COMPSCI 340 S2 2024

Assignment 1

15% of your grade
Due date: 11:59 pm Friday 16th August
Total – 15 marks

Introduction

We no longer live in a world where computing speeds increase along with Moore's Law. Instead, we maintain increased throughput in our computers by running more cores and increasing the amounts of parallelism. Parallelism on a multi-core (multi-processor) machine is looked after by the operating system.

In this assignment you have to parallelise a simple algorithm in a number of different ways and answer questions summarising your findings.

This assignment is to be done on a Unix based operating system. We recommend the flexit.auckland.ac.nz Ubuntu image (called Ubuntu Linux 20.04). The distributed code runs on Linux, but you can modify it to run on any Unix based operating system such as MacOS. It should run on the Windows subsystem for Linux. To get the results for this assignment you will need a machine (real or VM) with 4 or more cores. When you come to time your programs run them all on the same machine.

Markers will use the Ubuntu image provided on flexit.auckland.ac.nz. So please ensure that your submitted programs run in that environment. The FlexIT Ubuntu image has at least 4 cores.

Matrix Multiplication

The algorithm you have to parallelise is the matrix multiplication. Just to remind you, the matrix multiplication is an operation in which two matrices are multiplied together to produce a third matrix. Given two matrices A and B, with size *m x n* and *n x p* respectively, their product C is a matrix of size *m x p*. The element c*ij* of the resulting matrix C is obtained by taking the dot product of the *i*th row of A and the *j*th column of B.

The version of the matrix multiplication you have to use is written in C and available on the assignment Canvas page as `a1.0.c`. We are assuming both input matrices to be square matrices with size *n x n*.

In all of the work which follows do NOT optimise the C compilations. Always compile using:

`cc filename.c -o filename` (you will need to use `-pthread` for some programs).

Do all of the timings on the same (or very similar) machine and don't watch videos or do similar things while you are taking the timings. Take the timings at least 3 times and average them.

Things to do

Step 0

Read through and understand the code in the file `a1.0.c`.

Compile and run the program.

Run the program by including the matrix size parameter on the command line (and optional seed value) e.g.

```
./a1.0 [matrix_size] [optional_seed]
```

`Say ./a1.0 100` runs the program with two input matrices of size 100 x 100 where the input matrix elements get filled by random values. The program uses the seed value (if provided) to initialize the random number generator using the `srand()` function. If you call `srand()` with the same seed, you will get the same sequence of random numbers each time.

Also note, the program prints the randomly generated input matrices and the resultant matrix up to a `MAX_SIZE` of 4.

Determine the maximum matrix size that can be dealt by this program before you get a segmentation fault.

Segmentation faults happen when something goes wrong with memory accesses or allocations (a segment is an area of memory). In our case the program allocates space for the array of values, and extra working space, on the stack, and as the size of the array increases eventually, we run out of stack space.

Question 1 [0.5 mark]: What environment did you run the assignment on (e.g., operating system, number of cores, amount of real memory)? Hint: `man uname`, `man free` and `man lscpu`. The output of `uname -a` provides some of this information, `free` provides information on the amount of memory, and `lscpu` provides information on the number of CPUs. Also mention whether you were using a virtual machine and if so say which one. **[0.25 marks]**

What approximate size for the matrix can you get to in the original program before a segmentation error occurs**? [0.25 marks]**

You don't need to submit a1.0.c

Step 1

Modify the program (call it `a1.1.c`) to allocate the matrices on the heap using `calloc()` so that the program can at least deal with a size of 2000 x 2000. After using the allocated memory, it is important to free it using the `free()` function to avoid memory leaks.

Time how long it takes the program to multiply matrices of size 2000 x 2000 and record the result.

```
time ./a1.1 2000
```

Submit this program as a1.1.c [1 mark]

Question 2 **[0.5 mark]:** Run "ulimit -a". What is the maximum size of memory that can be allocated on the stack, and why is it limited? Also, what is the difference between stack and heap memory allocation in C programming?

Step 2

Modify the program (call it `a1.2.c`) to use two threads to perform the multiplication.

You will need to make sure you are running on a machine with at least 2 cores. If you are using a virtual machine you may need to change the configuration to use at least 2 cores. The FlexIT Ubuntu image has at least 4 cores.

Hint: `man pthread_create`, `pthread_join`

You may want to lock the critical section (i.e., to avoid threads accessing the shared output matrix at the same time). Hint: `man pthread_mutex_init`, `pthread_mutex_lock`, `pthread_mutex_unlock`.

