Example: Hand optimization of matrix x matrix multiplication

```
#include <stdio.h>
#define SIZE 500
int mm(double first[][SIZE],double second[][SIZE],double multiply[][SIZE])
{
  int i,j,k;
  double sum = 0;
  for (i = 0; i < SIZE; i++) { //rows in multiply
    for (j = 0; j < SIZE; j++) { //columns in multiply
      for (k = 0; k < SIZE; k++) { //columns in first,rows in second
            sum = sum + first[i][k]*second[k][j];
          }
          multiply[i][j] = sum;
          sum = 0;
    }
  }
  return 0;
}

int main( int argc, const char* argv[] )
{
  int i,j,iret;
  double first[SIZE][SIZE];
  double second[SIZE][SIZE];
  double multiply[SIZE][SIZE];
  for (i = 0; i < SIZE; i++) { //rows in first
    for (j = 0; j < SIZE; j++) { //columns in first
      first[i][j]=i+j;
      second[i][j]=i-j;
    }
  }
  iret=mm(first,second,multiply);
  return iret;
}
```

```
paszynsk@atari:~/optimize$ gcc mm1.c
paszynsk@atari:~/optimize$ time ./a.out
real    0m0.833s
user    0m0.820s
sys     0m0.020s
```

Ad.1

Use

`register unsigned int variable name;` instead of `int variable name;`

```
#include <stdio.h>
#define SIZE 500
int mm(double first[][SIZE],double second[][SIZE],double multiply[][SIZE])
{
  register unsgined int i,j,k;
  double sum = 0;
  for (i = 0; i < SIZE; i++) { //rows in multiply
    for (j = 0; j < SIZE; j++) { //columns in multiply
      for (k = 0; k < SIZE; k++) { //columns in first,rows in second
            sum = sum + first[i][k]*second[k][j];
          }
          multiply[i][j] = sum;
          sum = 0;
    }
  }
  return 0;
}

int main( int argc, const char* argv[] )
{
  register unsigned int i,j;
  int iret;
  double first[SIZE][SIZE];
  double second[SIZE][SIZE];
  double multiply[SIZE][SIZE];
  for (i = 0; i < SIZE; i++) { //rows in first
    for (j = 0; j < SIZE; j++) { //columns in first
      first[i][j]=i+j;
      second[i][j]=i-j;
    }
  }
  iret=mm(first,second,multiply);
  return iret;
}
```

```
paszynsk@atari:~/optimize$ gcc mm2.c
paszynsk@atari:~/optimize$ time ./a.out
real    0m0.839s
user    0m0.840s
sys     0m0.000s
```

Ad. 2 It is better to use integers instead of floating point numbers

```
#include <stdio.h>
#define SIZE 500
int mm(int first[][SIZE],int second[][SIZE],int multiply[][SIZE])
{
 register unsgined int i,j,k;
 int sum = 0;
 for (i = 0; i < SIZE; i++) { //rows in multiply
  for (j = 0; j < SIZE; j++) { //columns in multiply
   for (k = 0; k < SIZE; k++) { //columns in first,rows in second
      sum = sum + first[i][k]*second[k][j];
     }
     multiply[i][j] = sum;
     sum = 0;
  }
 }
 return 0;
}

int main( int argc, const char* argv[] )
{
 register unsigned int i,j;
 int iret;
 int first[SIZE][SIZE];
 int second[SIZE][SIZE];
 int multiply[SIZE][SIZE];
 for (i = 0; i < SIZE; i++) { //rows in first
  for (j = 0; j < SIZE; j++) { //columns in first
   first[i][j]=i+j;
   second[i][j]=i-j;
  }
 }
 iret=mm(first,second,multiply);
 return iret;
}
```

```
paszynsk@atari:~/optimize$ gcc mm3.c
paszynsk@atari:~/optimize$ time ./a.out

real    0m0.795s
user    0m0.790s
sys     0m0.000s
```

