Instructions:

1. This exam is open book, open notes.
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 interpreted as an academic offence.
3. Place all bags at the front or side of the room, or beneath your table, such that they are inaccessible.
4. There are five (5) questions. Not all are equally difficult.
5. The exam lasts **120** minutes and there are 100 marks.
6. Verify that your name and student ID number is on the cover page and that your examination code appears on the bottom of each page of the examination booklet.
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		20
2		14
3		26
4		20
5		20
Total		**100**

Question 1: Short Answer [20 marks]

Answer these questions using at most three sentences. Each is worth 2 marks.

(a) Give an example, other than if-else blocks or switch-case prediction, of an optimization possible with Profiler-Guided-Optimization (POGO).

(b) In Assignment 2, the output file `image.ppm` is slightly different when the compiler option `-fast` is used. This is equivalent to `-ffast-math` in `gcc`. Explain what this option does to make the output different.

(c) Normally when there are nested for loops we choose to parallelize the outermost loops. Give an example of a situation where it would be sensible to parallelize the inner loop.

(d) Explain the semantics of the OpenMP Directive `#pragma omp master`.

(e) Give an example of a situation where a profiler would give incorrect results about which instructions are costly.

(f) Why do heap profilers, such as Massif, record peak snapshots when memory is deallocated rather than when it is allocated?

(g) Write a brief example of code where Helgrind would report a lock ordering problem.

(h) Suppose we have a M/M/1 server situation where the service time s is 3 seconds and the arrival rate is λ is 0.25. Calculate the average completion time T_q for this system.

(i) Suppose `g(x)` changes `x` but `f(x)` does not, in the code below:

```
int y = f(x);
int z = g(x);
```

How would you change this so the WAR dependency is eliminated and these can be run in parallel?

(j) Why does it often make sense to put a `pthread_exit` call at the end of `main` in a program that uses pthreads?

Question 2: Harry Potter and the OpenMP House Elves [14 marks]

At Hogwarts School of Witchcraft and Wizardry, cooking and cleaning jobs are still done by House Elves, magical creatures who are not covered under any sort of employment standards or workers rights legislation, despite Hermione's best efforts to do something about that. Much as before, the elves still have to cook one feast per day and get supplies once a day, and while they are waiting to do these two tasks, they clean the school grounds.

This year at Hogwarts, however, these are OpenMP House Elves, which is to say they require a lot less ceremony before they can get on with their work. They are also much better organized because they have found a way to make Dobby (the main thread) pitch in and not just lie around until the day's work is done.

Assume the following function prototypes match to functions that implement the appropriate functionality (even though the implementations are not shown). The parameter `id` is the numerical identifier of the elf doing the task.

```
void cook_feast( int id );
void get_supplies( int id );
void clean_school( );
```

Consider the program below; it contains OpenMP specific functions and `structs` (most of which conveniently start with `omp_...`) as well as directives. Complete the table below the code to fill in for each OpenMP element, what it does and its pthread equivalent (if any exists).

```
1  #include <stdlib.h>
2  #include <stdio.h>
3  #include <stdbool.h>
4  #include <unistd.h>
5  #include <omp.h>
6
7  omp_lock_t cook_lock;
8  omp_lock_t supplies_lock;
9
10 void elf( int id ) {
11   bool cooked = false;
12   bool supplies = false;
13
14   while (!cooked || !supplies) {
15
16     if ( !cooked && 0 != omp_test_lock( &cook_lock ) ) {
17       cook_feast( id );
18       omp_unset_lock( &cook_lock );
19       cooked = true;
20     }
21     if ( !supplies && 0 != omp_test_lock( &supplies_lock ) ) {
22       get_supplies( id );
23       omp_unset_lock( &supplies_lock );
24       supplies = true;
25     }
26     clean_school( );
27   }
28 }

30 int main( int argc, char** argv ) {
31
32   omp_init_lock( &cook_lock );
33   omp_init_lock( &supplies_lock );
34
35   int id;
36   #pragma omp parallel private(id)
37   {
38     id = omp_get_thread_num();
39     elf( id );
40     printf("You␣are␣dismissed,␣elf␣%d\n", id);
41   }
42
43   omp_destroy_lock( &cook_lock );
44   omp_destroy_lock( &supplies_lock );
45   return 0;
46 }
```

