Systems Software HS15

Lab Exercises

Claudio Mura
claudio@ifi.uzh.ch

Practical notes on multi-threaded programming using PThreads

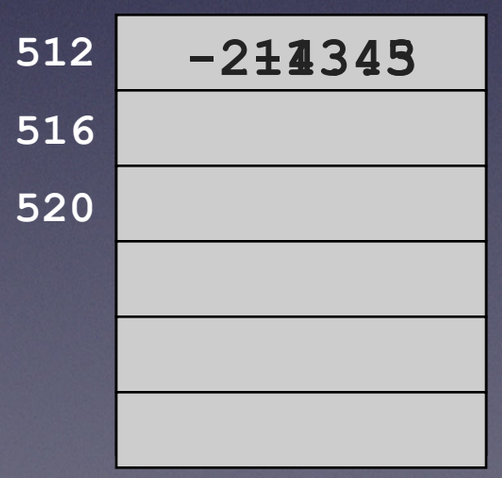
PThreads

- Thread = “lightweight process”
  - less overhead for execution of parallel code
  - threads are subsets of a process
  - share resources (code, data, files)
    - have own registers and stack
- Pthreads: POSIX standard for threads
  - procedures for process creation, management and synchronization

C/C++ Pointers

- Variable = memory location holding values
  - at that location, some memory has been allocated to hold the value
- Each memory location is represented by an address
- A pointer stores an address
  - corresponding to a location associated to a variable...
  - ...or to a location for which memory was not allocated!!!

```
float x = -214.43;
float* x_addr = &x;      // x_addr holds 512
std::cout << *x_addr;    // prints -214.43
*x_addr = -13.5;
std::cout << *x_addr;    // prints -13.5
float* y;
*y = 5;                  // error! no memory allocated
```



C/C++ Pointers

- Allocating / deleting memory explicitly with new / delete:

```
float* x = new float;      float* arr = new float[ 10 ];

delete x;  delete[] arr;
```

- Passing parameters by pointer:

```
int b = 5;
cout << b;        // prints 5
pass_val( b );
cout << b;        // prints 5
pass_ptr( &b );
cout << b;        // prints 15
```

```
void pass_ptr( int* a )
{ *a = 15; }

void pass_val( int a )
{ a = 15; }
```

- Pointers can refer to functions too!

```
void (*fct_ptr)( int * );
fct_ptr = pass_ptr;
```

Thread Creation

- A thread is identified by variable of type `pthread_t`
- Can be created using `pthread_create()`
  - takes in input *pointer* to `pthread_t` variable
  - pointer to struct with attributes for thread
  - pointer to routine to be executed
    - that is, the name of the function
  - pointer (`void*`) to variable
    - workaround to pass input attributes
    - can pass a pointer to struct variable of arbitrary type

Waiting for Termination

- After `pthread_create()`, your process consists of 2 threads
  - how to synchronize their execution?
- The "child" thread calls `pthread_exit()`
  - takes `void*` (might be NULL)
- The *main* thread waits for its termination using `pthread_join()`
  - takes `pthread_t` variable (thread to be waited for)
  - 2nd param is for possible values made available by child using `pthread_exit()` (might be NULL)

pthread_create(): Example

```
#include <pthread.h>
#include <iostream>

struct point {
  int x; int y;
};

void* func( void* ptr ) {
  point* p = ( point* )ptr;
  std::cout << p->x << " " << p->y << std::endl;
  pthread_exit( NULL );
}

int main() {
  point var;
  var.x = 10; var.y = 20;
  pthread_t thread;
  int ret = pthread_create( &thread, NULL, func, ( void* )( &var ) );
  pthread_join( thread, NULL );
  return 0;
}
```

pthread_create(): Example

```
struct point {
  int x; int y;
};

void* func( void* ptr ) {
  point* p = ( point* )ptr;
  std::cout << p->x << " " << p->y << std::endl;
  pthread_exit( NULL );
}

int main() {
  int ret;
  point var;
  var.x = 10; var.y = 20;
  pthread_t thread1, thread2;
  ret = pthread_create( &thread1, NULL, func, ( void* )( &var ) );
  var.x = 20; var.y = 40;
  ret = pthread_create( &thread2, NULL, func, ( void* )( &var ) );
  pthread_join( thread1, NULL );   pthread_join( thread2, NULL );
  return 0;
}
```

Error!

Compilation

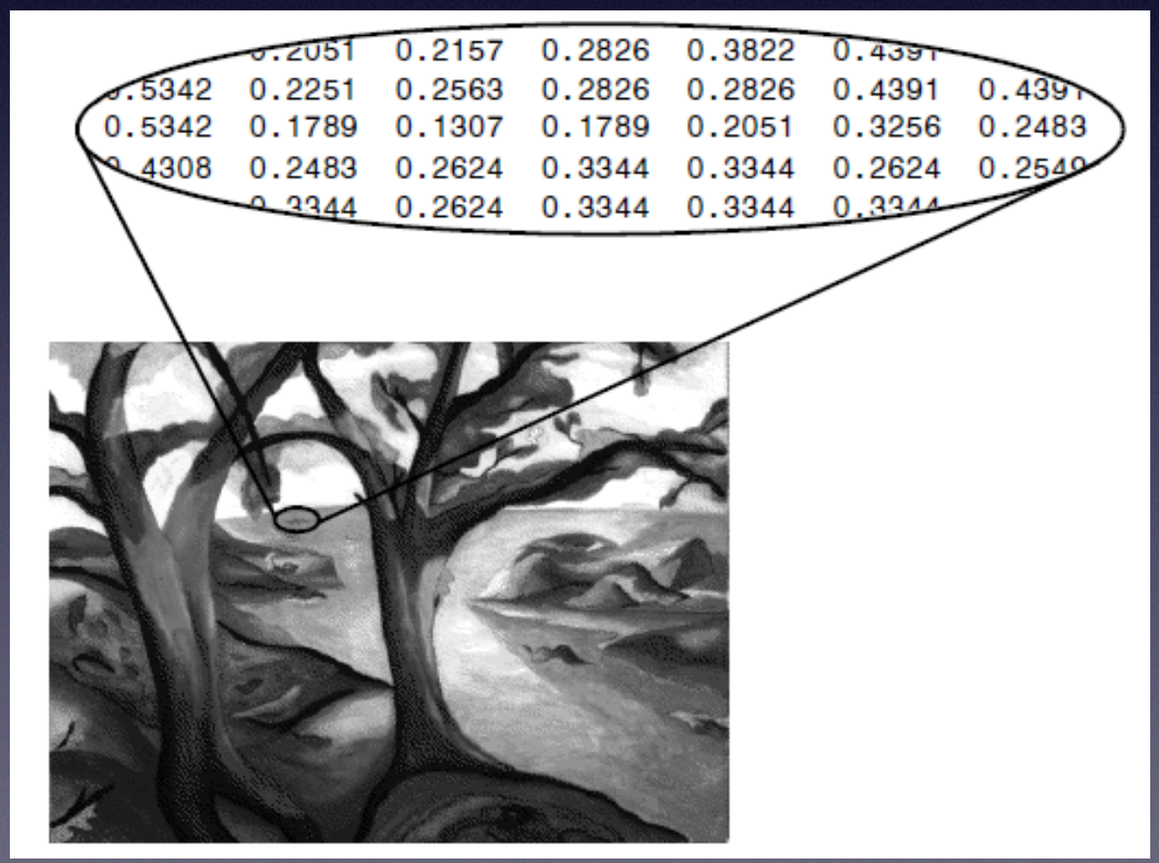
- You must add a flag to your compilation command
  - `-pthread / -lpthread`
  - `g++ main.cpp -pthread -o my_exec`
- The same holds for the Makefile
  - remember to modify the compilation rules accordingly

Exercise 3

Parallel Image Denoising using PThreads

Digital Images

- An image can be represented as a matrix of pixels
  - each pixel stores a color or intensity value (grayscale images)
  - typically, values between 0 (black) and 1 (white)



- Image acquisition: set value for each pixel

Noise & Image Denoising

- Problem: noise during the acquisition process
  - wrong values, due to the imaging sensor

- Solution: image denoising algorithm
  - replace each pixel with a “good” one from the surrounding area
  - meaning of “good” depends on the actual technique used…

Median Filtering

- Replace value of a pixel with median of surrounding values
  - surrounding values = values in a *window* of size *s*

