REPORT

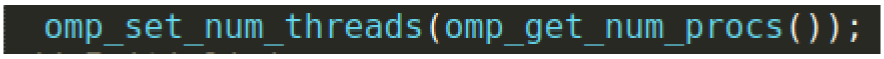
ROLLNO: 2019101055

Q1: MATRIX MULTIPLICATION(optimizing the existing code: (exising code is the one submitted for assignment 1))

OPTIMIZATIONS IN INITIAL CODE: transpose of matrix(caching),loops interchange,global matrices.

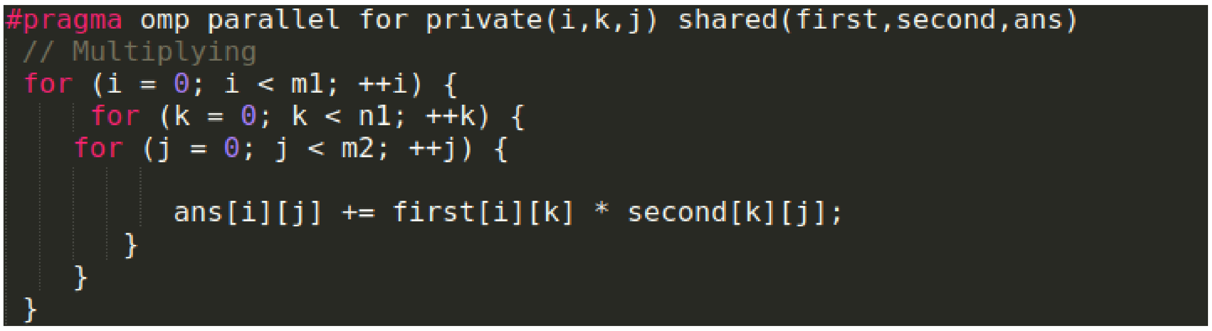
1: PARALELLISING THE MULTIPLICATION (best optimized version)

setting threads as the number of processors.



```
  omp_set_num_threads(omp_get_num_procs());
```

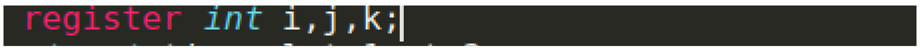
paralellizing the loop with i,j,k as private adn first array,second array and ans array as shared.Variables in shared context are visible to all threads running in associated parallel regions. Variables in private context are hidden from other threads. Each thread has its own private copy of the variable, and modifications made by a thread to its copy are not visible to other threads.



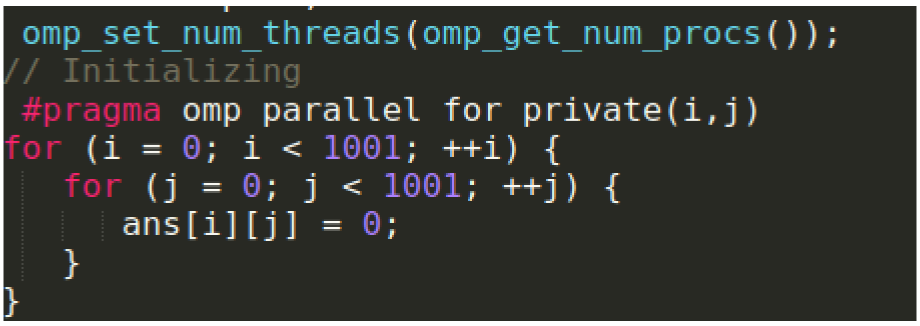
```
#pragma omp parallel for private(i,k,j) shared(first,second,ans)
 // Multiplying
 for (i = 0; i < m1; ++i) {
      for (k = 0; k < n1; ++k) {
    for (j = 0; j < m2; ++j) {

          ans[i][j] += first[i][k] * second[k][j];
       }
    }
 }
```

2: ADDING REGISTER VARAIBLES AND PARALLELISATION

Made all the local variables as register variables.



