

## ECE574: Cluster Computing – Homework 4

### POSIX Threads (pthreads)

**Due: Friday 26 February 2021, 5:00pm**

#### 1. Background

- In this homework we will take the sobel code from HW#3 and parallelize it using pthreads.
- A good tutorial on pthreads can be found here:  
<https://computing.llnl.gov/tutorials/pthreads/>

#### 2. Setup

- For this assignment, log into the same Haswell-EP machine we used in previous homeworks. As a reminder, use the username handed out in class and ssh in like this  

```
ssh -p 2131 username@weaver-lab.eece.maine.edu
```
- Download the code template from the webpage. You can do this directly via  

```
wget http://web.eece.maine.edu/~vweaver/classes/ece574_2021s/ece574_hw4_code.tar.gz
```

to avoid the hassle of copying it back and forth.
- Decompress the code  

```
tar -xzvf ece574_hw4_code.tar.gz
```
- Run make to compile the code.
- You may use your own code from HW#3 as a basis for this assignment. (If you had trouble with HW#3, I provide some simple and poorly-optimized sample code you can use instead). If you wish to use your own code, just copy your `sobel.c` file from HW#3 over top of the provided `sobel_coarse.c` file in the HW#4 directory.

#### 3. Coarse-grained Parallel Code (6 points)

Implement simple two-thread parallelism where you run `sobel_x` and `sobel_y` in parallel, but then join and do the combine step serially.

- Edit the file `sobel_coarse.c`
- Convert the code to use pthreads.
- You may need to add `#include <pthread.h>`
- Modify `generic_convolve` to be of `void *` type and take one `void *` argument. You will have to create a `struct` to hold the values you want to pass in and do some casting back and forth from the void pointer. This is some tricky C coding, so the provided `sobel_coarse.c` example shows you how to do this.
- Create one thread for each convolve operation using `pthread_create()`
- Once both threads are running, have the main thread wait for them using `pthread_join()`
- Be sure to comment your code!
- Compare the results generated to make sure they match the output given by your HW#3 code.

- Report results gathered on haswell-ep: run your code using  
`sbatch time_sobel.sh`  
 Which will use the provided `space_station_hires.jpg`  
 Report how long it takes to run compared to the time taken by your single-threaded HW#3 code.

#### 4. Instrument with PAPI (1 point)

Ideally PAPI should run just fine on multi-threaded code, but it sometimes can have some issues. So for this homework we will use a different feature of PAPI, which is using it to gather time results rather than performance counter results.

- If using your own code from HW#3, you can comment out the code that creates the eventset and starts/stops it, we won't be needing that.
- With PAPI you can gather a current timestamp with microsecond granularity via `PAPI_get_real_usec()`.
- To measure how long a routine is, just measure the timestamp before and after, then subtract. The value is a 64-bit one, so make sure you assign it to a value of type `long long` and print it using the `"%lld"` option in `printf()`.
- Have your code measure and print the following times:
  - (a) Total Convolution time (from just before you start the convolution to after both `sobelx` and `sobely` finish)
  - (b) Combine time (from before the combine starts to after it finished)
  - (c) JPEG Load Time
  - (d) JPEG Store Time

#### 5. Fine-grained Parallelism (2 points)

Getting more parallelism out of our code is possible, but is a bit more difficult. In this part we will attempt to parallelize the convolution code internally. Note: this can be complicated to get fully working.

- Instead of doing simple 2-thread parallelism, parallelize the entire code base at a fine-grained level.
- Copy your `sobel_coarse.c` file over `sobel_fine.c` and then modify `sobel_fine.c`
- Split up each operation into N number of parts, where N is configurable.
  - If you want, you can just make this a `#define` in your code and statically allocate all of the thread info. Doing this will require you to change and recompile your code if you vary the thread value.
  - Alternatively, you can get the thread number from a command line argument and dynamically allocate everything (something like `./sobel_fine IMAGE.jpg threads`)
  - Each element of the sobel operation is independent, so you can split up the input image into arbitrary sizes (say 8 for this example).
  - Create 8 threads, run `sobel_x` in parallel (each on 1/8th), join when done.
  - You will need to modify your `convolve()` function to take start/stop parameters, and only operate on the values from start to stop.

- Also be sure to run `sobel_y` in parallel, and also modify `combine()` in a similar way.
- If your image is not an integer multiple of `N` you will need to have fixup code at the end to make sure the edges get processed properly.
- Record the total time (using `time`) as well as the PAPI timing measurements for 1, 2, 4, and 8 threads in the README file.

## 6. Question (1 pt)

Put the answer to the following question in the README file.

- (a) You are running multi-threaded pthread code, and you have the following two functions that can be called by multiple threads at a time. To protect the critical sections, mutexes are used.

Can anything go wrong with this code? If so, describe a path through the code that can trigger a failure. What is this type of failure called?

```
void function_one(void) {
    pthread_mutex_lock(&mutex1);
    pthread_mutex_lock(&mutex2);
    /* critical section */
    pthread_mutex_unlock(&mutex2);
    pthread_mutex_unlock(&mutex1);
}

void function_two(void) {
    pthread_mutex_lock(&mutex2);
    pthread_mutex_lock(&mutex1);
    /* critical section */
    pthread_mutex_unlock(&mutex1);
    pthread_mutex_unlock(&mutex2);
}
```

## 7. Submitting your work.

- Be sure to edit the README to include your name, as well as the timing results, and any notes you want to add about your something cool.
- Run `make submit` and it should create a file called `hw04_submit.tar.gz`. E-mail this file to me.
- e-mail the file to me by the homework deadline.