Please print in pen:
Waterloo Student ID Number:

WatIAM/Quest Login Userid:

UNIVERSITY OF WATERLOO

Examination
Final
Winter 2019
ECE 459

Times: Wednesday 2019-04-24 at 12:30 to 15:00 (3PM)
Duration: 2 hours 30 minutes (150 minutes)
Exam ID: 4026613
Sections: ECE 459 LEC 001,002
Instructors: Jeff Zarnett, Patrick Lam

Open Book

Candidates may bring any reasonable aids. Open book, open notes. Calculators without communication capability permitted.

Instructions:

1. This exam is open book, open notes, calculators with no communication capability permitted.
2. Turn off all communication devices. Communication devices must be stored with your personal items for the duration of the exam. Taking a communication device to a washroom break during this examination is not allowed and will be considered an academic offence.
3. Place all bags at the front or side of the examination room, or beneath your table, so they are inaccessible.
4. There are six (6) questions. Not all are equally difficult.
5. The exam lasts **150** minutes and there are 120 marks.
6. Verify that your name and student ID number is on the cover page.
7. If you feel like you need to ask a question, know that the most likely answer is "Read the Question". No questions are permitted. If you find that a question requires clarification, proceed by clearly stating any reasonable assumptions necessary to complete the question. If your assumptions are reasonable, they will be taken into account during grading.
8. Do not fail this city.
9. After reading and understanding the instructions, sign your name in the space provided below.

Signature

Marking Scheme (For Examiner Use Only):

Question	Mark	Weight
1		30
2		21
3		19
4		25
5		10
6		15
Total		**120**

Please initial:

1 Short Answer [30 marks total]

Answer these questions using at most three sentences. Each question is worth 3 points.

(a) You're considering using value speculation on a particular piece of code. What numbers (facts) do you need to know to determine if it is worth using it?

(b) You have a small server and a program that is 25% sequential and 75% parallel. You have some budget to spend. You can either go from 8 CPUs to 10 or you can spend the money on development to reduce the sequential part by 2%. Which do you do?

(c) Is the following algorithm wait-free? Explain your answer.

```
while( !test_and_set( lock_var ) ) { /* Spin */ }
count++;
lock_var = 0;
```

(d) What are the tradeoffs of function inlining?

(e) You host your server on a cloud computing service. You notice it is running slowly; you need a bigger instance. How would you decide kind of instance you need?

(f) In our discussion of scaling to 10 million users (using AWS), the larger instances show a Write Master database and Read Replicas. Explain briefly why.

(g) How might the compiler decide that automatic parallelization of a loop is "not profitable"? Explain your answer.

Please initial:

(h) Consider the two OpenMP-annotated for loops below that use the `ordered` directive. Explain the difference between the two.

```
#pragma omp parallel for
for ( int i = 0; i < max; ++i ) {
  int id = do_something_useful();
  #pragma omp ordered
  printf("Completed item %d\n", id);
}
```

```
#pragma omp parallel for ordered
for ( int i = 0; i < max; ++i ) {
  int id = do_something_useful();
  printf("Completed item %d\n", id);
}
```

(i) Complete the following code to register the cancellation handler to perform cleanup as appropriate:

```
void * foo_thr( void * arg ) {


  struct bar* b = malloc(sizeof( struct bar ));


  /* Initialize b and do something with it */



  pthread_exit( b );
}
```

```
void foo_handler( void * ptr ) {



}
```

(j) There is a compiler optimization called *rematerialization*. This has nothing to do with Star Trek, but it means the compiler chooses to recompute a value instead of reading it from memory. Would this ever be a good idea? Why or why not?

2 Not-As-Short Answer [21 marks total]

2.1 Memory Consistency [5 Marks]

Consider the following pseudocode:

```
// assume x = y = 0

thread 1:                                    thread 2:
store 1 to x
load y into register r1
```

Write down pseudocode for thread 2 which accesses `x` and `y`. Your code must be able to generate final values for `x` and `y` which are not allowed under sequential consistency. Write down these final values, explain how they arise under a memory model that is weaker than sequential consistency, and explain how your final values could not happen under sequential consistency.