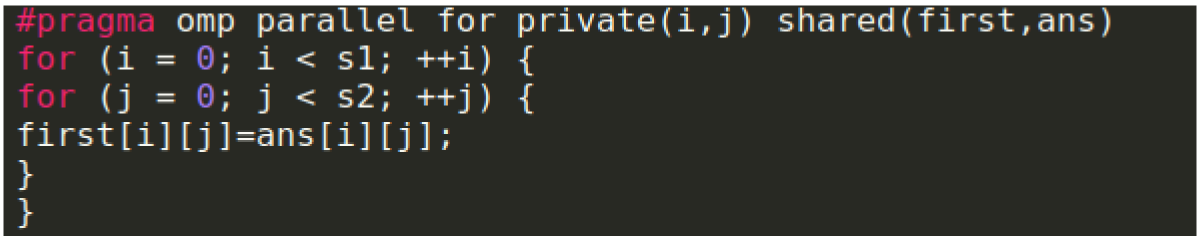
```
 register int i,j,k;
```



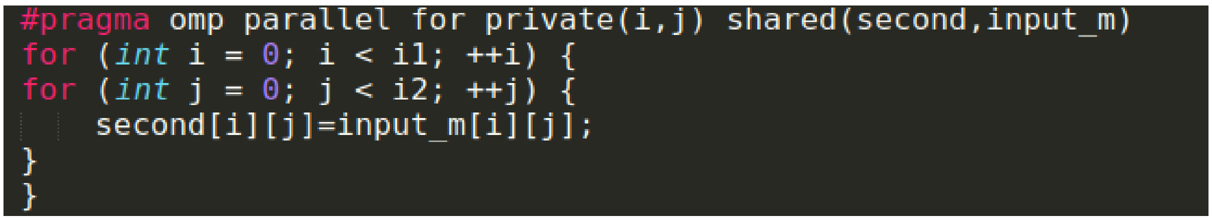
```
 omp_set_num_threads(omp_get_num_procs());
// Initializing
 #pragma omp parallel for private(i,j)
for (i = 0; i < 1001; ++i) {
   for (j = 0; j < 1001; ++j) {
      ans[i][j] = 0;
   }
}
```

Parallelizing the initialization loop with i,j as private.

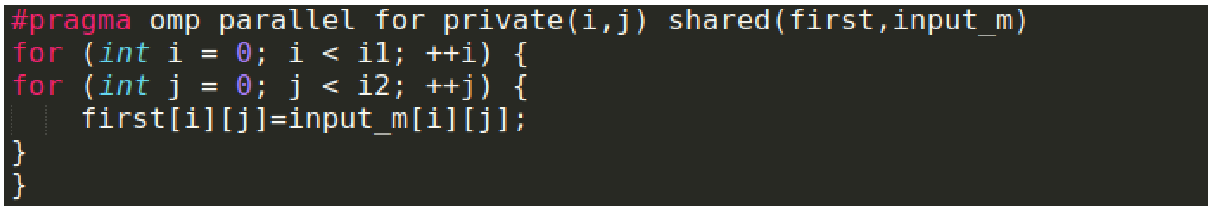
Parellilizing all 3 array copying loops.



```
#pragma omp parallel for private(i,j) shared(first,ans)
for (i = 0; i < s1; ++i) {
for (j = 0; j < s2; ++j) {
first[i][j]=ans[i][j];
}
}
```



