LAMMPS Report

National Tsing Hua University

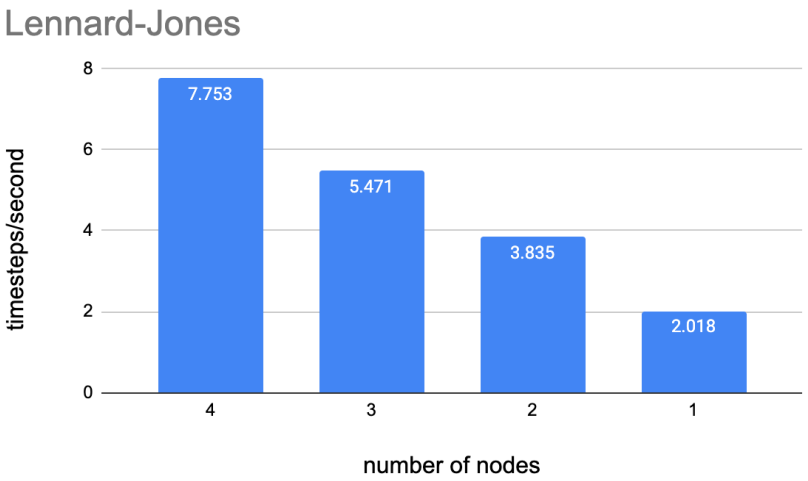
Niagara

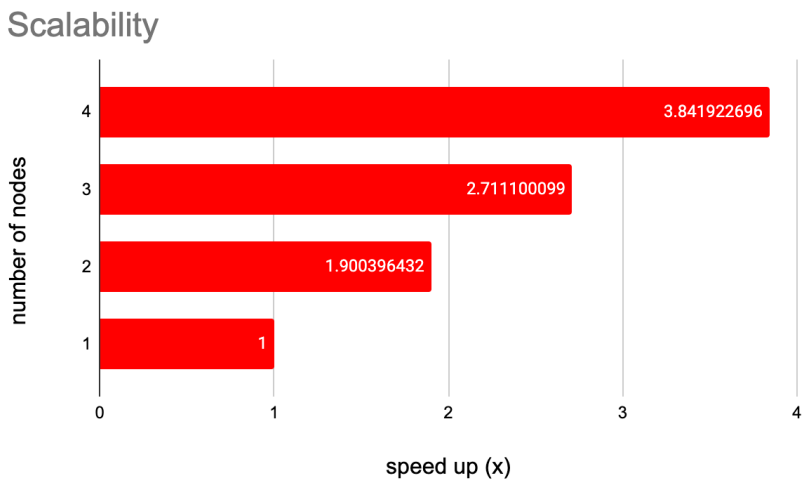
Experimental environment

LAMMPS version	29 Oct 2020
Compiler	Intel C++ Compiler 19.1.3.304
MPI	Intel MPI Version 2019 Update 9
FFT	Intel MKL Version 2020.0.4
CPU	Intel Xeon Gold 6148 CPU @ 2.40GHz
Vectorization	AVX-512

