On Assignment 5

Q. Part 1 (programming): Since we are starting with pmd_irecv.c **in assignment 4, do we only need to mark the OpenMP changes or both MPI & OpenMP changes?**

A. Please only mark the OpenMP changes.

Q. Part 3 (strong scaling): Should we plot & submit the runtime, speedup & parallel efficiency as a function of the number of threads, as in slide 20 of "Hybrid MPI+OpenMP MD" lecture, https://aiichironakano.github.io/cs596/05HMD.pdf**?**

A. No, please submit only the efficiency plot.

On Assignment 5, Part 3 (Scaling)

- **Goal: Measure multithread parallel efficiency on multiple cores within a single computing node**
- **CPU in standard output is total runtime (in seconds) including computing & communication; use it as $T(N, P)$ in the speedup formula, where N is the fixed problem size (proportional to the total # of atoms, *nglob* = 55296, but doesn't enter in efficiency calculation) & P is the # of cores (or threads, remember one thread per core seen using 'top' command)**

```
[liuhaora@discovery2 hw5]$ cat hmd-scale.out
8 threads
al = 4.103942e+01 4.103942e+01 4.103942e+01
lc = 16 16 16
rc = 2.564964e+00 2.564964e+00 2.564964e+00
nglob = 55296
CPU & COMT = 8.350226e+00 2.620383e-02
4 threads
al = 4.103942e+01 4.103942e+01 4.103942e+01
lc = 16 16 16
rc = 2.564964e+00 2.564964e+00 2.564964e+00
nglob = 55296
CPU & COMT = 3.940460e+00 2.325014e-02
2 threads
al = 4.103942e+01 4.103942e+01 4.103942e+01
lc = 16 16 16
rc = 2.564964e+00 2.564964e+00 2.564964e+00
nglob = 55296
CPU & COMT = 9.614403e+00 2.323241e-02
1 thread
al = 4.103942e+01 4.103942e+01 4.103942e+01
lc = 16 16 16
rc = 2.564964e+00 2.564964e+00 2.564964e+00
nglob = 55296
CPU & COMT = 1.492553e+01 2.285458e-02
```

$P = 1$	$T(N, 1) = 14.92553$
$P = 2$	$T(N, 2) = 9.614403$
$P = 4$	$T(N, 4) = 3.940460$
$P = 8$	$T(N, 8) = 8.350226$

$$S_2 = \frac{T(N,1)}{T(N,2)}$$

Strong-scaling (fixed problem size):

Speedup: $S_P = \frac{T(N,1)}{T(N,P)}$

Efficiency: $E_P = \frac{S_P}{P}$; **only plot this!**

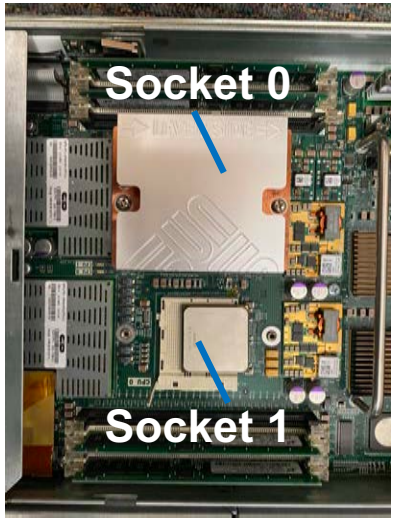
Why Dip in Runtime for $P = 4$?

- **Each of the two processors (or sockets) with multiple cores has fast local memory called cache (to be discussed in performance optimization lecture)**

In prior architectures (such as the Intel® Xeon® E5 v4 Processor family):

- The mid-level cache (MLC or also known as L2) was 256 KB per core.
- The last level cache (also known as L3) was a shared inclusive cache with 2.5 MB per core.

In the architecture of the Intel® Xeon® Scalable Processor family, the cache hierarchy has changed to provide a larger MLC of 1 MB per core and a smaller shared non-inclusive 1.375 MB LLC per core. A larger MLC increases the hit rate into the MLC resulting in lower effective memory latency and also lowers demand on the mesh interconnect and LLC. The shift to a non-inclusive cache for the LLC allows for more effective utilization of the overall cache on the chip versus an inclusive cache. https://www.intel.com