Ad. 8

if a loop uses a global variable, it is beneficial to make a local copy (before the loop) so the local copy can be assigned to a register.

```
#include <stdio.h>
#define SIZE 500
int mm(int first[][SIZE],int second[][SIZE],int multiply[][SIZE])
{
  register unsigned int i,j,k;
  register unsigned int local_size=SIZE;
  int sum = 0;
  for (i = 0; i < local_size; i++) { //rows in multiply
    for (j = 0; j < local_size; j++) { //columns in multiply
      for (k = 0; k < local_size; k++) { //columns in first,rows in second
            sum = sum + first[i][k]*second[k][j];
          }
          multiply[i][j] = sum;
          sum = 0;
    }
  }
  return 0;
}

int main( int argc, const char* argv[] )
{
  register unsigned int i,j;
  register unsigned int local_size=SIZE;
  int iret;
  int first[SIZE][SIZE];
  int second[SIZE][SIZE];
  int multiply[SIZE][SIZE];
  for (i = 0; i < local_size; i++) { //rows in first
    for (j = 0; j < local_size; j++) { //columns in first
      first[i][j]=i+j;
      second[i][j]=i-j;
    }
  }
  iret=mm(first,second,multiply);
  return iret;
}
```

```
paszynsk@atari:~/optimize$ gcc mm4.c
paszynsk@atari:~/optimize$ time ./a.out

real    0m0.845s
user    0m0.850s
sys     0m0.000s
```

Ad. 20

Loop termination

```
#include <stdio.h>

#define SIZE 500

int mm(int first[][SIZE], int second[][SIZE], int multiply[][SIZE])
{
  register unsigned int i,j,k;
  int sum = 0;
  for (i = SIZE; i-- ; ) { //rows in multiply
    for (j = SIZE; j-- ;) { //columns in multiply
      for (k = SIZE; k-- ; ) { //columns in first and rows in second
            sum = sum + first[i][k]*second[k][j];
          }
          multiply[i][j] = sum;
          sum = 0;
    }
  }
  return 0;
}

int main( int argc, const char* argv[] )
{
  register unsigned int i,j;
  int iret;
  int first[SIZE][SIZE];
  int second[SIZE][SIZE];
  int multiply[SIZE][SIZE];
  for (i = SIZE; i-- ; ) { //rows in first
    for (j = SIZE; j--; ) { //columns in first
      first[i][j]=i+j;
      second[i][j]=i-j;
    }
  }
  iret=mm(first,second,multiply);
  return iret;
}
```

```
paszynsk@atari:~/optimize$ gcc mm5.c
paszynsk@atari:~/optimize$ time ./a.out
real    0m0.763s
user    0m0.760s
sys     0m0.000s
```

Ad. 23. Loop unrolling

```
mm6.c

#include <stdio.h>

#define SIZE 500

int mm(int first[][SIZE], int second[][SIZE], int multiply[][SIZE])
{
  register unsigned int i,j,k;
  int sum = 0;
  for (i = SIZE; i-- ; ) { //rows in multiply
    for (j = SIZE; j-- ;) { //columns in multiply
      for (k = 0; k<SIZE ; ) { //columns in first and rows in second
            if(k<SIZE-8) {
                sum = sum + first[i][k]*second[k][j];
                sum = sum + first[i][k+1]*second[k+1][j];
                sum = sum + first[i][k+2]*second[k+2][j];
                sum = sum + first[i][k+3]*second[k+3][j];
                sum = sum + first[i][k+4]*second[k+4][j];
                sum = sum + first[i][k+5]*second[k+5][j];
                sum = sum + first[i][k+6]*second[k+6][j];
                sum = sum + first[i][k+7]*second[k+7][j];
                k=k+8;
            }
            else {
                sum = sum + first[i][k]*second[k][j];
                k++;
            }
          }
          multiply[i][j] = sum;
          sum = 0;
    }
  }
  return 0;
}

int main( int argc, const char* argv[] )
{
  register unsigned int i,j;
  int iret;
  int first[SIZE][SIZE];
  int second[SIZE][SIZE];
  int multiply[SIZE][SIZE];
  for (i = SIZE; i-- ; ) { //rows in first
    for (j = SIZE; j--; ) { //columns in first
      first[i][j]=i+j;
      second[i][j]=i-j;
    }
  }
  iret=mm(first,second,multiply);
  return iret;
}
```

