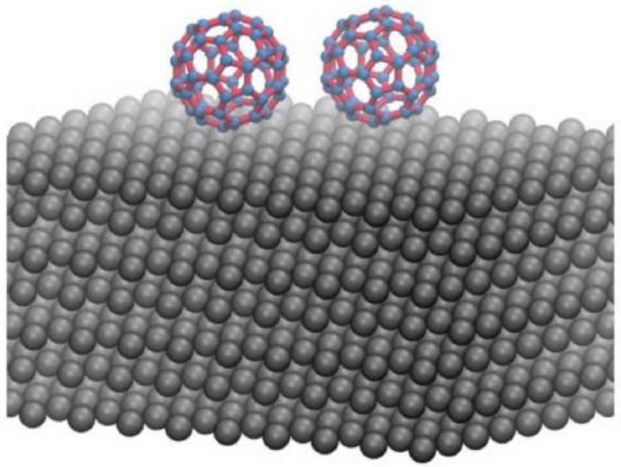
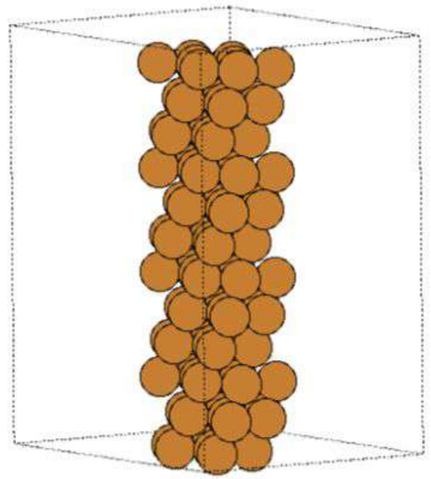
GPAW Introduction

- GPAW is a versatile software package for first-principles simulations of nanostructures utilizing density-functional theory and time-dependent density-functional theory.

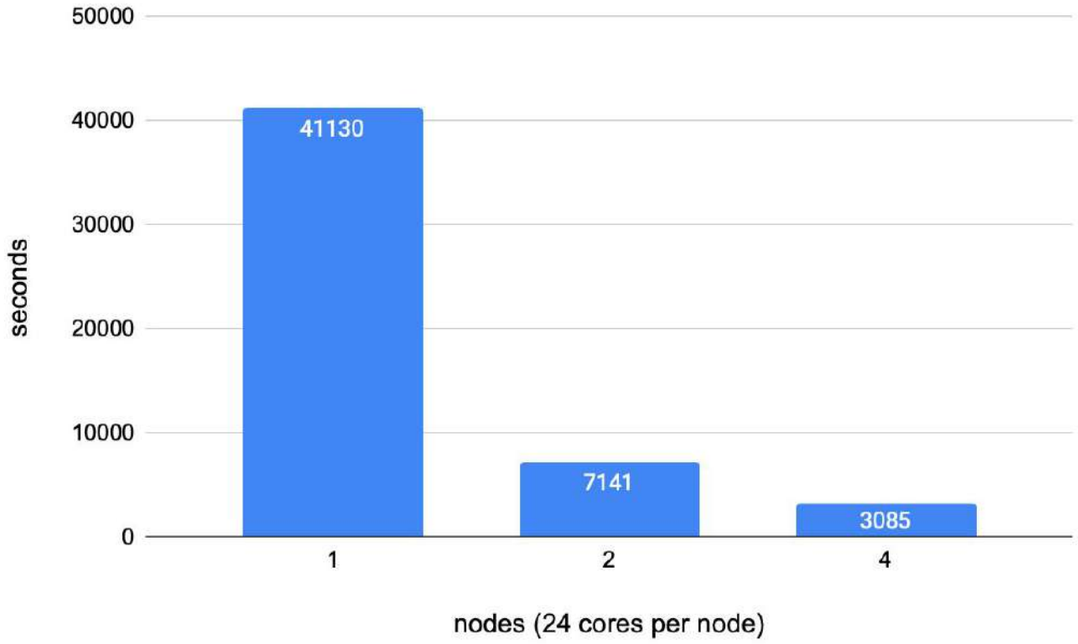
- The benchmark of GPAW in this competition is copper filament, periodic in z-direction Real-space basis, k-points in z- dimension.





NSCC using pure MPI parallelization

- **Multi-nodes has a good scalability on NSCC cluster.**
- **Using mpich to parallelize, the performance is bad.**