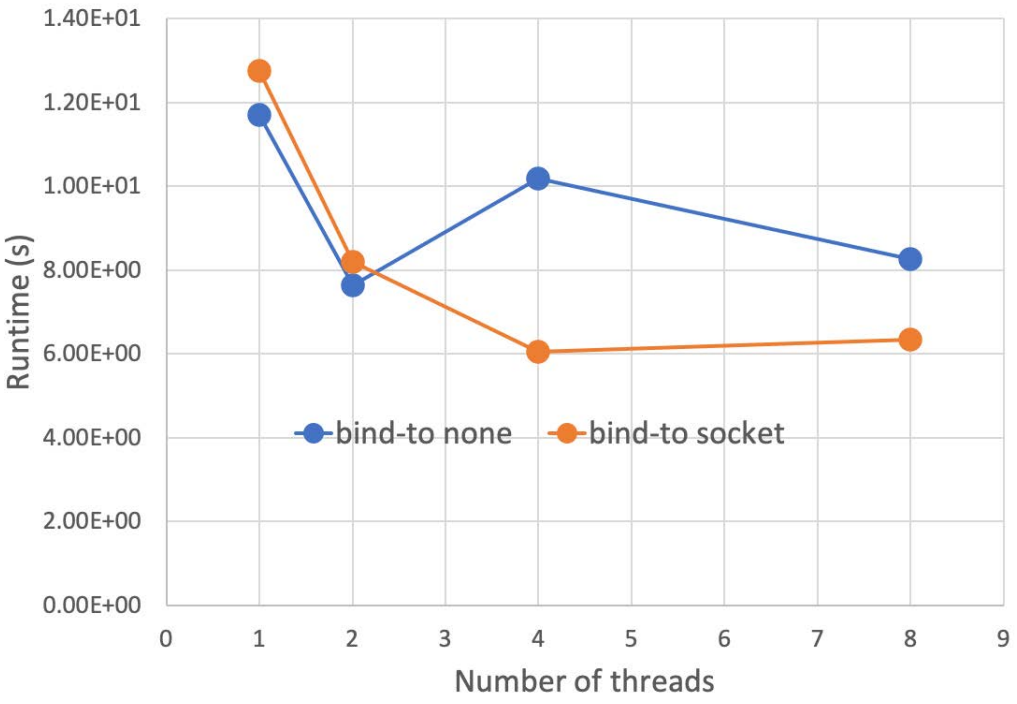


$P = 1$	$T(N, 1) = 14.92553$
$P = 2$	$T(N, 2) = 9.614403$
$P = 4$	$T(N, 4) = 3.940460$
$P = 8$	$T(N, 8) = 8.350226$

- **In addition to more arithmetic-logic operations, multiple cores provide larger caches to improve memory-access speed**
- **If threads are placed on different sockets, however, memory performance degrades**
- **Non-uniform memory access (NUMA): Memory design, where memory access time depends on memory location relative to the processor**
- **Again, there also is interference with other users in the same computing node**

Affinity

- **Processor (task) affinity: Controls binding (*i.e.*, pinning) of a process to a core or socket (`mpirun -bind-to none` unbinds a rank from single core or socket, while `mpirun -bind-to socket` pins all threads within one socket)**
  https://en.wikipedia.org/wiki/Processor_affinity
- **Binding can improve cache performance but degrade load balancing**



- **"There still is not an easy way for pinning MPI processes & OpenMP threads to CPU sockets & cores." *How to gain hybrid MPI-OpenMP code performance without changing a line of code a.k.a. dealing with task affinity*:**

  https://aciref.org/how-to-gain-hybrid-mpi-openmp-code-performance-without-changing-a-line-of-code-a-k-a-dealing-with-task-affinity/

- **Don't worry about nonmonotonic behavior & submit what your got (again not runtime but efficiency)**
- **Will revisit false sharing & affinity in performance-optimization lecture**

About srun

- **While mpirun is a command in MPI system to execute a parallel program, some versions of Slurm job-scheduling system & OpenMPI implementation of MPI support srun to execute an MPI program.**

2. Does Open MPI support "srun -n X my_mpi_application"?

Yes, if you have configured OMPI --with-pmi=foo, where foo is the path to the directory where pmi.h/pmi2.h is located. Slurm (> 2.6, > 14.03) installs PMI-2 support by default.

https://www.open-mpi.org/faq/?category=slurm#slurm-direct-srun-mpi-apps

- **Consult a system administrator about specific settings after each upgrade (following descriptions on discovery might be somewhat outdated)**

srun Launch parallel tasks (i.e., job steps) (typically for MPI jobs)

https://carc.usc.edu/user-information/user-guides/hpc-basics/running-jobs

```
module purge
module load gcc/8.3.0
module load openmpi/4.0.2
module load pmix/3.1.3

ulimit -s unlimited

export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK

srun --mpi=pmix_v2 --cpu-bind=ldoms -n $SLURM_NTASKS ./mpi_plus_openmp_program.x
```