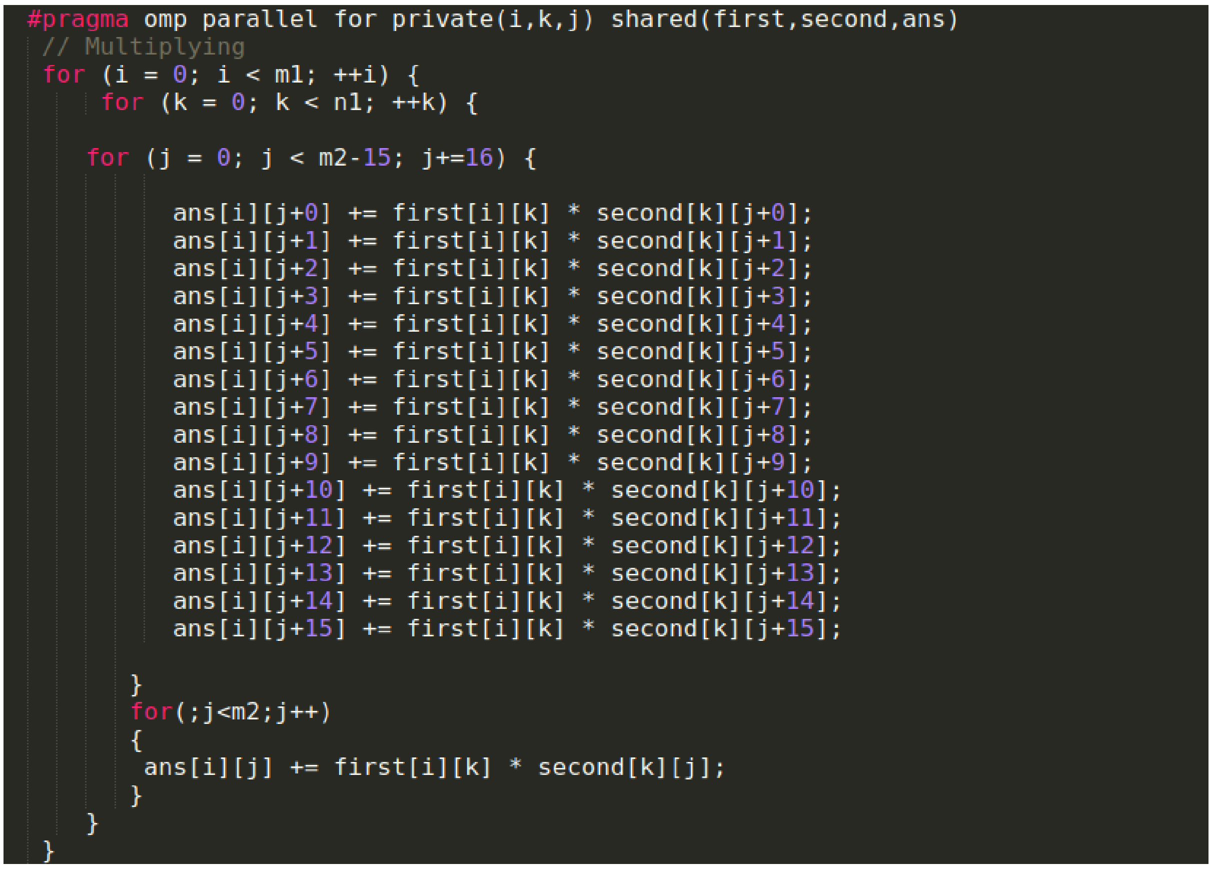
```
#pragma omp parallel for private(i,j) shared(second,input_m)
for (int i = 0; i < i1; ++i) {
for (int j = 0; j < i2; ++j) {
    second[i][j]=input_m[i][j];
}
}
```



```
#pragma omp parallel for private(i,j) shared(first,input_m)
for (int i = 0; i < i1; ++i) {
for (int j = 0; j < i2; ++j) {
    first[i][j]=input_m[i][j];
}
}
```

This doesn't make a big difference in time.
3:LOOP UNROLLING

Tried loop unrolling by 4,8,16 times and 16 times came out to be the most effecient loop unrolling.Loop unrolling increases the program’s speed by eliminating loop control instruction and loop test instructions.But by the below table we see that loop unrolling takes a bit more time than parellised program since paralell running is faster than loop unrolling and loop unrolling is not necessary since aldready parellisation is done.



```
#pragma omp parallel for private(i,k,j) shared(first,second,ans)
 // Multiplying
 for (i = 0; i < m1; ++i) {
     for (k = 0; k < n1; ++k) {

    for (j = 0; j < m2-15; j+=16) {

         ans[i][j+0] += first[i][k] * second[k][j+0];
         ans[i][j+1] += first[i][k] * second[k][j+1];
         ans[i][j+2] += first[i][k] * second[k][j+2];
         ans[i][j+3] += first[i][k] * second[k][j+3];
         ans[i][j+4] += first[i][k] * second[k][j+4];
         ans[i][j+5] += first[i][k] * second[k][j+5];
         ans[i][j+6] += first[i][k] * second[k][j+6];
         ans[i][j+7] += first[i][k] * second[k][j+7];
         ans[i][j+8] += first[i][k] * second[k][j+8];
         ans[i][j+9] += first[i][k] * second[k][j+9];
         ans[i][j+10] += first[i][k] * second[k][j+10];
         ans[i][j+11] += first[i][k] * second[k][j+11];
         ans[i][j+12] += first[i][k] * second[k][j+12];
         ans[i][j+13] += first[i][k] * second[k][j+13];
         ans[i][j+14] += first[i][k] * second[k][j+14];
         ans[i][j+15] += first[i][k] * second[k][j+15];

       }
       for(;j<m2;j++)
       {
        ans[i][j] += first[i][k] * second[k][j];
       }
    }
 }
```

