

Effective Caching for Matrix Multiplication

Cache Memory

- ▶ An important part of the memory hierarchy in any computer system
- ▶ Utilizes different technology compared to RAM
 - ▶ Faster
 - ▶ Uses more power
 - ▶ Costs more
- ▶ Gives the illusion of a large (RAM size), fast (cache speed) memory
- ▶ Compromise between cost, access time and size
- ▶ Program execution depends on efficiently utilizing the cache

Matrix Multiplication

- ▶ Matrices and matrix multiplication are very common in a variety of Engineering and Scientific Computing problems
- ▶ An $n \times n$ matrix A is mathematically written as:

$$A = \begin{bmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,n-1} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n-1,0} & a_{n-1,1} & \cdots & a_{n-1,n-1} \end{bmatrix}$$

- ▶ Matrix multiplication is mathematically written as:

$$C = AB$$

- ▶ Where A , B and C are all $n \times n$ matrices

Matrix Multiplication

- ▶ Each element in C requires n multiplications and $n - 1$ additions
- ▶ Each element in C is computed by multiplying a row of matrix A with a column of matrix B , for example,

$$\begin{bmatrix} c_{0,0} & c_{0,1} & \cdots & c_{0,n-1} \\ \boxed{c_{1,0}} & c_{1,1} & \cdots & c_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n-1,0} & c_{n-1,1} & \cdots & c_{n-1,n-1} \end{bmatrix} = \begin{bmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,n-1} \\ \boxed{a_{1,0} & a_{1,1} & \cdots & a_{1,n-1}} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n-1,0} & a_{n-1,1} & \cdots & a_{n-1,n-1} \end{bmatrix} * \begin{bmatrix} \boxed{b_{0,0}} & b_{0,1} & \cdots & b_{0,n-1} \\ \boxed{b_{1,0}} & b_{1,1} & \cdots & b_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ \boxed{b_{n-1,0}} & b_{n-1,1} & \cdots & b_{n-1,n-1} \end{bmatrix}$$

- ▶ $c_{1,0}$ is the sum of the products of the elements of row 1 of matrix A and column 0 of matrix B
- ▶ Or more generally,

$$c_{i,j} = \sum_{k=0}^{n-1} a_{i,k} b_{k,j}$$

Matrix Memory Access

- ▶ The main memory of a computer system can be thought of as a long linear array, where elements are stored at consecutive memory locations
- ▶ A matrix is stored in memory either by rows (row major order) or columns (column major order)
- ▶ The **C** language uses row major order, while Fortran uses column major order
- ▶ A matrix in **C** is stored as:



- ▶ Optimal cache utilization occurs when sequential memory elements are accessed
 - ▶ For the matrix multiplication example above, matrix A is efficiently accessed but matrix B is not

Matrix Multiplication Transpose

- ▶ Element access to matrix B can be made cache efficient by storing matrix B in column major order
- ▶ This is the same as computing the *transpose* of B
- ▶ In this case, the elements of the rows of matrix A are multiplied by the elements of the rows of matrix B

$$\begin{bmatrix} c_{0,0} & c_{0,1} & \cdots & c_{0,n-1} \\ c_{1,0} & c_{1,1} & \cdots & c_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n-1,0} & c_{n-1,1} & \cdots & c_{n-1,n-1} \end{bmatrix} = \begin{bmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,n-1} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n-1,0} & a_{n-1,1} & \cdots & a_{n-1,n-1} \end{bmatrix} * \begin{bmatrix} b_{0,0} & b_{1,0} & \cdots & b_{n-1,0} \\ b_{0,1} & b_{1,1} & \cdots & b_{n-1,1} \\ \vdots & \vdots & \ddots & \vdots \\ b_{0,n-1} & b_{1,n-1} & \cdots & b_{n-1,n-1} \end{bmatrix}$$

$$c_{i,j} = \sum_{k=0}^{n-1} a_{i,k} b_{j,k}$$

Matrix Multiplication

- ▶ Matrix multiplication using the transpose of the B matrix more efficiently utilizes the cache
- ▶ Both matrix multiplication techniques inefficiently use the cache because the data must be brought into the cache multiple times
 - ▶ Each row of matrix A is multiplied by each row of matrix B
 - ▶ When the matrices are large, the rows of matrix B will need to be brought into the cache for each row in matrix A

