Instructions:

1. This exam is open book, open notes, calculators permitted (with no communication capability).
2. Turn off all communication devices.
3. There are five (5) questions, some with multiple parts. Not all are equally difficult.
4. The exam lasts 150 minutes and there are 100 marks.
5. 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.
6. Do not fail this city.
7. 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		22
3		22
4		14
5		22
Total		**50**

Question 1: Short Answer [20 marks]

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

(a) What are the tradeoffs of function inlining?

(b) What is the difference between an open system and a closed system (per our discussion of queueing theory)?

(c) What is the motivation for creating a thread pool?

(d) Name two advantages of using non-blocking I/O, rather than threads.

(e) What are the tradeoffs of coarse-grained vs. fine-grained locks?

(f) How does the use of trylock synchronization primitives help prevent deadlock?

(g) Write a (simple) C function that is not reentrant.

(h) Suppose you are doing parallel execution of a mathematical function that has certain points that take a long time to evaluate. Your program terminates slow running threads “early”, leaving some points missing. Explain how you could fill in those missing points (and what assumptions are required for this to make sense).

(i) Explain conceptually the difference between *arrival rate* and *throughput*.

(j) Explain the purpose of the `mfence` instruction.

Question 2: Parallelism [22 marks total]

2A: OpenMP Dynamic Scheduling [4 marks]

Recall that in OpenMP you can request dynamic scheduling, where threads pick up work when they have nothing to do, rather than work being assigned upfront. Write a loop where the directive `#pragma omp parallel for schedule(dynamic)` would give better performance than not using a dynamic schedule.

2B: OpenMP Directives [8 marks]

Consider the code (from an online OpenMP tutorial) below which contains a number of OpenMP directives (in the form `#pragma omp...`). Next to each OpenMP directive, clearly and concisely describe what it does.

```
int main (int argc, char *argv[])  {

  int nthreads, tid, i;
  float a[N], b[N];
  omp_lock_t locka, lockb;
  omp_init_lock(&locka);
  omp_init_lock(&lockb);

  #pragma omp parallel shared(a, b, nthreads, locka, lockb) private(tid)
  {
  tid = omp_get_thread_num();
  #pragma omp master
    {
      nthreads = omp_get_num_threads();
      printf("Number of threads = %d\n", nthreads);
    }
  printf("Thread %d starting...\n", tid);
  #pragma omp barrier

  #pragma omp sections nowait
    {

    #pragma omp section
      {
        omp_set_lock(&locka);
        for (i=0; i<N; i++)
          a[i] = i * DELTA;
        omp_unset_lock(&locka);
        omp_set_lock(&lockb);
        for (i=0; i<N; i++)
          b[i] += a[i];
        omp_unset_lock(&lockb);
      }

    #pragma omp section
      {
        omp_set_lock(&lockb);
        for (i=0; i<N; i++)
          b[i] = i * PI;
        omp_unset_lock(&lockb);
        omp_set_lock(&locka);
        for (i=0; i<N; i++)
          a[i] += b[i];
        omp_unset_lock(&locka);
      }
    }
  }
}
```

2C: Speculative Execution: `throw new OptimisticProgrammerException();` [10 marks]

You come across this C code in your program:

```
void post_process( void* message ) {
  int error_code;
  error_code = store_message( message );
  if ( 0 == error_code ) {
    update_statistics( message );
  }
}
```

You know that storing the message takes a long time, and updating the statistics takes much longer. Therefore, you would like to parallelize this by doing speculative execution: you will create a pthread for each of those functions. Your code should follow this pattern:

Create and start a thread for `store_message` and one for `update_statistics`. Collect the results from the first thread. If the return value from `store_message` is anything other than zero, then terminate (with `pthread_kill` and signal 9) the statistics update thread, then re-run the statistics update function. If the return value of the storage thread was zero, simply wait for the second thread's completion. Hint: you will need a simple "wrapper" function for each function.

For convenience, some of the pthread functions available for your use (and remember you can give `NULL` for the attributes and return values):

```
pthread_create( pthread_t *thread, const pthread_attr_t *attributes,
                void *(*start_routine)( void * ), void *argument )
pthread_join( pthread_t thread, void **return_value )
pthread_cancel( pthread_t thread )
pthread_kill( pthread_t thread, int signal )
pthread_exit( void *return_value )
```

Question 3: OpenCL [22 marks total]

3A: GPU Password Cracking [6 marks]

Pretend for the purposes of this question that you have a legitimate reason to crack a password. All you have is the output (hashed value) and you know the password hash function is SHA256. (In real life there will also be the salt, but ignore that for the purposes of this question.) Describe, using words, how you would use OpenCL to crack this password in parallel. Be specific about what work you would do in host code, what you would transfer to the kernel via buffers, and what the kernel code would do. Please be concise.

3B: OpenCL Matrix Multiplication [16 marks]

In this question, you will do matrix multiplication using OpenCL. Multiplying matrix A times B to produce matrix C in some sequential C code looks like this, found in an OpenCL lecture at the University of Bristol (thanks Google):

```
void mat_mul(int Mdim, int Ndim, int Pdim, float *A, float *B, float *C) {
 int i, j, k;
  for (i=0; i< Ndim; i++){
    for (j=0; j< Mdim; j++){
      for (k=0; k< Pdim; k++) { // C(i,j) = sum(over k) A(i,k) * B(k,j)
         C[i*Ndim+j] += A[i*Ndim+k] * B[k*Pdim+j];
       }
    }
  }
}
```

Assume that the matrices have been appropriately allocated and initialized for your code. Now you are going to convert this to an OpenCL kernel.

Part 1 (6 marks) For each pointer parameter to the function, you need a buffer in the host code. For each buffer, indicate whether it should read-only, write-only, or read/write. (You don't need them for the integer arguments.)

Part 2 (10 marks) Write the OpenCL kernel implementation below. It does not need to be optimal, but as a hint, you should have at most one `for` loop in your code. Remember to prefix the function with `__kernel` and your buffer arguments with `__global` or and the integers with the `const` prefix. Assume it is set up and called appropriately from the host code. Recall also the function `get_global_id( int dimension )`, and that this is a 2-dimensional problem!

Question 4: Profiling [14 marks]

4A: Profiler Guided Optimization [6 marks]

Explain the steps for Profiler Guided Optimization (POGO). Your explanation should use an example with a simple code fragment in C/C++ to explain where POGO would be applied and what the benefit of it would be.

4B: Profiler Inaccuracies [2 marks]

Sometimes a profiler like oprofile reports samples at the very start or end of a function, even though there are no executable statements there. Explain what is executing at the time of the sample to cause the sample to be assigned to the start or end of the function.

4C: Profiling the Impossible [6 marks]

Imagine you are a student of ECE 459 and the evil, mean professor has given you assignment 4 code. He wants you to find a speedup of 2.0, but you are pretty sure this is impossible. How could you use profiling to gather evidence that a 2.0 speedup impossible, and convince him that a speedup of 1.2 is much more reasonable (though still difficult)?

Question 5: Scalability & Queuing Theory [22 marks total]

5A: Fly the Friendly Queues [4 marks]

Queuesville airlines knows that on average, 5% of the people who have made a reservation for a given flight do not show up (for whatever reasons). Their model for this assumes that each person independently does not show up with a probability of 5%. Airlines hate having an empty seat on the plane, because that's lost revenue they could be earning. They would much rather have someone "bumped" from the flight and put on a later one (because they hate passengers?). Thus, they oversell the flight (that is, sell more tickets than seats) in the expectation that some people will not show up and all will work out in the end.

Part 1 (1 mark) What probability distribution describes the airline's model of passenger behaviour?

Part 2 (1 mark) What probability distribution should you use to decide how many seats to sell for a given flight?

Part 3 (2 marks) If the plane in question has 50 seats, how many tickets should Queuesville Airlines sell for this flight? If your answer is fractional, round down to an integer value.

5B: M/M/1 Server [8 marks]

Suppose we have a server that completes a request, on average, in 4 ms, and the time to complete the requests is exponentially distributed. In a typical hour, 500 000 jobs arrive. For this system, calculate the following values:

Property	Value
Utilization (ρ)	
Completion time average (T_q)	
Average queue length (W)	
Maximum arrivals per second the system can handle (full utilization)	

Now suppose the system receives a hardware upgrade so that the average completion time of a request decreases from 4 ms to 3 ms. None of the other properties of the system change. For the updated system, calculate the following values:

Property	Value
Utilization (ρ)	
Completion time average (T_q)	
Average queue length (W)	
Maximum arrivals per second the system can handle (full utilization)	

5C: Total Throughput [4 marks]

Banks run reconciliation of transactions (e.g., foreign currency exchange) at night as a batch job (closed system). To vastly oversimplify how this works, let us pretend the bank does this as two consecutive activities: (1) `sumTransactions` – add up all the transactions for each currency, whether buying or selling, and (2) `performExchange` – buy/sell the appropriate currencies on the market to balance out their books. Imagine a typical day has 25 000 such transactions. Summing transactions takes 8 minutes and 9 seconds; performing the exchange purchases/sales on the foreign exchange market takes 58 seconds.

Calculate the throughput (X_n) for each of the two activities, and then the combined throughput, X_0 per the formula below. Report your answers to two decimal places.

$$X_0 = \frac{X_1 \times X_2}{X_1 + X_2}$$

5D: Database Replication [6 marks]

In the video about practical scalability with Amazon AWS, the database was shown to have (1) a write master, (2) a slave, and (3) multiple read replicas. For each of those three databases, explain its function, and why it is advantageous to separate that function from the others. Your answer should be specific; "higher performance" is not adequate as an answer.