Programming Assignment 1

1.
commented source code:

```
  // To be modified by you to improve performance
  //
  for(jt=0;jt<N;jt=jt+N/4)
    for (i=0; i<N; i++)
      for (k=i; k<N; k++)
       for (j=jt; j<jt+N/4; j++)
        B[i][j] += A[i][k]*B[k][j];
  //
```

output:
icc –fast:
Matrix Size = 2048
Base Symm-MatMult: 206.1 MFLOPS; Time = 41.706 sec;
Test Symm-MatMult: 1695.6 MFLOPS; Time = 5.068 sec;
No differences found between base and test versions

gcc –O3:
Matrix Size = 2048
Base Symm-MatMult: 203.3 MFLOPS; Time = 42.283 sec;
Test Symm-MatMult: 1626.1 MFLOPS; Time = 5.285 sec;
No differences found between base and test versions

Analysis:

Cache size is 12288kb. The whole data will take 2048*2048*8/1024=32768kb.

	A	B
k	N/B	N
j	1	N/B
I	N	N

Do permute from ijk to ikj

	A	B
j	I	N/B
k	N/B	N
i	N	N

Then do tiling for j. we can get

	A	B
j	I	N/B
k	N/B	N
i	N	1

for icc –fast, it is 41.706/5.068=8.23 times faster.
For gcc –O3, it is 42.283/5.285=8.00 times faster.

2.
commented source code version 1:

```
// To be modified by you to improve performance
//
  for (lt=0;lt<N;lt=lt+N/8)
    for (kt=0;kt<N;kt=kt+N/8)
      for (i=0; i<N; i++)
       for (j=0; j<N; j++)
        for (k=kt; k<kt+N/8; k++)
          for (l=lt; l<lt+N/8; l++)
          {
           C[i][j] += A[l][i][k]*B[k][j][l];
          }
//
```

Output:
icc –fast:
Tensor Size = 128
Base-TensorMult: 841.3 MFLOPS; Time = 0.638 sec;
Test-TensorMult: 1396.3 MFLOPS; Time = 0.384 sec;
No differences found between base and test versions

gcc –O3:
Tensor Size = 128
Base-TensorMult: 237.2 MFLOPS; Time = 2.263 sec;
Test-TensorMult: 826.7 MFLOPS; Time = 0.649 sec;
No differences found between base and test versions

Analysis:
Cache size: 12288kb. Data size 128^3*8/1024=16384kb > cache size.
So we can do tiling to optimize code.
For icc –fast, it is 0.638/0.384=1.6615 times faster.
For gcc –O3, it is 2.263/0.649=3.487 times faster.

Commented source code version 2:

```
// To be modified by you to improve performance
//
   for (i=0; i<N; i++)
    for (j=0; j<N; j++)
      for (k=0; k<N; k++)
        for (l=0; l<N; l=l+4)
        {
          C[i][j] += A[l][i][k]*B[k][j][l];
          C[i][j] += A[l+1][i][k]*B[k][j][l+1];
```

```
          C[i][j] += A[l+2][i][k]*B[k][j][l+2];
          C[i][j] += A[l+3][i][k]*B[k][j][l+3];
        }
//
```

Output:
icc –fast:
Tensor Size = 128
Base-TensorMult: 850.1 MFLOPS; Time = 0.632 sec;
Test-TensorMult: 1396.3 MFLOPS; Time = 0.384 sec;
No differences found between base and test versions

gcc –O3:
Tensor Size = 128
Base-TensorMult: 264.6 MFLOPS; Time = 2.029 sec;
Test-TensorMult: 321.8 MFLOPS; Time = 1.668 sec;
No differences found between base and test versions

Analysis:
Unrolling is also another way to optimize the code.
For icc –fast, it is 0.632/0.384=1.6458 times faster.
For gcc –fast, it is 2.029/1.668=1.2164 times faster.

3.
a.

```
// To be modified by you to improve performance
//
   for (it=0;it<N;it=it+N/16)
    for (jt=0;jt<N;jt=jt+N/16)
     for(i=it;i<it+N/16;i++)
      for(j=jt;j<jt+N/16;j++)
       BB[i][j] = 0.5*(AA[i][j]+AA[j][i]);
//
```

Output:
icc –fast:
Matrix Size = 4096
Base Symmetrizer: 120.7 MFLOPS; Time = 0.278 sec;
Test version: 259.9 MFLOPS; Time = 0.129 sec;
No differences found between base and test versions

gcc –O3:
Matrix Size = 4096
Base Symmetrizer: 157.1 MFLOPS; Time = 0.214 sec;
Test version: 239.5 MFLOPS; Time = 0.140 sec;
No differences found between base and test versions

Analysis:
Cache size 12288kb. Data size 4096^2*8/1024=131072.
Cache size/Data size=3/32
Do we do tiling to optimize code.
For icc –fast, it is 0.278/0.129=2.155 times faster.
For gcc –fast, it is 0.214/0.140=1.53 times faster.

b.
In this case, I changed the data structure of AA[N][N] to AA[N][N+1].
commented source code:

```
double A[N][N], B[N][N], AA[N][N+1], BB[N][N];
.....
.....
//
  for(it=0;it<N;it=it+N/16)
    for (i=it; i<it+N/16; i++)
      for (j=0; j<N; j++)
       BB[i][j] = 0.5*(AA[i][j]+AA[j][i]);
  //
```

Output:
icc –fast:
Matrix Size = 4096
Base Symmetrizer: 119.3 MFLOPS; Time = 0.281 sec;
Test version: 426.0 MFLOPS; Time = 0.079 sec;
No differences found between base and test versions

gcc –O3:
Matrix Size = 4096
Base Symmetrizer: 145.4 MFLOPS; Time = 0.231 sec;
Test version: 322.9 MFLOPS; Time = 0.104 sec;
No differences found between base and test versions

Analysis:
If we can change the data structure, we can just pad the array AA[N][N] to AA[N][N+1] so that the cache is interlaced filled for the AA[N][N+1].

For icc –fast, it is 0.281/0.079=3.557 times faster.
For gcc –O3, it is 0.231/0.104=2.221 times faster.