CS 4230/6230

Programming Assignment 1

Due Monday, 10/17/2022, 11:59pm

For this programming assignment, you are to parallelize several provided sequential codes by using OpenMP. For questions 2,3,4, template C code is provided in three files, with suffixes _main.c, _seq.c and _par.c. The _main.c file contains initialization code and a test driver that performs multiple executions of a reference sequential version (code is in the _seq.c file) and your parallelized version, which is to be created by editing the _par.c file. Initially the _par.c file just contains the sequential code in the _seq.c file. You should not make any modifications to the _main.c or _seq.c files, but only modify the _par.c files. Compile using "*gcc -fopenmp -O3 foo_main.c foo_seq.c foo_par.c*". Correctness checking is performed by the driver in the _main.c files.

For parallel code development and testing, use any CHPC node. After code development, performance should be reported on a specific type of node via batch submission (specifics will be provided). Specified performance targets should be attained to get full credit but partial credit will be awarded if some speedup is achieved with correct parallel execution.
Submission: 1) A PDF report on Canvas providing explanations as specified for each question, 2) Submission of ***only*** your final _par.c files for questions 2/3/4, using exactly the same file names as provided – they will be used by the TAs to test performance. For example, for question 2, the file you submit will be hist_par.c.

1. (10 points) Compile and execute the program test.c ("*gcc -O3 -fopenmp test.c*"). It has two variants of parallel code for incrementing the values of elements in an array. You should observe very different performance for the two versions when executed in parallel. Why?

2. (25 points) The following code generates a histogram of the number of occurrences of each data element in the range ***0..nbins-1*** in the ***data*** array.
```
  for (i=0;i<nbins;i++) hist[i]=0;
  for (i = 0; i < nelts; i++) hist[data[i]] += 1;
```
Parallelize the histogram computation using OpenMP. **Performance target will be provided.**

3. (25 points) The following codes both performs matrix multiplication of two lower-triangular matrices (stored as standard dense matrices with the upper triangular parts storing zero values) to produce a lower triangular matrix.
```
for (k=0; k<N; k++)
 for (i=k; i<N; i++)          // trimm_kij
  for (j=0; j<=k; j++)
   C[i][j]+= A[i][k]*B[k][j];

for (i=0; i<N; i++)
 for (j=0; j<=i; j++)        // trimm_ijk
  for (k=j; k<=i; j++)
   C[i][j]+= A[i][k]*B[k][j];
```
Create parallel versions of both the above code variants using OpenMP.
   a) (10 points) Perform three executions, one each by placing a work-sharing "#pragma omp for" just above each of the three loops. Provide a clear explanation for the differences you observe regarding the code's execution in the 3 cases.
   b) (15 points) Make any further changes to achieve the performance target. Explain what you did differently (if anything) for the two code versions. **Performance targets will be provided.**

4. (40 points) Create a parallel merge-sort implementation using OpenMP. **Performance target will be provided.**

```
void Merge(a,b,lo,mid,hi)
int a[],b[], lo,mid,hi;
{ int h,i,j,k;
  h = lo; i = lo; j = mid+1;
  while ((h<=mid) && (j<=hi))
  {if (a[h]<=a[j]) b[i++] = a[h++]; else b[i++] = a[j++]; }
 if (h>mid)
   { for(k=j;k<=hi;k++) b[i++] = a[k]; }
  else
   { for(k=h;k<=mid;k++) b[i++] = a[k]; }
 for(k=lo;k<=hi;k++) a[k] = b[k];
}
```

```
void Merge_Sort(a,b, lo, hi)
 int a[],b[], lo,hi;
 { int temp,mid;
  if (lo < hi)
  { if (hi == lo+1)
       { if (a[hi]<a[lo]) {temp=a[hi];a[hi]=a[lo];a[lo]=temp;}}
          else
       { mid = (lo+hi)/2;
         Merge_Sort(a,b,lo,mid);
         Merge_Sort(a,b,mid+1,hi);
         Merge(a,b,lo,mid,hi);
       }
   }
}
```