CS403 Parallel Programming
Lab03 - Introduction to OpenMP

Topics covered:

1. How to measure time for a block of C code
2. Introduction to OpenMP

Measure time for a block of C-code

To calculate time taken by a process, we can use `clock()` function which is available `time.h`. We can call the clock function at the beginning and end of the code for which we measure time, subtract the values, and then divide by `CLOCKS_PER_SEC` (the number of clock ticks per second) to get processor time.

`CLOCKS_PER_SEC` is a constant which is declared in `time.h`.

The sample code used for measurement is given below:

```c
#include <time.h>

int main()
{
    clock_t tic = clock();

    my_expensive_function_which_can_spawn_threads();

    clock_t toc = clock();

    printf("Elapsed: %f seconds\n", (double)(toc - tic) / CLOCKS_PER_SEC);

    return 0;
}
```

Note that this returns the time as a floating point type. This can be more precise than a second (e.g. you measure 4.52 seconds).

Introduction to OpenMP

OpenMP is an Application Program Interface (API), jointly defined by a group of major computer hardware and software vendors. OpenMP provides a portable, scalable model for developers of shared memory parallel applications. The API supports C/C++ and Fortran on a wide variety of architectures.

To compile:

```
gcc -fopenmp omp_hello.c -o hello
```

Sample code:
https://computing.llnl.gov/tutorials/openMP/samples/C/omp_hello.c
https://computing.llnl.gov/tutorials/openMP/samples/C/omp_workshare1.c
https://computing.llnl.gov/tutorials/openMP/samples/C/omp_reduction.c

Lab problems:

1. Familiarize yourself with the OpenMP code given in this handout.

2. Using the command in the section on “How to measure time for a block of C-code”, find the time required in the thread function call in the previous lab assignment.

3. Write a C-code using OpenMP threads to create an unbalanced load using `sleep` command and `omp_hello.c`. The sample sleep times are given below:
    a. thread-1: 10 sec, thread-2: 5 sec, thread-3: 20 sec, thread-4: 30 sec.
    b. Measure the total time taken for the complete execution of code with and without the additional sleep command.

    Hint: (i) You will require to include `unistd.h` for successful compilation.
    (ii) Use `sections` in OpenMP to create the unbalanced load.

4. Write a threaded code using OpenMP on matrix multiplication:
    a. Take overall execution time measurement using `time` command for different application size and thread count for the serial and parallel code.

Execution Time

Application Size	*p* = 1	*p* = 2	*p* = 4	*p* = 8	*p* = 16	*p* = 32
5x5						
10x10						
100x100						
500x500						
1000x1000						
5000x5000						
10,000x10,000						

$$\text{Speedup} = \frac{Execution\ Time\ (p)}{Execution\ Time\ (serial\ code)}$$

Speedup Time

Application Size	*p* = 2	*p* = 4	*p* = 8	*p* = 16	*p* = 32
5x5					
10x10					
100x100					
500x500					
1000x1000					
5000x5000					
10,000x10,000					

    b. Observe `gnome-system-monitor` output as your fire up different thread counts.

c. Use the overall execution time measurements to plot and comment upon the speed-up. Verify that:
    - Parallel execution time is more compared to the serial time when the application size is small.
    - For large application size, initially the speedup increases as you increase the number of threads. However, for very large number of threads (e.g. 64 threads) your performance becomes much worse w.r.t. serial code and lower thread count.

5. Multi-access threaded queue - using OpenMP

   Implement a multi-access threaded queue with multiple threads inserting and multiple threads extracting from the queue. Use mutex-locks to synchronize access to this queue. Document the time for 1000 insertion and 1000 extractions each with 4 insertion threads (producers) and 4 extraction threads (consumers).

   - Buffer size = 250

Lab-report:
Programming problem
- Objective or summary of problem statement
- Pseudo-code
- Measurements / results
- Conclusion

Attachments:
a) Lab report
b) unbalanced_load_hello.c
c) matmul_parallel.tar.gz (contains C code, executable)
d) producer_consumer.tar.gz (contains C code, executable)

NOTE: (1) In your lab-report, attach screenshots whenever necessary.
(2) For the programming questions, include your pseudo-code (in lab-report) and C-code *separately*.

References:
General:
https://computing.llnl.gov/tutorials/openMP/

How to measure time for a block of C code:
http://www.geeksforgeeks.org/how-to-measure-time-taken-by-a-program-in-c/
http://stackoverflow.com/questions/5248915/execution-time-of-c-program

Sections:
https://computing.llnl.gov/tutorials/openMP/#SECTIONS

http://stackoverflow.com/questions/2770911/how-does-the-sections-directive-in-openmp-distribute-work

Producer-consumer:
http://heather.cs.ucdavis.edu/~matloff/OpenMP/Examples/NM/ProdCons.c
http://www.cs.utah.edu/~mhall/cs4961f11/CS4961-L9.pdf