HPC Spring 2017 – Project 1

2 Determine Machine Timer Accuracy

Q:For each timer tested, report its resolution and precision
-DUSE_TIMES
Timer resolution is 0.010000000 seconds
With MINSEC=0.1 timing precision is at least 1 digit(s)

-DUSE_GETRUSAGE
Timer resolution is less than or equal to 0.000001000 seconds
With MINSEC=0.1 timing precision is at least 5 digit(s)

-DUSE_GETTIMEOFDAY
Timer resolution is 0.000001000 seconds
With MINSEC=0.1 timing precision is at least 5 digit(s)

Q:Report the timer that gives the best wall-clock time accuracy and works naturally in a multi- threaded machine.
-DUSE_GETTIMEOFDAY

3 Profiling

Q:What is the CPU cycle count of line 16 and the and the L1 D-cache miss ratio (misses / refs) for line 16?
initial j,i,k order:
CPU Cycles: 123622701673 (85.20%)
L1 D-cache hit: 158038406489 (86.77%)
L1 D-cache miss: 5288053677 (81.14%)
L1 D-cache miss ratio(miss/hit): 3.35%
after revising to j,k,i order:
CPU Cycles: 93373236980 (80.05%)
L1 D-cache hit: 154715081103 (82.10%)
L1 D-cache miss: 22407006 (10.94%)
L1 D-cache miss ratio(miss/hit): 1.4483e-04
From the result above, we will find the miss ratio is significant decreased for the revised loop order.
Q:Was there any improvement in this ratio given that we changed the memory access pattern in the loop nest?

From the result above, we will find the miss ratio is significant decreased for the revised loop order.

Q:Explain the impact of the loop change based on the profiling results.What is the FP:M ratio of the code that spans the code of the innermost loop and its body[2]?How did the change improve the spatial and/or temporal locality of memory access?

After loop change the miss rate decrease significantly, that make speed of memory access increase.

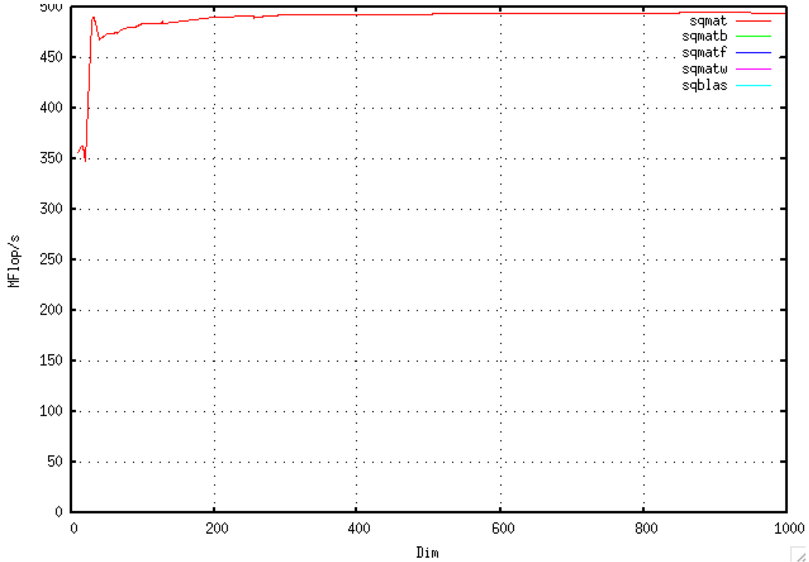
FP:M = 1:1

```
for (j = 0; j < n; j++)
    for (k = 0; k < n; k++)
        for (i = 0; i < n; i++)
            C[j*n + i] = C[j*n + i] + A[k*n + i]*B[j*n + k];
```