mm7.c

```
#include <stdio.h>

#define SIZE 500

int mm(int first[][SIZE], int second[][SIZE], int multiply[][SIZE])
{
  register unsigned int i,j,k;
  int sum = 0;
  for (i = SIZE; i-- ; ) { //rows in multiply
    for (j = SIZE; j-- ;) { //columns in multiply
      for (k = 0; k<SIZE ; ) { //columns in first and rows in second
            if(k<SIZE-16) {
                sum = sum + first[i][k]*second[k][j];
                sum = sum + first[i][k+1]*second[k+1][j];
                sum = sum + first[i][k+2]*second[k+2][j];
                sum = sum + first[i][k+3]*second[k+3][j];
                sum = sum + first[i][k+4]*second[k+4][j];
                sum = sum + first[i][k+5]*second[k+5][j];
                sum = sum + first[i][k+6]*second[k+6][j];
                sum = sum + first[i][k+7]*second[k+7][j];
                sum = sum + first[i][k+8]*second[k+8][j];
                sum = sum + first[i][k+9]*second[k+9][j];
                sum = sum + first[i][k+10]*second[k+10][j];
                sum = sum + first[i][k+11]*second[k+11][j];
                sum = sum + first[i][k+12]*second[k+12][j];
                sum = sum + first[i][k+13]*second[k+13][j];
                sum = sum + first[i][k+14]*second[k+14][j];
                sum = sum + first[i][k+15]*second[k+15][j];
                k=k+16;
            }
            else {
                sum = sum + first[i][k]*second[k][j];
                k++;
            }
          }
          multiply[i][j] = sum;
          sum = 0;
    }
  }
  return 0;
}
```

mm8.c

```
#include <stdio.h>

#define SIZE 500

int mm(int first[][SIZE], int second[][SIZE], int multiply[][SIZE])
{
  register unsigned int i,j,k;
  int sum = 0;
```

```
 for (i = SIZE; i-- ; ) { //rows in multiply
  for (j = SIZE; j-- ;) { //columns in multiply
   for (k = 0; k<SIZE ; ) { //columns in first and rows in second
      if(k<SIZE-32) {
        sum = sum + first[i][k]*second[k][j];
        sum = sum + first[i][k+1]*second[k+1][j];
        sum = sum + first[i][k+2]*second[k+2][j];
        sum = sum + first[i][k+3]*second[k+3][j];
        sum = sum + first[i][k+4]*second[k+4][j];
        sum = sum + first[i][k+5]*second[k+5][j];
        sum = sum + first[i][k+6]*second[k+6][j];
        sum = sum + first[i][k+7]*second[k+7][j];
        sum = sum + first[i][k+8]*second[k+8][j];
        sum = sum + first[i][k+9]*second[k+9][j];
        sum = sum + first[i][k+10]*second[k+10][j];
        sum = sum + first[i][k+11]*second[k+11][j];
        sum = sum + first[i][k+12]*second[k+12][j];
        sum = sum + first[i][k+13]*second[k+13][j];
        sum = sum + first[i][k+14]*second[k+14][j];
        sum = sum + first[i][k+15]*second[k+15][j];
        sum = sum + first[i][k+16]*second[k+16][j];
        sum = sum + first[i][k+17]*second[k+17][j];
        sum = sum + first[i][k+18]*second[k+18][j];
        sum = sum + first[i][k+19]*second[k+19][j];
        sum = sum + first[i][k+20]*second[k+20][j];
        sum = sum + first[i][k+21]*second[k+21][j];
        sum = sum + first[i][k+22]*second[k+22][j];
        sum = sum + first[i][k+23]*second[k+23][j];
        sum = sum + first[i][k+24]*second[k+24][j];
        sum = sum + first[i][k+25]*second[k+25][j];
        sum = sum + first[i][k+26]*second[k+26][j];
        sum = sum + first[i][k+27]*second[k+27][j];
        sum = sum + first[i][k+28]*second[k+28][j];
        sum = sum + first[i][k+29]*second[k+29][j];
        sum = sum + first[i][k+30]*second[k+30][j];
        sum = sum + first[i][k+31]*second[k+31][j];
        k=k+32;
      }
      else {
        sum = sum + first[i][k]*second[k][j];
        k++;
      }
     }
     multiply[i][j] = sum;
     sum = 0;
  }
 }
 return 0;
}
```

```
paszynsk@atari:~/optimize$ gcc mm5.c
paszynsk@atari:~/optimize$ time ./a.out
real    0m0.762s
user    0m0.760s
sys     0m0.000s

paszynsk@atari:~/optimize$ gcc mm6.c
paszynsk@atari:~/optimize$ time ./a.out
real    0m0.644s
user    0m0.650s
sys     0m0.000s

paszynsk@atari:~/optimize$ gcc mm7.c
paszynsk@atari:~/optimize$ time ./a.out
real    0m0.632s
user    0m0.640s
sys     0m0.010s
```

```
paszynsk@atari:~/optimize$ gcc mm8.c
paszynsk@atari:~/optimize$ time ./a.out
real    0m0.637s
user    0m0.630s
sys     0m0.000s
```

