OpenMP Exercises

These exercises will introduce you to using OpenMP for parallel programming. There are four exercises:

1. OMP Hello World
2. Worksharing Loop
3. OMP Functions
4. Hand-coding vs. MKL

To begin, log onto an interactive node of Lonestar using your account:

ssh <user-name>@lonestar.tacc.utexas.edu

Untar the openmp_lab.tar file (in ~tg459572) into your directory:

tar xvf ~tg459572/LABS/lab_openmp.tar

cd into the lab_openmp directory

cd lab_openmp

The makefile that comes with these exercises is set up to use the Intel 11.1 compilers, which should be the default when you log on. Also, one of the exercises requires the Intel Math Kernel Library 10.3, so add that to your list of modules:

module add mkl/10.3

OMP Hello world

The *Hello world* example is very short, so for convenience we will run it on the interactive node where you're logged in. The other examples will run longer and will involve measuring performance, so they will be done on dedicated nodes through the batch system.

Look at the code in *hello.c* and/or *hello.f90*. This code simply reports OpenMP thread IDs in a parallel region. Compile hello.c or hello.f90 using the makefiles provided and execute first with 3 threads and then with 2 to 16 threads. (If you want Fortran, substitute hello_f90 for hello_c below.)

make hello_c
export OMP_NUM_THREADS=3
./hello_c
make run_hello_c

Worksharing Loop

Look at the code in file daxpy.f90. The nested loop repeats a simple DAXPY type of operation (double-precision ax+y, scalar times vector plus vector). It is repeated ten times in order to gather statistics on performance. Parameter N determines the size of the vector: N=48*1024*1024 is the default. A more detailed comparison will be done in the batch job. (If you prefer, the makefile lets you “make run_work” interactively.)

make daxpy
export OMP_NUM_THREADS=3
./daxpy

OMP Functions

Look at the code in work.f90. Threads perform some work in a subroutine called pwork. The timer returns wall-clock time. Compile work.f90 and run it with one set of threads to verify that it built properly. Running with other numbers of threads will be done in a batch job after all of the executables have been built.

make work
export OMP_NUM_THREADS=3
./work

Now look at work_serial.f90. We no longer use omp_lib, and numeric values are substituted for the calls to OMP_ functions. The OpenMP directives are ignored because the code is not compiled with OpenMP. As expected, this code runs with nearly the same speed as the work.f90 code with 1 thread. The overhead due to OpenMP is minimal in this case, because all threads are forked at the beginning and the parallel region contains all the work.

make work_serial
./work_serial
export OMP_NUM_THREADS=1
./work

Hand-coding vs. MKL

Look at the code in file daxpy2.f90. The nested loop performs a DAXPY operation for each outer loop. The DAXPY routine comes from the Intel MKL library, which is already parallelized with OpenMP (!). You should have loaded the MKL module at the beginning of these exercises. All you have to do is change the value of OMP_NUM_THREADS. Compare the performance to what you saw in earlier exercise with the hand-coded OpenMP version of DAXPY.

make daxpy2
export OMP_NUM_THREADS=3
./daxpy2

Next prepare to run a batch job. Edit the file *job* to put your account number after the -A flag. Note the –pe option is commented out; we will specify that option later. There are two notification lines you may wish to uncomment.

vi job

Now run a batch job that makes more detailed comparisons on a dedicated node. (If you prefer, the makefile has various interactive run_ and plot_ options; the run options can be done in batch, and the plot_ options can be done interactively.)

qsub –pe 12way 12 job

showq -u

Note, the number of OpenMP threads can exceed the number of physical cores.