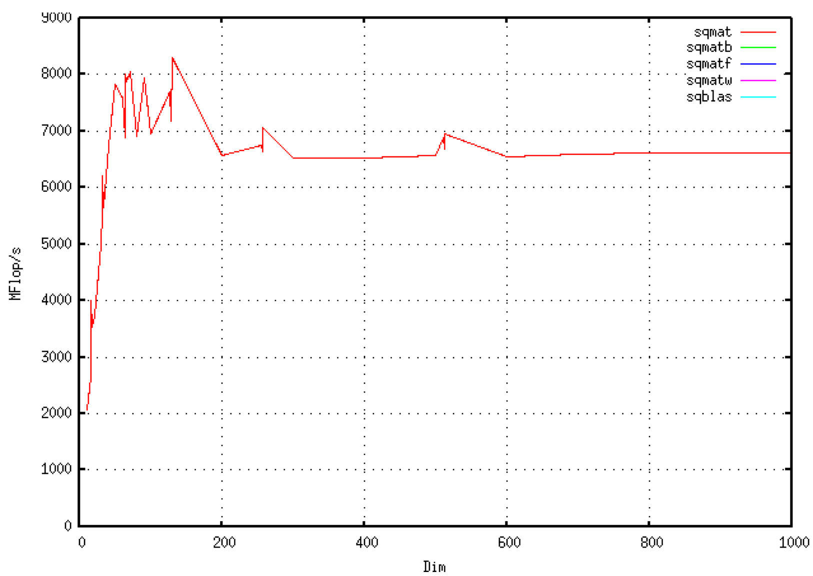
Look at the code, for inner loop, we only need to access the value of B[j*n+k] once. That will impact the spatial and temporal locality of memory access, enhancing the spatial and temporal locality means giving chance to decease miss rate.

4 Compiler Optimizations

MFlops plot
baseline:



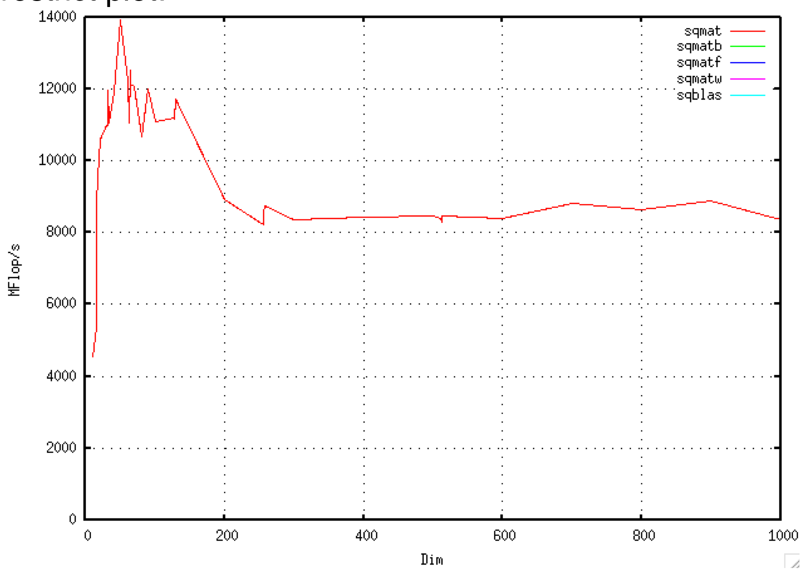
Q:Put the new plot in your report and estimate the speedup obtained with the -fast option compared to the baseline performance.
-fast:



I estimate the speedup is more than 10 times.

Q:What optimizations have been applied? Which optimization(s) do you believe have contributed significantly to the performance increase?
Dynamic-alias-disambiguation, this will contributed significantly to performance.
Also loop micro-vectorized is used

restrict plot:



Q:if any speedups were obtained with “restrict” compared to the previous experiment.?
Yes, it’s obviously from the plot.

Q:What optimizations have been applied? Which optimization(s) do you believe have contributed significantly to the performance increase?
micro-vectorized and unroll and jam
Unroll and jam contribute more.

Source file: sqmat.c
Object file: sqmat
Load Object: sqmat

```
    1. #include "global.h"
    2.
    3. void
    4. sqmat_mult(const double *restrict A, const double *restrict B, double *restrict C, int n)
      <Function: sqmat_mult>
    5. {
    6.   int i, j, k, n2;
    7.
    8.   n2 = n*n;
    9.

  Source loop below has tag L1
  L1 cloned for microvectorizing-epilog.  Clone is L12
  L1 is micro-vectorized
   10.   for (i = 0; i < n2; i++)
   11.     C[i] = 0.0;
   12.
```

```
Source loop below has tag L2
 13.   for (j = 0; j < n; j++)

Source loop below has tag L3
L3 cloned for unrolling-epilog.  Clone is L5
L5 is outer-unrolled 2 times as part of unroll and jam
 14.     for (k = 0; k < n; k++)

Source loop below has tag L4
L4 cloned for unrolling-epilog.  Clone is L7
All 2 copies of L7 are fused together as part of unroll and jam
L4 cloned for microvectorizing-epilog.  Clone is L16
L7 cloned for microvectorizing-epilog.  Clone is L14
L7 is micro-vectorized
L4 is micro-vectorized
 15.       for (i = 0; i < n; i++)
 16.         C[j*n + i] = C[j*n + i] + A[k*n + i]*B[j*n + k];
 17. }
```

