September 26, 2016

Fast Matrix-Matrix multiplication

In this class:

- *How the computer architecture affects how we implement an algorithm*

Next class

Solving Ax = b
G&C – Chapter 7

Next next class

HOMEWORK DUE!
More Ax = b
G&C – Chapter 7

Matrix methods

Matrix multiplication	$\boldsymbol{AB} = \boldsymbol{C}$
Linear systems	$\mathbf{Ax} = \mathbf{b}$
Eigenvalue problems	$\mathbf{Ax} = \lambda\mathbf{x}$

GEMM

GEneral Matrix-Multiply

One of the BLAS routines

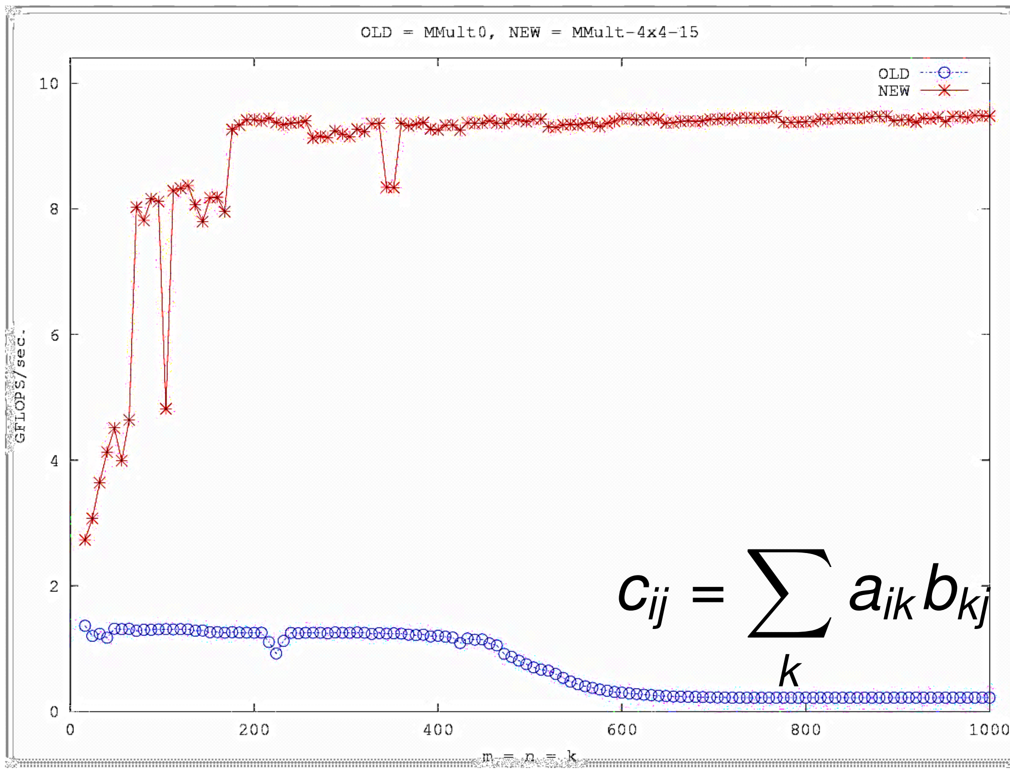
“Basic Linear Algebra Subroutines”

Used by LAPACK

“Linear Algebra” Package

Solving Ax=b, Eigenvalues, etc.