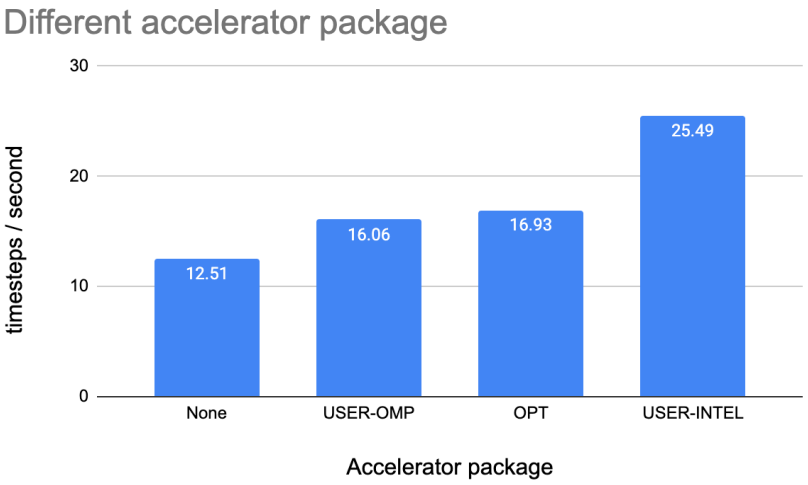
Lennard-Jones

The first step to enhance the performance is straightforwardly to use as many nodes as I can, where I am allowed to use 4 nodes on the Niagara cluster. And in case the scalability of this task is not good enough that more nodes may even lead to a worse performance, I tested the scalability first, which is actually pretty nice. Besides, I take the value of timesteps per second as the basis of viewing performance in all the following experiments. The more the better.

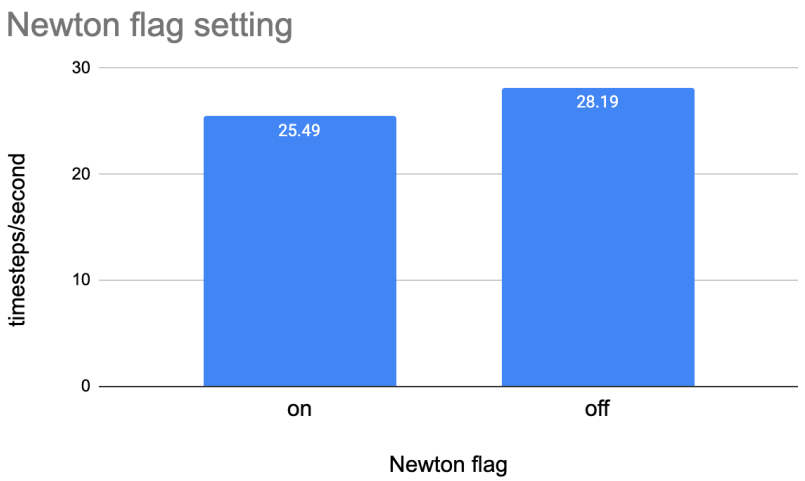




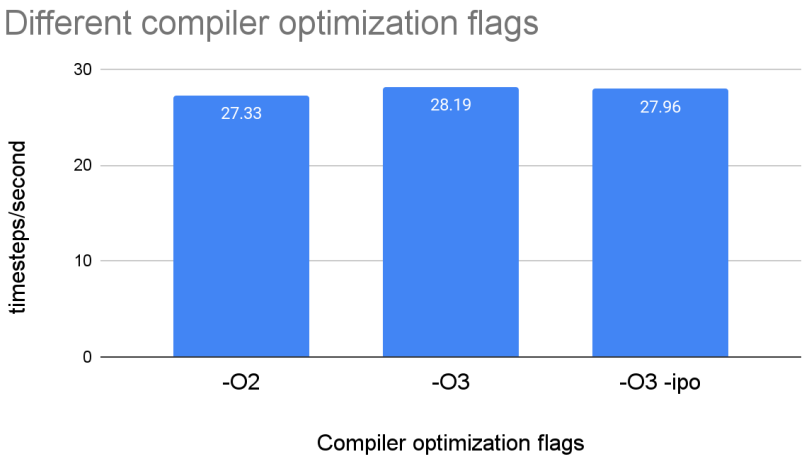
As for the accelerator package, I tested OPT, USER-OMP, and USER-INTEL, and the performance of USER-INTEL package is much better than the other two since the Niagara cluster are using Intel CPUs inside and I used Intel C++ compiler to compile LAMMPS, which maximizes the benefits that USER-INTEL package can brought.