Please initial:

2.2 Automatic Parallelization [5 marks]

We've discussed the use of the `restrict` keyword as a way to convince the compiler that it is safe to do automatic parallelization of a given for loop, such as the one below:

```
void add_arrays( int * restrict c, int * restrict a, int * restrict b, int length ) {
  for( int i = 0; i < length; ++i ) {
    c[i] = a[i] + b[i];
  }
}
```

Draw a diagram showing the steps to produce an error that occurs if the user lies to the compiler and violates the terms of `restrict` when calling `add_arrays`, but does not occur if the sequential version runs.

2.3 Cache Coherence [5 marks]

In lectures we discussed cache coherence with the MESI protocol (Modified/Exclusive/Shared/Invalid). Suppose we have 2 CPUs in the system, each with its own cache. Suppose there are data blocks `x` and `y` in the system and that cache has space for exactly two blocks. All cache lines are initially in the "I" (invalid) state. The two CPUs issue the following instructions, in this order:

1. CPU0 reads `y`.
2. CPU1 writes `x`.
3. CPU0 writes `y`.
4. CPU1 reads `y`.
5. CPU1 writes `x`.

Complete the table below to reflect the MESI state of the cache of each CPU after each step in the above list has taken place. Use the letters M, E, S, and I to indicate the states of Modified, Exclusive, Shared, and Invalid, respectively. The actual values of `x` and `y` read and written are not relevant.

	CPU 0		**CPU 1**	
Step	**x**	**y**	**x**	**y**
0.	I	I	I	I
1.				
2.				
3.				
4.				
5.				

2.4 Profiler-Guided Optimization [6 marks]

As discussed in the lecture on the topic, many well-known programs use Profiler-Guided Optimization (POGO), including web browsers. Training data is used to optimize the code. Give three (3) different examples of test scenarios you would suggest for inclusion in the generation of training data for Mozilla Firefox, and justify your selections.

3 Nonblocking I/O [19 marks]

If you think you are tired of accreditation-themed questions, imagine how Professors Wright and Aagaard feel! This time, we are in a parallel universe where Spock has a goatee and the threads/OpenMP versions of the accreditation tool have not been built. What we are going to create instead is a non-blocking I/O version of the solution. The simple implementation of the program is:

```
course_term* get_next( );
double query_db( course_term* ct );
double total = 0.0;

int main( int argc, char** argv ) {
  while( 1 ) {
    course_term* next = get_next( );
    if ( next == NULL ) {
      break;
    }
    total += query_db( next );
    free( next );
  }
  printf( "Total AUs: %g\n.", total );
  return 0;
}
```

To make this work using non-blocking I/O, you will use curl multi, as in assignment 1. You should get all the `course_term` structures to be queried and turn them into easy handles (using the `convert` function as specified below), and put them in the curl multi handle. Once they are all ready, you can start the requests. Once all requests have been started, use a loop to check the number that are still running, using the `wait_for_curl` function (also specified below), in the body of the loop. Once all requests are finished, the provided part of `main()` to check the results and clean up the easy handle should run. After that, you can print out the total AUs, clean up, and exit.

Complete the code for `callback` and `main` below to implement the desired behaviour using non-blocking I/O. Your code should not have any race conditions or memory leaks.

```
/* Global variables here */
CURLM* cm;
double total = 0.0;









/* This function takes a pointer to a cURL multi handle
     and waits until something happens. Use this
     function in the body of a loop that is waiting for
     handles to complete. Implementation not shown for
     space reasons. */
void wait_for_curl( CURLM* cm );

/* This function takes a pointer to a course_term
     structure and creates and initializes a cURL easy
     handle with the data it needs to complete the
     request. The easy handle will be set up to call
     the callback function callback() when data is
     received. After this function has run, course_term
     structure is no longer needed. */
CURL* convert( course_term* ct );


/* This function will be run as the callback on a curl
     easy handle */
size_t callback(char *d, size_t n, size_t l, void *p) {
   /* Parses cURL response to extract answer as double
        */
   double aus = parse_response( d, n*l );









   return n*l;
}
```

```
int main( int argc, char** argv ) {
    curl_global_init( CURL_GLOBAL_ALL );









    /* All requests finished, now check they are OK and
         clean up easy handles */
    int left = 0;
    CURLMsg *m = NULL;
    while (( m = curl_multi_info_read( cm, &left ))) {
        if ( m->msg == CURLMSG_DONE ) {
            CURL* eh = m->easy_handle;

            CURLcode return_code = m->data.result;
            if ( return_code != CURLE_OK ) {
                printf("Error %d\n", m->data.result );
                continue;
            }
            curl_multi_remove_handle( cm, eh );
            curl_easy_cleanup( eh );
        } else {
            printf("Error %d\n", m->msg );
        }
    }
    /* All requests finished/callbacks executed */









    curl_global_cleanup();
    return 0;
}
```