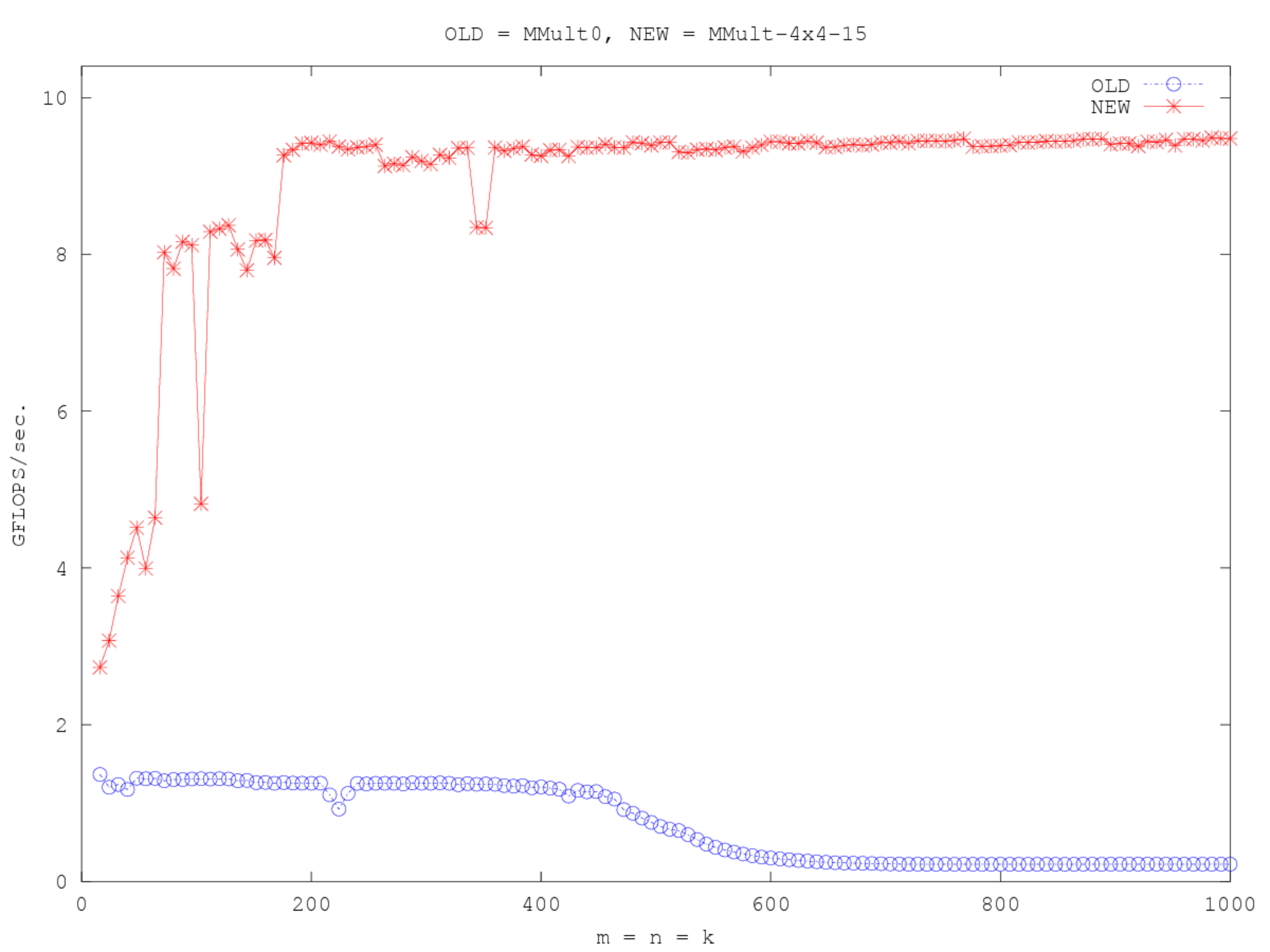
Optimizing GEMM



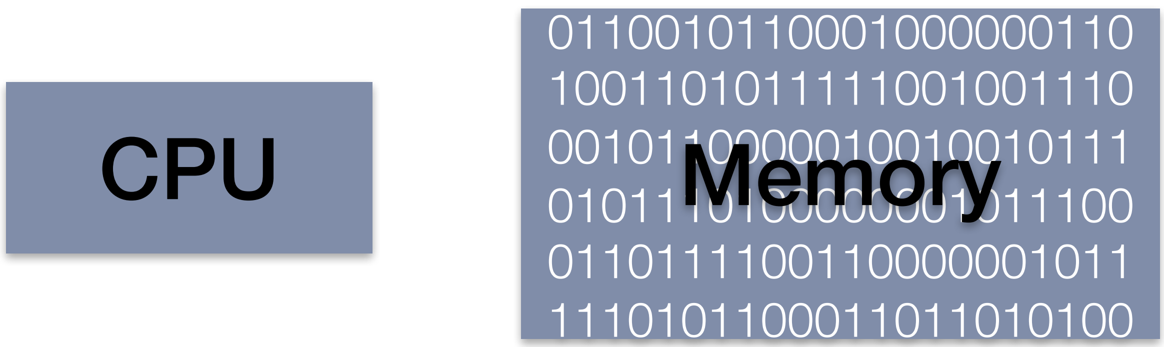
$$c_{ij} = \sum_k a_{ik} b_{kj} \text{ for all } i, j$$