Block Matrix Multiplication

- ▶ A better way of utilizing the cache is to perform block matrix multiplication, where the matrices are broken into blocks or tiles
- ▶ For example, a 4 X 4 matrix A can be made of 2 X 2 blocks or tiles each of size 2 X 2

$$A = \begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} a_{0,0} & a_{0,1} \end{bmatrix} & \begin{bmatrix} a_{0,2} & a_{0,3} \end{bmatrix} \\ \begin{bmatrix} a_{1,0} & a_{1,1} \end{bmatrix} & \begin{bmatrix} a_{1,2} & a_{1,3} \end{bmatrix} \\ \begin{bmatrix} a_{2,0} & a_{2,1} \end{bmatrix} & \begin{bmatrix} a_{2,2} & a_{2,3} \end{bmatrix} \\ \begin{bmatrix} a_{3,0} & a_{3,1} \end{bmatrix} & \begin{bmatrix} a_{3,2} & a_{3,3} \end{bmatrix} \end{bmatrix} = \begin{bmatrix} A_{0,0} & A_{0,1} \\ A_{1,0} & A_{1,1} \end{bmatrix}$$

- ▶ Block matrix multiplications is

$$\begin{bmatrix} C_{0,0} & C_{0,1} \\ C_{1,0} & C_{1,1} \end{bmatrix} = \begin{bmatrix} A_{0,0} & A_{0,1} \\ A_{1,0} & A_{1,1} \end{bmatrix} \begin{bmatrix} B_{0,0} & B_{0,1} \\ B_{1,0} & B_{1,1} \end{bmatrix}$$

Block Matrix Multiplication

- The product matrix

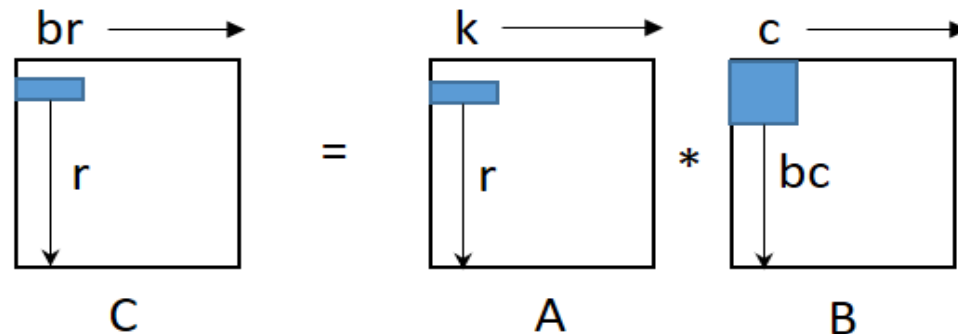
$$C_{0,0} = A_{0,0}B_{0,0} + A_{0,1}B_{1,0}$$

$$C_{0,1} = A_{0,0}B_{0,1} + A_{0,1}B_{1,1}$$

$$C_{1,0} = A_{1,0}B_{0,0} + A_{1,1}B_{1,0}$$

$$C_{1,1} = A_{1,0}B_{0,1} + A_{1,1}B_{1,1}$$

- This technique efficiently utilizes the cache because all the elements in a block in matrix B are completely used for all computations and are not needed again

