CSE 5441
Programming Assignment 2
Akshay M. Nikam (nikam.5)

Problem 1:

Dependences exist in the loop nesting. However all permutations are valid for serial execution.
OpenMP construct #pragma omp for was tested on each of the loops with various permutations. It was successful only with j loop. Using omp for construct with loop i and k results in differences. Parallelizing i loop is not a good idea since it will change the order of execution in B and alter the values that have to be read later. Similar reason for k loop. After this analysis, transformations are considered with a parallelized j loop.
All 6 permutations were attempted and permutation ikj was found to be the best performing of them. This permutation causes unit stride access to B. Hence continuing with ikj permutation.
Loop unrolling: Unrolling i and k was efficient. Since j is innermost loop, it doesn't yield significant performance improvement over minor unrolling.
Tiling: 3D tiling was tested and found to be very efficient compared to earlier transformations. Tile size 64 is optimal. Loop jt is made parallel.
K was unrolled after tiling and improved the performance.

No.	Transformations	GCC 12 cores	ICC 12 cores
0	pvt(I,j,k)	0.7	0.7
1	ijk	0.7	0.7
2	ikj	1.9	1.9
3	jik	0.8	0.7
4	jki	0.3	0.3
5	kij	1.7	1.5
6	kji	0.3	0.3
7	Unroll i	2.8	3.3
8	Unroll j	1.8	2.1
9	Unroll k	3	3.2
10	Tile ikj T=32	13.9	20.1
11	Tile ikj T=64	15.8	23.5
12	Tile ikj T=128	14.4	17.6
13	Tile ikj T=256	15.6	23.2
14	Tile ikj T=512	8.9	14.3
15	Tile ikj T=64 2-way Unroll k	20.4	25.9

Optimal code:

```
int it, jt, kt, T=64;
   #pragma omp for
for(jt=0; jt<N; jt+=T) {
for(it=0; it<N; it+=T) {
for(kt=it; kt<N; kt+=T) {
   for (i=it; i<it+T; i++) {
         if(i>kt) {
               if(i%2 == 1)
                     for (j=jt; j<jt+T; j++) {
                           B[i][j] += A[i][i]*B[i][j];
                     }
               for (k=i+i%2; k<kt+T; k+=2) {
                     for (j=jt; j<jt+T; j++) {
                           B[i][j] += A[i][k]*B[k][j];
                           B[i][j] += A[i][k+1]*B[k+1][j];
                      }
                }
          }
          else {
                for (k= kt; k<kt+T; k+=2) {
                      for (j=jt; j<jt+T; j++) {
                            B[i][j] += A[i][k]*B[k][j];
                            B[i][j] += A[i][k+1]*B[k+1][j];
                      }
                }
          }
    }
 }}}
```

Performance of this code on various number of threads:

# Threads	GCC GFLOPS	ICC GFLOPS
1	2.1	2.8
2	3.9	5.2
3	5.9	7.8
4	8	10.8
5	9.2	11.9
6	10.5	13.8
7	12.5	16.6
8	15.5	20.4
9	15.3	20.4
10	15.1	20.1
11	20.4	26.2
12	20.4	25.9

Performance of this code on various number of cores: (ppn in runjob.sh was changed)

# Cores	GCC GFLOPS	ICC GFLOPS
12	20.4	25.9
10	15.3	20.3
8	14.6	18.6
6	10.4	10.8
4	6.5	9.0
2	4.1	3.1
1	2.1	2.8

Problem 2:

Omp for construct was applied to k or l loop as it gave increased performance.

Stride analysis:

	A		B		C
l	N^2	0	0	N^2	1
k	0	N^2	N^2	0	N
j	1	1	1	1	0
i	N	N	N	N	0

It can be observed that since j loop gives unit stride access to both A and B, it should be the innermost loop in an ideal permutation. After j, there should be i loop. After i and j, l and k can be permuted as outermost. It was found that permutation lkij gives best performance compared to other permutations. Hence, considering lkij here onwards.

4D tiling of loop structure was tested and found to increase performance. Tile size 32 is optimal.

After tiling, unrolling i and j was tested.