Line #	Code	What it does	pthread equivalent
7, 8	`omp_lock_t`		
16, 21	`omp_test_lock`		
18, 23	`omp_unset_lock`		
32, 33	`omp_init_lock`		
36	`#pragma omp parallel private(id)`		
38	`omp_get_thread_num`		
43, 44	`omp_destroy_lock`		

Question 3: $\sqrt{-1}\ 2^3\ \Sigma$ OpenCL π [26 marks]

Consider the following code that computes the value of pi (π) using a monte carlo method: random points are generated and we count the number of those points that are in the first quadrant of the circle, and then multiply that by 4 to get an estimate for pi. The math checks out. OpenCL does not support the `rand` function and populating the initial vector with random numbers becomes a bottleneck. So there's a "random" function provided in this code which can be ported to OpenCL. See the simple C code below:

```
#include <stdlib.h>
#include <stdio.h>

#define INT_MAX 4294967296

/* Intel rand function OK for OpenCL */
unsigned int random( unsigned int x ) {
  unsigned int value = x;
  value = (value ^ 61) ^ (value>>16);
  value *= 9;
  value ^= value << 4;
  value *= 0x27d4eb2d;
  value ^= value >> 15;
  return value;
}
```

```
/* Dartmouth code modified for this question */
int main(int argc, char** argv){
  int iterations = atoi( argv[1] );
  int count = 0; /* # in quadrant 1 of unit circle */

  for ( int i = 0; i < iterations; i++ ) {
    float x = (float)random( i ) / INT_MAX;
    float y = (float)random( i ^ random( i )) / INT_MAX;
    if ( (x*x + y*y) <= 1) {
      count++;
    }
  }
  float pi = (float)count / iterations * 4;
  printf("Estimate of pi is %g \n", pi);
  return 0;
}
```

This is an easily parallelizable task and can be converted to use OpenCL for significant speedup. There are two parts to this, as you will recall from the OpenCL assignments, the host code (C++) and the kernel code (in the OpenCL C-like language). In this question you will do both parts (and you may find it easier to write the kernel first, but it's up to you). You will need to write both parts and you must write them such that they work together.

