Matrix-Matrix Multiplication

cache blocking,loop unrolling,OpenMP tasks,Strassen

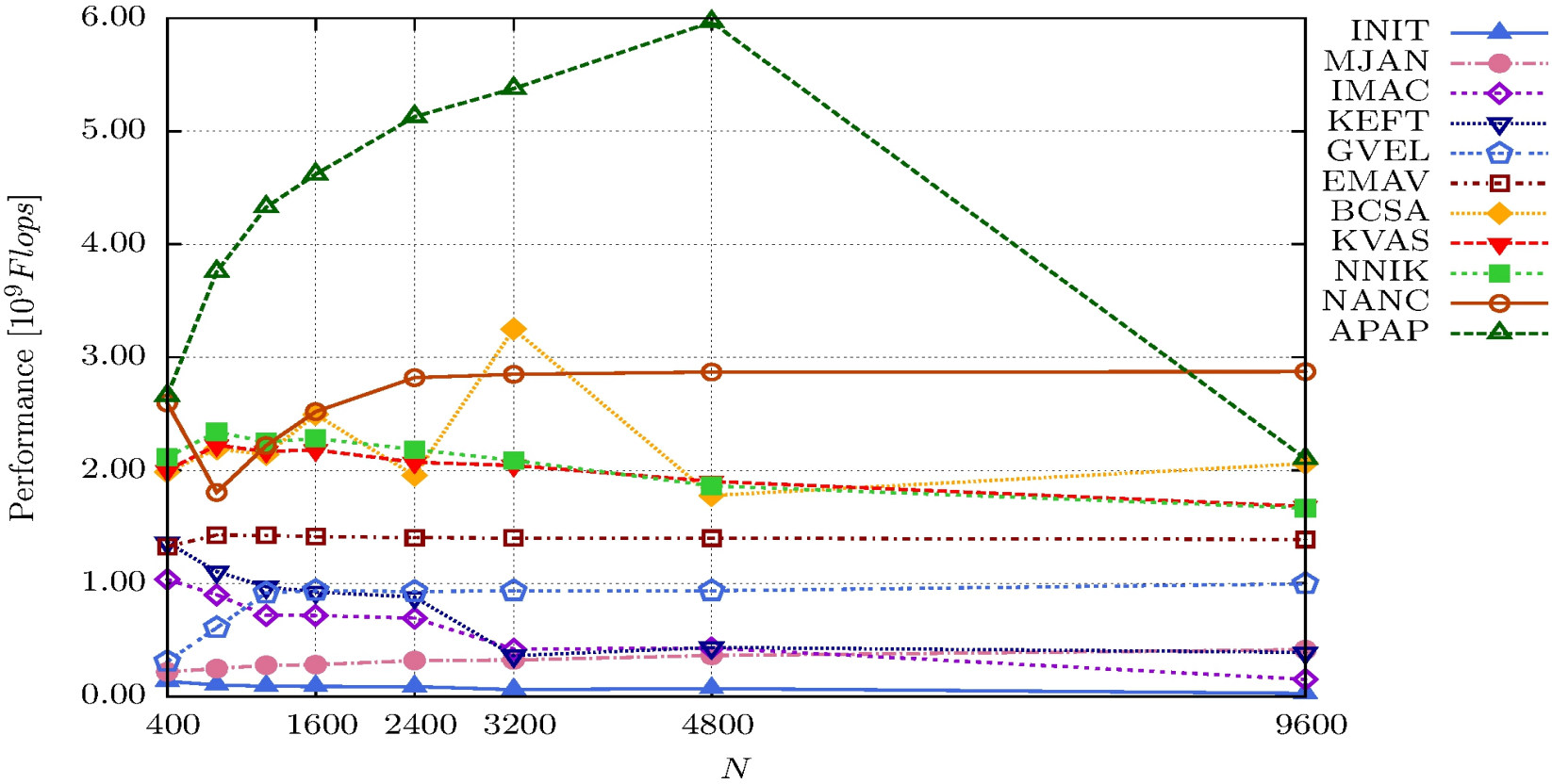
HP-SEE Computing Challenge

"We appreciate your programming skills, but at the same time we offer you a challenge! Are you able to write the fastest matrix-matrix multiplication code?"

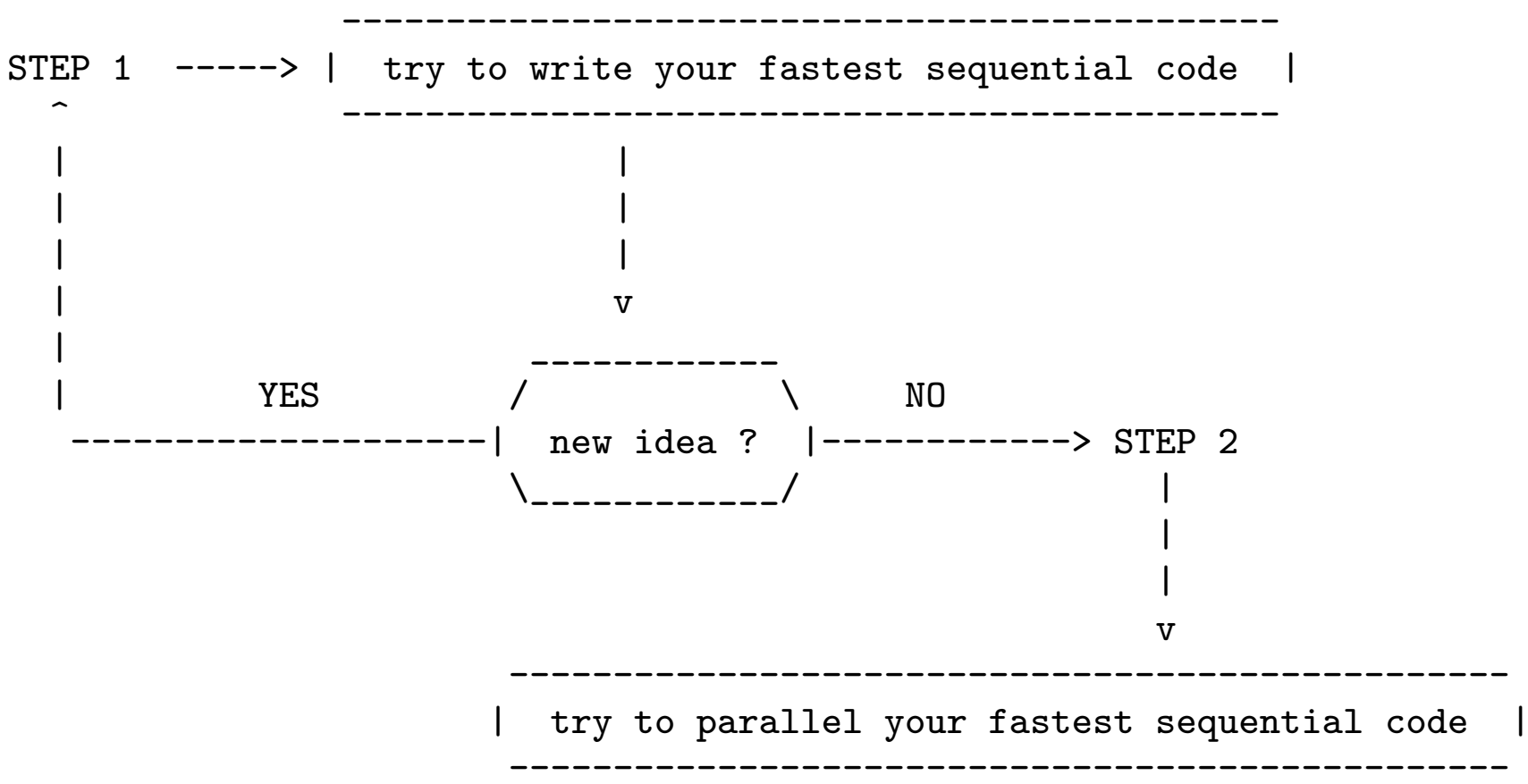
```
0. void matmul(long N, double *a, double *b, double *c) {
1.     long i, j, k;
2.
3.     for (i = 0; i < N; i ++)
4.         for (j = 0; j < N; j ++)
5.             for (k = 0; k < N; k ++)
6.                 c[i * N + j] += a[i * N + k] * b[k * N + j];
7. }
```

Try to modify this code and improve its performance. The modifications can be implemented in C and/or assembly language, with OpenMP parallelization.

The performance of your code will be measured on a computer with two quad-core Intel Xeon E5345 @ 2.33 GHz CPUs (4 MB of L3 cache) and 32 GB of RAM. The code will be compiled with the Intel compiler (version 13.0.0) without optimization (-O0) and with the flag for OpenMP parallelization (-openmp). No external library may be called.

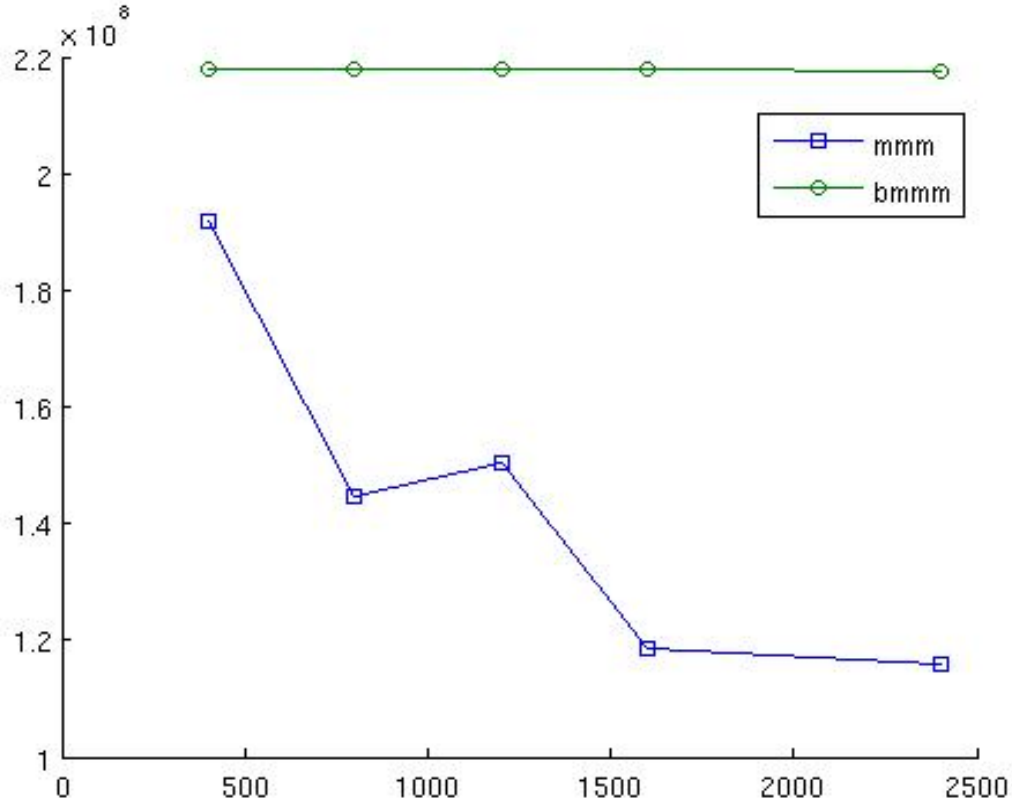


http://www.hp-see.eu/hp-see-computing-challenge

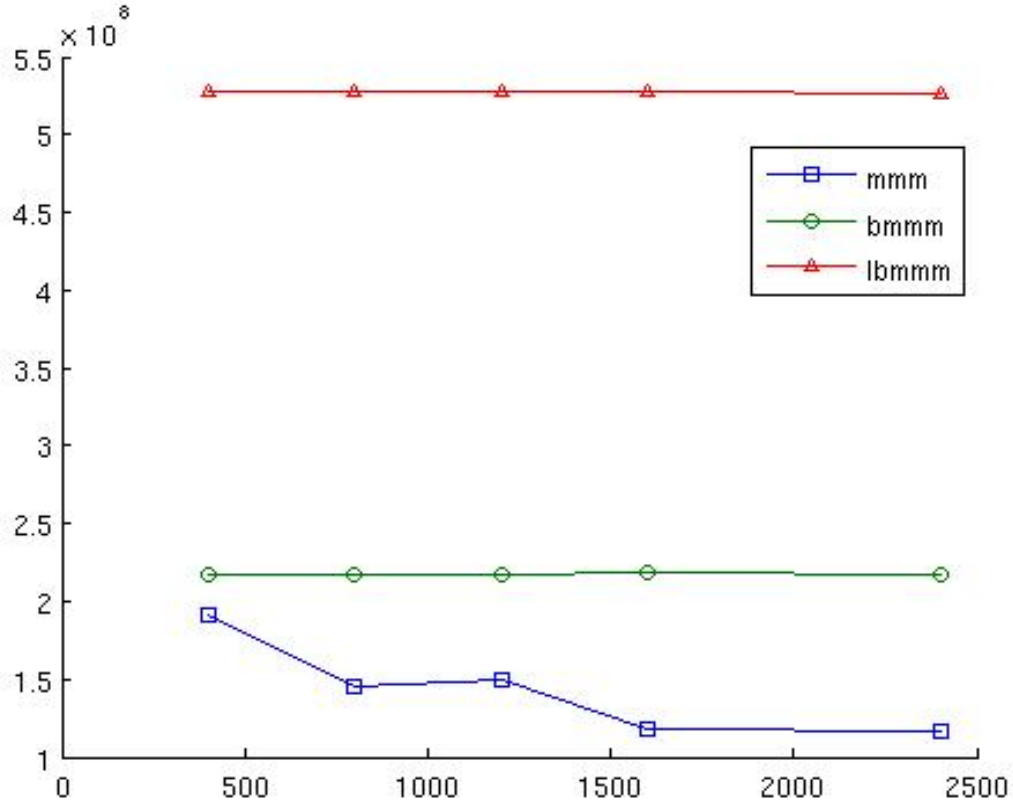

STEP 1
try to write your fastest sequential code
new idea ?
YES
NO
STEP 2
try to parallel your fastest sequential code

Cache Blocking

```
0. b_t:=b transpose
1. num=N/BLOCK_SIZE;
2.
3. for(i=0;i<num;i++){
4.     for (j=0; j<num; j++){
5.         for (k=0; k<BLOCK_SIZE; k++){
6.             for (m=0; m<BLOCK_SIZE; m++) {
7.                 double sum=0.0;
8.                 for (r = 0; r < num; r++){
10.                    for(p=0; p<BLOCK_SIZE; p++){
11.                        sum += a[i*BLOCK_SIZE*N + r*BLOCK_SIZE + k*N + p]*
12.                             b_t[j*BLOCK_SIZE*N + r*BLOCK_SIZE + m*N + p];
13.                    }
14.                }
15.                c[i*BLOCK_SIZE*N + j*BLOCK_SIZE + k*N + m] = sum;
16.            }
17.        }
18.    }
19. }
```


x 10^8
2.2
2
1.8
1.6
1.4
1.2
1
0
500
1000
1500
2000
2500
mmm
bmmm

Full loop unrolling of p-for-loop


× 10^8
5.5
5
4.5
4
3.5
3
2.5
2
1.5
1
0
500
1000
1500
2000
2500
mmm
bmmm
lbmmm

Matrices X and Y are split into four smaller (N/2)x(N/2) matrices as follows:

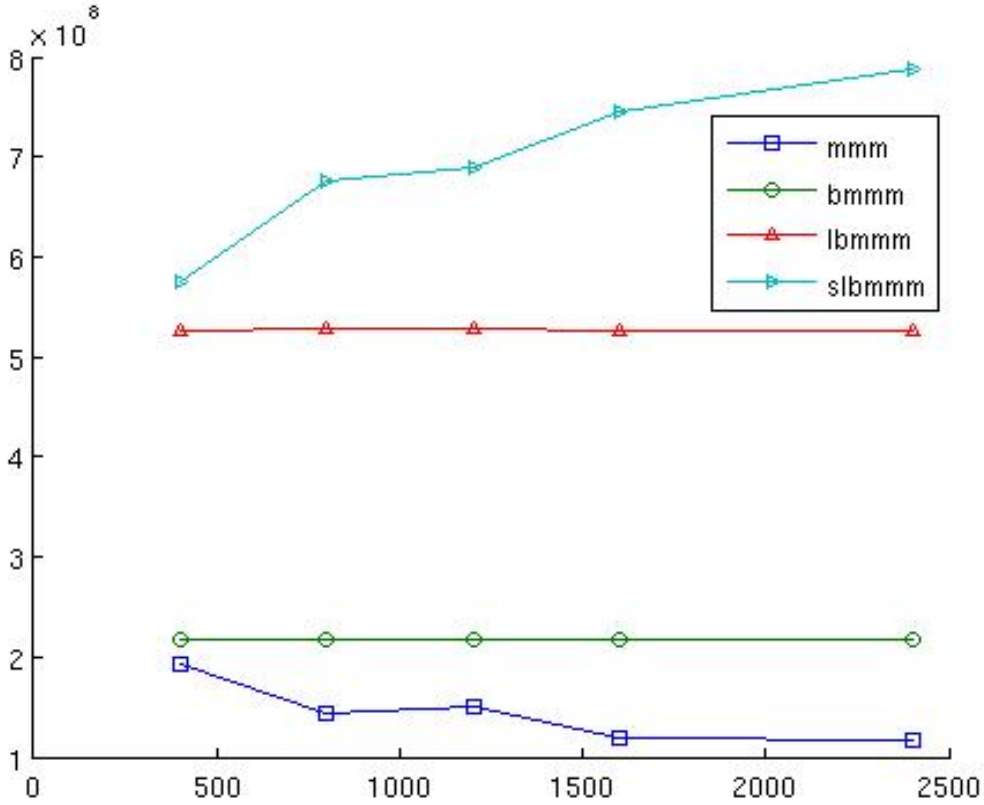
$$X = \begin{vmatrix} A & B \\ C & D \end{vmatrix} \qquad Y = \begin{vmatrix} E & F \\ G & H \end{vmatrix}$$

Then we build the following 7 matrices (requiring seven (N/2)x(N/2) matrix multiplications -- this is where the 2.807 = log2(7) improvement comes from):

```
P0 = A*(F - H)
P1 = (A + B)*H
P2 = (C + D)*E
P3 = D*(G - E)
P4 = (A + D)*(E + H)
P5 = (B - D)*(G + H)
P6 = (A - C)*(E + F)
```

