PA2: OPENMP

DIHAN DAI

February 13, 2020

Question 1

In merge sort, the divided subproblems can be solved concurrently and the merging at every level can also be performed concurrently, which can be achieved using #pragma omp parallel for.

```
#include <omp.h>
#include <stdio.h>
void Merge_Sort(int a[], int b[], int lo, int hi);
void Merge(int a[], int b[], int lo, int mid,int hi);
void Merge_Sort_Par(int a[], int b[], int n, int nThreads)
{
   int b_size = n/nThreads;
   int rem = n%nThreads;
   int b_rem;
   int b_borders[nThreads+1];// The borders of subproblems
   int b_number = nThreads, btmp;
   omp_set_num_threads(nThreads);

    b_borders[0] = 0;
    #pragma omp parallel for
    for (int i=1; i<nThreads; i++)
    {
            b_borders[i] = rem+i*b_size;
    }
    b_borders[nThreads] = n;
    #pragma omp parallel for
    for (int i=0; i<nThreads; i++)
    {
            Merge_Sort(a, b, b_borders[i], b_borders[i+1]-1);
    }
do
 {
      b_rem = b_number%2;
      b_number = b_number/2;
            #pragma omp parallel for
            for (int i=1; i<=b_number; i++)
            {
                        Merge(a, b, b_borders[2*i-2], b_borders[2*i-1]-1, b_borders[2*i]-1);
            }
            if(b_rem!=0)
            {
                        Merge(a, b, b_borders[2*b_number-2], b_borders[2*b_number]-1,b_borders[2*b_number+1]-1);
            }

            for (int i=0; i<b_number; i++)
            {
                        b_borders[i] = b_borders[2*i];
            }
            b_borders[b_number] = n;
 }while(b_number!=1);
}
```

Question 2

In template code, i, j, k defined outside the parallel region are shared variables, which results in the mistake. To overcome this, one simply need to defined i, j, k inside the parallel region. On the other hand, the sum should be written as c[j][i]+=a[k][i]*b[k][j] for parallel accumulation. In my program, I chose to solve the block version concurrently. In each block, I also perform loop interchange to reduce cache misses.

```
#include<omp.h>
void pa2_p2_sol(int n, float a[][n], float b[][n], float c[][n], int nThreads)
{
int i, j, k;
int it, jt, kt, iub, jub, kub;
int T = 36;

omp_set_num_threads(nThreads);

#pragma omp parallel for collapse(2) private(i, j, k, it, jt, kt, iub, jub, kub) shared(a,b,c)
for(it=0; it<n; it+=T)//Tiling
        for(jt=0; jt<n; jt+=T)
                for(kt=0; kt<n; kt+=T){
                        iub=it+T; jub=jt+T; kub=kt+T;
                        if (iub>n) iub=n;
                        if (jub>n) jub=n;
                        if (kub>n) kub=n;
                        for(k=kt; k<kub; k++)
                                for(i=it; i<iub; i++)
                                        for(j=jt; j<jub; j++){
                                                c[i][j] += a[k][j]*b[k][i];
}
}
}
```

Question 3

Similar to question 2, I also chose to perform tiling first and calculate the results for each block concurrently. However, it is in general slower than solving question 2. The time varies during exam (with ±0.05s at most), and the speed varies from 60flops to 100flops. I can't figure out the reason and the improvement for it.

```
#include<omp.h>
#include<stdio.h>
void pa2_p3_sol(int n, float a[][n], float b[][n], float c[][n], int nThreads) {
int i, j, k;
int it, jt, kt, iub, jub, kub;
int T = 36;

omp_set_num_threads(nThreads);

#pragma omp parallel for collapse(2) private(i, j, k, it, jt, kt, iub, jub, kub) shared(a,b,c,n,T)
for(it=0; it<n; it+=T)
    for(jt=0; jt<n; jt+=T)
    {
        for(kt=0; kt<=it; kt+=T)
        {
            iub=it+T; jub=jt+T;
            kub=kt+T;
            if (iub>n) iub=n;
            if (jub>n) jub=n;
            if (kub>n) kub=n;

            for(k=kt; k<kub; k++)
            {
                if (kt<it)
                {
                    for(j=jt; j<jub; j++)
                            for(i=it; i<iub; i++)
                            {
                                c[j][i] +=a[k][i]*b[k][j];
                            }
                }

                else{
                    for(j=jt; j<jub; j++)
                        for(i=k; i<iub; i++)
                        {
                            c[j][i] +=a[k][i]*b[k][j];
                        }

                }
            }
        }
    }
}
```