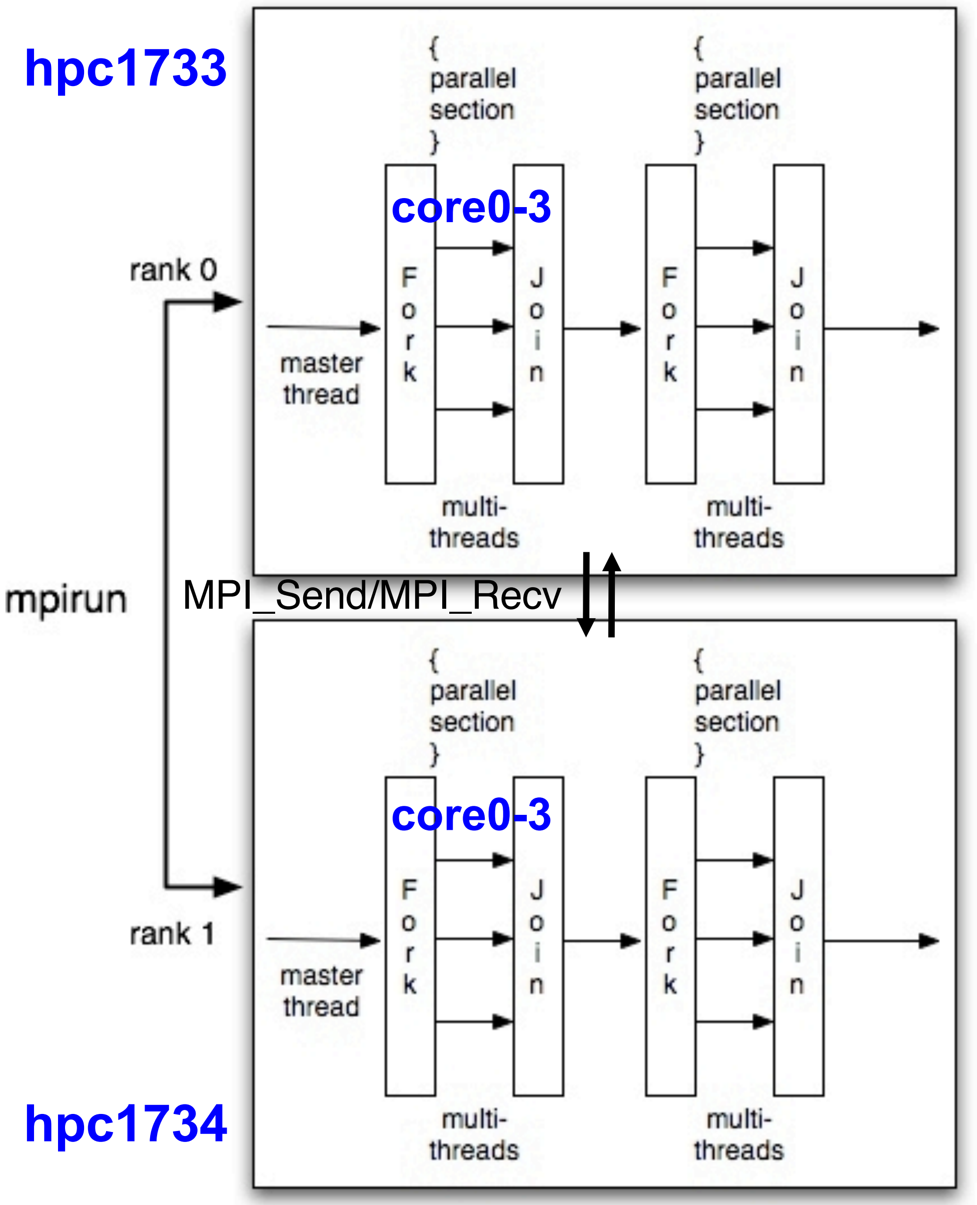
How Hybrid MPI+OpenMP MD Runs



In hmd.h:

```
int vproc[3] = {1,1,2}, nproc = 2;
int vthrd[3] = {2,2,1}, nthrd = 4;
```

In hmd.c:

```
omp_set_num_threads(nthrd);
```

On hpc-login3:

```
salloc --nodes=2 --ntasks-per-node=1
  --cpus-per-task=4 –t 30
```

On hpc1733:

```
srun –n 2 ./hmd
```

On hpc1733 & hpc1734:

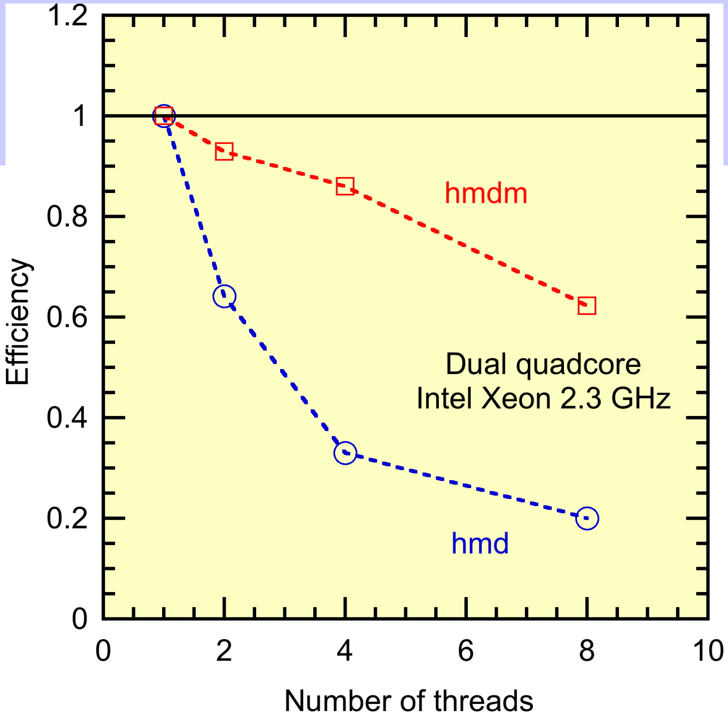
```
top (then type H and 1)
```

More on Multithreading MD

- **Large overhead is involved in opening an OpenMP parallel section**
  → Open it only once in the main function

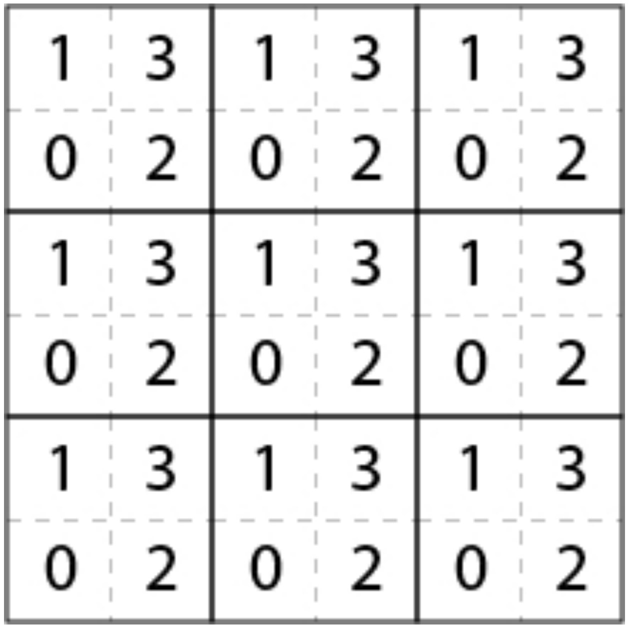
In hmdm.c:

```
int main() {
  ...
  omp_set_num_threads(nthrd);
  #pragma omp parallel
  {
    #pragma omp master
    {// Do serial computations here}
    ...
    #pragma omp barrier // When threads need be synchronized
    ...
  }
  ...
}
```



More on Avoiding Race Conditions

- Program **hmd.c**: (1) used data privatization; (2) disabled the use of Newton's third law → this doubled computation
- Cell-coloring
  > Race condition-free multithreading without duplicating pair computations
  > Color cells such that no cells of the same color are adjacent to each other
  > Threads process cells of the same color at a time in a color loop



1	3	1	3	1	3
0	2	0	2	0	2
1	3	1	3	1	3
0	2	0	2	0	2
1	3	1	3	1	3
0	2	0	2	0	2

H. S. Byun *et al.*,
Comput. Phys. Commun.
219, 246 ('17)

- Use graph coloring in more general computations

False Sharing

- **While eliminating race conditions by data privatization, the use of consecutive per-thread accumulators, `lpe_td[nthrd]`, degrades performance by causing excessive cache misses**