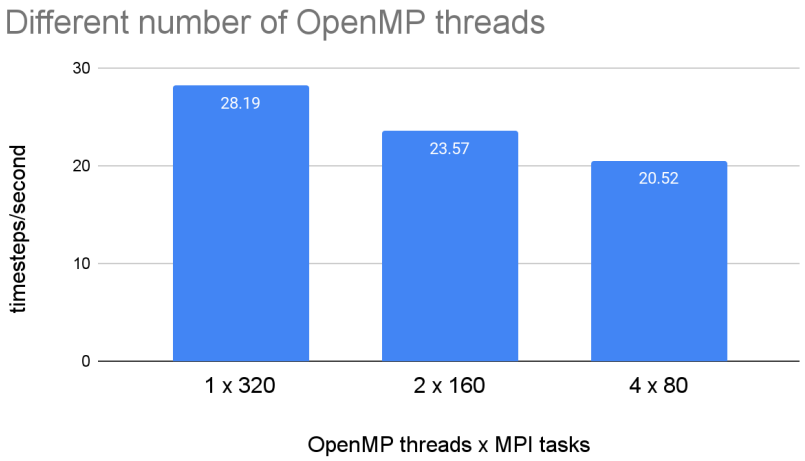
The architecture of this benchmark is quite simple that the communication between MPI tasks becomes the bottleneck of the performance, so I turned Newton's third law off for pairwise and bonded interactions to reduce the communication.



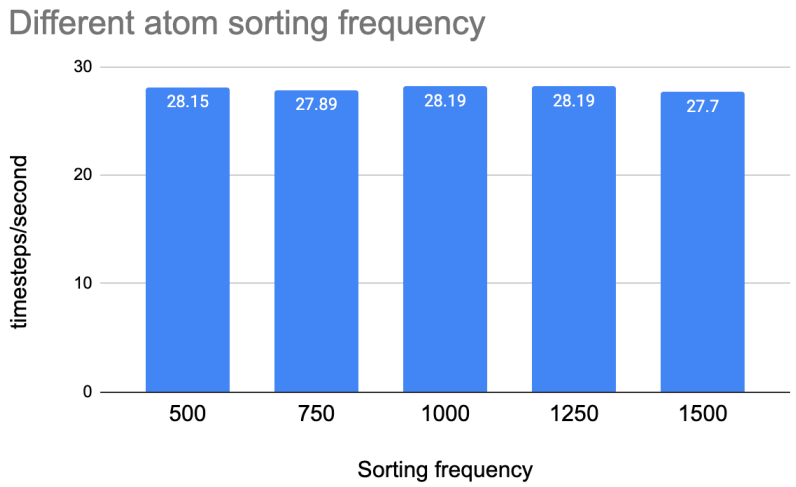
Afterward, I modify the optimization flags, which does bring me some benefits.



Next, I tried to use OpenMP to divide each MPI task to see whether I can reach a better performance, but it turns out that it would be better not to use OpenMP.



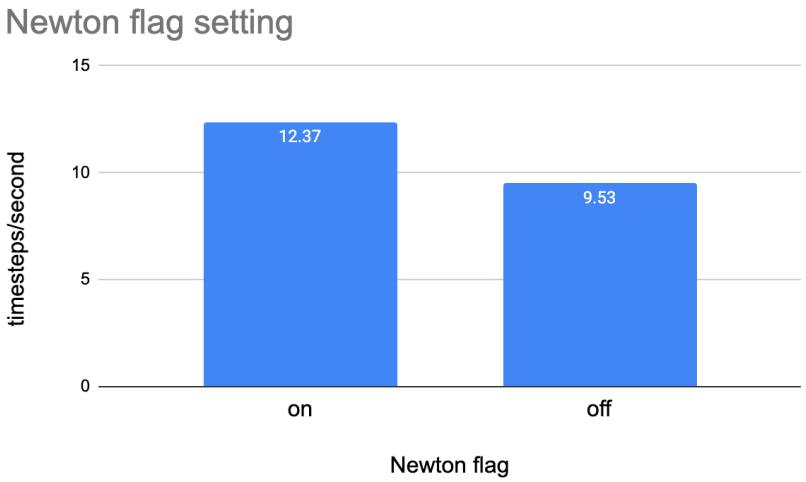
And I also change the frequency of spatial sorting, where the default value is to sort every 1000 timesteps. I changed the frequency to 500, 750, 1250, 1500 timesteps, but the influence on the performance is so little that I consider the gap between them to be just inevitable experimental errors.