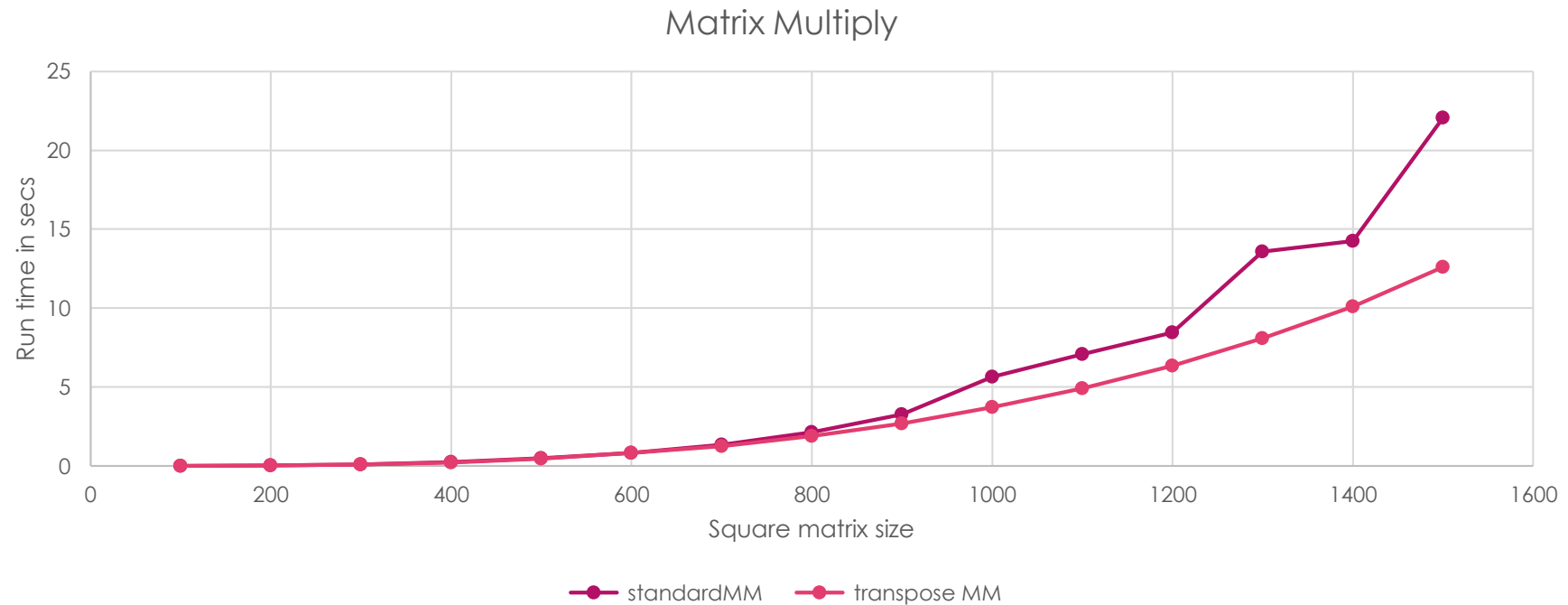


Matrix Multiply Program

- ▶ Two programs were written to time the standard matrix multiplication with both the transpose and the block matrix techniques
- ▶ Command line arguments are used to determine the size of the matrices
- ▶ All matrices utilize double-precision floating-point numbers
- ▶ Computational results are compared to ensure the product matrix is correctly determined

