CSE 5441

Programming Assignment 1

Vineeth Thumma (thumma.6)

Problem 1

```
#define N 4096
double x[N], y[N], z[N], m[N][N];
for(t=0;t<Niter;t++)
        for(i=0;i<N;i++)
                for(j=0;j<N;j++)
                {
                        y[j] = y[j] + m[i][j]*x[i];
                        z[j] = z[j] + m[j][i]*x[i];
                }
```

Reason for Low Performance:

Base-Version:

Compiler	L3 cache misses	Resource Stall Cycles	Performance
GCC	172.26 M	7089.34 M	0.26 GFLOPS
ICC	176.8 M	8030.85 M	0.24 GFLOPS

The reasons for low performance are:

- It can be observed that in the above snippet, the access stride for 'm' array in the computation of 'z' is not 1 but N because of which there are more cache misses

*z[j] = z[j] + m[j][i]*x[i];*

- There are also more number of resource stall cycles because there will be cache misses and the data has to fetched from main memory each time there is a cache miss

Code Transformations:

```
for(t = 0; t < Niter; t++)
     {
    for(i = 0; i < N; i++)
       for(j = td_id * bk_size; j < (td_id +1) * bk_size && j< N; j++)
                 {
                     y[j] = y[j] + m[i][j] * x[i];
                  }
```

```
for(j = td_id * bk_size; j < (td_id +1) * bk_size && j< N; j++)
             for(i = 0; i < N; i++)
```

```
                {
                  z[j] = z[j] + m[j][i] * x[i];
                }

#pragma omp barrier
for(i = td_id * bk_size; i < (td_id +1) * bk_size && i<N; i++)
            {
                x[i] += 1e-8;
            }
#pragma omp barrier
}
```

-First to reduce the no. of cache misses, we split y[j] and z[j] computations into two different loops as shown in the code snippet above so that 2D-array 'm' has access stride of 1 in both the computations.

- After doing the above loop transformation, the code is parallelized using work distribution by assigning each thread a fraction of the total computations.

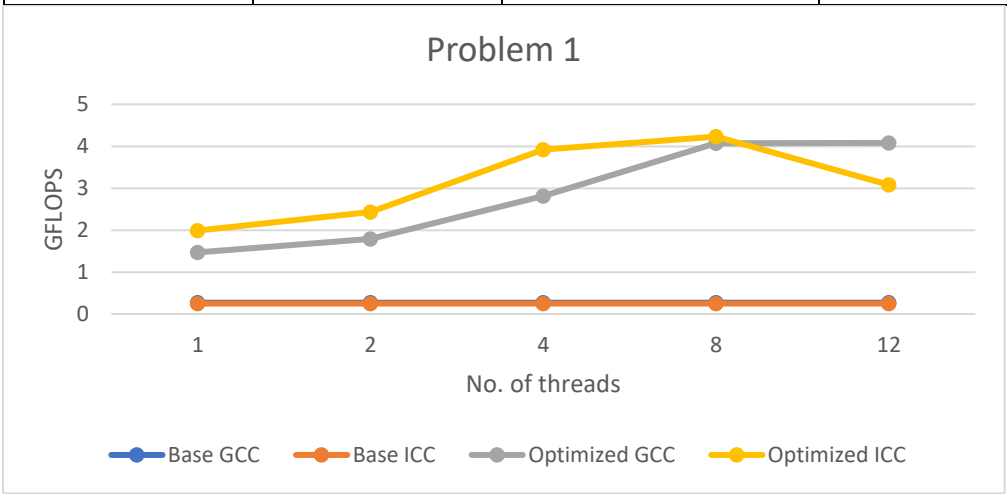
- The code is not parallelized on loop 't' because after each iteration of 't' we need to update the values of 'x'. The code is not parallelized on 'i' while computing 'y[j]' because there is a possibility of two different threads over-writing the same 'y[j]' which is not desired.

- We also parallelize the computation of 'x[i]' at the end of each iteration of 't' loop

Performance Improvements:

Optimized & Parallelized -Version: (No. of threads = 12, cores = 12)

Compiler	L3 cache misses	Resource Stall Cycles	Performance
GCC	1.71 M	199.26 M	3.74 GFLOPS
ICC	2.39 M	222.90 M	3.08 GFLOPS



Problem 2

```
#define N 128
double A[N][N][N], C[N][N], B[N][N][N];
for(k=0;k<N;k++)
  for(l=0;l<N;l++)
    for(i=0;i<N;i++)
      for(j=0;j<N;j++)
        C[j][k][l] += A[j][i][l]*B[k][i];
```

Reason for Low Performance:

Base-Version:

Compiler	L3 cache misses	Resource Stall Cycles	Performance
GCC	330.16 M	15129.17 M	0.1 GFLOPS
ICC	0.08 M	164.44 M	3.95 GFLOPS

The reasons for low performance are:
- There is no spatial locality (access stride is not 1) while accessing the arrays A,C in the given permutation of k,l,i,j because of which there are more number of cache misses and more resource stall cycles as the data has to be fetched from main memory
- Access-Stride for 'C' is N^2 , Access-Stride for 'A' is N^2

Code Transformations:

```
#pragma omp parallel for private(i,k,l)
  for(j=0;j<N;j++)
      for(k=0;k<N;k++)
        for(i=0;i<N;i++)
          for(l=0;l<N;l++)
           C[j][k][l] += A[j][i][l]*B[k][i];
  }
```

- To reduce the number of Cache Misses, we do Loop Permutation in such a way that arrays A, B and C have Access-Stride of 1 (i.e., change k,l,i,j to j,k,i,l)

- The above permutation doesn't change the values that are computed as compared to base version which means permutation is valid

- The code is not parallelized on 'i' while computing 'C[j][k][l]' because there is a possibility of two different threads over-writing the same ''C[j][k][l]' which is not desired.

Performance Improvements:

Optimized & Parallelized -Version: (No. of threads = 12, cores = 12)

Compiler	L3 cache misses	Resource Stall Cycles	Performance
GCC	0.01 M	1.90 M	22.0 GFLOPS
ICC	0.01 M	14.29 M	8.80 GFLOPS