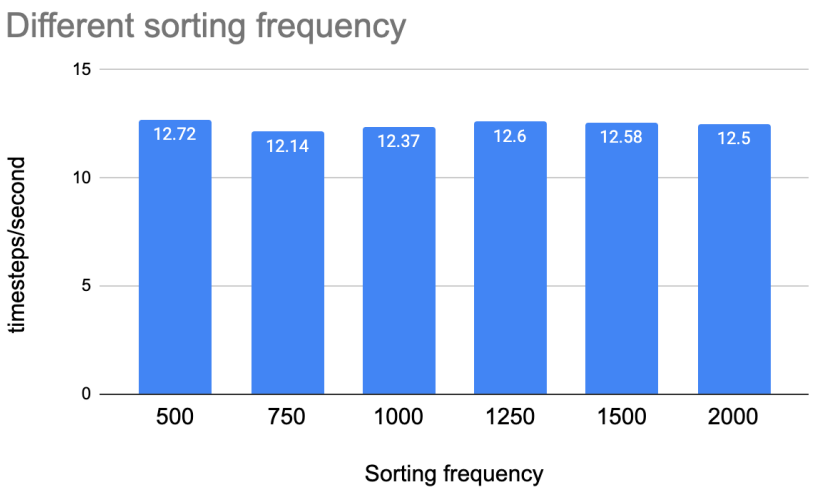
Rhodopsin

With the experience before, I used 4 nodes with 80 MPI tasks per node and no OpenMP while running this benchmark.

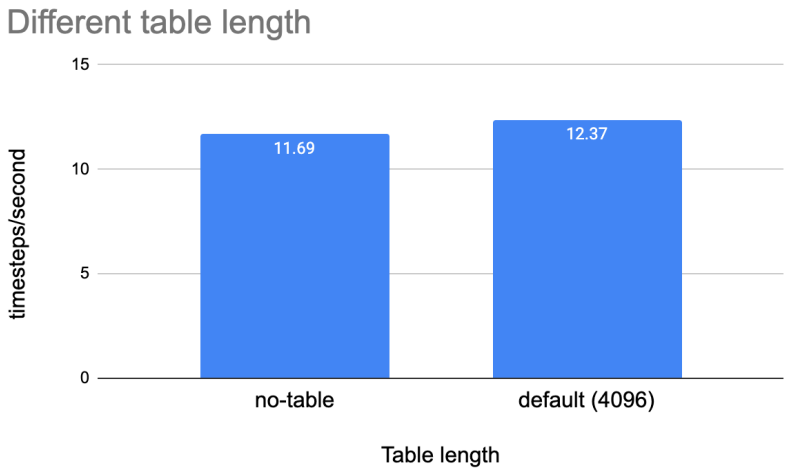
The computational loading of this benchmark is heavier than Lennard-Jones, so I got a worse performance when I turned Newton's third law off.



And I also changed the frequency of spatial sorting. I changed the frequency to be 500, 750, 1250, 1500, 2000 timesteps. Again, the influence on the performance is so little that I consider the gap between them to be just inevitable experimental errors.



Furthermore, I turn off tabulation in pair_style, but it leads to a worse performance.



Finally, the configurations I modified for each benchmark to reach my best performance on the Niagara cluster are as following:

	Lennard-Jones	Rhodopsin
compiler optimization flag	-O3 -fno-alias	-O3 -fno-alias
accelerator package	USER-INTEL	USER-INTEL
Newton flag	off	on
# MPI tasks	320	320
sorting frequency	1250 timesteps	500 timesteps

NSCC

Experimental environment on Aspire-1

LAMMPS version	29 Oct 2020
Compiler	Intel C++ Compiler 19.0.0.117
MPI	Intel MPI Version 5.1.2
FFT	Intel MKL Version 19.0.0.117
GNU Binary Utilities	binutil 2.36
CPU	Intel Xeon CPU E5-2690 v3 @ 2.60GHz
Vectorization	AVX-2