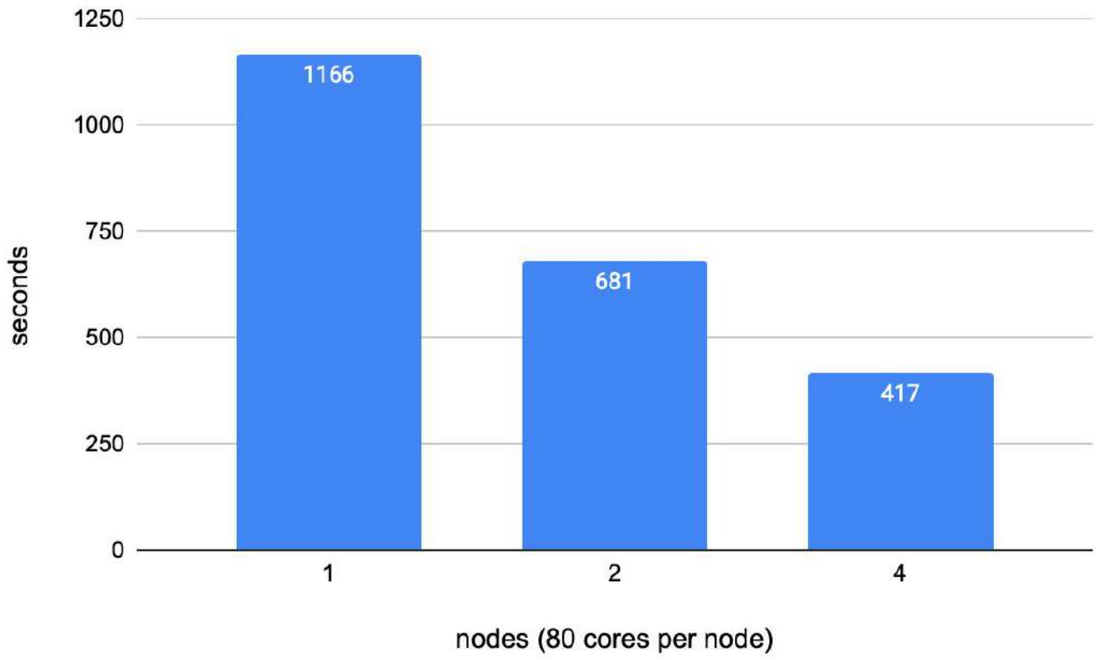


gcc	5.1.0
mpich	3.3.0
libxc	4.3.4
Blas	3.8.0
python	3.8.3

Total cores	Nodes	cores Per node	Time (second)
24	1	24	41130
48	2	24	7141
96	4	24	3085

Niagara using pure MPI parallelization

- **Multi-nodes has a good scalability on Niagara cluster.**
- **Using Intelmpi to parallelize, the performance is good.**

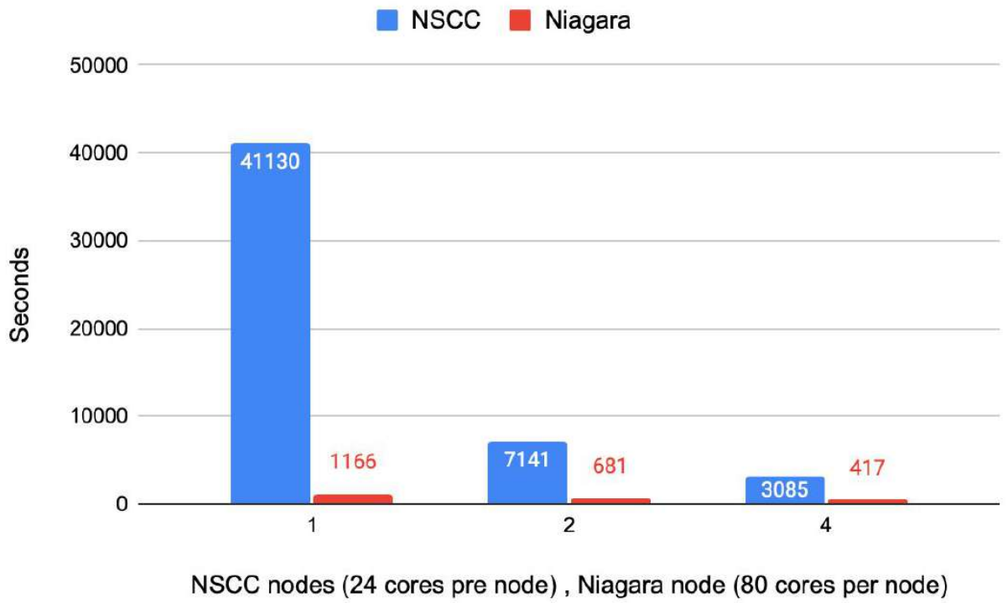


icc	2019u4
Intelmpi	2019u4
libxc	4.3.4
openblas	0.3.7
python	3.8.5

Total cores	Nodes	cores Per node	Time (second)
80	1	80	1166
160	2	80	681
320	4	80	417

NSCC vs. Niagara using pure MPI parallelization

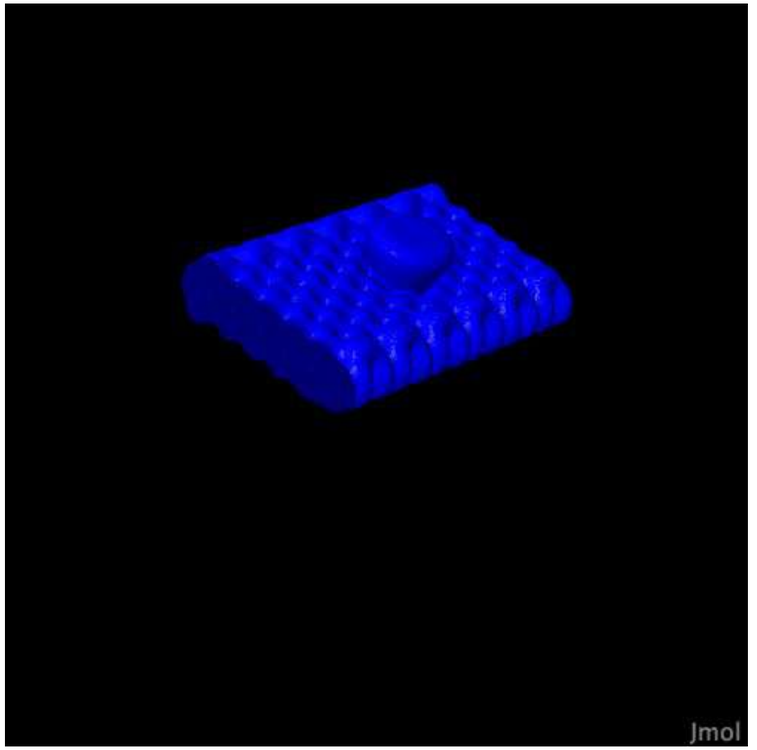
- Multi-nodes has a good scalability on both of Niagara cluster and NSCC cluster.
- Niagara cluster has more cores than NSCC cluster, but we can discover that performance of Intelmpi is better than mpich.