Transformations	GCC GFLOPS		ICC GFLOPS
Permutation kijl serial	0.2		0.2
Permutation kijl omp for k	0.8		0.7
Permutation klij omp for k	7.4		6.4
Permutation lkij omp for l	7.1		7.7
4D tiling lkij T=16	6.5		9.1
4D tiling lkij T=32	8.5		11
4D tiling lkij T=64	8		9.7
Tiling and unrolling i	10.4		14.1

Tiling and unrolling l and j	9.7		11.8
Tiling and unrolling j	10.8		8.9
Tiling unrolling l (4 way) and j (2 way)	12		13
Tiling unrolling i (4 way)	10.5		14.4

GCC Best code:

```
int it, jt, kt, lt, T = 32;
for(it=0; it<N; it+=T) {
for(jt=0; jt<N; jt+=T) {
for(lt=0; lt<N; lt+=T) {
for(kt=0; kt<lt+1; kt+=T) {
#pragma omp for
for (l=(lt==0 ? 1:lt); l<lt+T; l++) {
   for (k=kt; k<(kt==lt ? l : kt+T); k++) {
      for (i=it; i<it+T; i+=4) {
         for (j=jt; j<jt+T; j+=2) {
            C[k][l] += 0.5*(A[l][i][j]*B[k][i][j]+A[k][i][j]*B[l][i][j]);
            C[k][l] += 0.5*(A[l][i][j+1]*B[k][i][j+1]+A[k][i][j+1]*B[l][i][j+1]);

            C[k][l] += 0.5*(A[l][i+1][j]*B[k][i+1][j]+A[k][i+1][j]*B[l][i+1][j]);
            C[k][l] += 0.5*(A[l][i+1][j+1]*B[k][i+1][j+1]+A[k][i+1][j+1]*B[l][i+1][j+1]);

            C[k][l] += 0.5*(A[l][i+2][j]*B[k][i+2][j]+A[k][i+2][j]*B[l][i+2][j]);
            C[k][l] += 0.5*(A[l][i+2][j+1]*B[k][i+2][j+1]+A[k][i+2][j+1]*B[l][i+2][j+1]);

            C[k][l] += 0.5*(A[l][i+3][j]*B[k][i+3][j]+A[k][i+3][j]*B[l][i+3][j]);
            C[k][l] += 0.5*(A[l][i+3][j+1]*B[k][i+3][j+1]+A[k][i+3][j+1]*B[l][i+3][j+1]);
         }
      }
   }
 }
 }}}}
```

ICC Best code:

```
int it, jt, kt, lt, T = 32;
for(it=0; it<N; it+=T) {
for(jt=0; jt<N; jt+=T) {
for(lt=0; lt<N; lt+=T) {
for(kt=0; kt<lt+1; kt+=T) {
#pragma omp for
for (l=(lt==0 ? 1:lt); l<lt+T; l++) {
   for (k=kt; k<(kt==lt ? l : kt+T); k++) {
      for (i=it; i<it+T; i+=4) {
```

```
                for (j=jt; j<jt+T; j++) {
                    C[k][l] += 0.5*(A[l][i][j]*B[k][i][j]+A[k][i][j]*B[l][i][j]);
                    C[k][l] += 0.5*(A[l][i+1][j]*B[k][i+1][j]+A[k][i+1][j]*B[l][i+1][j]);
                    C[k][l] += 0.5*(A[l][i+2][j]*B[k][i+2][j]+A[k][i+2][j]*B[l][i+2][j]);
                    C[k][l] += 0.5*(A[l][i+3][j]*B[k][i+3][j]+A[k][i+3][j]*B[l][i+3][j]);
                }
            }
        }
    }
    }}}}
```

Performance of GCC best code on various number of threads:

		GCC
# threads		GFLOPS
1		2.1
2		2.8
3		5
4		6.3
5		7.1
6		7.8
7		8.8
8		10.4
9		10.4
10		10.3
11		12
12		12

Performance of ICC best code on various number of threads:

		ICC
# threads		GFLOPS
1		2.4
2		4.4
3		5.8
4		8.5
5		9.9
6		11.6
7		11.7
8		13
9		12.8

10		13.6
11		14
12		14.4

Performance of GCC best code on various number of cores: (ppn in runjob.sh was changed)

# Cores	GCC GFLOPS
12	12
10	10.4
8	10.2
6	4.2
4	5.7
2	3.3
1	2

Performance of ICC best code on various number of cores: (ppn in runjob.sh was changed)

# Cores	ICC GFLOPS
12	14.4
10	13.6
8	11.5
6	10.4
4	7.5
2	5.5
1	3.2