Lab #2: Exploring OpenMP parallelism

PCSE 2015

Justification

OpenMP uses so-called 'fork/join' parallelism, where a thread can dynamically create a 'team' of threads. In this lab you will create a few small programs to test this dynamic behaviour, and how it relates to actually available parallelism.

To create a parallel region, you use the `parallel` directive. In C:

```
#pragma omp parallel
{
   // parallel region: statement or block
}
```

In Fortran:

```
!$OMP parallel
  ! parallel region
!$OMP end parallel
```

Preliminaries

At the top of your program, make sure to include for C

```
#include "omp.h"
```

and for Fortran:

```
use omp_lib
```

When compiling and linking your program, include the flag `-openmp` for the Intel compiler (stampede default), or `-fopenmp` for gcc.

Lab assignment

Do the following exercises. You can do most of this exercise on your laptop or other non-TACC machine, but questions 5 and 7 have to be done on stampede.

Each time the questions asks you to alter the source, copy your source file.

1. Start with the `hello.c` or `hello.F90` program. Compile and run it. What is the output?
2. OpenMP parallelism is controlled by the `OMP_NUM_THREADS` variable. Confirm that changing the value of this variable changes the output of the program, without needing to recompile.
3. The functions `omp_get_num_threads` and `omp_get_thread_num` return the number of threads in a team, and the number of the current threads. Use these functions to custommize the message to `hello world from thread 0 out of 4` et cetera.
   Note: in C, declare the variables inside the parallel region. In Fortran, add a clause `private( var1,var2,... )` to the directive. What happens if you leave that clause out?
4. The function `omp_in_parallel` tells you whether you are in a parallel region. Customize the message further to `Inside a parallel region, hello from` *m* `out of` *n* or `Outside a parallel region, hello from` *m* `out of` *n*.
5. The function `omp_get_num_procs` returns the number of threads that the hardware can physically support. Use this function. What is the value it reports, and can you explain that value?
6. Can you increase the number of threads to a value larger than what you got in the previous question?
7. On stampede, if you add the flag `-mmic` to your compilation and linking, you get an executable that will execute on the Intel Xeon Phi co-processor. Do this with the program that has the `omp_get_num_procs` call. What value do you get? Wow!
8. Finally, write a subprogram that has a parallel region in it. The subprogram should have a trace message both outside and inside the parallel region. Now call this subprogram both from outside and inside the parallel region in the main program. How many messages are you expecting? Try it out with 4 threads specified in the environment.
9. If in the last exercise you didn't get the result you were expecting, that is because nested OpenMP parallelism is controlled by the environment variable `OMP_NESTED`. Set this to `TRUE` and run the previous test again.