Automatically bind tasks to NUMA locality domains

https://carc.usc.edu/user-information/user-guides/hpc-basics/slurm-templates

Recap: Slide 20 in Hybrid MPI+MD Lecture

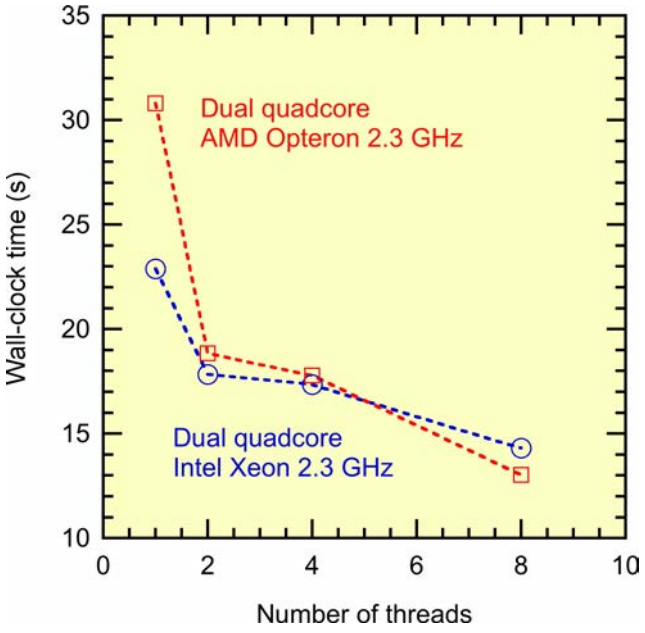
1 MPI process; 1-8 threads

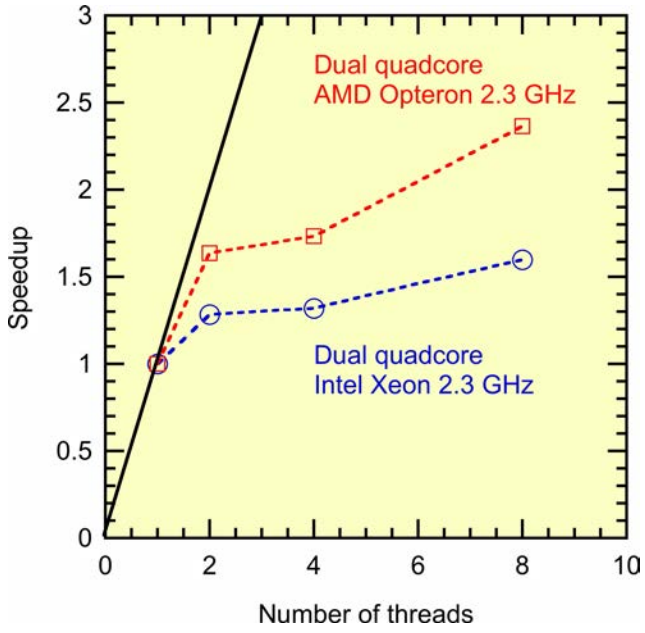
In hmd.h:

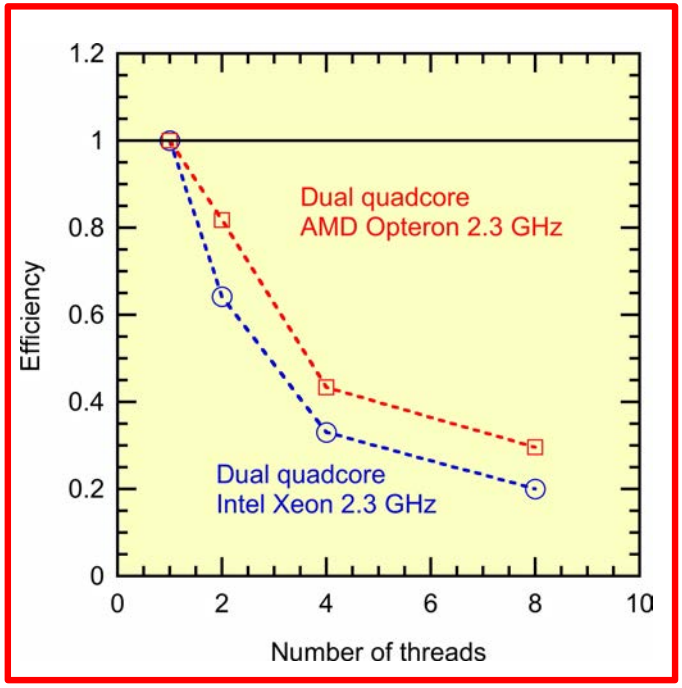
```
vproc = {1,1,1}, nproc = 1;
vthrd = {1,1,1}, nthrd = 1;
         2 1 1           2
         2 2 1           4
         2 2 2           8
```

pmd.in

```
24 24 24 InitUcell[3]
0.8      Density
1.0      InitTemp
0.005    DeltaT
100      StepLimit
101      StepAvg
```







`InitUcell[] = {24,24,24}`

$N = 4 \times 24^3$
$= 55296$ atoms

$$S_P = \frac{T(N,1)}{T(N,P)}$$

P: Number of cores

$$E_P = \frac{S_P}{P}$$

Just one curve (no need to compare different nodes)