https://github.com/flame/how-to-optimize-gemm/wiki

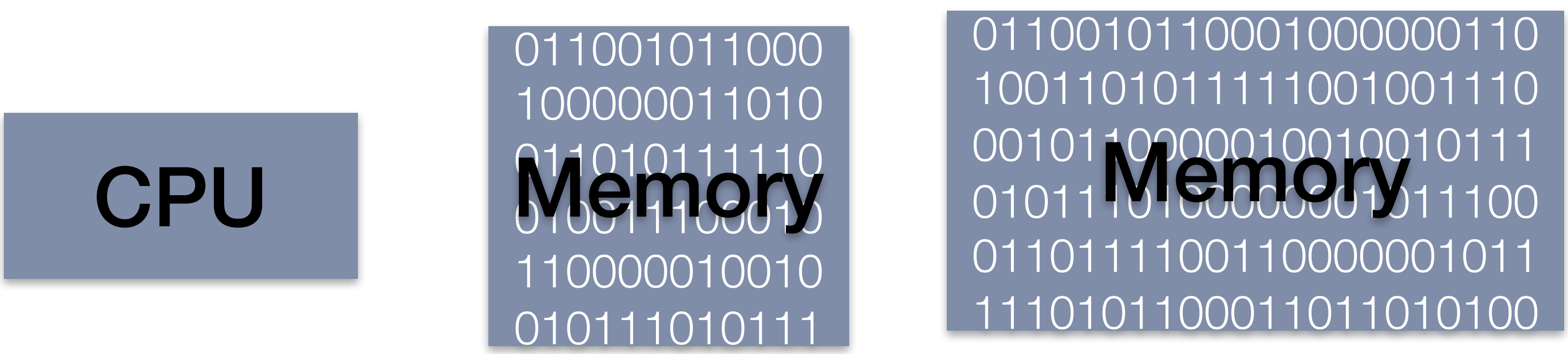
By Robert Van De Gejin


OLD = MMult0, NEW = MMult-4x4-15
OLD
NEW
GFLOPS/sec.
0
2
4
6
8
10
0
200
400
600
800
1000
m = n = k

What a computer looks like!
(To a programming language)



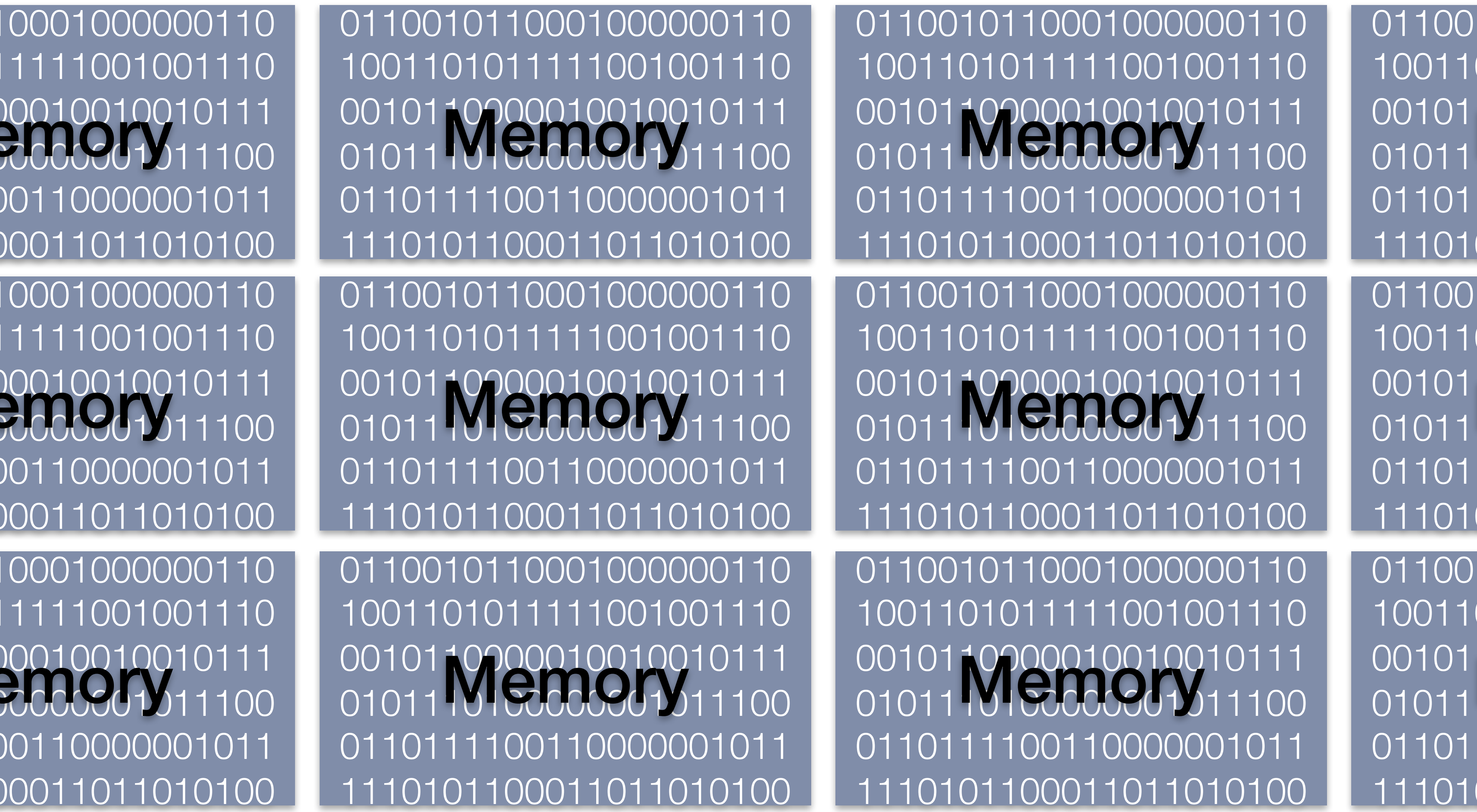
What a computer looks like!
(To an algorithm)