Compile flags: /panfs/storage.local/opt/studio/developerstudio12.5/bin/cc -xinline=no -xarch=native -g -fast -I. -DUSE_GETTIMEOFDAY -DUSE_SUNPERF -DCOMPILER='"suncc"' -DOFLAGS='"-xinline=no -xarch=native -g -fast"' -DMFLAGS='"-DUSE_GETTIMEOFDAY -DUSE_SUNPERF"' -DPLATFORM='"x86_64-unknown-linux-gnu"' -xlic_lib=sunperf -lm sqmat.c -W0,-xp.XAQ6qKEcuKuYkny.

Q: Edit sqmat.c to modify the test sizes as follows for three profiling experiments we will conduct:

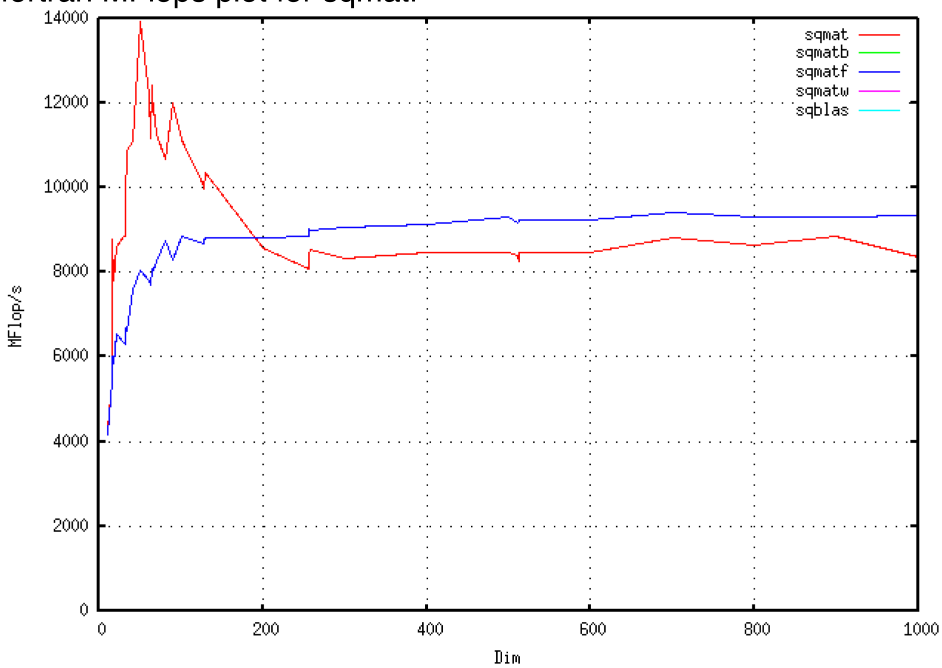
• Small : int sqmat_test_size[] = { 50, 60, 70, 80, 90, 100 };
CPU cycle count:3425078832 (93.86%)
L1 D- cache hits:2172178177 (96.02%)
L1 D- cache misses:393725360 (100.00%)
L1 D-cache miss ratio(miss/hit):18.13%

• Medium: int sqmat_test_size[] = { 200, 300, 400, 500, 600, 700, 800 };
CPU cycle count:3809196747 (88.81%)
L1 D- cache hits:1081083014 (95.58%)
L1 D- cache misses:361714703 (82.48%)
L1 D-cache miss ratio(miss/hit):33.46%

• Large: int sqmat_test_size[] = { 900, 1000 };
CPU cycle count:2208695629 (76.67%)
L1 D- cache hits:440441444 (89.80%)
L1 D- cache misses:393725434 (80.39%)
L1 D-cache miss ratio(miss/hit):89.39%