New idea = efficient reutilization of cache

```c
#include <stdio.h>

#define SIZE 500

int mm(int first[][SIZE], int second[][SIZE], int multiply[][SIZE])
{
  register unsigned int i,j,k;
  int sum = 0;
  for (i = SIZE; i-- ; ) { //rows in multiply
    for (j = SIZE; j-- ;) { //columns in multiply
      for (k = 0; k<SIZE ; ) { //columns in first and rows in second
            if(k<SIZE-16) {
                sum = sum + first[i][k]*second[j][k];
                sum = sum + first[i][k+1]*second[j][k+1];
                sum = sum + first[i][k+2]*second[j][k+2];
                sum = sum + first[i][k+3]*second[j][k+3];
                sum = sum + first[i][k+4]*second[j][k+4];
                sum = sum + first[i][k+5]*second[j][k+5];
                sum = sum + first[i][k+6]*second[j][k+6];
                sum = sum + first[i][k+7]*second[j][k+7];
                sum = sum + first[i][k+8]*second[j][k+8];
                sum = sum + first[i][k+9]*second[j][k+9];
                sum = sum + first[i][k+10]*second[j][k+10];
                sum = sum + first[i][k+11]*second[j][k+11];
                sum = sum + first[i][k+12]*second[j][k+12];
                sum = sum + first[i][k+13]*second[j][k+13];
                sum = sum + first[i][k+14]*second[j][k+14];
                sum = sum + first[i][k+15]*second[j][k+15];
                k=k+16;
            }
            else {
                sum = sum + first[i][k]*second[j][k];
                k++;
            }
          }
          multiply[i][j] = sum;
          sum = 0;
    }
  }
  return 0;
}

int main( int argc, const char* argv[] )
{
  register unsigned int i,j;
  int iret;
  int first[SIZE][SIZE];
  int second[SIZE][SIZE];
  int multiply[SIZE][SIZE];
  for (i = SIZE; i-- ; ) { //rows in first
    for (j = SIZE; j--; ) { //columns in first
      first[i][j]=i+j;
      second[j][i]=i-j;
    }
  }
  iret=mm(first,second,multiply);
  return iret;
}
```

```
paszynsk@atari:~/optimize$ gcc mm10.c
paszynsk@atari:~/optimize$ time ./a.out

real    0m0.551s
```

```
user	0m0.550s
sys	0m0.000s
```

What happens if we move matrix to structure?

mmA.c

```
#include <stdio.h>

#define SIZE 400

typedef struct {
  double M[SIZE][SIZE];
} Matrix;

int mm(Matrix first, Matrix second, Matrix multiply)
{
  int i,j,k;
  double sum = 0;
  for (i = 0; i < SIZE; i++) { //rows in multiply
    for (j = 0; j < SIZE; j++) { //columns in multiply
      for (k = 0; k < SIZE; k++) { //columns in first and rows in second
            sum = sum + first.M[i][k]*second.M[k][j];
          }
          multiply.M[i][j] = sum;
          sum = 0;
    }
  }
  return 0;
}

int main( int argc, const char* argv[] )
{
  int i,j,iret;
  Matrix first, second, multiply;
  for (i = 0; i < SIZE; i++) { //rows in first
    for (j = 0; j < SIZE; j++) { //columns in first
      first.M[i][j]=i+j;
      second.M[i][j]=i-j;
    }
  }
  iret=mm(first,second,multiply);
  return iret;
}
```