Super fast memory!

Fast memory!

Your RAM


01100101100010000000110
10011010111111001001110
Memory
01101111001100000001011
11101011000110111010100
01100101100010000000110
10011010111111001001110
Memory
01101111001100000001011
11101011000110111010100
01100101100010000000110
10011010111111001001110
Memory
01101111001100000001011
11101011000110111010100
01100101100010000000110
10011010111111001001110
Memory
01101111001100000001011
11101011000110111010100
01100101100010000000110
10011010111111001001110
Memory
01101111001100000001011
11101011000110111010100
01100101100010000000110
10011010111111001001110
Memory
01101111001100000001011
11101011000110111010100

But …

CPU – Access in 0.5 nanoseconds – 32 doubles

L1 - Super-fast – Access 1-2 nanoseconds – 1000 doubles

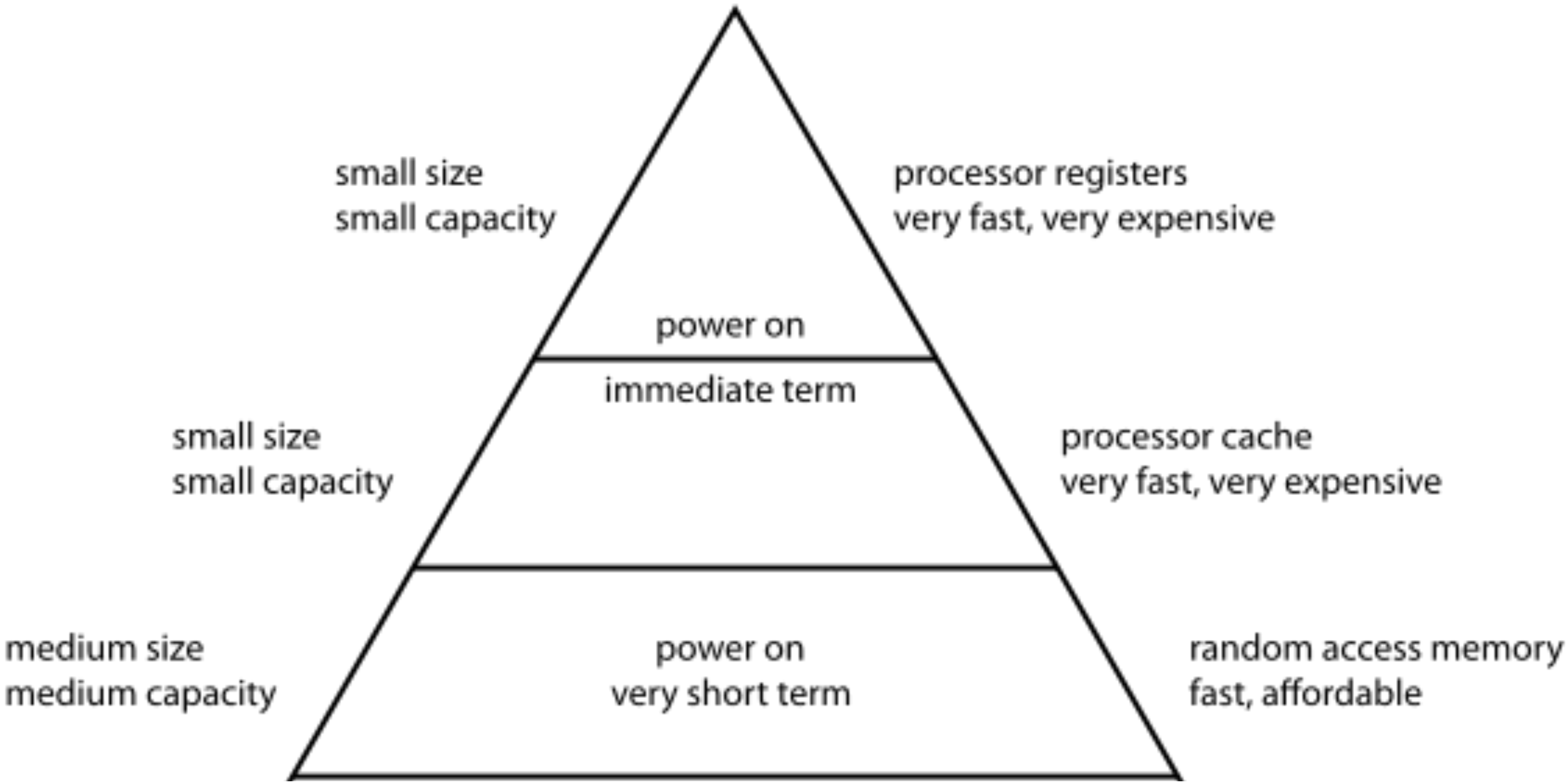
L2 - Fast – Access in 5-10 nanoseconds – 20000 doubles

