Document Title: Laboratory Sheet 7

Subject: CPP

Q1: Suppose two 100×100 matrices are to be multiplied using 8 threads. How many dot products, i.e., operations performed by the innermost for loop, must each thread compute if different approaches to parallelizing the two outermost for loops of matrix multiplication illustrated in Figure 3.6 are used?

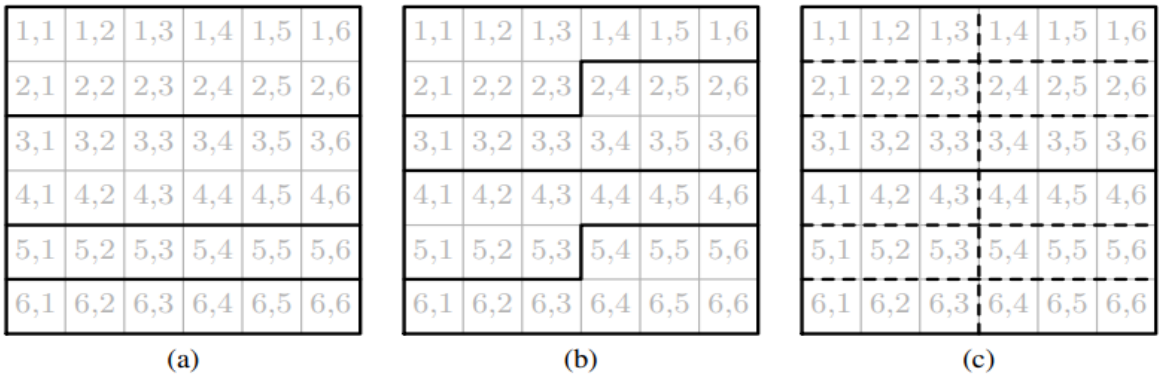

Fig. 3.6: Partition of the problem domain when all pairs of integers from 1 to 6 must be printed using 4 threads: (a) if only the outer `for` loop is parallelized, (b) if both `for` loops are parallelized together, and (c) if both `for` loops are parallelized separately.

Q2: Perform parallel implementation of the Quicksort algorithm where each recursive call is performed as a new task.

Q3: While computing the sum of all elements of sums in Listing 3.23, the program creates new threads within every iteration of the outer loop. Rewrite the code so that creation of new threads in every iteration of the outer loop is avoided.

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

int main (int argc, char *argv[]) {
  int max; sscanf (argv[1], "%d", &max);
  int ts = omp_get_max_threads ();
  if (max % ts != 0) return 1;
  int sums[ts];
  #pragma omp parallel
    {
      int t = omp_get_thread_num ();
      int lo = (max / ts) * (t + 0) + 1;
      int hi = (max / ts) * (t + 1) + 0;
      sums[t] = 0;
      for (int i = lo; i <= hi; i++)
        sums[t] = sums[t] + i;
    }
  int sum = 0;
  for (int t = 0; t < ts; t++) sum = sum + sums[t];
  printf ("%d\n", sum);
  return 0;
}
```

Listing 3.22: Implementing efficient summation of integers by hand using simple reduction.