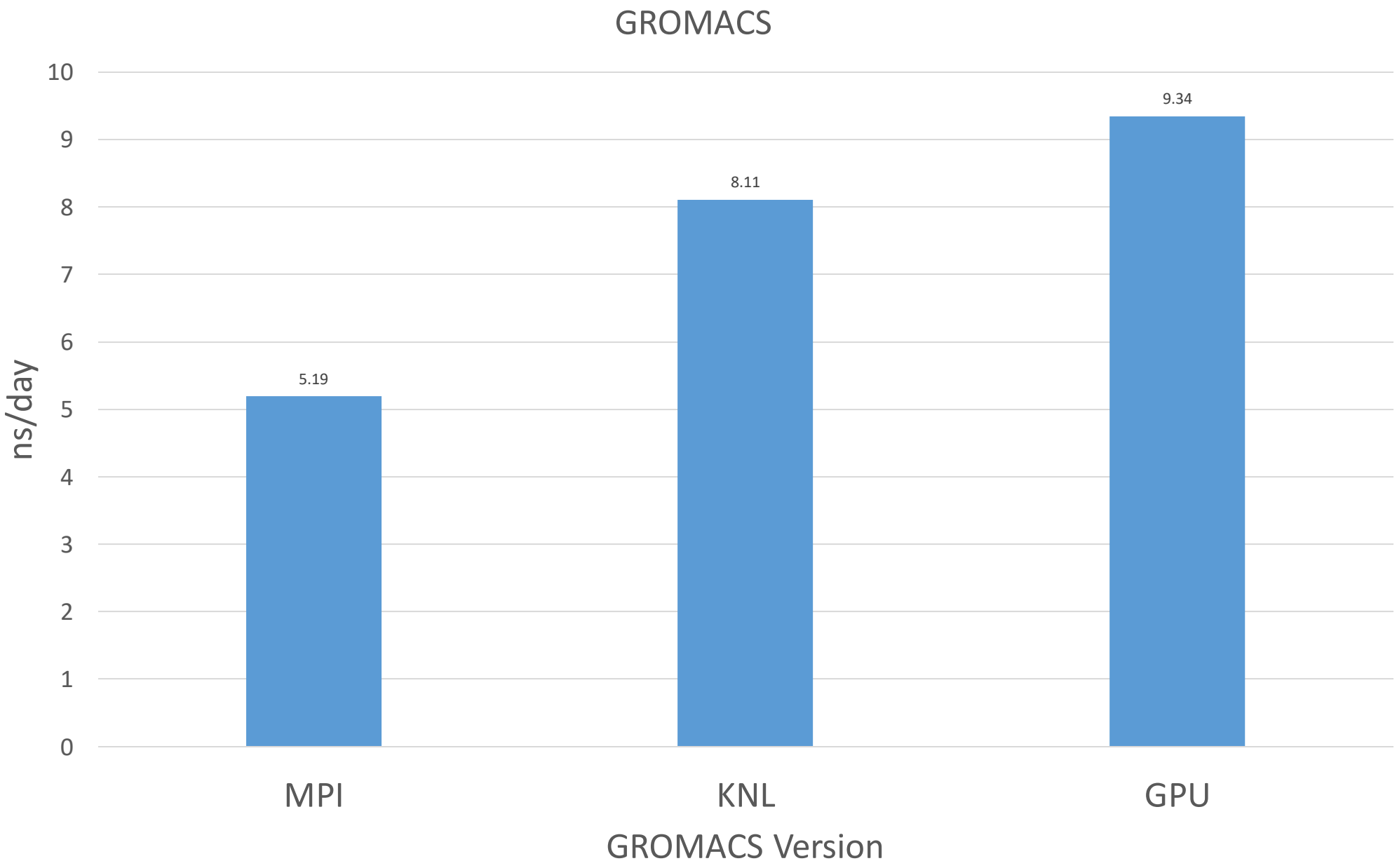
JCTC, 9, 3878 (2013)


```
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
```

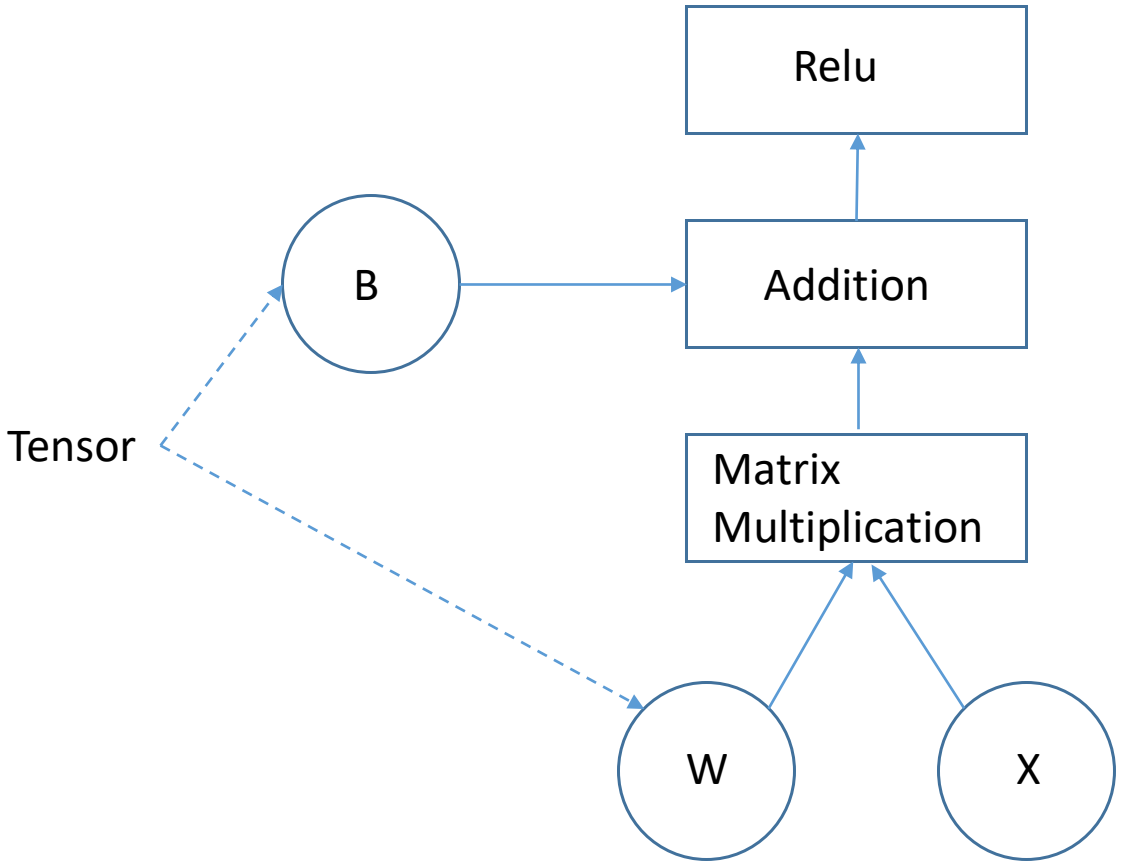

AMBER
7
6
5
4
3
2
1
0
ns/day
6.36
1.76
0.57
SANDER-MPI
PMEMD-MPI
PMEMD-GPU
AMBER Version

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
10
9
8
7
6
5
4
3
2
1
0
ns/day
5.19
8.11
9.34
MPI
KNL
GPU
GROMACS Version

Tensorflow



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
```

```
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