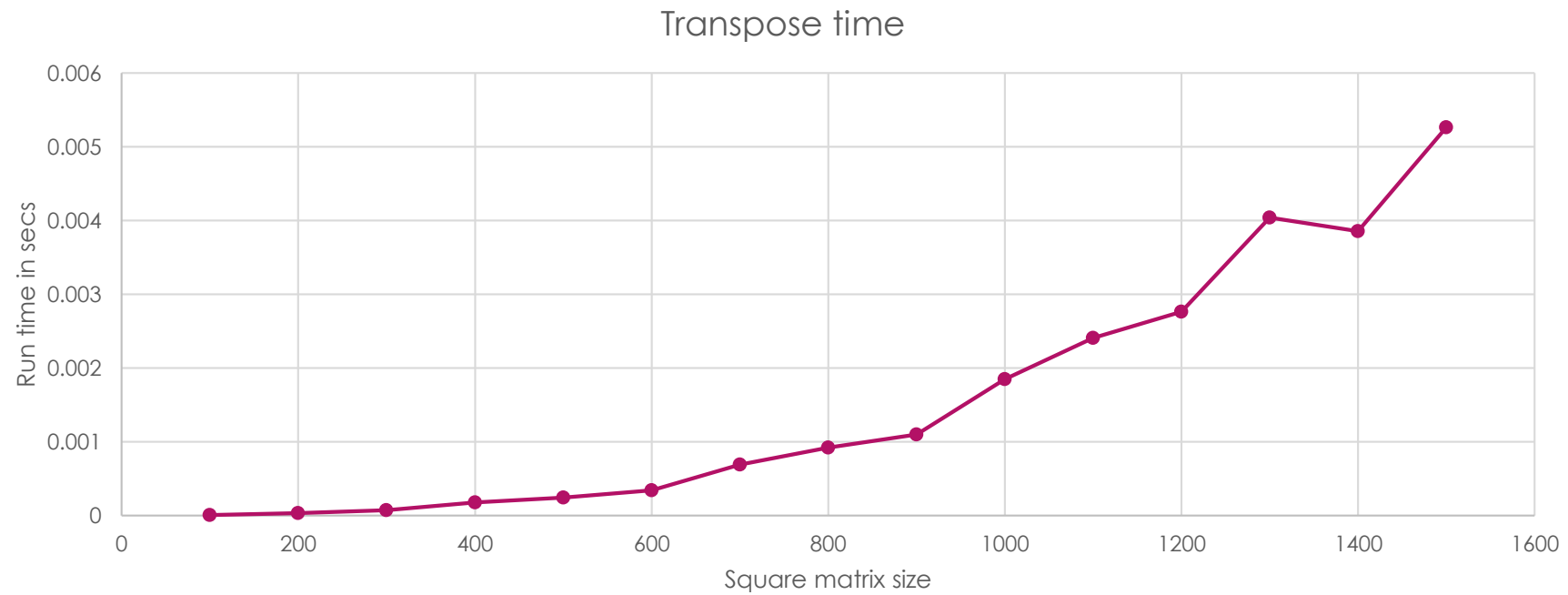
Program Results

- Timing results for standard matrix multiplication compared to transpose matrix multiplication



Program Results

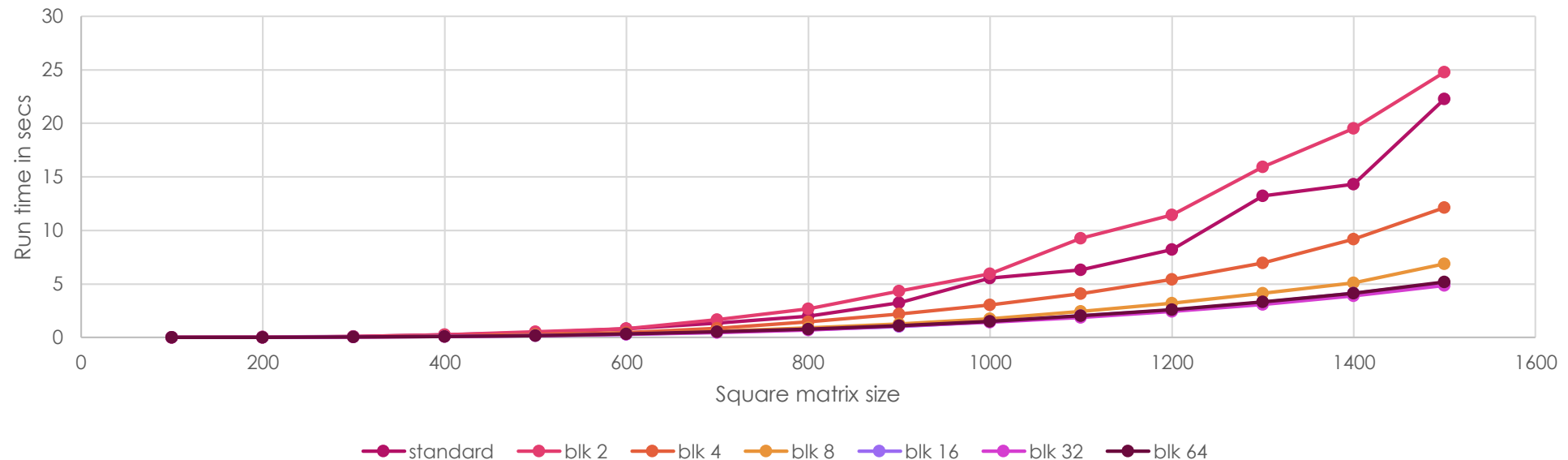
- The time to transpose the matrix is negligible



1

- The time to perform block matrix multiplication depends on the size of the block
- Blocks of size 8 X 8 or 16 X 16 appear to give the best performance

Block Matrix Multiply with doubles



Program Results

- ▶ Block matrix multiply with blocks of size 16 X 16 appear to give the best performance