Experimental environment on DGX-1

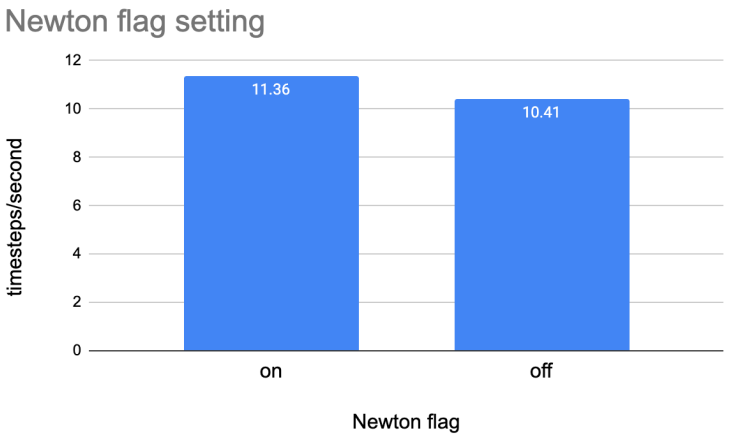
LAMMPS version	29 Oct 2020
Compiler	GCC 4.4.7
MPI	Open MPI 4.1.1
CUDA	CUDA 10.0
FFT	cuFFT 10.0
CPU	Intel Xeon CPU E5-2698 v4 @ 2.20GHz
GPU	Nvidia V100 (16GB)

Lennard-Jones

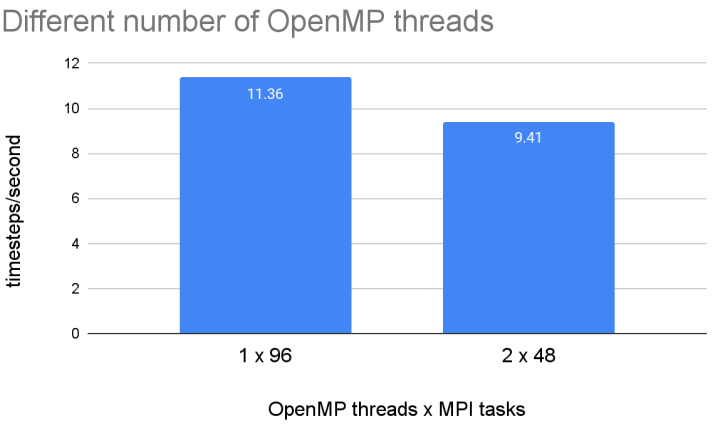
We are allowed to use either 4 CPUs nodes or 4 GPUs on the NSCC cluster. I ran the two tasks with 4 CPU nodes first and then with 4 GPUs after, and tried to compare the difference between the performances.

Aspire 1

Since the CPUs inside Aspire 1 are also manufactured by Intel, I can make some use of the experience from the Niagara cluster. I compile the executable file in the same way with the best configuration I know on the Niagara cluster and use 4 nodes with the USER-INTEL accelerator package directly. I tried to turn Newton's third law off first, but the performance worsened since the computational ability of the CPUs inside Aspire 1 are worse than those inside Niagara and therefore this task is now computation-bound.



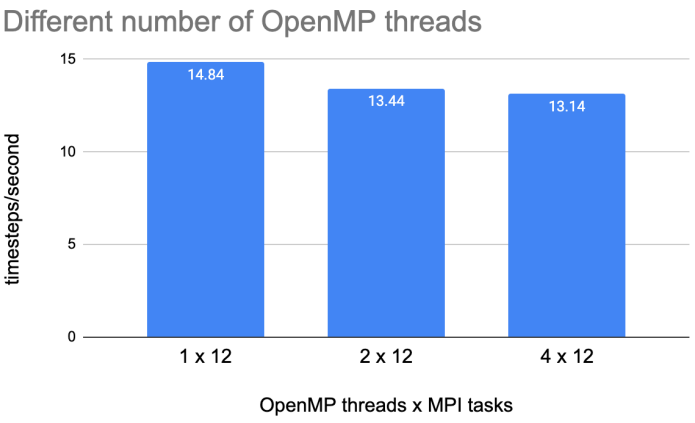
Then I also tried to make use of OpenMP, but the performance decreased.