.02	.04	.05	.03	.01
.02	.03	.03	.04	.03
.01	.02	.99	.03	.00
.00	.03	.03	.00	.03
.00	.01	.01	.04	.03

Median Filtering

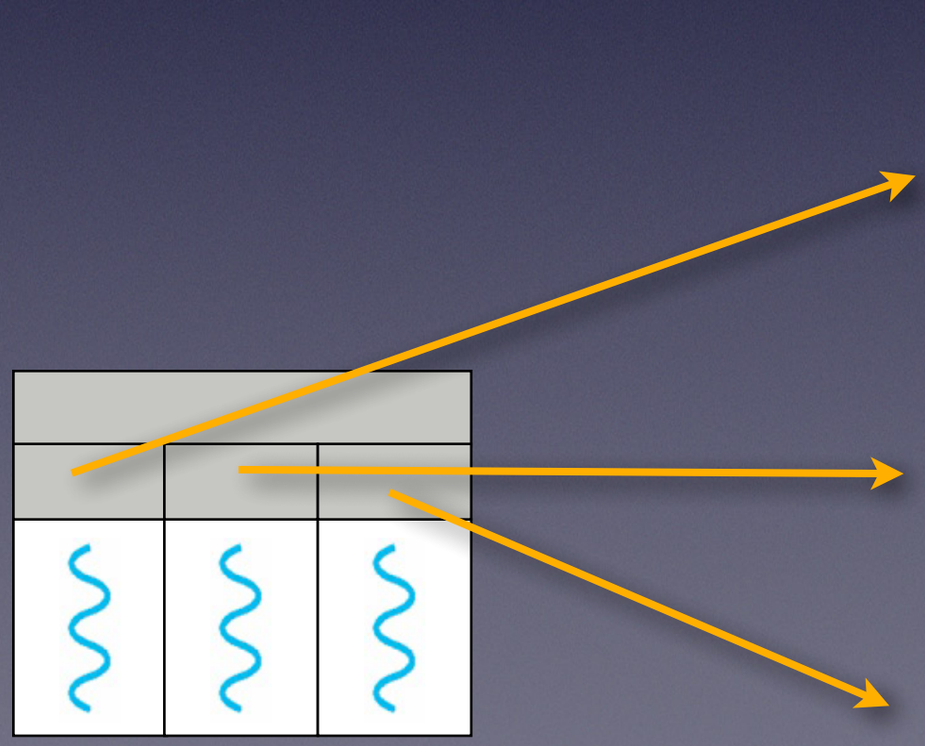
- Replace value of a pixel with median of surrounding values
  - surrounding values = values in a *window* of size *s*

.02	.04	.05	.03	.01
.02	.03	.03	.04	.03
.01	.02	.03	.03	.00
.00	.03	.03	.00	.03
.00	.01	.01	.04	.03

.00	.02	.03	.03	.03	.03	.03	.04	.99

Parallel Median Filtering

- Filtering of each pixel is independent
  - don't need filtered values of other pixels! no data dependencies
- Can parallelise the problem if we have multiple threads!
  - assign every thread a portion of the matrix to compute
  - portions of equal size = better performance



.02	.04	.05	.03	.01
.02	.03	.03	.04	.03
.01	.02	.99	.03	.00
.00	.03	.03	.00	.03
.00	.01	.01	.04	.03

Your task

- Write an application that performs median filtering on a grayscale image
  - must run in serial and, more importantly, in parallel mode
- Create multiple threads using the PThread library
  - assign a range of pixels to each thread
  - this assignment defines how the workload is split among threads
- Task of each thread: for each pixel in its assigned range
  - access pixels in the window surrounding the pixel
  - compute the median value
  - set the median value as new (filtered) value of the pixel
- Where to store it?
  - use a separate output matrix; avoid read-write conflicts!

Additional details

- Read input image (matrix of floats) from text file
- Command line arguments:
  - filename of text file with input matrix
  - *window size* (positive integer)
  - *no_threads*: number of threads to be created (positive integer)
  - *mode* (binary value, 0 or 1): flag that controls whether the application should run serially or in parallel
- Write filtered image to text file named `filtered.txt`
- Be careful with pixels on the boundaries!
  - the window might fall outside the image
  - only consider pixels *inside*