See false sharing Wiki page

- **Solution 1: Padding**

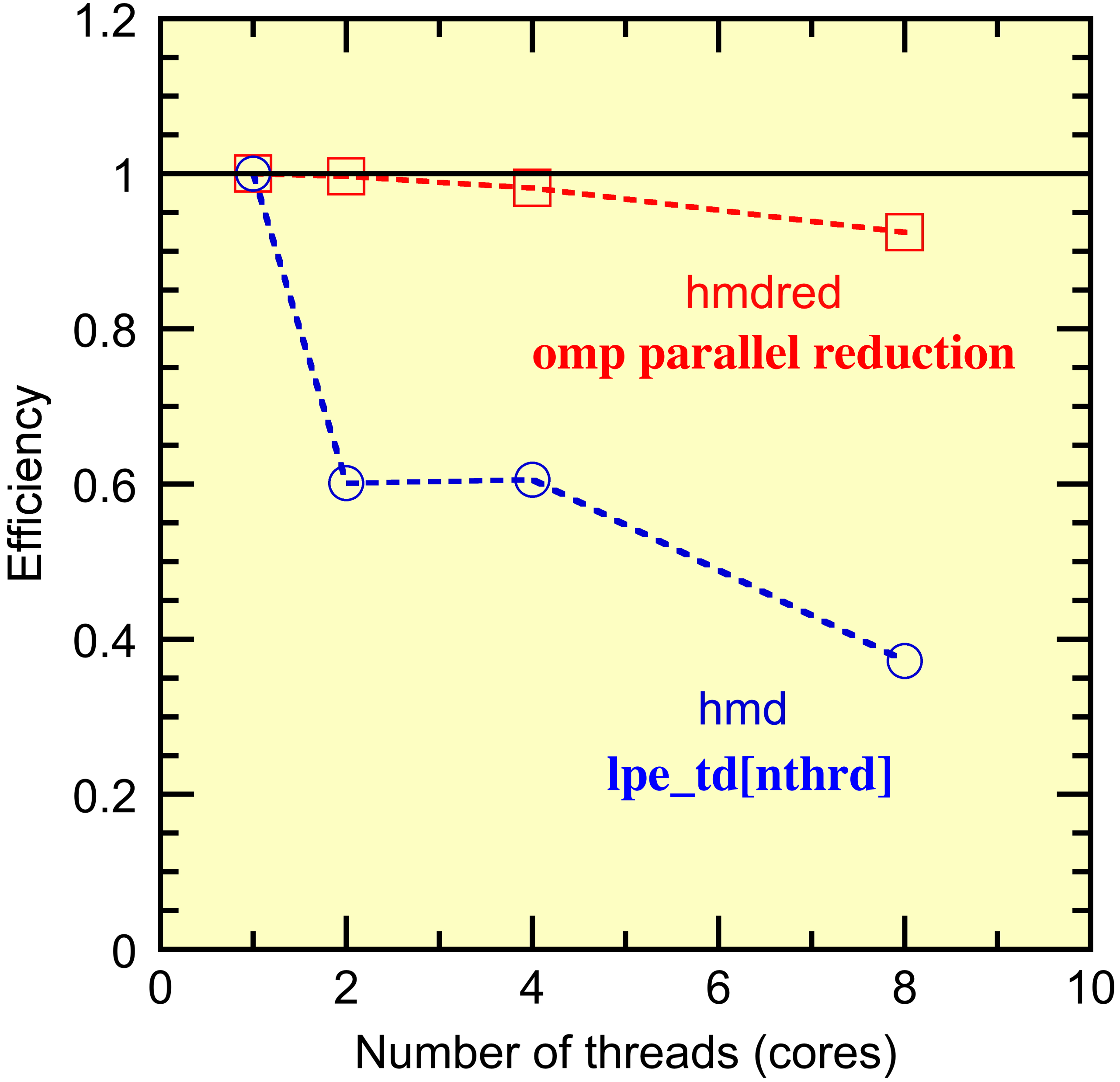
```
struct lpe_t {
  double lpe;
  double pads[7]; // assume intel CPU with 64 byte cache line
};
struct lpe_t lpe_td[nthrd];
```

- **Solution 2: System-supported data privatization**

```
 #pragma omp parallel private (...) reduction(+:lpe)
{
  ...
  lpe += 0.5*vVal;
  ...
}
// No reduction over the threads is required here
```

1. **Create private copies of the variable (`lpe`) in the reduction clause for all the threads**
2. **Perform the specified reduction operation (+) on the variable at the end of the parallel section**

Scalability Test



Some Like It as Arguments

- Use command line arguments for scaling tests without re-compiling multiple times
- **`hmd.c`** → **`hmdarg.c`** by adding the following lines in **`main()`**

```
int main(int argc, char **argv) {
  ...
  vthrd[0] = atoi(argv[1]);
  vthrd[1] = atoi(argv[2]);
  vthrd[2] = atoi(argv[3]);
  nthrd = vthrd[0]*vthrd[1]*vthrd[2];
  printf("Number of threads = %d\n", nthrd);
}
```

string-to-integer conversion

command-line argument

- Compiling

```
mpicc –o hmdarg hmdarg.c –fopenmp -lm
```

Strong-Scaling Test with `hmdarg.c`

```
[anakano@hpc-login3 cs596]$ salloc --nodes=1 --ntask-per-node=1 --cpus-per-
task=8 –t 59
...
[anakano@hpc1727 cs596]$ srun -n 1 ./hmdarg 1 1 1
Number of threads = 1
al = 4.103942e+01 4.103942e+01 4.103942e+01
lc   = 16 16 16
rc   = 2.564964e+00 2.564964e+00 2.564964e+00
thbk = 16 16 16
nglob = 55296
CPU & COMT = 1.073547e+01 2.005649e-02
[anakano@hpc1727 cs596]$ srun -n 1 ./hmdarg 2 1 1
Number of threads = 2
...
thbk = 8 16 16
nglob = 55296
CPU & COMT = 6.804797e+00 1.980424e-02
[anakano@hpc1727 cs596]$ srun -n 1 ./hmdarg 2 2 1
Number of threads = 4
...
thbk = 8 8 16
nglob = 55296
CPU & COMT = 4.956142e+00 1.981378e-02
[anakano@hpc1727 cs596]$ srun -n 1 ./hmdarg 2 2 2
Number of threads = 8
...
thbk = 8 8 8
nglob = 55296
CPU & COMT = 4.078273e+00 2.253795e-02
```