Question 1: Short Answer

One point each. Answer these questions in your exam booklet.

(a) Two library functions for doing communication in MPI are _______ and _______.
(b) Doing more work in parallel increases a system’s _______.
(c) Modern CPUs spend most of their time waiting due to _______.
(d) When using OpenCL, you may choose to divide the computation space into _______.
(e) Gustafson’s Law says that parallelization isn’t hopeless when you can increase the _______.
(f) One condition that would impede automatic parallelization is _______.
(g) One primary design goal of DTrace was _______.
(h) A different primary design goal of DTrace was _______.
(i) An sfence instruction prevents reordering of _______.
(j) To effectively parallelize an OpenMP loop where different iterations run for different amounts of time, you want to use _______.
(k) oprofile is an example of a _______ profiler.
(l) The three steps in using profile-guided optimization are: _______.
(m) The main difference between combine and reduce in MapReduce is that _______.
(n) The obvious way to get rid of a race condition is by using _______.
(o) A technique to get rid of a WAR dependency is _______.
(p) The first thing you should do before trying to improve performance of some code is to _______ it.
(q) The term for speeding things up by doing many things at once is ________.
(r) Re-entrancy for a lock means that you can ________.
(s) Botnets are a good example of this parallelization pattern: ________.
(t) The bit of code that runs massively parallel in GPU programming is a ________.

Question 2: OpenMP

Consider the following code\(^1\).

```c
#define NV 4

/* don't worry about mind, connected */
#pragma omp parallel /* private, shared etc */
{
  my_id = omp_get_thread_num();
  nth = omp_get_num_threads();
  my_first = (my_id * NV) / nth;
  my_last = ((my_id + 1) * NV) / nth - 1;
  #pragma omp single
  {
    printf("%d: Parallel region begins with %d threads\n", my_id, nth);
    printf("\n");
  }
  printf(stdout, "%d: First= %d Last= %d\n", my_id, my_first, my_last);
  for (my_step = 1; my_step < NV; my_step++)
  {
    #pragma omp single
    {
      md = i4_huge;
      mv = -1;
    }
    find_nearest (my_first, my_last, mind, connected, &my_md, &my_mv);
    #pragma omp barrier
  }
}
```

(a) Explain what each of the OpenMP pragmas does.

(b) Assume that **OMP_NUM_THREADS** is 4 and draw a diagram explaining what the threads do.

\(^1\)http://people.sc.fsu.edu/~jburkardt/c_src/dijkstra_open_mp/dijkstra_open_mp.html.
Question 3: Reductions

We saw this example of a reduction in the notes.

```c
double sum (double *array, int length)
{
  double total = 0;
  for (int i = 0; i < length; i++)
    total += array[i];
  return total;
}
```

I mentioned that the Solaris compiler could detect that it was a reduction and parallelize it.

(a) Write down, in reasonably-detailed pseudocode, the corresponding parallelized code.

(b) Write down the assumptions that you’re making about the + operator in your parallelization.

Question 4: Memory Barriers and Consistency Models

Consider the following code; all variables are initially 0.

```
T1: x = 1; r1 = y;
T2: y = x; r2 = x;
```

Assume the architecture is not sequentially consistent.

- Show me all possible (intermediate and final) memory values and how they arise.

Question 5:

Here is some C code from meschach.

```c
double zm_norm1 (ZMAT *A)
{
  int i, j, m, n;
  Real maxval, sum;
  if (A == ZMNULL)
    error (E_NULL,”zm_norm1”);
  m = A->m; n = A->n;
  maxval = 0.0;
```
```c
for ( j = 0; j < n; j++)
{
    sum = 0.0;
    for ( i = 0; i < m; i++)
    {
        sum += zabs(A->me[i][j]);
        maxval = max(maxval,sum);
    }
}
return maxval;
```

- Describe 2 compiler optimizations that could apply to the code and the resulting code after optimization. Briefly summarize the conditions that need to hold for the optimizations to be safe; you may add qualifiers to the code if you want.

**Question 6: GPU Programming**

Here is some C code\(^2\).

```c
#define M 500
#define N 500

int i, j;
double diff;
double u[M][N], w[M][N];

diff = 0.0;
for ( i = 1; i < M - 1; i++ ) {
    for ( j = 1; j < N - 1; j++) {
        w[i][j] = ( u[i-1][j] + u[i+1][j] + u[i][j-1] + u[i][j+1] ) / 4.0;
        if ( diff < fabs ( w[i][j] - u[i][j] ) )
            diff = fabs ( w[i][j] - u[i][j] );
    }
}
```

- Express the code as one or many OpenCL kernels. (I don’t care about host code, just the kernel itself.)
- Indicate the floating-point operations in the kernel.
- Do you need to worry about synchronization?

---

\(^2\)http://people.sc.fsu.edu/~jburkardt/c_src/heated_plate/heated_plate.c