mmB.c

```
#include <stdio.h>

#define SIZE 400

typedef struct {
  double M[SIZE][SIZE];
} Matrix;

int mm(Matrix* first, Matrix* second, Matrix* multiply)
{
  int i,j,k;
  double sum = 0;
  for (i = 0; i < SIZE; i++) { //rows in multiply
    for (j = 0; j < SIZE; j++) { //columns in multiply
      for (k = 0; k < SIZE; k++) { //columns in first and rows in second
            sum = sum + first->M[i][k]*second->M[k][j];
          }
          multiply->M[i][j] = sum;
          sum = 0;
    }
  }
  return 0;
}

int main( int argc, const char* argv[] )
{
  int i,j,iret;
  Matrix first, second, multiply;
  for (i = 0; i < SIZE; i++) { //rows in first
    for (j = 0; j < SIZE; j++) { //columns in first
      first.M[i][j]=i+j;
      second.M[i][j]=i-j;
    }
  }
  iret=mm(&first,&second,&multiply);
  return iret;
}
```

```
paszynsk@atari:~/optimize$ gcc mmB.c <-- pointer
paszynsk@atari:~/optimize$ time ./a.out

real    0m0.444s
user    0m0.430s
sys     0m0.010s
```

No difference with respect to our implementation mm1.c with 2D tables:

```
paszynsk@atari:~/optimize$ gcc mm1.c
paszynsk@atari:~/optimize$ time ./a.out

real    0m0.450s
user    0m0.450s
sys     0m0.010s
```

Gcc VS intel

```
paszynsk@atari:~/optimize$ gcc mm1.c
paszynsk@atari:~/optimize$ time ./a.out
real    0m0.833s
user    0m0.820s
sys     0m0.020s
```

```
paszynsk@atari:~/optimize$ icc mm1.c
mm1.c(28): (col. 5) remark: LOOP WAS VECTORIZED.
mm1.c(33): (col. 8) remark: LOOP WAS VECTORIZED.
```

```
mm1.c(11): (col. 7) remark: LOOP WAS VECTORIZED.
paszynsk@atari:~/optimize$ time ./a.out

real    0m0.245s
user    0m0.240s
sys     0m0.000s
```

Zmniejszenie liczby operacji +

```
int mm(int first[][SIZE], int second[][SIZE], int multiply[][SIZE])
{
  register unsigned int i,j,k;
  int sum = 0;
  for (i = SIZE; i-- ; ) { //rows in multiply
    for (j = SIZE; j-- ;) { //columns in multiply
      for (k = 0; k<SIZE ; ) { //columns in first and rows in second
            if(k<SIZE-16) {
                sum = sum + first[i][k]*second[k][j]
                  + first[i][k+1]*second[k+1][j]
                  + first[i][k+2]*second[k+2][j]
                  + first[i][k+3]*second[k+3][j]
                  + first[i][k+4]*second[k+4][j]
                  + first[i][k+5]*second[k+5][j]
                  + first[i][k+6]*second[k+6][j]
                  + first[i][k+7]*second[k+7][j]
                  + first[i][k+8]*second[k+8][j]
                  + first[i][k+9]*second[k+9][j]
                  + first[i][k+10]*second[k+10][j]
                  + first[i][k+11]*second[k+11][j]
                  + first[i][k+12]*second[k+12][j]
                  + first[i][k+13]*second[k+13][j]
                  + first[i][k+14]*second[k+14][j]
                  + first[i][k+15]*second[k+15][j];
                k=k+16;
            }
            else {
                sum = sum + first[i][k]*second[k][j];
                k++;
            }
          }
          multiply[i][j] = sum;
          sum = 0;
    }
  }
  return 0;
}
```