Part 1: The Host Code [18 marks]. Complete the host code below to make this program function as expected. The provided code is a modification of the assignment 3 starter code. The comment lines tell you what steps are needed to accomplish the setup, launch, and collecting the results of the OpenCL kernel. Attached to the end of the exam is an OpenCL Reference that gives you the function signatures that are relevant to completing the tasks the comments ask you to do.

```
#define __CL_ENABLE_EXCEPTIONS
#include <CL/cl.hpp>
#include <iostream>
#include <fstream>
#include <string>
#include <utility>
#include <vector>

int main(int argc, char** argv) {

    int iterations = atoi( argv[1] );
    cl_int* results = (int*) malloc( iterations * sizeof ( cl_int ) );
    try {
        // Get available platforms
        std::vector<cl::Platform> platforms;
        cl::Platform::get(&platforms);

        // Select the default platform and create a context using this platform and the GPU
        cl_context_properties cps[3] = {
            CL_CONTEXT_PLATFORM,
            (cl_context_properties)(platforms[0])(),
            0
        };
        cl::Context context(CL_DEVICE_TYPE_GPU, cps);

        // Get a list of devices on this platform
        std::vector<cl::Device> devices = context.getInfo<CL_CONTEXT_DEVICES>();

        // Create a command queue and use the first device
        cl::CommandQueue queue = cl::CommandQueue(context, devices[0]);

        // Read source file
        std::ifstream sourceFile("montepi_kernel.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));

        // Make program of the source code in the context
        cl::Program program = cl::Program(context, source);

        // Build program for these specific devices
        try {
            program.build(devices);
        } catch(cl::Error error) {
            std::cerr << program.getBuildInfo<CL_PROGRAM_BUILD_LOG>(devices[0]) << std::endl;
            throw;
        }
```

```
        // Make kernel
        cl::Kernel kernel(program, "montepi");

        // Create buffers









        // Write buffers











        // Set arguments to kernel











        // Run the kernel on specific N-D range











        // Read buffer















        cl_int sum = 0;
        #pragma omp parallel for reduction(+:sum)
        for (int i = 0; i < iterations; i++) {
            sum += results[i];
        }

        float pi = (float) sum / iterations * 4;

        std::cout << "Estimate␣of␣pi␣is␣" << pi << std::endl;

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

Part 2: Creating the Kernel [8 marks] Create the file `montepi_kernel.cl` file by writing the OpenCL kernel function (which must be named `montepi`) below. Make it as optimal as possible (loops are bad, conditionals are less bad but still not great).

Remember to prefix the function with `__kernel` and your buffer arguments with `__global` and any integers with the `const` prefix. Assume it is set up and called appropriately from the host code, but you should match your arguments and buffers with what you wrote in the host code. This is written in the OpenCL language which is C-like, but without some features. Recall also the function `get_global_id( int dimension )`. The random number function has been moved here for you since it is a pain to rewrite.

```
uint random( uint x ) {
    uint value = x;
    value = (value ^ 61) ^ (value>>16);
    value *= 9;
    value ^= value << 4;
    value *= 0x27d4eb2d;
    value ^= value >> 15;
    return value;
}
```

Question 4: Swap Meet! [20 marks]

During a code review, you find the following block of code in a program you are reading. Assume the locks are declared, created, and initialized properly elsewhere in the program. Assume also the locks are used only for this part of the program.

```
void swap( container_t * x, container_t * y ) {
    pthread_mutex_lock( &(x->lock) );
    data_t temp = x->data;
    pthread_mutex_lock( &(y->lock) );
    x->data = y->data;
    pthread_mutex_unlock( &(x->lock) );
    y->data = temp;
    pthread_mutex_unlock( &(y->lock) );
}
```

Part 1: Finding the Problem Identify two ways this function can be called that would lead to deadlock. Show only the function calls that would cause the problem.

Part 2: Fixing the Problem Modify this function such that both problems are solved. Your solution should still use fine-grained locking and should be thread safe. That is, you should not introduce any new deadlocks or race conditions.

Question 5: Queuing Theory & Scalability [20 marks total]

5A: Queueing For Performance [14 marks]

The reason we learned about queueing theory was to take stock of our system and make meaningful predictions about its capabilities. In this question, we are going to assess the performance of a database server processing large amounts of data (22 500 transactions per hour). Complete the table below and compute the three major values below the table.

Device	**Data/Hour**	λ	S	V	ρ	$V \times S$
Transactions	22 500					
CPU					30%	
RAM Disk	765 000		2ms			
Solid State Drive	945 000		3ms			
Network	11 250		25ms			

Bottleneck Device:

Maximum Rate of transactions (saturation):

Average transaction time:

5B: Measure Thrice, Compile Once [6 marks]

Three TAs, Alex, Taylor, and Jordan, are having an argument about how to evaluate assignment 4. Everyone agrees that 1 000 000 jobs is a reasonable total number of jobs, but they disagree after that:

- Alex thinks a single run with the jobs parameter set to 1 000 000 is correct.
- Taylor thinks the right approach is to perform 20 runs of 50 000 jobs each and average those results.
- Jordan says it would be better to do 5 000 runs of 200 jobs each and average those results.

Which of the TAs, if any, are right? Explain your reasoning.

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
```