Please initial:

4 Locking and Synchronization [25 marks]

You are implementing a cache for `item` structures. You do not need to know any of the details of the `item` structure, except that it has an `unsigned int` field `id` which we can use to identify the item when needed. The cache may be used multiple times in the same program, so your implementation needs to be thread-safe and cannot use global variables. The cache will use the replacement strategy of first-in-first-out. For performance reasons, it will use reader-writer locks (`pthread_rwlock_t`).

The following operations are supported: `itemcache_search`, `itemcache_put`, `itemcache_inv` (invalidate), and `itemcache_clear`. See the comments above each of these function to see what the function takes as arguments, what it returns, and what it is supposed to do. Remember that you can copy an item into the array using a simple assignment operator. Complete the C code below to implement the behaviour of each function as specified. Your code should not have any race conditions or memory leaks.

```
typdef struct {
  int replace_index;
  int max_size;
  item* array;
  pthread_rwlock_t lock;
  /* You can add more fields if you need */




} itemcache;

/* Create a copy of the given item; returns a pointer
     to it. */
item* copy_item( item* i ) {
  /* Implementation not shown */
}

/* Initialize the itemcache structure with the array
     the size of cache_size items */
void itemcache_init( itemcache *c, int cache_size ) {









}

/* Cleanup and deallocate anything allocated or created
     in the init function */
void itemcache_destroy( itemcache *c ) {









}

/* Return a pointer to a new copy of the item with the
     specified ID, if it is found in the cache. Use the
      copy function above to make a copy of the item
     in question; it allocates the memory for the copy.
      Deallocation of the copy is the responsibility of
      the caller of itemcache_search.
   If no item with the id is found in the cache, return
        NULL.
*/
item* itemcache_search( unsigned int id ) {









}
```

```
/* Put a new item in the cache. If an item with the
     same id is already in the cache, then it should
     replace the older version. If the item is not
     already in the cache, then replace using the above
     -specified replacement strategy of FIFO.
*/
void itemcache_put( itemcache *c, item * i ) {









}

/* Invalidate erases the item with the specified ID, if
      that item exists in the cache. If it is not found
     , do nothing. An item is removed by overwriting it
      with 0s.
*/
void itemcache_inv( itemcache *c, unsigned int id ) {









}

/* Invalidate all items in the cache (please do so
     efficiently, i.e. not by calling invalidate N
     times) */
void itemcache_clear( itemcache *c ) {









}
```


Please initial:

5 Rust or Queueing Theory [10 marks]

You choose which of 5.1 or 5.2 we will mark. We will mark the one you answer. If both have something written, we will mark the first. If you change your mind, cross out the one you don't want marked.

5.1 Rust

Consider this broken partial implementation of a C++ `Box` class.

```
1  template<typename T> class Box  {
2      T* storage = nullptr;
3      void deepCopy(const Box &other) {
4          delete storage;
5          storage = nullptr;
6          if (other.storage) {
7              T newStorage;
8              storage = &newStorage;
9              *storage = *other.storage;
10         }
11     }
12 }
```

(a) (2 points) What is wrong with this C++ `deepCopy` implementation?

(b) (5 points) In Rust you would call `other.storage.clone()` instead of what we do on lines 7–9[1]. Provide a function signature for a Rust `deepCopy` implementation; we're looking for the correct types, not syntax. (We didn't discuss member functions but it's like the Rust functions seen in class). Discuss the ownership situation for the `other` parameter in your Rust `deepCopy` before, during, and after this function call; consider refs and mutability.

(c) (3 points) You would clone `storage` in your Rust `deepCopy` implementation. Rust doesn't have a `delete` operator; when would the old `storage` get freed in a Rust `deepCopy` implementation? Which variable owns the new `storage`, and when would that `storage` get dropped?

