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, http://cacs.usc.edu/education/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)

<table>
<thead>
<tr>
<th>$P$</th>
<th>$T(N, 1)$</th>
<th>$T(N, 2)$</th>
<th>$T(N, 4)$</th>
<th>$T(N, 8)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>14.92553</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>9.614403</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td>3.940460</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td>8.350226</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>$P$</th>
<th>$T(N, P)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>14.92553</td>
</tr>
<tr>
<td>2</td>
<td>9.614403</td>
</tr>
<tr>
<td>4</td>
<td>3.940460</td>
</tr>
<tr>
<td>8</td>
<td>8.350226</td>
</tr>
</tbody>
</table>

- 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 a 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
Recap: Slide 20 in Hybrid MPI+MD Lecture

1 MPI process; 1-8 threads

In hmd.h:

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

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

\[ N = 4 \times 24^3 = 55296 \text{ atoms} \]

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

\[ E_P = \frac{S_P}{P} \]

Just on curve (no need to compare different nodes)