	NSCC	Niagara
C compiler	5.1.0	Intel 2019u4
mpi	mpich -3.3	Intelmpi 2019u4
libxc	4.3.4	4.3.4
Blas	3.8.0	No use
Openblas	No use	0.3.7
python	3.8.3	3.8.5

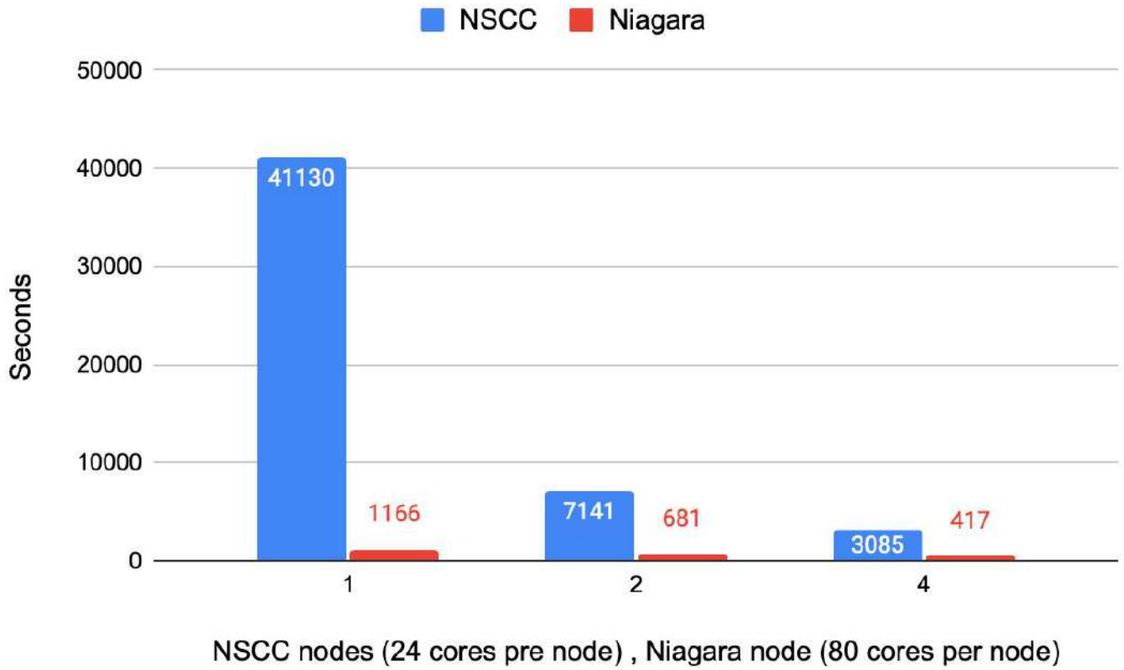
GPAW visualization

- Using jmol simulator to simulate the file elf_ribbon.cube file.
- One is image and the other is animation.



Choose which cluster to optimize.

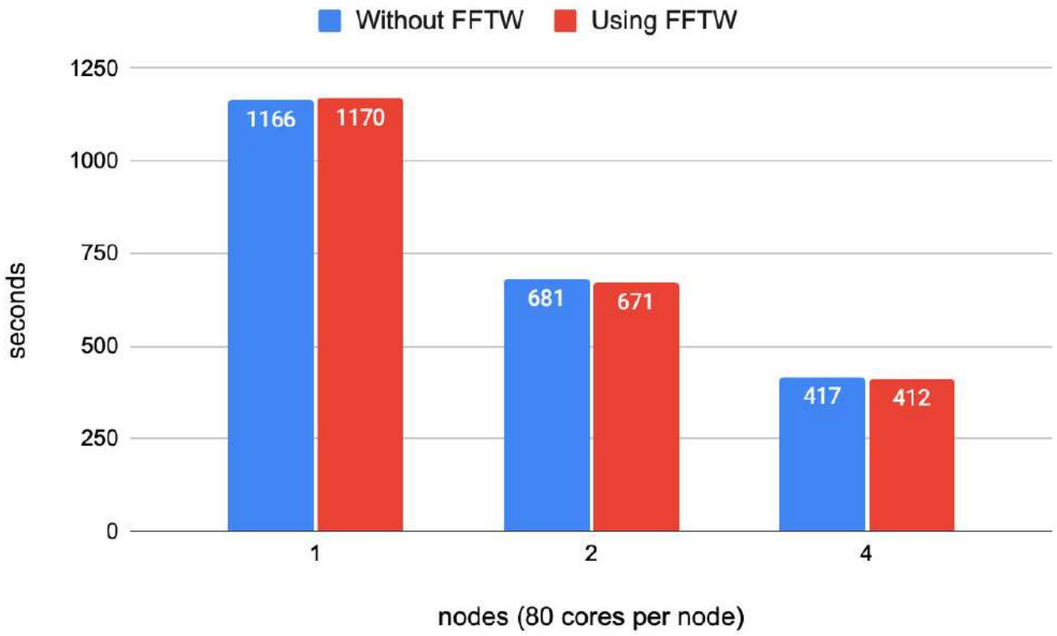
- When using four nodes and each node using all of cores per node to compare.
- The performance of Niagara cluster is much better than NSCC cluster.
- We choose Niagara cluster to optimize.



	NSCC	Niagara
Nodes	4	4
cores per node	24	80
Total cores	96	320
Time (seconds)	3085	417

FFTW

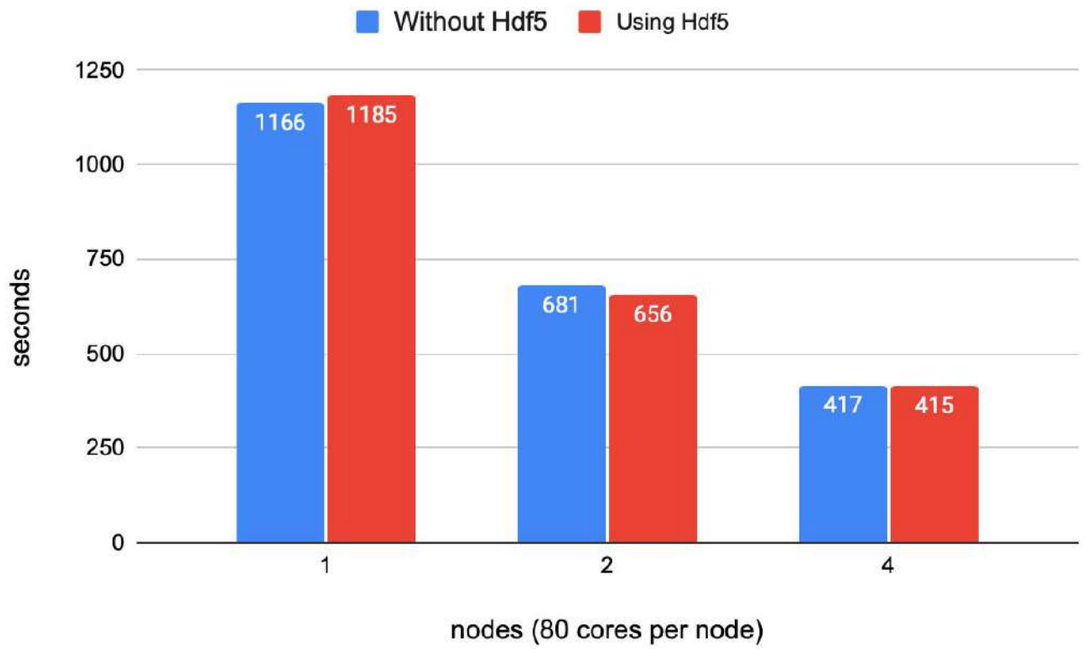
- **There is only a little speedup when using FFTW to optimize.**



cNode	Without FFTW (seconds)	Using FFTW (seconds)
1 (80 cores per node)	1166	1170
2 (80 cores per node)	681	671
4 (80 cores per node)	417	412

Hdf5

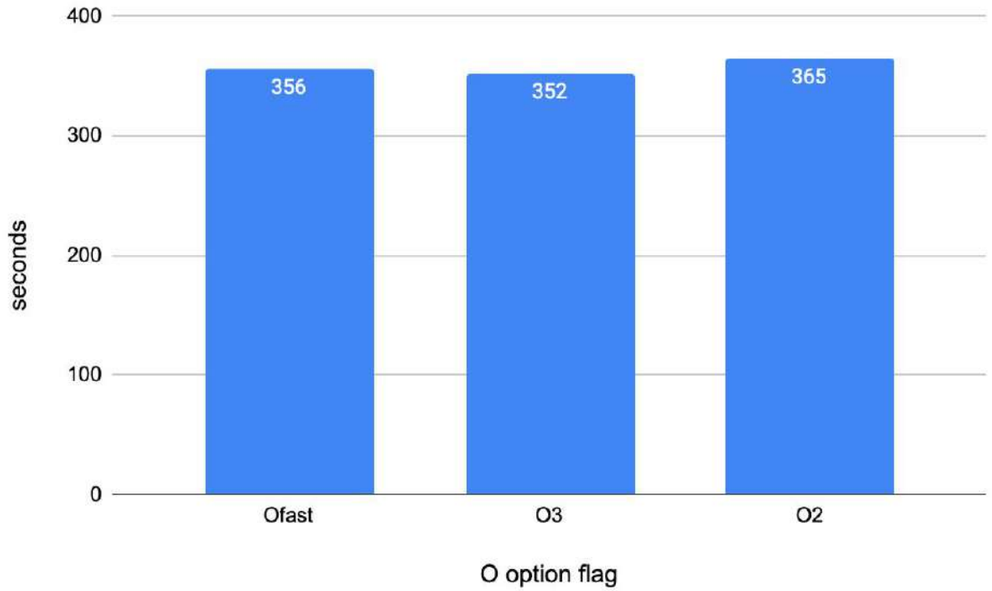
- **There is only a little speedup when using Hdf5 to optimize.**



Node	Without Hdf5	Using Hdf5
1 (80 cores per node)	1166	1185
2 (80 cores per node)	681	656
4 (80 cores per node)	417	415

- O option flag

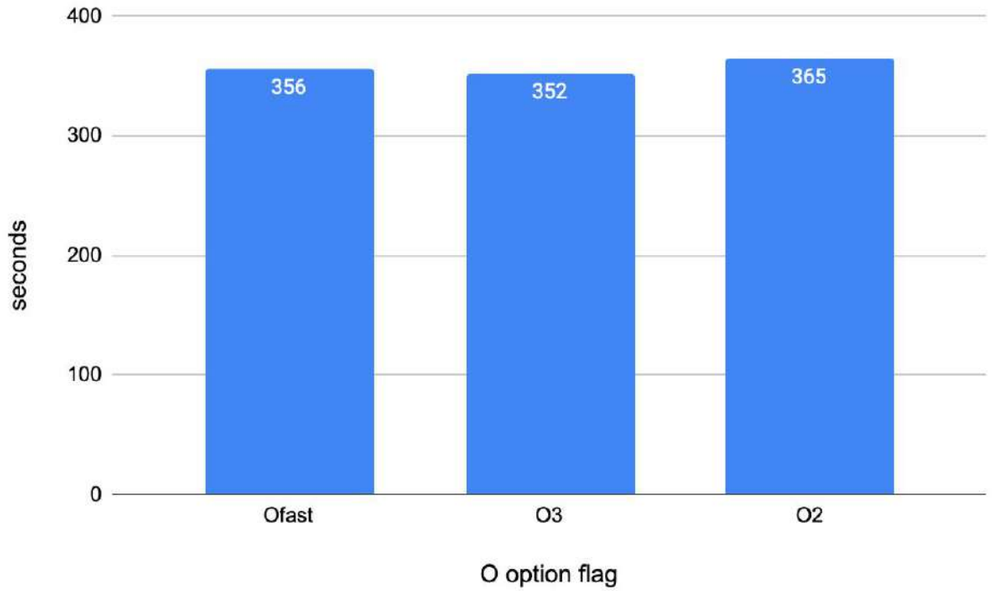
- Run on Niagara four nodes (80 cores per node).
- Speed : O3 > Ofast > O2
- The result of Ofast is correct, but the speed of O3 is fastest.



O option flag	Time (seconds)
Ofast	356
O3	352
O2	365

- O option flag

- Run on Niagara four nodes (80 cores per node).
- Speed : O3 > Ofast > O2
- The result of Ofast is correct, but the speed of O3 is fastest.



O option flag	Time (seconds)
Ofast	356
O3	352
O2	365

Vtune to find out the hotpot

- Using Intel vtune profiler to find out the hotspot.

Function Stack	CPU Time: Total ▼	CPU Time: Self	Module	Function (Full)	Source File	Start Address
Total	100.0%	0ms				
_start	60.0%	0ms	mpiex...	_start		0x4047c0
__libc_start_main	60.0%	0ms	libc.so.6	__libc_start_...		0x22460
main	60.0%	0ms	mpiex...	main	mpiexec.c	0x4048b0
mpiexec_get_parameters	50.0%	0ms	mpiex...	mpiexec_get...	mpiexec_...	0x41f270
i_set_default_ppn	30.0%	0ms	mpiex...	i_set_default...	i_mpiexec...	0x4201a6
i_read_default_env	20.0%	0ms	mpiex...	i_read_defau...	i_mpiexec...	0x422620
push_env_downstream	10.0%	0ms	mpiex...	push_env_d...	mpiexec.c	0x40ac40
_start	40.0%	0ms	srun	_start		0x406d70

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
openat	libc.so.6	0.040s
OS_BARESYSCALL_DoCallAsmIntel64Linux	libc-dynamic.so	0.020s
fscanf	libc.so.6	0.010s
pthread_create	libpthread.so.0	0.010s
__snprintf_chk	libc.so.6	0.010s
[Others]	libpin3dwarf.so	0.010s

*N/A is applied to non-summable metrics.

Hotspots Hotspots by CPU Utilization

Analysis Configuration | Collection Log | Summary | Bottom-up | Caller/Callee | Top-down Tree | Platform | topology-linux.c

Function	CPU Time: Total — Effective Time by Utilization (Idle, Poor, Ok, Ideal, Over)	Spin Time	Overhead Time	CPU Time: Self	Module	
__libc_start_main	100.0%	0.0%	0.0%	0ms	libc.so.6	__libc_start_main
_start	60.0%	0.0%	0.0%	0ms	mpiexec.hydra	_start
main	60.0%	0.0%	0.0%	0ms	mpiexec.hydra	main
hwloc_look_linuxfs	50.0%	0.0%	0.0%	0ms	mpiexec.hydra	hwloc_look_linuxf
mpiexec_get_parameters	50.0%	0.0%	0.0%	0ms	mpiexec.hydra	mpiexec_get_para
ipl_entrance	50.0%	0.0%	0.0%	0ms	mpiexec.hydra	ipl_entrance
ipl_processor_info	50.0%	0.0%	0.0%	0ms	mpiexec.hydra	ipl_processor_info
ipl_detect_machine_topology	50.0%	0.0%	0.0%	0ms	mpiexec.hydra	ipl_detect_machin
hwloc_discover	50.0%	0.0%	0.0%	0ms	mpiexec.hydra	hwloc_discover
hwloc_topology_load	50.0%	0.0%	0.0%	0ms	mpiexec.hydra	hwloc_topology_l
main	40.0%	0.0%	0.0%	0ms	srun	main
_start	40.0%	0.0%	0.0%	0ms	srun	_start
look_sysfscpu	40.0%	0.0%	0.0%	0ms	mpiexec.hydra	look_sysfscpu
openat	40.0%	0.0%	0.0%	39.988ms	libc.so.6	openat
srun	30.0%	0.0%	0.0%	0ms	srun	srun
i_set_default_ppn	30.0%	0.0%	0.0%	0ms	mpiexec.hydra	i_set_default_ppn
hwloc__alloc_read_path_as_cpumask	30.0%	0.0%	0.0%	0ms	mpiexec.hydra	hwloc__alloc_rea
hwloc_open	30.0%	0.0%	0.0%	0ms	mpiexec.hydra	hwloc_open
hwloc__read_path_as_cpumask	30.0%	0.0%	0.0%	0ms	mpiexec.hydra	hwloc__read_path
OS_BARESYSCALL_DoCallAsmIntel64Lin	20.0%	0.0%	0.0%	20.014ms	libc-dynamic.so	OS_BARESYSCA
OS_SyscallDo	20.0%	0.0%	0.0%	0ms	libc-dynamic.so	OS_SyscallDo
i_read_default_env	20.0%	0.0%	0.0%	0ms	mpiexec.hydra	i_read_default_en
plugin_load_from_file	20.0%	0.0%	0.0%	0ms	libslurmfull.so	plugin_load_from_
plugin_context_create	20.0%	0.0%	0.0%	0ms	libslurmfull.so	plugin_context_cr
plugin_load_and_link	20.0%	0.0%	0.0%	0ms	libslurmfull.so	plugin_load_and_
dlopen	20.0%	0.0%	0.0%	0ms	libdl.so.2	dlopen
LoadDwarfForFile	20.0%	0.0%	0.0%	0ms	libpin3dwarf.so	LoadDwarfForFile
GetSubprogramsListInImage	20.0%	0.0%	0.0%	0ms	libpin3dwarf.so	GetSubprogramsL
list_for_each_max	20.0%	0.0%	0.0%	0ms	libslurmfull.so	list_for_each_ma

Scalable Python

- We reference the paper “Optimizing GPAW” try to use scalable python to optimize.
- We install a scalable version of python, but the version is based on python2 which couldn’t support GPAW 20.10.0.

Available on-line at www.prace-ri.eu

Partnership for Advanced Computing in Europe

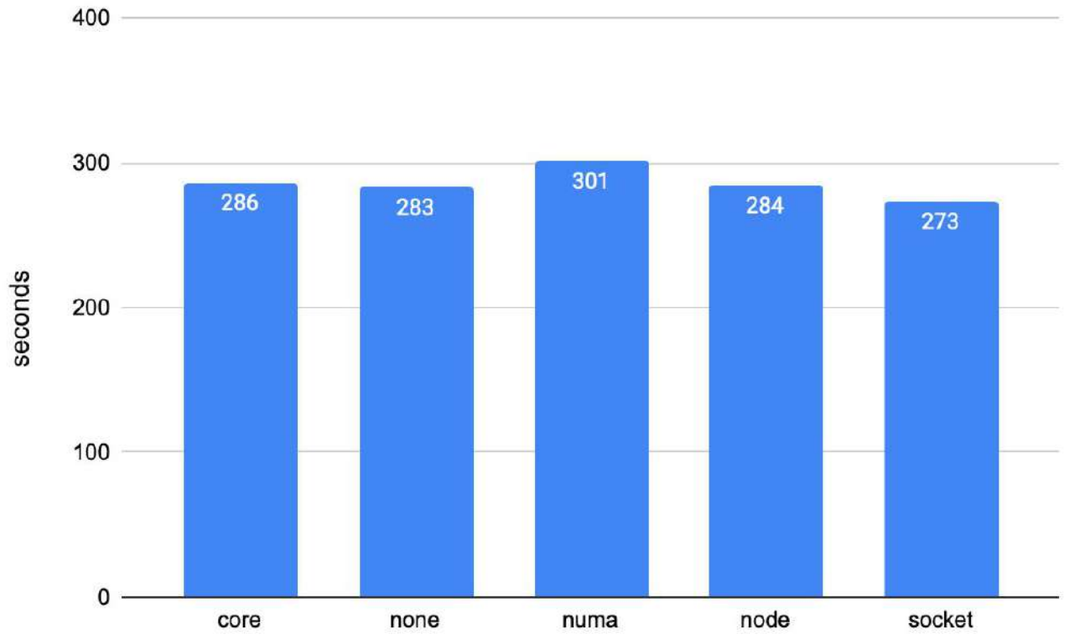
Optimizing GPAW

Jussi Enkovaara[a,*], Martti Louhivuori[a], Petar Jovanovic[b], Vladimir Slavnic[b], Mikael Rännar[c]

[a]CSC - IT Center for Science, P.O. Box 405 FI-02101 Espoo Finland
[b]Scientific Computing Laboratory, Institute of Physics Belgrade, Pregrevica 118, 11080 Belgrade, Serbia
[c]Department of Computing Science, Umea University, SE-901 87 Umea, Sweden