5.2 Queueing Theory

We have two backend servers. Each server completes a request, on average, in 10 ms; the time to complete requests is exponentially distributed. Jobs arrive at a single central queue and are processed FIFO. On average, 180 000 jobs arrive per hour in a Markov process. For this system, write down/calculate the values below:

Property	**Value**
Kendall Notation Description	
Utilization (ρ)	
Intermediate Value (K)	
Completion time average (T_q)	
Average queue length (W)	
Maximum arrivals/hour the system can handle (full utilization)	
Maximum average service time the system can tolerate	

[1] You'd also use an `Option` for `storage` but that doesn't seem to come up in this question.

Please initial:

6 We Built This City on OpenCL [15 marks]

The city of Bielefeld wants to decide where to place segregated bike lanes. You are given a 2-dimensional array P, where each element is a grid square recording the popularity of that part of the city. The city is hilly; you are given a 4-dimensional array A such that $A[x_0][y_0][dx][dy]$ represents the difficulty of going from (x_0, y_0) to $(x_0 + dx - 1, y_0 + dy - 1)$. Here, dx and dy are each in $[0, 2]$ so that we can record difficulties for adjacent squares. All numbers are `cl_floats`. Further, the elements of A are `cl_floats` between 0 and 1. Array P contains data from indices $[1][1]$ through [SIZE][SIZE] and a border of 0s around the data elements.

Your task is to use OpenCL to find the largest weighted sum

$$\begin{aligned} P[x][y] \quad &+ \quad A[x][y][1][0]P[x][y-1] + A[x][y][0][1]P[x-1][y] \\ &+ \quad A[x][y][1][2]P[x][y+1] + A[x][y][2][1]P[x+1][y]. \end{aligned}$$

We recommend using two kernels: one to compute weighted sums and one to find the maximum in the array.

Arrays P and A are available on host side but you'll need to make them available to the kernels.

Write host code and kernel code below to (1) create any buffers you will need, (2) enqueue those buffers, (3) set kernel arguments, (4) run your kernels, and (5) read the results. Your host function should return the coordinates of the winning grid square (through the C++ reference parameters).

Host Code. Complete the host code below. Complete the kernel(s) on the next page.

```
void findBestLocation( int & x_index, int & y_index ) {
  /* Other setup including the A and P arrays not shown */
  try {
    std::vector<cl::Platform> platforms;
    cl::Platform::get(&platforms);
    cl_context_properties cps[3] = {CL_CONTEXT_PLATFORM,(cl_context_properties)(platforms[0])(), 0};
    cl::Context context(CL_DEVICE_TYPE_GPU, cps);
    std::vector<cl::Device> devices = context.getInfo<CL_CONTEXT_DEVICES>();
    cl::CommandQueue queue = cl::CommandQueue(context, devices[0]);
    std::ifstream sourceFile("bikelanes.cl");
    if(!sourceFile.is_open()){
       std::cerr << "Cannot find kernel file" << std::endl;
       throw;
    }
    std::string sourceCode(std::istreambuf_iterator<char>(sourceFile), (std::istreambuf_iterator<char>()));
    cl::Program::Sources source(1, std::make_pair(sourceCode.c_str(), sourceCode.length()+1));
    cl::Program program = cl::Program(context, source);
    try {
         program.build(devices);
    } catch(cl::Error error) {
         std::cerr << program.getBuildInfo<CL_PROGRAM_BUILD_LOG>(devices[0]) << std::endl;
         throw;
    }
    // Complete the code below
```

```
 } catch(cl::Error error) {
        std::cout << error.what() << "(" << error.err() << ")" << std::endl;
 }

}
```


Please initial:

Kernel(s). Write down your kernel(s) below.

Extra Space. Use this space if needed. Be sure to indicate on the original question to refer to this extra space.

Please initial:

OpenCL Reference

The following constructors may be helpful:

```
cl::Buffer (const Context &context, cl_mem_flags flags,::size_t size, void *host_ptr=NULL, cl_int *err=NULL)
cl::NDRange ()
cl::NDRange (::size_t size0)
cl::NDRange (::size_t size0,::size_t size1)
cl::NDRange (::size_t size0,::size_t size1,::size_t size2)
```

Kernel arguments can be set with set with:

```
void setArg( int argumentIndex, cl::Buffer buffer )
```

Note this is not actually the signature but you can use this and it works.

