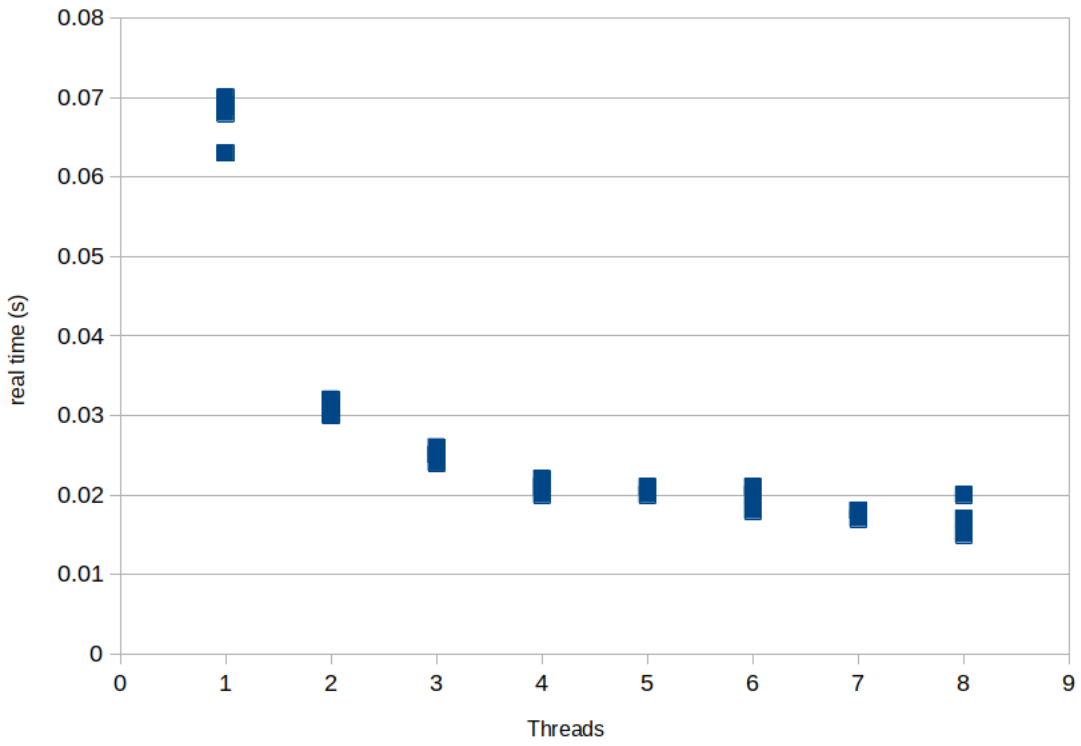
Introduction to High Performance Computing
Assignment 2

Wesley Liao
2019-09-27

1. Because the dot_prod is assigned to its current value plus the calculated vector product, this creates an opportunity for threads to use an old value of dot_prod. Code on page 2.
2. The order of the chunk execution and the printout of “Thread id: x working on index i” may change but the end result of the program will be the same.
3.
   (a) For 3 OpenMP threads, the initialization for loop will run in all 3 threads, and each omp section will execute in parallel.
   (b) For 5 OpenMP threads, the initialization for loop will run in all 5 threads, but after that there are only 3 sections, so there will be only work for 3 of the threads.
   (c) With just 1 OpenMP thread, each of the three sections will essentially be executed sequentially.
   (d) In general, there will be very little to no performance improvement when there are more threads than parallel sections.
4. Code on page 3. When compiled, p4 accepts 1 argument, an integer number of threads to run. Performance was measured using the “time” utility, running 1 through 8 threads 4 times each and plotting the resulting time reported below:



Code for part 1:

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

// compute the dot product of two vectors

int main() {
  int const N=100;
  int i;
  double a[N], b[N];
  double dot_prod = 0.0;

  int thread_id;
  //Arbitrarily initialize vectors a and b
  for (i = 0; i < N; i++) {
    a[i] = 3.14;
    b[i] = 6.67;
  }

  #pragma omp parallel private(thread_id)
  {
    thread_id = omp_get_thread_num();
    printf("This thread is: %d\n", thread_id);
    #pragma omp for
    for(i=0; i < N; i++) {
      // sum up the element-wise product of the two arrays
      #pragma omp atomic
      {
        dot_prod = dot_prod + (a[i] * b[i]);
      }
    }
  }

  printf("Dot product of the two vectors is %g\n", dot_prod);

  return 0;
}
```

Code for part 4:

```
#include <stdio.h>
#include <stdlib.h>
// link with -lm at compile time
#include <math.h>
#include <omp.h>


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

  int threads = atoi(argv[1]);
  int const N=1000;
  int i, j, ij;

  double A[N*N];
  double x[N], b[N];

  // initialize the matrix and the vector
  #pragma omp parallel for num_threads(threads)
  for (ij = 0; ij < N*N; ij++) {
    A[ij] = sin(0.01*(ij));
  }

  #pragma omp parallel for num_threads(threads)
  for (i = 0; i < N; i++) {
    b[i] = cos(0.01*i);
    x[i] = 0.0;
  }

  // matrix vector multiplication
  #pragma omp parallel for num_threads(threads)
  for (ij = 0; ij < N*N; ij++) {
    #pragma omp atomic
    x[(ij - (ij%N))/N] = x[(ij - (ij%N))/N] + (A[ij]*b[ij%N]);
  }


  printf("x[%d] = %g\n", 505, x[505]);

  return 0;
}
```