L3 - Fast- - Access in 10 nanoseconds – 250,000 doubles

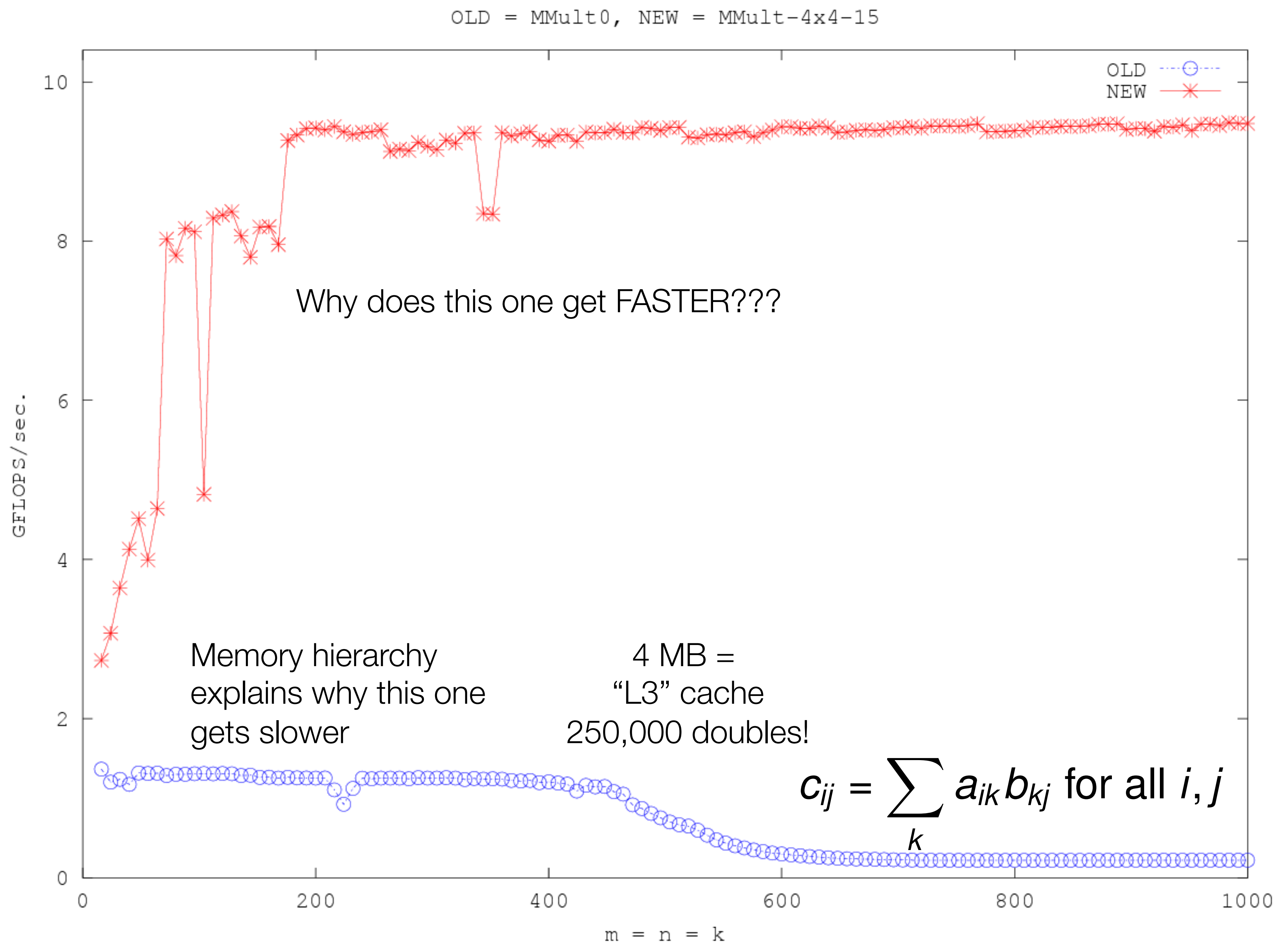
RAM – Access in 25-100 nanoseconds – 100,000,000 doubles

Each layer is much slower!