The following functions can be invoked on an instance of `cl::CommandQueue`:

```
cl_int  enqueueReadBuffer (const Buffer &buffer, cl_bool blocking,::size_t offset,::size_t size, void *ptr, const
     VECTOR_CLASS< Event > *events=NULL, Event *event=NULL) const

cl_int  enqueueWriteBuffer (const Buffer &buffer, cl_bool blocking,::size_t offset,::size_t size, const void *ptr
    , const VECTOR_CLASS< Event > *events=NULL, Event *event=NULL) const

cl_int  enqueueCopyBuffer (const Buffer &src, const Buffer &dst,::size_t src_offset,::size_t dst_offset,::size_t
    size, const VECTOR_CLASS< Event > *events=NULL, Event *event=NULL) const

cl_int  enqueueReadImage (const Image &image, cl_bool blocking, const size_t< 3 > &origin, const size_t< 3 > &
    region,::size_t row_pitch,::size_t slice_pitch, void *ptr, const VECTOR_CLASS< Event > *events=NULL, Event *
    event=NULL) const

cl_int  enqueueWriteImage (const Image &image, cl_bool blocking, const size_t< 3 > &origin, const size_t< 3 > &
    region,::size_t row_pitch,::size_t slice_pitch, void *ptr, const VECTOR_CLASS< Event > *events=NULL, Event *
    event=NULL) const

cl_int  enqueueCopyImage (const Image &src, const Image &dst, const size_t< 3 > &src_origin, const size_t< 3 > &
    dst_origin, const size_t< 3 > &region, const VECTOR_CLASS< Event > *events=NULL, Event *event=NULL) const

cl_int  enqueueCopyImageToBuffer (const Image &src, const Buffer &dst, const size_t< 3 > &src_origin, const
    size_t< 3 > &region,::size_t dst_offset, const VECTOR_CLASS< Event > *events=NULL, Event *event=NULL) const

cl_int  enqueueCopyBufferToImage (const Buffer &src, const Image &dst,::size_t src_offset, const size_t< 3 > &
    dst_origin, const size_t< 3 > &region, const VECTOR_CLASS< Event > *events=NULL, Event *event=NULL) const

void *  enqueueMapBuffer (const Buffer &buffer, cl_bool blocking, cl_map_flags flags,::size_t offset,::size_t
    size, const VECTOR_CLASS< Event > *events=NULL, Event *event=NULL, cl_int *err=NULL) const

void *  enqueueMapImage (const Image &buffer, cl_bool blocking, cl_map_flags flags, const size_t< 3 > &origin,
    const size_t< 3 > &region,::size_t *row_pitch,::size_t *slice_pitch, const VECTOR_CLASS< Event > *events=
    NULL, Event *event=NULL, cl_int *err=NULL) const

cl_int  enqueueUnmapMemObject (const Memory &memory, void *mapped_ptr, const VECTOR_CLASS< Event > *events=NULL,
    Event *event=NULL) const

cl_int  enqueueNDRangeKernel (const Kernel &kernel, const NDRange &offset, const NDRange &global, const NDRange &
    local, const VECTOR_CLASS< Event > *events=NULL, Event *event=NULL) const

cl_int  enqueueTask (const Kernel &kernel, const VECTOR_CLASS< Event > *events=NULL, Event *event=NULL) const

cl_int  enqueueNativeKernel (void(CL_CALLBACK *userFptr)(void *), std::pair< void *,::size_t > args, const
    VECTOR_CLASS< Memory > *mem_objects=NULL, const VECTOR_CLASS< const void * > *mem_locs=NULL, const
    VECTOR_CLASS< Event > *events=NULL, Event *event=NULL) const

cl_int  enqueueMarker (Event *event=NULL) const

cl_int  enqueueWaitForEvents (const VECTOR_CLASS< Event > &events) const

cl_int  enqueueAcquireGLObjects (const VECTOR_CLASS< Memory > *mem_objects=NULL, const VECTOR_CLASS< Event > *
    events=NULL, Event *event=NULL) const

cl_int  enqueueReleaseGLObjects (const VECTOR_CLASS< Memory > *mem_objects=NULL, const VECTOR_CLASS< Event > *
    events=NULL, Event *event=NULL) const

cl_int  enqueueBarrier () const

cl_int  flush () const

cl_int  finish () const
```


Please initial: