CS 6230/5960
Programming Assignment 2

Due Wednesday, 2/12/2020, 11:59PM

1. (40 points) Create a parallel merge-sort implementation using OpenMP. Report performance for sorting 100 Million elements on one node of Notchpeak (target performance: 75 Million elements per second on some number of threads of your choosing; 10% extra credit for achieving twice target performance). Template code will be available on Canvas.

```
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);
        }
     }
}
```

2. (35 points) The following code performs a variant of matrix-matrix multiplication: $C = B_T A$ (which is equivalent to: $C_T = A_T B$, which is how the code is expressed). Your goal is to create a parallel OpenMP version of this computation to achieve a target parallel performance of 100 GFLOPs using any number of threads of your choice on one node of Notchpeak, for matrices of size 1999 × 1999 (10% extra credit for achieving twice target performance). You may use any valid loop transformations in creating the optimized parallel version.

   A buggy template code will be made available on Canvas for this exercise.

```
float c[N][N],b[N][N],a[N][N];
for(i=0;i<N;i++) for(j=0;j<N;j++)
  for(k=0;k<N;k++)
```

```
      c[j][i] = c[j][i] + a[k][i]*b[k][j];
```

3. (30 points) Consider a variant of the previous problem where only the upper-triangular portion of *A* is used as the left matrix in the matrix product. In this case, the summation loop over *k* only runs over the range from 0 to *i*, as shown below. Create an efficient parallel version of this code. Target parallel performance of 100 GFLOPs using any number of threads of your choice on one node of Notchpeak, for matrices of size 1999 × 1999 (10% extra credit for achieving twice target performance). Template code will be available on Canvas.

```
float c[N][N],b[N][N],a[N][N];
for(i=0;i<N;i++)
  for(j=0;j<N;j++)
    for(k=0;k<=i;k++)
      c[j][i] = c[j][i] + a[k][i]*b[k][j];
```

Report achieved parallel performance in GFLOPs (p2/p3) or MUpdates/sec (p1) for executing the parallel version on different number of threads, using one node on Notchpeak. You may observe that the best performance is not achieved using a number of threads corresponding to the number of physical cores on each node of the system (32 physical cores; 64 hyperthreaded cores). It is only required that the target performance be achieved for at least one of the cases reported. Note that while you may tune the code to achieve high performance for the specific problem sizes specified, the code should work correctly for other problem sizes. Present all results using the GNU gcc compiler (gcc -O3 -fopenmp).

The base code versions can be extremely slow (especially for p2 and p3). You can change the problem size during development to reduce turnaround time, and then change it back for final testing and reporting of performance.

Submit a report on Canvas including commented source code, performance data from execution of the codes and explanation/interpretation.