There are 6 questions, one question per page.
Please answer all questions in your exam booklet.

1 Short Answer (30 marks)

1.1 (10 marks)

Give the definition of concurrency and parallelism. What is the main programming concern for each concept?

1.2 (10 marks)

Consider the following two lines of code:

```
data = result1 + 2;
data = result2 + 2;
```

We want these two lines to run in parallel. Identify the variable that is involved in a dependency and the type of dependency. Show how we could modify the code to run the lines in parallel.

1.3 (10 marks)

Consider the following code:

```
#pragma omp parallel for
for ( i = 0 ; i < N; ++i ) {
    #pragma omp flush(max)
    if (arr[i] > max) {
        #pragma omp critical
        {
            if (arr[i] > max) max = arr[i];
        }
    }
}
```

Explain why we need an `omp flush(max)`. Recall that flush is a `__sync_synchronize()` call to the compiler and an `mfence` instruction to the CPU.

2 Parallel Limitations (20 marks)

We have a problem we need to solve. We always need 10 seconds of sequential setup time. After the setup, we can perfectly parallelize the rest of the code.

(5 marks) [**First scenario**] We have 4 processors and 20 seconds. What is our speedup over a sequential execution?

(5 marks) [**Second scenario**] Now we have 1 000 seconds and the same 4 processors. What is our speedup now?

(10 marks) Assume the we have a **fixed** problem size, with the same amount of work as the first scenario. How many processors do we need to get the same speedup as the second scenario?

3 Synchronization (25 marks)

Consider the following thread-safe code:

```
void swap(int* x, int* y) {
    pthread_mutex_lock(&swap_lock);
    int temp = *x;
    *x = *y;
    *y = temp;
    pthread_mutex_unlock(&swap_lock);
}
```

After profiling a threaded version of our program we discovered that the majority of execution time is spent in the `swap` function. One of your colleagues made the locking more fine-grained by adding an individual lock to each piece of data, giving the following code (assume the locks are created and destroyed properly elsewhere):

Listing 1: Modified code

```
void swap(Data* x, Data* y) {
    pthread_mutex_lock(&x->lock);
    int 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);
}
```

(10 marks) Show a serious sychronization problem with the code in **Listing 1**. Give a clear example when this problem may occur.

(10 marks) Propose a fix to this problem. Give reasonably detailed pseudocode and explain your approach in a few sentences.

(5 marks) Argue that your solution is still thread-safe.

4 Parallelization (25 marks)

Consider the following code for multiplying a matrix by a vector, which we want to parallelize:

```
1  void matVec(double **mat, double *vec, double *out, int row, int col)
2  {
3      int i, j;
4      for (i = 0; i < row; i++) {
5          out[i] = 0;
6          for (j = 0; j < col; j++) {
7              out[i] += mat[i][j] * vec[j];
8          }
9      }
10 }
```

Assume the bounds are correct and you don't have to worry about any error-checking. When we try to parallelize the code automatically we get the following error:

```
% cc -O3 -xloopinfo -xautopar fploop.c
"fploop.c", line 4: not parallelized, unsafe dependence
"fploop.c", line 6: not parallelized, unsafe dependence
```

(5 marks) Which loop should we parallelize to maximize performance? Why?

(10 marks) Parallelize the code using just a `#pragma omp parallel for`. Explicitly state the data types for each of the 5 variables (private, threadprivate, shared), don't worry about any other clauses. Argue that your parallelization is safe.

(10 marks) How could we change the code so that a compiler could automatically parallelize it? (Hint: think keywords) Why would this change allow automatic parallelization?

5 MapReduce (25 marks)

We want to leverage MapReduce to count the number of URL accesses. We have an input file consisting of multiple URLs (one entry for each access). Our goal is to have MapReduce output the number of accesses to each URL. Below is an example of the input chunks each node would receive. In this example, for the output, we would want to see that `twitter.com` had 2 accesses.

Listing 2: Input Chunk 1

```
reddit.com
facebook.com
twitter.com
```

Listing 3: Input Chunk 2

```
reddit.com
twitter.com
reddit.com
```

Listing 4: Input Chunk 3

```
reddit.com
wikipedia.org
```

(8 marks) Create a suitable mapper function. Explain the input (from the file), algorithm and output of the function (optionally explain your combiner).

(8 marks) Create a suitable reducer function. Explain the input, algorithm and output of the function.

(9 marks) Using the example provided, show the input/output of each stage of MapReduce with your functions. The reduction will be done on a single node. The final result should match the description.

6 OpenCL (25 marks)

Consider the following code for adding two matrices together, which we want to run on a GPU:

```
void matAdd(float **mat1, float **mat2, double **out, int row, int col)
{
    int i, j;
    for (i = 0; i < row; i++) {
        for (j = 0; j < col; j++) {
            out[i][j] = mat1[i][j] + mat2[i][j];
        }
    }
}
```

Don't worry about any OpenCL host code, except for the questions below.

(6 marks) Explain all the buffers you would use in your host code. For each buffer, state whether it should be read only, write only or read/write.

(4 marks) State what global range you would use when you call `enqueueNDRangeKernel`. As a reminder, the range is specificed using `NDRange(int dim0[, int dim1[, int dim2]])`.

(15 marks) Write suitable pseudocode for the kernel. Reminder: prefix the function with `__kernel` and use `__global` for any buffer arguments. Here are the built-in kernel functions you may also need to use: `get_global_id(int dim)` and `get_global_size(int dim)`.