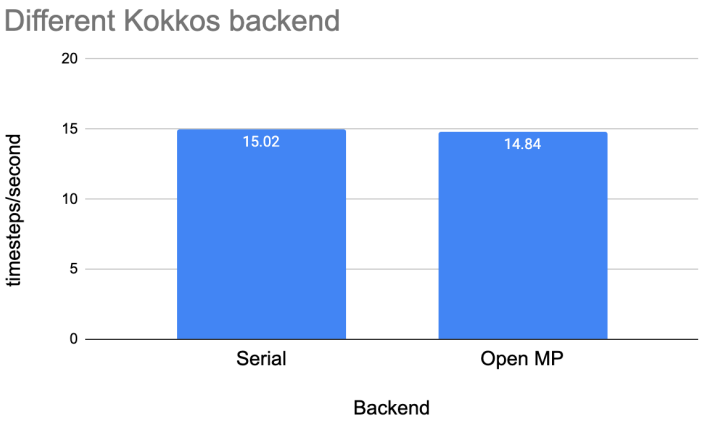
DGX-1

I tried both the GPU package and the Kokkos package on our own cluster, and the performance with the help of Kokkos package is much better than the GPU package. So on the DGX server, I tried to run the task on 4 GPUs with the help of Kokkos package.

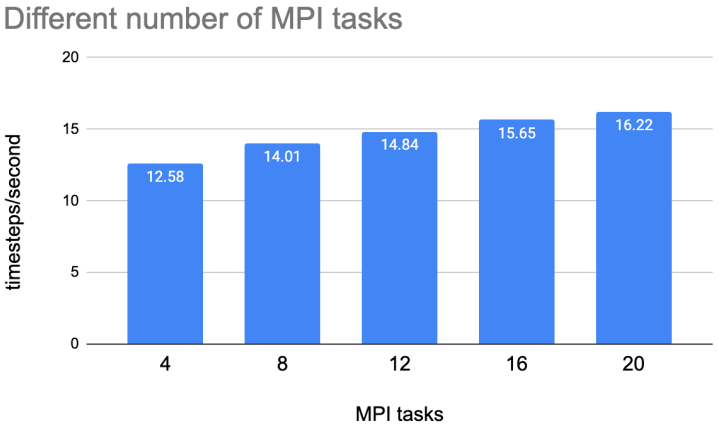
I tried to use OpenMP to divide each MPI task to see whether I can reach a better performance, but it turns out that it would be better not to use OpenMP.



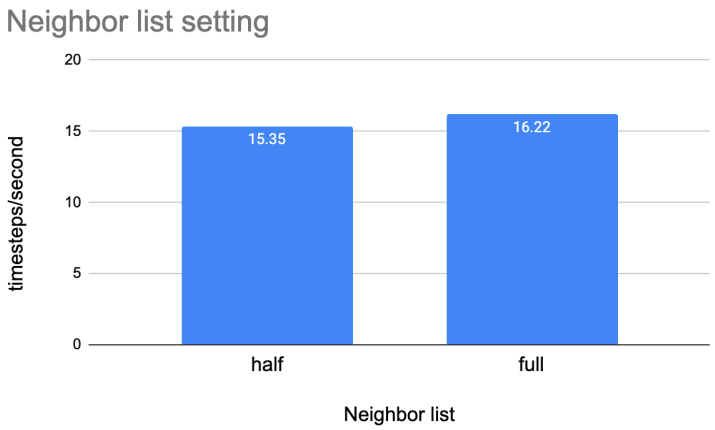
What is worth-mentioning is that if I decide not to use OpenMP, i.e. one OMP thread per MPI task, it would be better to use Kokkos without OpenMP backend.



