Will McNeill
18 April 2012
HPC Lab 8

NUMA, KMP, OpenMP/MPI Hybrid

1. Running stream_numa with numactl –interleave=all results in significantly faster benchmarks. We can attribute this to numa's memory policy placing chunks of memory in better places, allowing it to be accessed faster. This memory policy reduces the need to use memory on a different socket, thus reducing the need to use the interconnect, and reducing the associated time penalty.

2.
A-Setting KMP_AFFINITY to scatter results in faster benchmarks.
B-This is because scatter places threads as far apart from each other as possible. This distributes the threads across both sockets, allowing for maximized memory bandwidth, as we are using memory on both sockets. Compact places threads as close to each other as possible. This means all threads are on the same socket and share the same memory, which incurs a memory bandwidth penalty. Individual threads do not have to synchronize frequently in this example, making scatter the better policy. If threads had to communicate a lot or operate on lots of the same memory, scatter would not have such a strong advantage.

3. Raw data:

OpenMP only, 6 threads:	1 50 2.61409	speedup=5.996
OpenMP only, 12 threads:	1 50 1.3325	speedup=11.764
Hybrid, 6 threads, 4 MPI procs:	4 50 1.81019	speedup=8.659
Hybrid, 6 threads, 6 MPI procs:	6 50 1.8748	speedup=8.361

Using OpenMP only, we get very good performance with nearly linear speedup, very similar to what we see using MPI only. The hybrid configurations have better speedups than either pure MPI or pure OpenMP, but not by very much. Having overheads from both MPI and OpenMP contribute significantly to this effect (to further compound this issue, more MPI processes means more OpenMP overhead for the same problem size; note that the 6 MPI process configuration was actually slower than with 4).

To execute all these configurations, I wrote the following script:

```
#!/bin/bash
cd /home/mcneillw/HPC/lab8/work
. /curc/tools/utils/dkinit #so we can use the "reuse" command
reuse .openmpi-1.4.3_gcc-4.5.2_torque-2.5.8_ib
mpicxx -O2 -fopenmp trap.cpp -o trap

export OMP_NUM_THREADS=12
mpiexec -np 1 ./trap
export OMP_NUM_THREADS=6
mpiexec -np 1 ./trap
mpiexec -np 4 ./trap
mpiexec -np 6 ./trap
```

My modified trap code snippet:

```
#pragma omp parallel for reduction(+:integral) private(i)
for(i=1; i<local_n-1; ++i){
        double x = start_n +i*step;
        integral += f(x);
}
```