5 Fortran vs C

fortran MFlops plot for sqmatf




```
<Function: sqmult_>
        SUBROUTINE SQMULT(A, B, C, n)
        DOUBLE PRECISION A(n,n), B(n,n), C(n,n)

Source loop below has tag L2
L2 interchanged with L1
L2 cloned for microvectorizing-epilog.  Clone is L6
L2 is micro-vectorized
        DO i = 1,n

Source loop below has tag L1
L1 interchanged with L2
          DO j = 1,n
            C(i,j) = 0
          ENDDO
        ENDDO

!       TODO: Implement the rest of matrix multiply

Function __f95_dgemm_ not inlined because the compiler has not seen the body of the
function.  Use -xcrossfile or -xipo in order to inline it
        DO j = 1,n
          DO k = 1,n
            DO i = 1,n
              C(i,j) = C(i,j) + A(i,k) * B(k,j)
```

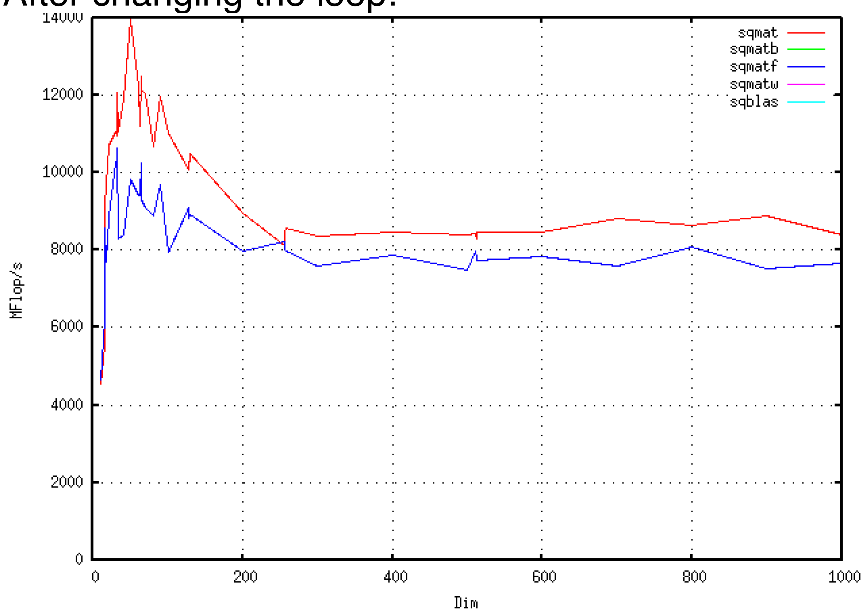
```
          ENDDO
        ENDDO
      ENDDO


      END
```

Compile flags: /panfs/storage.local/opt/studio/developerstudio12.5/bin/f90 -inline=no -xarch=native -g -fast -I. -DCOMPILER='"sunf95"' -DOFLAGS='"-inline=no -xarch=native -g -fast"' -DMFLAGS='"-DUSE_GETTIMEOFDAY -DUSE_SUNPERF"' -DPLATFORM='"x86_64-unknown-linux-gnu"' -c -qoption f90comp -h.XAQ6qKD7yduYkbj. sqmatf.f

Compiler has raise the degmm to optimize the multiplication.
f95 dgemm benchmark is a simple, multi-threaded, dense-matrix multiply benchmark. The code is designed to measure the sustained, floating-point computational rate of a single node.

Q:Try a different loop ordering in sqmatf.f to compile and run as a separate experiment and report what happens in that case, i.e. compiler optimizations and timings.

After changing the loop:



<Function: sqmult_>

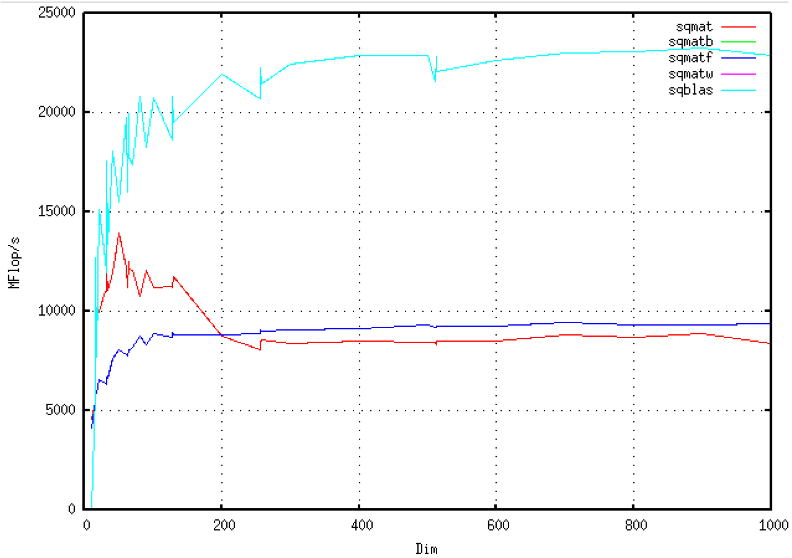
```
      SUBROUTINE SQMULT(A, B, C, n)
      DOUBLE PRECISION A(n,n), B(n,n), C(n,n)

   Source loop below has tag L2
   L2 interchanged with L1
   L2 cloned for microvectorizing-epilog.  Clone is L21
   L2 is micro-vectorized
      DO i = 1,n

   Source loop below has tag L1
```

```
L1 interchanged with L2
        DO j = 1,n
          C(i,j) = 0
        ENDDO
      ENDDO

!     TODO: Implement the rest of matrix multiply

Source loop below has tag L5
      DO j = 1,n

Source loop below has tag L4
L4 interchanged with L3
L4 blocked by 32 for improved memory hierarchy performance, new inner loop L10
L4 cloned for unrolling-epilog.  Clone is L16
All 2 copies of L16 are fused together as part of unroll and jam
L4 cloned for microvectorizing-epilog.  Clone is L25
L16 cloned for microvectorizing-epilog.  Clone is L23
L16 is micro-vectorized
L4 is micro-vectorized
        DO i = 1,n
          DO k = 1,n
            C(i,j) = C(i,j) + A(i,k) * B(k,j)
          ENDDO
        ENDDO
      ENDDO
```

Compile flags: /panfs/storage.local/opt/studio/developerstudio12.5/bin/f90 -inline=no -xarch=native -g -fast -I. -DCOMPILER='"sunf95"' -DOFLAGS='"-inline=no -xarch=native -g -fast"' -DMFLAGS='"-DUSE_GETTIMEOFDAY -DUSE_SUNPERF"' -DPLATFORM='"x86_64-unknown-linux-gnu"' -c -qoption f90comp -h.XAQ6qKDE8duYk0o. sqmatf.f

We will find this kind of loop type do not use degmm is slower.

6 BLAS Level 3: DGEMM



Q: From the plots, compare performances of the sqmat, sqmatf, and sqblas benchmarks.
We will find the the performance is
sqblas(using degmm)>sqmatf>sqmat(without degmm), when test size dim more than 200.
sqmat>sqmatf, when test size dim under 200

7 Blocking

```
 int ii,jj,kk,iibound,jjbound,kkbound;
  iibound = (i + BLKSIZE) < n ? (i + BLKSIZE):n;
  jjbound = (j + BLKSIZE) < n ? (j + BLKSIZE):n;
  kkbound = (k + BLKSIZE) < n ? (k + BLKSIZE):n;
  for (ii = i; ii < iibound; ii++)
    for(jj = j; jj < jjbound; jj++)
      for(kk = k; kk < kkbound; kk++)
            C[jj*n + ii] = C[jj * n + ii] + A[kk * n + ii]*B[jj*n + kk];
```

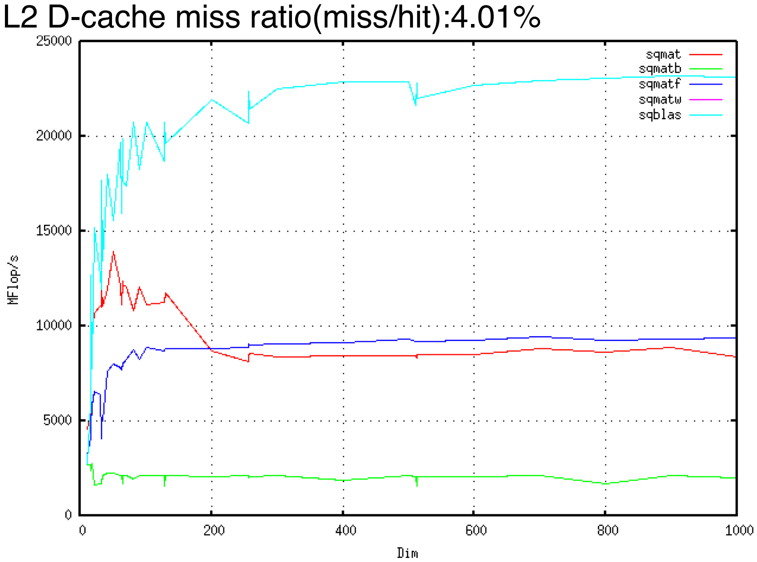
I choose the block size as 32, because I found when we run the order of j,i,k in fortran, it use 32 as the block size for unroll and jam. That's because of the memory hierarchy, when we using the block of 2^n, it can decrease the miss ratio.