TIME TAKEN: is the time taken for matrix multiplication.
ELAPSED TIME: is time taken in each matrix multiplication
TOTAL ELAPSED TIME: gives the total time after input to end of program(1.1.total time of running after input)

TEST CASES:	TIME TAKEN for n = 2 size = 100	TIME TAKEN for n = 5 size = 300	TIME TAKEN for n = 3 size = 500	TIME TAKEN for n = 4 size = 1000
ORIGINAL CODE (COMPILED WITH O2)	< elapsed time = 0.009371 seconds. < total elapsed time = 0.030877 seconds.	< elapsed time = 0.094219 seconds. < elapsed time = 0.090123 seconds. < elapsed time = 0.090802 seconds. < elapsed time = 0.091401 seconds. < total elapsed time = 0.410216 seconds.	< elapsed time = 0.431473 seconds. < elapsed time = 0.429230 seconds. < total elapsed time = 0.902999 seconds.	< elapsed time = 3.445069 seconds. < elapsed time = 3.381860 seconds. < elapsed time = 3.410231 seconds. < total elapsed time = 10.449878 seconds.

1.pararellisation and using all processors using threads.	< elapsed time = 0.014543 seconds. < total elapsed time = 0.021800 seconds.	< elapsed time = 0.024529 seconds. < elapsed time = 0.037594 seconds. < elapsed time = 0.029007 seconds. < elapsed time = 0.023435 seconds. < total elapsed time = 0.161386 seconds.	< elapsed time = 0.116537 seconds. < elapsed time = 0.110847 seconds. < total elapsed time = 0.275435 seconds.	< elapsed time = 0.907792 seconds. < elapsed time = 0.991014 seconds. < elapsed time = 0.921718 seconds. < total elapsed time = 3.060677 seconds.
2. register int and extra paralellisation	< elapsed time = 0.000866 seconds. < total elapsed time = 0.015549 seconds.	< elapsed time = 0.039165 seconds. < elapsed time = 0.037122 seconds. < elapsed time = 0.025462 seconds. < elapsed time = 0.025310 seconds. < total elapsed time = 0.222343 seconds.	< elapsed time = 0.123317 seconds. < elapsed time = 0.112537 seconds. < total elapsed time = 0.283928 seconds.	< elapsed time = 0.910338 seconds. < elapsed time = 0.968709 seconds. < elapsed time = 0.927784 seconds. < total elapsed time = 3.053896 seconds.
3. loop unrolling	< elapsed time = 0.000950 seconds. < total elapsed time = 0.004475 seconds.	< elapsed time = 0.046017 seconds. < elapsed time = 0.043102 seconds.	< elapsed time = 0.138090 seconds. < elapsed time = 0.124568 seconds.	< elapsed time = 1.101894 seconds. < elapsed time = 1.087929 seconds.

		< elapsed time = 0.027082 seconds. < elapsed time = 0.026754 seconds. < total elapsed time = 0.240670 seconds.	< total elapsed time = 0.316568 seconds.	< elapsed time = 1.073883 seconds. < total elapsed time = 3.516383 seconds.