Computer Memory Hierarchy



http://en.wikipedia.org/wiki/Memory_hierarchy

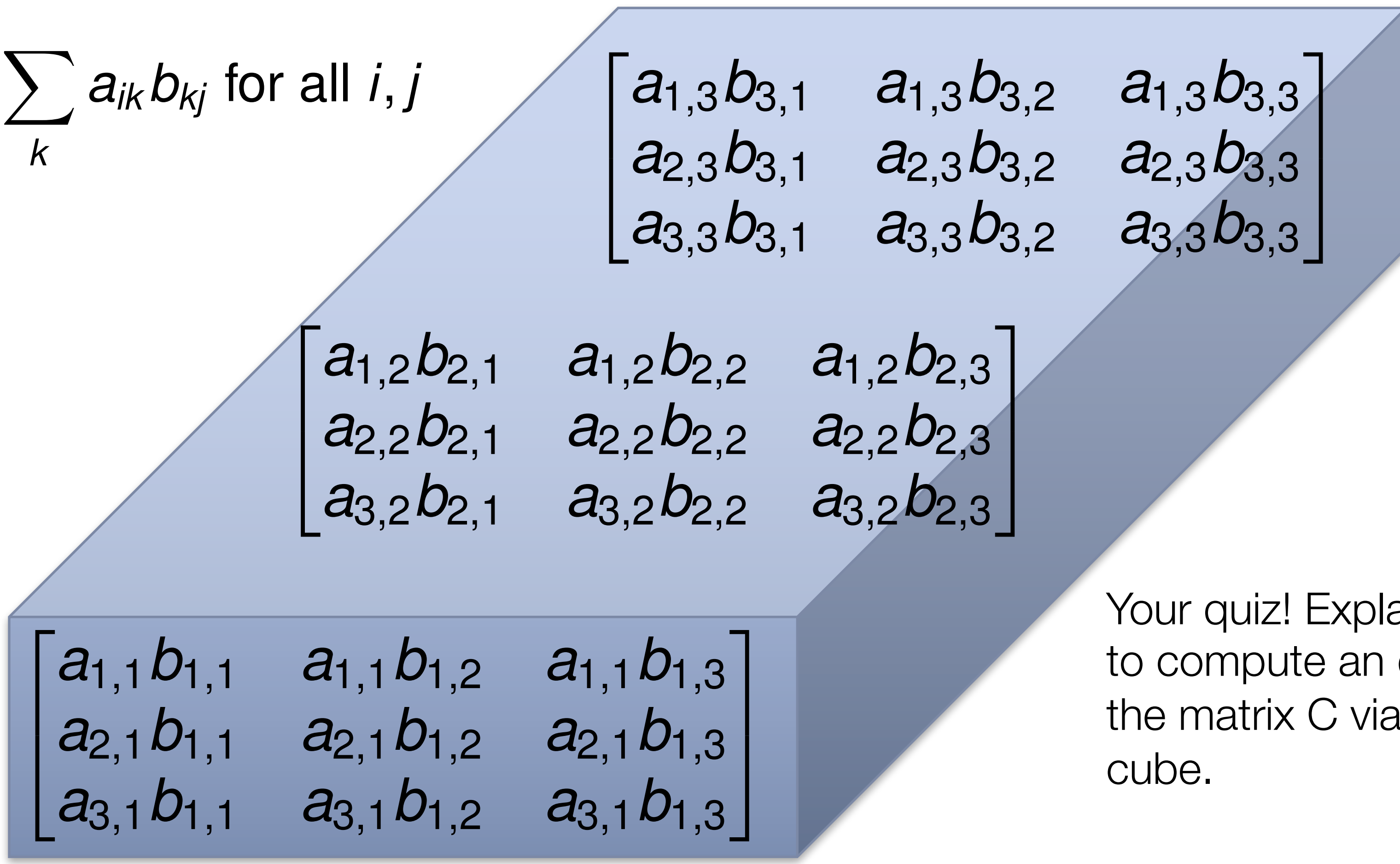


OLD = MMult0, NEW = MMult-4x4-15
OLD
NEW
GFLOPS/sec.
m = n = k
0
2
4
6
8
10
0
200
400
600
800
1000
Why does this one get FASTER???
Memory hierarchy
explains why this one
gets slower
4 MB =
"L3" cache
250,000 doubles!
$c_{ij} = \sum_k a_{ik} b_{kj}$ for all i, j

The MatMul Cube!

$$\boldsymbol{A} = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} \quad \boldsymbol{B} = \begin{bmatrix} b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,1} & b_{3,2} & b_{3,3} \end{bmatrix}$$

$$c_{ij} = \sum_k a_{ik} b_{kj} \text{ for all } i, j$$



Your quiz! Explain how to compute an entry in the matrix C via this cube.

```
void MY_MMult( int m, int n, int k,
double *a, int lda,
double *b, int ldb,
double *c, int ldc ){
  int i, p, pb, ib;  /* This time, we
compute a mc x n block of C by a call
to the InnerKernel */
  for ( p=0; p<k; p+=kc ){
    pb = min( k-p, kc );