-bind-to flag

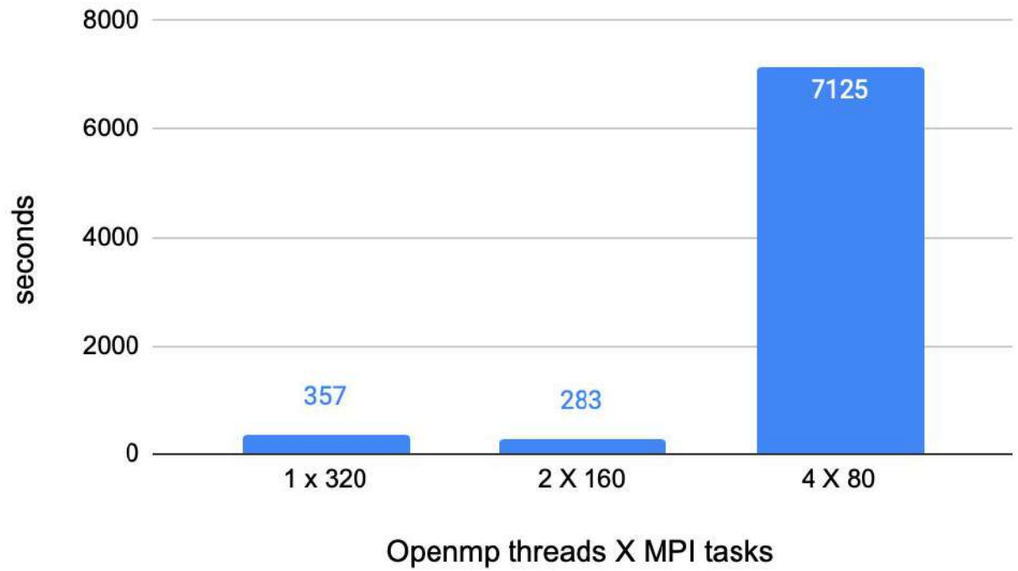
- Running GPAW on four node using different -bind-to flag.



-bind-to flag	Time (seconds)
core	286
none	283
node	284
numa	301
socket	273

Openmp threads

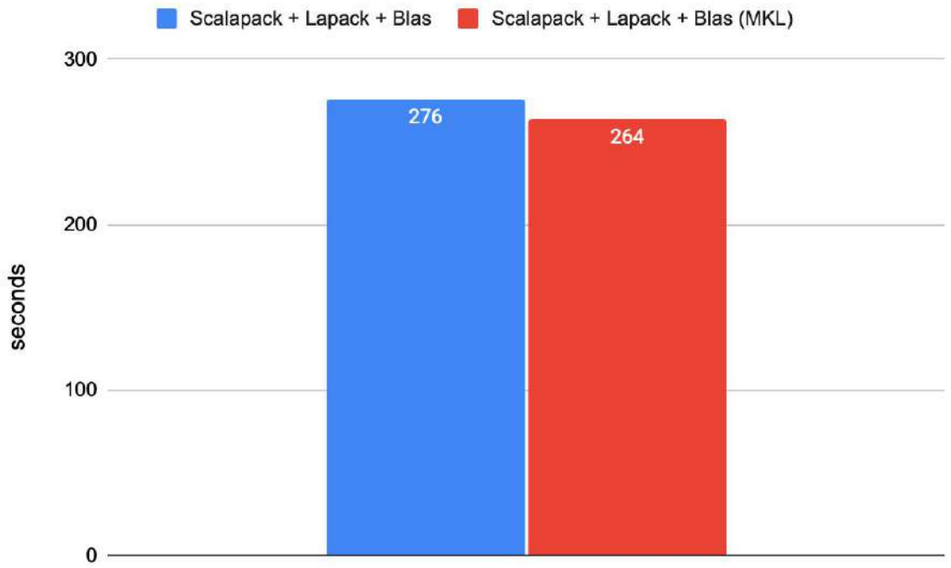
- **Using 320 cores to compare performance of different openmp threads number.**



	Threads	MPI tasks	Time (seconds)
1 X 320	1	320	357
2 X 160	2	160	283
4 X 80	4	80	7125

Scalapack, Lapack, and Blas with MKL

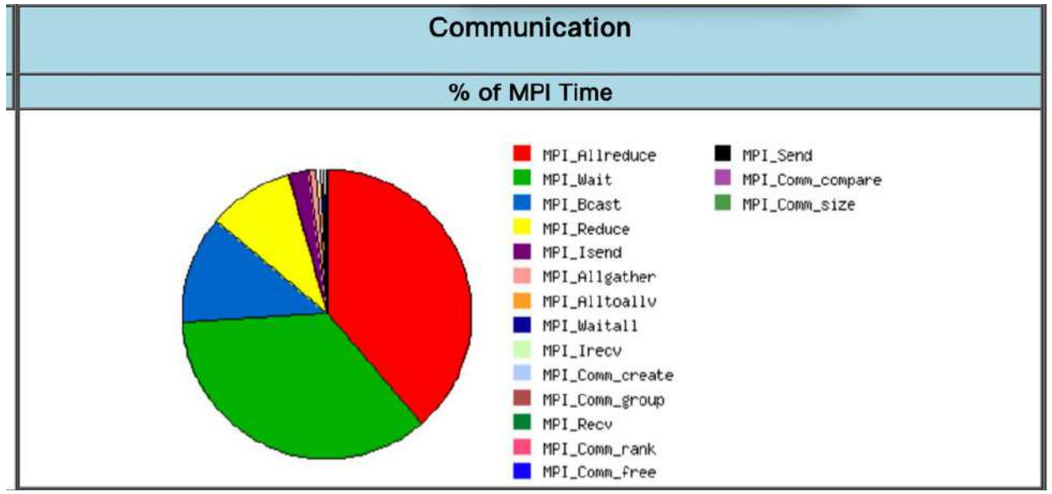
- Using MKL to optimize the math libraries.
- MKL make GPAW



