

Matrix Multiplication

```
28
29 void getMatrixA()
30 {
31     FILE *fp;
32
33     fp = fopen("/Users/fetouh/Desktop/matrixthread/matrixA.txt", "r");
34     fscanf(fp, "%d %d", &rowA,&columnA);
35
36     while(!feof(fp))
37     {
38         for(int i=0; i<rowA;i++)
39         {
40             for(int j=0; j<columnA;j++)
41
42                 fscanf (fp, "%d", &matrixA[i][j]);
43
44         }
45     }
46
47     fclose(fp);
48 }
49
50 void getMatrixB()
51 {
52     FILE *fp;
53
54     fp = fopen("/Users/fetouh/Desktop/matrixthread/matrixB.txt", "r");
55     fscanf(fp, "%d %d", &rowB,&columnB);
56
57     while(!feof(fp))
58     {
59         for(int i=0; i<rowB;i++)
60         {
61             for(int j=0; j<columnB;j++)
62
63                 fscanf (fp, "%d", &matrixB[i][j]);
64
65         }
66     }
67
68     fclose(fp);
69 }
```

input both matrices using getMatrixA and getMatrixB

Multiply functions used to multiply matrices in both ways

```
void* multiply1(void *data)
{
    for(int j=0; j<columnB;j++)
    {
        for(int k=0;k<rowB;k++)
            matrixC1[(int)data][j]+=matrixA[(int)data][k]*matrixB[k][j];
    }

    pthread_exit(NULL);
}

void* multiply2(void *data)|
{
    argo* x= (argo*)data;

    int row= x->row;
    int column=x->column;

    for(int k=0;k<rowB;k++)
        matrixC2[row][column]+=matrixA[row][k]*matrixB[k][column];

    pthread_exit(NULL);
}
```

```

20
21
22 void output1(double time)
23 {
24     FILE *fp;
25     fp = fopen("/Users/fetouh/Desktop/matrixthread/output1.txt", "w");
26     for(int i=0; i<rowA; i++)
27     { for(int j=0; j<columnB; j++)
28     {
29         fprintf(fp, "%d\t", matrixC1[i][j]);
30     }
31     fprintf(fp, "\n");
32
33
34
35 }
36     fprintf(fp, "%f", time);
37
38
39
40
41
42
43
44
45 }
46 void output2(double time)
47 {
48     FILE *fp;
49     fp = fopen("/Users/fetouh/Desktop/matrixthread/output2.txt", "w");
50     for(int i=0; i<rowA; i++)
51     { for(int j=0; j<columnB; j++)
52     {
53         fprintf(fp, "%d\t", matrixC2[i][j]);
54     }
55     fprintf(fp, "\n");
56
57
58
59 }
60     fprintf(fp, "%f", time);
61
62
63
64
65
66

```

print both matrices and time taken in a text file using output functions and print time taken time taken using threads for each row is less than thread less than each element .

output1.txt

```

-1    10    -15    -28
-3    -10    15    -36
5      -2     -9    -20
0.000234

```

output2.txt

```

-1    10    -15    -28
-3    -10    15    -36
5      -2     -9    -20
0.000551

```

Merge Sort

```
6
7
8
9 void input_array(int array[100])
10 {
11     FILE *fp;
12     //input array from file
13     fp = fopen("/Users/fetouh/Desktop/matrixthread/input.txt", "r");
14     fscanf(fp, "%d", &arraySize);
15     int i=0;
16     while(i<arraySize && !feof(fp))
17     {
18
19
20         fscanf (fp, "%d", &array[i]);
21
22         i++;
23     }
24     fclose(fp);
25 }
```

input array and array size from text file

Create new thread and call merge sort. Use pthread join to wait for all threads to finish.
Print array after mergesort

```
127 int main(int argc, const char * argv[]) {
128     argo x;
129     int array[100];
130     input_array(array);
131     int i=0;
132     x.array=array;
133     x.start=0;
134     x.end=arraySize-1;
135     pthread_t threads[1];
136     pthread_create(&threads[0], NULL, mergeS, (void *)&x);
137     pthread_join(threads[0], NULL);
138
139
140
141
142
143     while(i<arraySize-1)
144     {printf("%d",x.array[i]);
145         i++;
146     }
147
148
149
150
```