Time how long it takes the program to multiply matrices of size 2000 x 2000 and record the result.

Submit this program as a1.2.c [1 mark]

Question 3 **[0.5 mark]:** Explain how you parallelized the multiplication and compare the times you observe with the times from Step 1. Explain why the times are different.

Step 3

Modify the program (call it `a1.3.c`) to use as many threads as there are cores to perform the multiplication, and no more.

You will need to make sure you are running on a machine with at least 4 cores. If you are using a virtual machine you may need to change the configuration to use at least 4 cores (if your computer can do this). The FlexIT Ubuntu image has at least 4 cores.

First you need a way to determine from within the program how many cores you have in the environment you are using. Hint: `man sysconf` or `man get_nprocs_conf`

You don't have to worry about the other work going on in the computer just proceed as if all cores are available for your multiplication program.

One way could be to use some shared state to keep track of how many threads are currently active. Every time a new thread is started you should add one to a counter and every time a thread stops running you should reduce that counter OR you could split the matrix rows among the available threads.

You may want to lock the critical section (i.e., to avoid threads accessing the shared output matrix at the same time). Hint: `man pthread_mutex_init`, `pthread_mutex_lock`, `pthread_mutex_unlock`.

Time how long it takes the program to multiply matrices of size 2000 x 2000 and record the result.

You should also take screen shots of the `System Monitor` program showing the `Resources` tab as your program runs. This will prove that all cores are being used.

Submit this program as a1.3.c [1 mark]

Question 4 [1 mark]: Explain how you parallelized the matrix multiplication and compare the times you observe with the times from Step 2. Explain why the times are different.

Step 4

Go back to step 2 and modify the program (call it `a1.4.c`) to use two processes rather than two threads.

Processes normally don't share memory with each other and so there will have to be some communication between the processes. Hint: `man fork`, `pipe`.

One of the interesting things is that the `fork` system call copies the data in the parent process so that the child can see the data from the parent (at the time of the `fork`). This means the child process does not need to copy data from the parent to the child. However, after the child has performed multiplication, the resulting values have to be sent back to the parent in order for it to compute the output matrix.

Time how long it takes the program to multiply matrices of size 2000 x 2000 and record the result.

Submit this program as a1.4.c [1 mark]

Question 5 [1.5 mark]: Explain how you set up the pipes and send the multiplied data back to the parent. In your answer include how the parent process knows when a child process has completed its multiplication, and how it puts the received data into the correct place in the resultant matrix.

Question 6 [1.5 mark]: Compare the times and memory usage of this program with the two threads version (Step 2). Explain the results.

Step 5

Same as step 4 but rather than passing information back to the parent process we share the memory to be involved in multiplication in the processes. Hint: man mmap, wait.

Time how long it takes the program to multiply matrices of size 2000 x 2000 and record the result. You should also take a screen shot of the System Monitor program showing the Resources tab as your program runs.

Submit this program as a1.5.c [1 mark]

Question 7 [1 mark]: Explain how the parent process sets up the shared memory for the child processes. In particular, say whether or not the parent is sharing the same data with all the children, and why you did it that way.

Question 8 [1 mark]: Explain how the parent process knows when a child process has completed its multiplication.

Question 9 **[1 mark]:** Compare the times and memory usage of this program with the earlier steps (mainly Step 2, Step 4). Explain the results.

Question 10 **[1.5 marks]:** Which of the steps do you consider gives the best solution? You may order the techniques from slowest to fastest and include a brief explanation of what is happening in each of them and how that relates to their performance. You may include relevant timing information and screen shots from the `System Monitor`.

Submission

1. Enter your zipped source files in the "Assignment 1 Programs" Assignment on Canvas. Remember to include your login in each file.
2. Enter your answers to the questions in the "**Assignment 1 Questions**" Canvas quiz. As this is set up as a quiz, I recommend you write your answers in a file and prepare them before running the quiz and pasting the answers into the corresponding question field.

- **The program files are each 1 mark, please ensure your program prints the input and resultant matrices up to a MAX_SIZE of 4, and for each version the output should be the same for the same seed value.**
- **Questions from each step will be marked only if you have written and submitted the program from that step.**
- **The submitted source code files must include your name and UPI.**

By submitting a file, you are testifying that you and you alone wrote the contents of that file (except for the component given to you as part of the assignment).