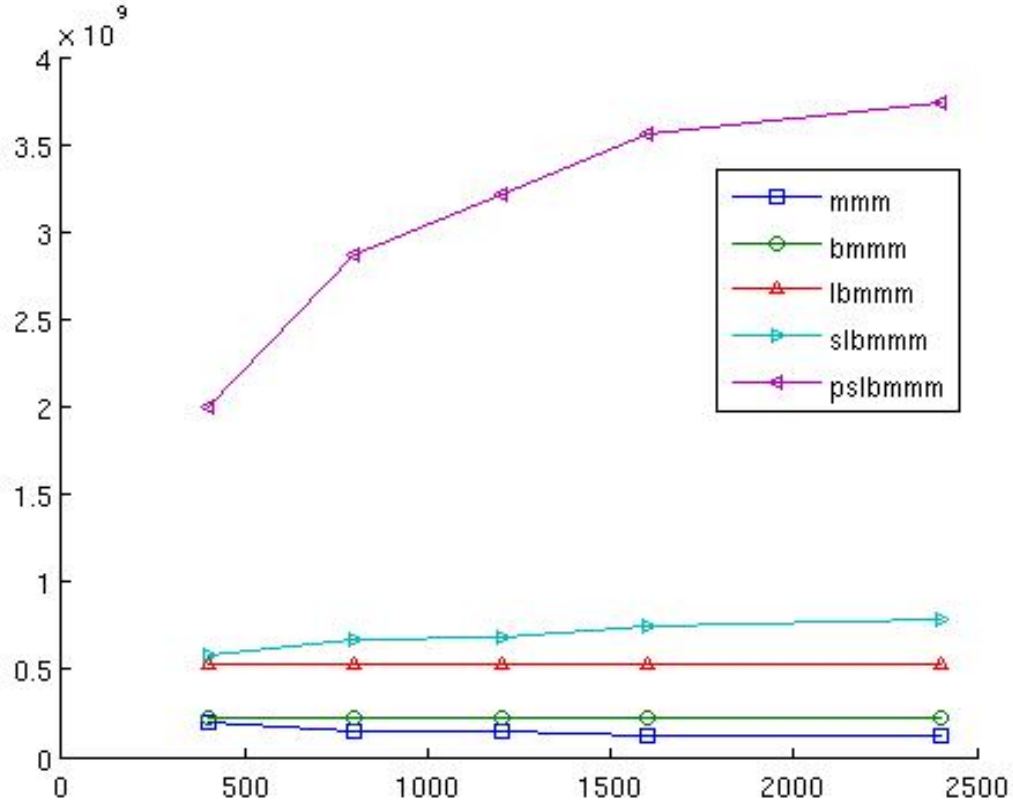
The final result is

$$Z = \begin{vmatrix} (P3 + P4) + (P5 - P1) & P0 + P1 \\ P2 + P3 & (P0 + P4) - (P2 + P6) \end{vmatrix}$$


x 10^8
8
7
6
5
4
3
2
1
0
500
1000
1500
2000
2500
mmm
bmmm
lbmmm
slbmmm

```
0. void matmul(long N, double *a, double *b, double *c) {
1.     #pragma omp parallel num_threads(NUM_THREADS)
2.     {
3.         #pragma omp single
4.         {
5.             strassen(N, a, b, c);
6.         }
7.     }
8. }
```

```
0. void strassen(long N, double *a, double *b, double *c) {
1.      if (N<=THRESHOLD_MULT){
2.          lbmmm(N,a,b,c);
3.          return;
4.      }
5.      ...
6.      #pragma omp task
7.      {
8.          // P0 = A*(F - H);
9.          msub(n, F, H, T);
10.         strassen(n, A, T, P[0]);
11.     }
12.     ...
13.     #pragma omp taskwait
14.     ...
15. }
```