CACHEGRIND:
1. n = 2 size = 100

```
==103073==
==103073== I   refs:      106,574,592
==103073== I1  misses:          1,888
==103073== LLi misses:          1,851
==103073== I1  miss rate:        0.00%
==103073== LLi miss rate:        0.00%
==103073==
==103073== D   refs:       33,359,381  (27,927,575 rd   + 5,431,806 wr)
==103073== D1  misses:        279,029  (   145,812 rd   +   133,217 wr)
==103073== LLd misses:        137,295  (     6,957 rd   +   130,338 wr)
==103073== D1  miss rate:         0.8% (       0.5%     +       2.5%  )
==103073== LLd miss rate:         0.4% (       0.0%     +       2.4%  )
==103073==
==103073== LL refs:           280,917  (   147,700 rd   +   133,217 wr)
==103073== LL misses:         139,146  (     8,808 rd   +   130,338 wr)
==103073== LL miss rate:          0.1% (       0.0%     +       2.4%  )
```

1.n = 3 size = 500

```
==103601==
==103601== I   refs:      4,965,479,270
==103601== I1  misses:            1,363
==103601== LLi misses:            1,350
==103601== I1  miss rate:          0.00%
==103601== LLi miss rate:          0.00%
==103601==
==103601== D   refs:      1,796,609,968  (1,626,402,633 rd   + 170,207,335 wr)
==103601== D1  misses:       14,687,752  (   14,024,686 rd   +     663,066 wr)
==103601== LLd misses:          676,281  (      140,029 rd   +     536,252 wr)
==103601== D1  miss rate:           0.8% (          0.9%     +         0.4%  )
```

```
==103601== LLd miss rate:           0.0% (          0.0%     +         0.3%  )
==103601==
==103601== LL refs:          14,689,115  (   14,026,049 rd   +     663,066 wr)
==103601== LL misses:           677,631  (      141,379 rd   +     536,252 wr)
==103601== LL miss rate:            0.0% (          0.0%     +         0.3%  )
2.n = 2 size = 100
==103073==
==103073== I   refs:      106,574,592
==103073== I1  misses:          1,888
==103073== LLi misses:          1,851
==103073== I1  miss rate:        0.00%
==103073== LLi miss rate:        0.00%
==103073==
==103073== D   refs:       33,359,381  (27,927,575 rd   + 5,431,806 wr)
==103073== D1  misses:        279,029  (   145,812 rd   +   133,217 wr)
==103073== LLd misses:        137,295  (     6,957 rd   +   130,338 wr)
==103073== D1  miss rate:         0.8% (       0.5%     +       2.5%  )
==103073== LLd miss rate:         0.4% (       0.0%     +       2.4%  )
==103073==
==103073== LL refs:           280,917  (   147,700 rd   +   133,217 wr)
==103073== LL misses:         139,146  (     8,808 rd   +   130,338 wr)
==103073== LL miss rate:          0.1% (       0.0%     +       2.4%  )
2.n = 3 size = 500
==103288==
==103288== I   refs:      5,191,674,012
==103288== I1  misses:            2,165
==103288== LLi misses:            2,135
==103288== I1  miss rate:          0.00%
==103288== LLi miss rate:          0.00%
==103288==
==103288== D   refs:      1,821,145,936  (1,647,581,560 rd   + 173,564,376 wr)
==103288== D1  misses:       14,689,680  (   14,025,888 rd   +     663,792 wr)
==103288== LLd misses:          677,813  (      140,928 rd   +     536,885 wr)
==103288== D1  miss rate:           0.8% (          0.9%     +         0.4%  )
==103288== LLd miss rate:           0.0% (          0.0%     +         0.3%  )
==103288==
==103288== LL refs:          14,691,845  (   14,028,053 rd   +     663,792 wr)
==103288== LL misses:           679,948  (      143,063 rd   +     536,885 wr)
==103288== LL miss rate:            0.0% (          0.0%     +         0.3%  )
```

