CO471 - Parallel Programming

M0 - Programming with POSIX Threads

Contents

Note: (a) Deadline: 9AM, Monday, Jan, 2. Pack your code, screenshots and other files in an archive and mail to `co471.nitk@gmail.com`. (b) This is a team assignment. At most two students per team. One submission per team.

Q1. Hello World Program - 0

Hello World Program - 0

Create 5 threads with the `pthread_create()` routine. Each thread prints a "Hello World!" message, and then terminates with a call to `pthread_exit()`.

Q2. Hello World Program - 1

Hello World Program - 1

Modify the above program. Pass a thread id to each thread and have them print it.

Q3. Know Your System

Know Your System

Find a multicore environment where you can use at leat two cores. Investigate the environment and answer the following:

1. What is the model, make and core count of your CPU?
2. What are the L1, L2, L3, (L4?) cache, and memory sizes?
3. Clock speed, the MIPS and the MFLOPS rating for your CPU?
4. Disk seek, latency, and transfer times (your local hard disk)?
5. How to know if your processor is 64b or 32b? How do you know if the processor supports "hyperthreading"?
6. OS version? Is 32 or 64 bit?
7. Available VM size for user processes. Can it be changed?
8. Limits on the stack space, heap space and static area. Can they be changed?
9. What is a memory leak and how can you detect one?
10. How can one cause a signal on stack overflow?
11. How can one tell how many page faults a process had?
12. How can one tell how much user, system and elapsed time a process used.
13. How can one time a procedure in a program? How accurate is this?

Q4. DAXPY Loop

DAXPY Loop

D stands for Double precision, A is a scalar value, X and Y are one-dimensional vectors of size 2^{16} each, P stands for Plus. The operation to be completed in one iteration is X[i] = a*X[i] + Y[i]. Your task is to compare the speedup (in execution time) gained by increasing the number of threads. Start from a 2 thread implementation. In your implementation, have the master thread "wait" for all threads to complete before exiting the program (use `pthread_join()`). How many threads give the max speedup? What happens if no. of threads are increased beyond this point? Why?

Note: Speedup = $\frac{T_N}{T_1}$. T_N: Execution time of an n-threaded program; T_1: Execution time of the uniprocessor implementation.

Q5. Dot Product with Mutex

Dot Product with Mutex

Write a program to compute dot product of two vectors and calculate its running sum. An iteration in a sequential implementation would look like `Sum = Sum + (X[i] * Y[i]);` . Use mutex variables in this program. Arrays X and Y, and variable Sum are available to all threads through a globally accessible structure. Each thread works on a different part of the data. The main thread waits for all the threads to complete their computations, and then it prints the resulting sum.

Hint: Wrap the critical section between the `pthread_mutex_lock()` and `pthread_mutex_unlock()` calls. Use the `pthread_mutex_init()` and the `pthread_mutex_destroy()` calls to create and destroy mutexes.

Q6. Signalling using Condition Variables

Signalling using Condition Variables

Implement the following behaviour in a pthreads program using mutexes and condition variables. Create 2 increment-count threads and 1 watch-count thread. Increment-count threads increment a count variable (shared by both) till a threshold is reached. On reaching the threshold, a signal is sent to the watch-count thread (use `pthread_cond_signal`). The watch-count thread locks the count variable, and waits for the signal (use `pthread_cond_wait()`) from one of the increment-count threads. As signal arrives, the watch-count thread releases lock and exits. The other two threads exit too.

Q7. Matrix Multiply

Matrix Multiply

Build a pthread implementation of multiplication of large matrices (Eg. size could be 250x250 - populate your matrices using a random number generator). Repeat the experiment from the DAXPY question for this implementation. Think about how to partition the work amongst the threads - which elements of the product array will be calculated by each thread?

Additionally, what are the largest size MM programs that can be run? How much do they page fault? What are the largest size (n using matmul) programs that can be run with limited pagefaults? Using this program, investigate the limits of static, stack and heap sizes on your environment. You should also check the `ulimit` bash built in, and several useful system calls, such as `getrlimit()` and `setrlimt()`.