    for ( i=0; i<m; i+=mc ){
      ib = min( m-i, mc );
      InnerKernel( ib, n, pb,
        &A( i,p ), lda,
        &B(p, 0 ), ldb,
        &C( i,0 ), ldc, i==0 );
    }
  }
}
```

```
void InnerKernel( int m, int n, int k,
double *a, int lda,
double *b, int ldb,
double *c, int ldc, int first_time ){
  int i, j;
  double packedA[ m * k ];
  static double packedB[ kc*nb ]; /*
Note: using a static buffer is not
thread safe... */
  for ( j=0; j<n; j+=4 ){  /* Loop over
the columns of C, unrolled by 4 */
    if ( first_time )
      PackMatrixB( k, &B( 0, j ), ldb,
         &packedB[ j*k ] );
    for ( i=0; i<m; i+=4 ){
       /* Loop over the rows of C */
       /* Update C( i,j ), C( i,j+1 ),
C( i,j+2 ), and C( i,j+3 ) in
one routine (four inner products) */
       if ( j == 0 )
         PackMatrixA( k, &A( i, 0 ),
           lda, &packedA[ i*k ] );
       AddDot4x4( k, &packedA[ i*k ],
         4, &packedB[ j*k ], k,
         &C( i,j ), ldc );
    }
  }
}
```