sqmat:

L1 D- cache hits:18028068993 (97.25%)
L1 D- cache misses:1927013961 (90.94%)
L1 D-cache miss ratio(miss/hit):10.69%
L2 D- cache hits:1309217094 (98.08%)
L2 D- cache misses:603617452 (77.01%)
L2 D-cache miss ratio(miss/hit):46.11%

sqmatb:

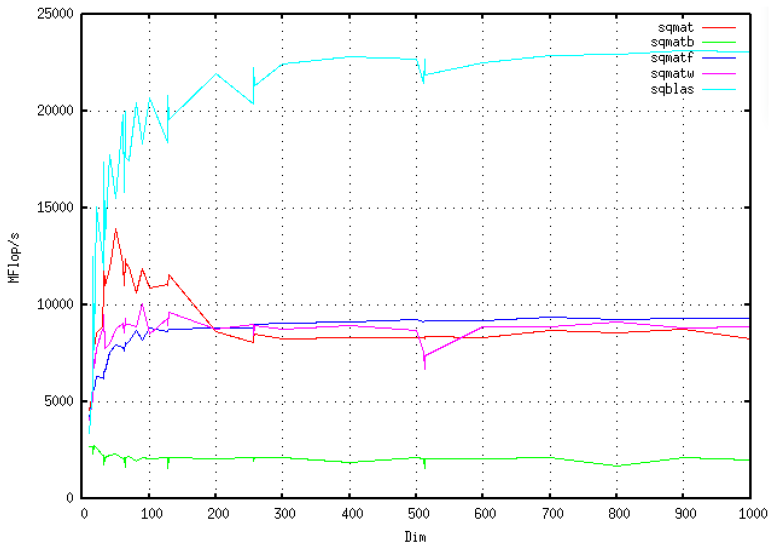
L1 D- cache hits:25485543576 (94.75%)
L1 D- cache misses:822666599 (80.06%)
L1 D-cache miss ratio(miss/hit):3.23%
L2 D- cache hits:774652722 (96.03%)
L2 D- cache misses:31031882 (14.42%)
L2 D-cache miss ratio(miss/hit):4.01%



Q: What is a possible explanation of the slower parts and what could be causing these spikes?

For the memory hierarchy, at size of 2^n, we need to load the next level memory and it will take some time.

8 Winograd's Algorithm




```
SUBROUTINE SQMULT(A, B, C, n)
DOUBLE PRECISION A(n,n), B(n,n), C(n,n),x(n),y(n),sum
INTEGER i,j,k

DO i = 1,n
  x(i)=0
  y(i)=0
  DO j = 1,n
    C(i,j) = 0
  ENDDO
ENDDO
!      TODO: Implement Winograd's matrix multiply
DO i = 1,n
  DO k= 1,n/2
    x(i) = x(i)+A(i,2*k-1)*A(i,2*k)
    y(i) = y(i)+B(2*k-1,i)*B(2*k,i)
  ENDDO
ENDDO

 DO i=1,n
   DO j=1,n
     C(i,j)=-x(i)-y(j)
     DO k=1,n/2
       sum = (A(i,2*k)+B(2*k-1,j))*(A(i,2*k-1)+B(2*k,j))
       C(i,j)=C(i,j)+sum
     ENDDO
     IF (MOD(n,2)==1) THEN
         C(i,j)=C(i,j)+A(i,n)*B(n,j)
     END IF
```

```
   ENDDO
ENDDO

END
```

FP:M of Winograd's algorithm is 1:1
FP:M of basic square matrix multiply is 1:1