PERF stat:
when n = 3 size = 500
original code :

```
            889.67 msec task-clock                 #    0.107 CPUs utilized
           447       context-switches          #    0.502 K/sec
            33       cpu-migrations            #    0.037 K/sec
         4,946       page-faults               #    0.006 M/sec
  3,36,03,26,825       cycles                    #    3.777 GHz
 11,29,38,12,074       instructions              #    3.36  insn per cycle
    40,73,35,925       branches                  #  457.850 M/sec
       14,05,362       branch-misses             #    0.35% of all branches
```

after optimisation :

```
            1,994.94 msec task-clock               #    0.150 CPUs utilized
         2,060       context-switches          #    0.001 M/sec
            18       cpu-migrations            #    0.009 K/sec
         4,984       page-faults               #    0.002 M/sec
  7,33,28,87,251       cycles                    #    3.676 GHz
 11,64,55,39,795       instructions              #    1.59  insn per cycle
    68,25,25,350       branches                  #  342.129 M/sec
       14,77,706       branch-misses             #    0.22% of all branches
```

GPROF:

ORIGINAL CODE:

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 99.20      0.75     0.75        2   376.96   376.96  multiply_2
  0.00      0.75     0.00        3     0.00     0.00  input
  0.00      0.75     0.00        1     0.00     0.00  print_matrix
                              Call graph
```

```
granularity: each sample hit covers 2 byte(s) for 1.33% of 0.75 seconds

index % time    self  children    called     name
                0.75    0.00       2/2           main [2]
[1]    100.0    0.75    0.00       2         multiply_2 [1]
-----------------------------------------------
                                                 <spontaneous>
[2]    100.0    0.00    0.75                 main [2]
                0.75    0.00       2/2           multiply_2 [1]
                0.00    0.00       3/3           input [3]
                0.00    0.00       1/1           print_matrix [4]
-----------------------------------------------
                0.00    0.00       3/3           main [2]
[3]      0.0    0.00    0.00       3         input [3]
-----------------------------------------------
                0.00    0.00       1/1           main [2]
[4]      0.0    0.00    0.00       1         print_matrix [4]
-----------------------------------------------
Index by function name

   [3] input                  [1] multiply_2             [4] print_matrix


OPTIMIZED ONE:
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 94.91      0.51     0.51                             main
  3.72      0.53     0.02        2    10.05    10.05  multiply_2
  1.86      0.54     0.01        1    10.05    10.05  print_matrix
  0.00      0.54     0.00        3     0.00     0.00  input
                        Call graph


granularity: each sample hit covers 2 byte(s) for 1.84% of 0.54 seconds

index % time    self  children    called     name
                                                 <spontaneous>
[1]    100.0    0.51    0.03                 main [1]
                0.02    0.00       2/2           multiply_2 [2]
                0.01    0.00       1/1           print_matrix [3]
```

```
               0.00    0.00       3/3           input [4]
-----------------------------------------------
               0.02    0.00       2/2           main [1]
[2]      3.7    0.02    0.00       2         multiply_2 [2]
-----------------------------------------------
               0.01    0.00       1/1           main [1]
[3]      1.9    0.01    0.00       1         print_matrix [3]
-----------------------------------------------
               0.00    0.00       3/3           main [1]
[4]      0.0    0.00    0.00       3         input [4]
-----------------------------------------------
Index by function name

   [4] input                   [2] multiply_2
   [1] main                    [3] print_matrix
```

3.LOOP UNROLLED:

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 98.42      0.55     0.55                             main
  1.79      0.56     0.01        1    10.02    10.02  print_matrix
  0.00      0.56     0.00        3     0.00     0.00  input
  0.00      0.56     0.00        2     0.00     0.00  multiply_2
                              Call graph


granularity: each sample hit covers 2 byte(s) for 1.78% of 0.56 seconds

index % time    self  children    called     name
                                                 <spontaneous>
[1]    100.0    0.55    0.01                 main [1]
               0.01    0.00       1/1           print_matrix [2]
               0.00    0.00       3/3           input [3]
               0.00    0.00       2/2           multiply_2 [4]
-----------------------------------------------
               0.01    0.00       1/1           main [1]
[2]      1.8    0.01    0.00       1         print_matrix [2]
-----------------------------------------------
               0.00    0.00       3/3           main [1]
[3]      0.0    0.00    0.00       3         input [3]
```

```
-----------------------------------------------
                0.00    0.00       2/2          main [1]
[4]      0.0    0.00    0.00       2        multiply_2 [4]
-----------------------------------------------
Index by function name

   [3] input                   [4] multiply_2
   [1] main                    [2] print_matrix
```