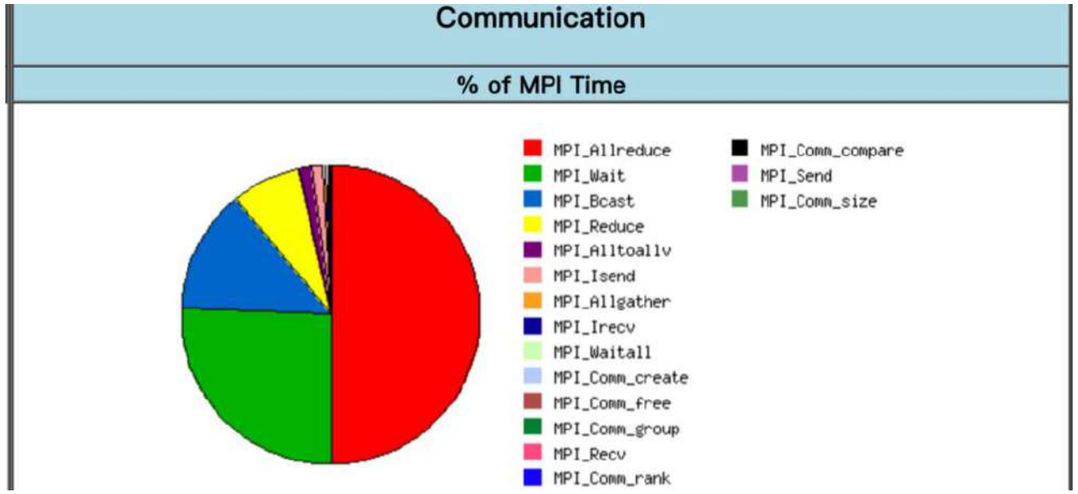
	Without MKL	Using MKL
Time (seconds)	276	264

Using IPM profiler to profile the

- When we run the GPAW on optimizing stage, we compare the both of profiling results to analysis.



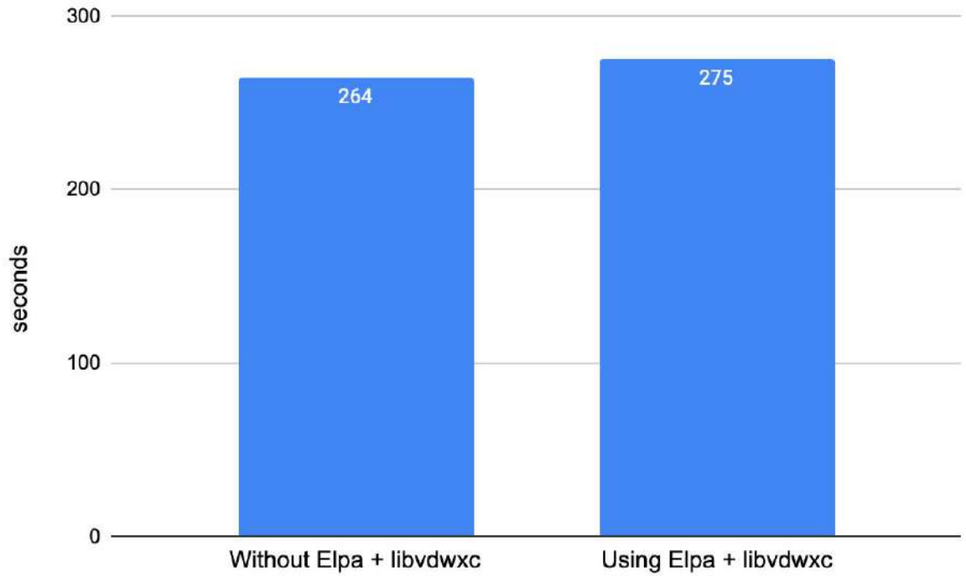
Before



After

Elpa & libvdwxc

- **Using Elap and libvdwxc to speedup the performance of GPAW.**



	Without Elpa + libvdwxc	Using Elpa + libvdwxc
Time (seconds)	264	275

Eigensolver and parallel runs option

- We try to use different eigensolver to the copper.py.
- Fix some parallel runs options.

Eigensolver	Time (seconds)
rmm-diis (default)	261
cg	upto 3600
dav	441

```
40
41 # setup parameters
42 args = {'h': h,
43         'nbands': -20,
44         'occupations': FermiDirac(0.2),
45         'kpts': kpts,
46         'xc': 'PBE',
47         'mixer': Mixer(0.1, 5, 100),
48         'eigensolver': RMMDIIS(),
49         'maxiter': maxiter,
50         'txt': txt,
51         'parallel' : {'band' : 2 ,'use_elpa': 'True', 'elpasolver' : '2stage'}
52         }
53
54 calc = GPAW(**args)
55 atoms.set_calculator(calc)
56
```

Final Result - Niagara Cluster

- **Software compilation version**
- **Hardware usage**

intel	2019u4
intelmpi	2019u4
openblas	0.3.7
libxc	4.3.4
python	3.8.5
Openmp	2019u4
fftw	mkl (intel 2019u4)
Scalapack	mkl (intel 2019u4)
Lapack	mkl (intel 2019u4)
Blas	mkl (intel 2019u4)

Blacs	mkl (intel 2019u4)
libvdwxc	0.4.0
Elpa	2021.05.001
Hdf5	1.8.21
O option flag	O3
-bind-to	socket
Openmp threads	2

Nodes	4
Total cores	320
Cores per node	80

Time: 00:04:24
= 264s

Speed up: 58%