SOFE 3950U / CSCI 3020U: Operating Systems

TUTORIAL #9: OpenMP

Objectives

- Learn the fundamentals of parallel processing
- Gain experience writing parallel code using OpenMP

Important Notes

- Work in groups of **four** students
- All reports must be submitted as a PDF on blackboard, if source code is included submit everything as an archive (e.g. zip, tar.gz)
- Save the file as <tutorial_number>_<first student's id>.pdf (e.g. tutorial9_100123456.pdf)
- If you cannot submit the document on blackboard then please contact the TA with your submission at **jonathan.gillett@uoit.net**

Notice

It is recommended for this lab activity and others that you save/bookmark the following resources as they are very useful for C programming.

- http://en.cppreference.com/w/c
- http://www.cplusplus.com/reference/clibrary/
- http://users.ece.utexas.edu/~adnan/c-refcard.pdf
- http://gribblelab.org/CBootcamp

The following resources are helpful as you will need to use OpenMP to complete this tutorial.

- https://computing.llnl.gov/tutorials/openMP/
- http://openmp.org/mp-documents/omp-hands-on-SC08.pdf

Conceptual Questions

1. Explain what OpenMP is, what are it's benefits?

2. What are the **#pragma** definitions, what do they do?

3. Write the OpenMP #pragma definition to execute a loop in parallel.

4. What does the **reduction** do in the #pragma definition in OpenMP?

5. Explain the **critical** and **private()** declarations used in OpenMP.

Application Questions

All of your programs for this activity can be completed using the template provided, where you fill in the remaining content. A makefile is not necessary, to compile your programs use the following command in the terminal. **For this tutorial you MUST use gcc instead of clang. If you are still having issues please use -std=gnu99 instead of c99. You must add -lm as the very last argument for the compiler.**

```
gcc -Wall -Wextra -std=c99 -fopenmp <program name>.c -o <program name> -lm
```

Example:

```
gcc -Wall -Wextra -std=c99 -fopenmp question1.c -o question1 -lm
```

You can then execute and test your program by running it with the following command.

```
./<program name>
```

Example:

```
./question1
```

Template

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <omp.h>

int main(void)
{ ... }
```

Notice

You **MUST** use **gcc** for this tutorial, as there are issues using clang with OpenMP.

1. Ensure that you are able to execute and compile the following two demo programs: **demo_1.c** and **demo_2.c.**
    - Once you have both programs compiled and running add comments above each **#ifdef** and **#pragma** line in the code explaining what each statement does.
    - You should be able to get both examples working if you are having issues try and install the following: **apt-get install gcc-multilib**

2. Create a program using OpenMP that does the following.
    - Takes the number of threads to use as a command line argument, and sets the number of OpenMP threads to use (see demo examples).
    - Sets a variable **n** to 100 million

- Create an array **y** containing 100 million double values
- Calculates the step size **dx** as 1 / (n+1)
- Executes a for loop in parallel from i=0 to **n** with private variable **x**
- Each time calculates **x = i * dx**
- Calculates the following and store it in the array **y**
  y[i] = exp(x) * cos (x) * sin (x) * sqrt (5 * x + 6.0)
- If you are getting segmentation fault issues try: **ulimit -s unlimited**
- Try running your program with 1 up to 8 or more threads, you should see an increase in performance up to a certain point.

3. Create a program using OpenMP that does the following.
   - Initializes an integer array **x** of 100 million with random integer values between 0 and 100.
   - Performs a **serial** for loop i=0 to 100 million (don't use OpenMP), and records the wall time to calculate the norm of all of the values using a **serial for loop**.
     norm += fabs(x[i]);
   - Perform a **parallel** loop using OpenMP using a **reduction** on the summation of the norm value. Record the wall time to calculate the **parallel** loop using a **reduction**.
   - What is the difference in Wall time between the serial for loop and the parallel for loop with a reduction using OpenMP?
   - If you are getting segmentation fault issues try: **ulimit -s unlimited**
   - To accurately record the wall time of the serial and OpenMP loops use function **omp_get_wtime()** before after the for loops to get the wall time in seconds.

4. Create a program using OpenMP that does the following.
   - Create two 2-dimensional integer arrays **a,** and **b** that are 100x100
   - Initialize each of the 2-d arrays to a specified integer value (e.g. to the current index of the for loop for example).
   - Implement **parallel matrix multiplication** using OpenMP, you will need to think carefully how you implement it as it will require a triple nested loop.
   - You should use another tool to perform the matrix multiplication to verify that your parallel solution is correct, it is likely you may have race conditions and other issues if you do not set private variables correctly.
   - For details on matrix multiplication please see: https://en.wikipedia.org/wiki/Matrix_multiplication

5. Create a program using OpenMP that does the following.
    - Takes the number of threads to use as a command line argument, and sets the number of OpenMP threads to use (see demo examples).
    - Sets a variable **n** to 100 million
    - Calculates the step size **dx** as 1 / (n+1)
    - Executes for loop in parallel from i=0 to **n** with private variable **x**
    - Each time in the loop calculate
      x = i * dx
      y = exp(x) * cos (x) * sin (x) * sqrt (5 * x + 6.0)
    - For every **1 millionth calculation**, store the current index **i**, and the values for **x and y** to a file called **calculations.txt**
    - You will need to use the **critical section** feature of OpenMP in order to ensure the file is written to safely.
    - Try running your program with 1 up to 8 or more threads, you should see a certain number of threads that results in the optimal performance.
    - If you are getting segmentation fault issues try: **ulimit -s unlimited**