We process now two dimensional blocks

```
int mm(double first[][SIZE], double second[][SIZE], double multiply[]
[SIZE])
{
  register unsigned int i,j,k;
  register unsigned int local_size=SIZE;
  double sum = 0;
  for (i = SIZE; i-- ; ) {
    for (j = 0; j < SIZE ; ) {
      for (k = 0; k < SIZE ; ) {
        if(j<SIZE-8 && k<SIZE-8) {
          sum = sum + first[i][k]*second[j][k];
          sum = sum + first[i][k+1]*second[j][k+1];
          sum = sum + first[i][k+2]*second[j][k+2];
          sum = sum + first[i][k+3]*second[j][k+3];
          sum = sum + first[i][k+4]*second[j][k+4];
          sum = sum + first[i][k+5]*second[j][k+5];
          sum = sum + first[i][k+6]*second[j][k+6];
          sum = sum + first[i][k+7]*second[j][k+7];

        sum = sum + first[i][k]*second[j+1][k];
        sum = sum + first[i][k+1]*second[j+1][k+1];
        sum = sum + first[i][k+2]*second[j+1][k+2];
        sum = sum + first[i][k+3]*second[j+1][k+3];
        sum = sum + first[i][k+4]*second[j+1][k+4];
        sum = sum + first[i][k+5]*second[j+1][k+5];
        sum = sum + first[i][k+6]*second[j+1][k+6];
        sum = sum + first[i][k+7]*second[j+1][k+7];

        sum = sum + first[i][k]*second[j+2][k];)
        sum = sum + first[i][k+1]*second[j+2][k+1];
        sum = sum + first[i][k+2]*second[j+2][k+2];
        sum = sum + first[i][k+3]*second[j+2][k+3];
        sum = sum + first[i][k+4]*second[j+2][k+4];
        sum = sum + first[i][k+5]*second[j+2][k+5];
        sum = sum + first[i][k+6]*second[j+2][k+6];
        sum = sum + first[i][k+7]*second[j+2][k+7];

        sum = sum + first[i][k]*second[j+3][k];
        sum = sum + first[i][k+1]*second[j+3][k+1];
        sum = sum + first[i][k+2]*second[j+3][k+2];
        sum = sum + first[i][k+3]*second[j+3][k+3];
        sum = sum + first[i][k+4]*second[j+3][k+4];
        sum = sum + first[i][k+5]*second[j+3][k+5];
        sum = sum + first[i][k+6]*second[j+3][k+6];
        sum = sum + first[i][k+7]*second[j+3][k+7];

        sum = sum + first[i][k]*second[j+4][k];
        sum = sum + first[i][k+1]*second[j+4][k+1];
        sum = sum + first[i][k+2]*second[j+4][k+2];
        sum = sum + first[i][k+3]*second[j+4][k+3];
        sum = sum + first[i][k+4]*second[j+4][k+4];
        sum = sum + first[i][k+5]*second[j+4][k+5];
        sum = sum + first[i][k+6]*second[j+4][k+6];
        sum = sum + first[i][k+7]*second[j+4][k+7];

        sum = sum + first[i][k]*second[j+5][k];
        sum = sum + first[i][k+1]*second[j+5][k+1];
        sum = sum + first[i][k+2]*second[j+5][k+2];
        sum = sum + first[i][k+3]*second[j+5][k+3];
        sum = sum + first[i][k+4]*second[j+5][k+4];
        sum = sum + first[i][k+5]*second[j+5][k+5];
        sum = sum + first[i][k+6]*second[j+5][k+6];
        sum = sum + first[i][k+7]*second[j+5][k+7];

        sum = sum + first[i][k]*second[j+6][k];
        sum = sum + first[i][k+1]*second[j+6][k+1];
        sum = sum + first[i][k+2]*second[j+6][k+2];
        sum = sum + first[i][k+3]*second[j+6][k+3];
        sum = sum + first[i][k+4]*second[j+6][k+4];
        sum = sum + first[i][k+5]*second[j+6][k+5];
        sum = sum + first[i][k+6]*second[j+6][k+6];
        sum = sum + first[i][k+7]*second[j+6][k+7];

        sum = sum + first[i][k]*second[j+7][k];
        sum = sum + first[i][k+1]*second[j+7][k+1];
        sum = sum + first[i][k+2]*second[j+7][k+2];
        sum = sum + first[i][k+3]*second[j+7][k+3];
        sum = sum + first[i][k+4]*second[j+7][k+4];
        sum = sum + first[i][k+5]*second[j+7][k+5];
        sum = sum + first[i][k+6]*second[j+7][k+6];
        sum = sum + first[i][k+7]*second[j+7][k+7];

          j=j+8;
          k=k+8;
        }
        else {
          sum = sum + first[i][k]*second[j][k];
          k++;
        }
      }
    }
  }
  return 0;
}
```

```
paszynsk@atari:~/optimize/MM_dtime$ gcc -O3 mm1.c    // automatic optimization

paszynsk@atari:~/optimize/MM_dtime$ ./a.out

Time: 2.720970e-01

paszynsk@atari:~/optimize/MM_dtime$ gcc mm6.c //blocking in rows

paszynsk@atari:~/optimize/MM_dtime$ ./a.out

Time: 5.649290e-01

paszynsk@atari:~/optimize/MM_dtime$ gcc mm7.c //blocking in rows and columns

paszynsk@atari:~/optimize/MM_dtime$ ./a.out

Time: 8.970000e-03
```