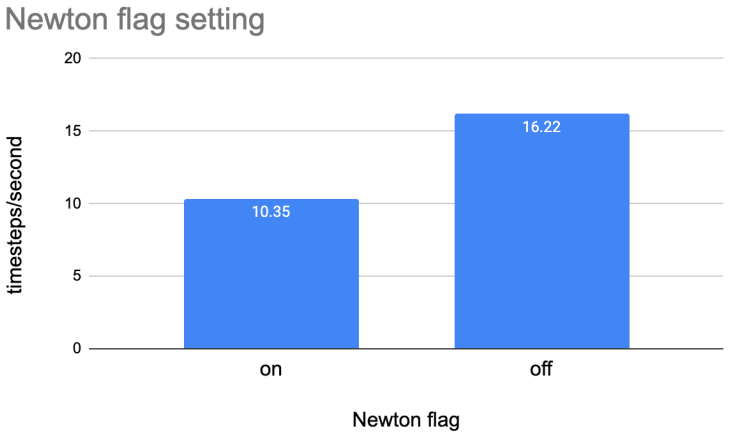
Then, I changed the number of MPI tasks first, and found that 20 MPI tasks gave me the best performance.



Afterward, I tried to turn the neighbor list to be half but it gave me no advantage.



Furthermore, Newton's third law is off by default when using Kokkos package with GPUs. I tried to turn it on and not surprisingly gave me a worse performance since the computational ability of the Nvidia V100 is so good that the bottleneck is not computation but communication.



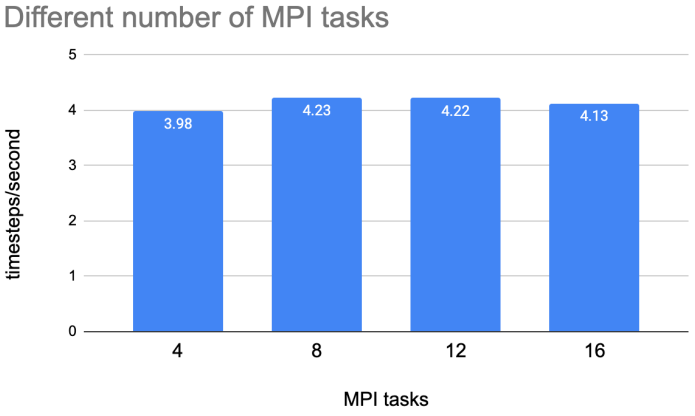
Rhodopsin

Aspire-1

I tried to compare the performance between the CPU and GPU but failed to run this task on CPU nodes. It would mysteriously stop running without any error message.

DGX-1

With all the knowledge above and considering the limited GPU hours on NSCC, I decided not to try OpenMP and let the Newton's flag stay to be off on this task. Besides, the neighbor list has no choice but to be half in order to run this module successfully. I changed the number of MPI tasks first, and found that 8 MPI tasks gave me the best performance.



Since I compiled the executable with “--default-stream per-thread” flag, I am able to overlap kspace style and molecular topology running on the host cpu with a pair style running on the GPU by adding “/kk/host” suffix in the input file. I added the suffix on different styles and found that overlapping bond_style and improper_style gave the optimal performance, and the results are shown in the following table. The kspace_style is pppm, which invokes a particle-particle particle-mesh solver that maps atom charge to a 3d mesh, and uses 3d FFTs to solve Poisson’s equation on the mesh, then interpolates electric fields on the mesh points back to the atoms. And since I choose cuFFT as the fast Fourier transform library, I am not able to assign it to be executed by the host.

/kk/host appended behind which style	timesteps / second
none	4.234
bond	4.436
angle	3.803
dihedral	2.992
improper	4.445
bond, angle	3.999
bond, improper	4.467
bond, angle, dihedral, improper	2.740

Then, I tried to perform “pack/unpack” on the device and the host, but the performances are identical.

Finally, the configurations I modified for each benchmark to reach my best performance on the NSCC cluster are as following:

	Lennard-Jones	Rhodopsin
run on	DGX-1	DGX-1
accelerator package	Kokkos	Kokkos
Kokkos backend	Serial	Serial
# MPI tasks	20	8
/kk/host	-	bond, improper