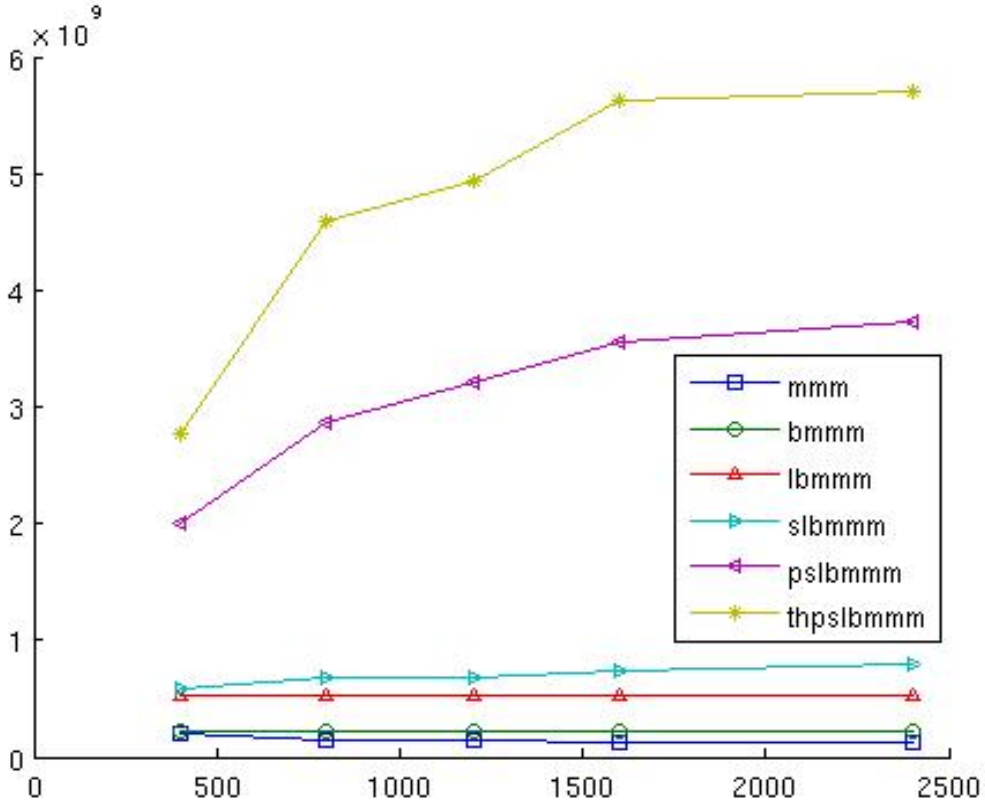

x 10^9
4
3.5
3
2.5
2
1.5
1
0.5
0
0
500
1000
1500
2000
2500
mmm
bmmm
lbmmm
slbmmm
pslbmmm

```
0. void strassen(long N, double *a, double *b, double *c) {
1.      if (N<=THRESHOLD_MULT){                 ----------------------------
2.          lbmmm(N,a,b,c);     ----------> | lbmmm is a sequential code |--
3.          return;                         ----------------------------   |
4.      }                                                                    |
5.      ...                                                                  |
                            ----------------------------------------        |
                           | parallel the most external for-loop ?  | <----
                            ----------------------------------------
                                                |
                              -------------     |
                             | BINGOoo !!! | <-----
                              -------------
                                   |
                                   |
                                   v
   ------------------------------------------------------------------------
  | ... but wait, it is in a OpenMP task...."pragma omp for" will crash it |
  |  "#pragma omp parallel for" will cause too much overhead               |
   ------------------------------------------------------------------------
```

-___- hmmmm.....

```
0. #pragma omp task
1. {
2.     for(i=0; i<N; i++)
3.         job();
4. }
```

```
0.  #pragma omp task
1.  {
2.      int tol=N/NUM_THREADS;
3.
4.      for(thread=0;thread<NUM_THREADS;thread++)
5.          #pragma omp task firstprivate(thread) private(i)
6.          {
7.              for (i=thread*tol;i<(thread+1)*tol;i++)
8.                  job();
9.          }
10.
11.     #pragma omp taskwait
12.
13.     for(i=NUM_THREADS*tol; i<N; i++)
14.         job();
15. }
```


× 10^9
6
5
4
3
2
1
0
0
500
1000
1500
2000
2500
mmm
bmmm
lbmmm
slbmmm
pslbmmm
thpslbmmm

http://students.ceid.upatras.gr/~papadakis/mmm.c