SACRAMENTO STATE

CSc 131
Computer Software Engineering

Chapter 8
Matrix Multiply

Herbert G. Mayer, CSU CSc

Status 10/27/2019

Syllabus

- Definition
- Goal
- P1 Store Max Value
- P2 Matrix Multiply
- Summary
- References

Definition

- A matrix is a two-dimensional data object, organized in rows and columns
- Common operations on matrices are the "Matrix Multiply", and "Cramer's Rule" for solving multiple unknowns in linearly independent equations
- Many other uses!
- Focus here getting acquainted with simple matrices operations
- Implement two projects, P1 Store Max Value and P2 Matrix Multiply

Goal of P1 Store Max Value

- Project **P1 Store Max Value** initializes all elements of a 2-Dim square integer matrix
- For each row, **P1** extracts the row's largest integer, and stores it in an extra element at the end of that row, at index a[SZ], **SZ** being the symbolic integer constant of the matrix size
- The actual data structure for **P1** this is not a square matrix, but a rectangular matrix, with one extra element at each row
- Finally, **P1** prints the "almost square" matrix with all max values of each row repeated in the last position of that respective row

Goals of P2 Matrix Multiply

- Project P2 Matrix Multiply is a square matrix multiply problem
- Both source matrices are square, simplifying the upper bound computation
- All elements are integers
- Pre-assign the source matrices via initialization, not by reading values from files or from the console
- Compute result into a same-sized matrix c[ ][ ]

Project P1 Store Max Value

Specify P1

- Matrix a[SZ][SZ + 1] is an integer matrix, sized via symbolic constant SZ, AKA macro in C++
- A[][] is printed twice, once before finding the maximum value of each row, and once after placing the max value of a[row] into position a[row][SZ]

Implement P1, Initialize

```
#include <iostream.h>
#define SZ 5          // small matrices
typedef int m_tp[ SZ ][ SZ + 1 ];    // use typedef!

// actual data in rectangular matrix a[ ][ ]:
m_tp a = { {  1,-2, 3,-4, 2, 0 },
           {  8, 7,-6, 5, 9, 0 },
           {  6,-5, 4,-3, 0, 0 },
           { -4, 5,-6, 7, 8, 0 },
           {  6,-5, 4, 3, 1, 0 } };

void print( char * msg, m_tp m )
{ // print
    cout << "Printing " << msg << endl;
    for( int row = 0; row < SZ; row++ ) {
        for( int col = 0; col < SZ + 1; col++ ) {
            cout << m[ row ][ col ] << " ";
            // no newline: array "known to be small" ☹
        } //end for
        cout << endl;
    } //end for
    cout << endl;
} //end print
```

Implement P1, Find Max

```
// input parameter is a[], a single-dimensional array
// find max in a[ SZ ] and store on a[ SZ ]
// i.e. object is passed as parameter, no globals

void extract( int a[] ) // a[] passed by reference
{ // extract
   int max = a[ 0 ];    // initial guess: this is max
   for( int col = 1; col < SZ; col++ ) {
      max = ( a[ col ] > max ) ? a[ col ] : max;
   } //end for
  a[ SZ ] = max;  // now we really know max
} // end extract
```

Implement P1, One Row at a Time

```
void extract_mat()
{ // extract_mat
   // handle all rows of global matrix a[][]
   for( int row = 0; row < SZ; row++ ) {
      // pass matrix row to function extract()
      extract( a[ row ] );     // handle 1 row
   } // end for
} // end extract_mat

int main()
{ // main
   print( "before", a );
   extract_mat();
   print( "after ", a );
   return 0;
} //end main
```

Project P2 Matrix Multiply

Matrix Multiply

- **Matrix Multiply:** 2 source matrices (here a[][] and b[][]) and 1 destination matrix (here x[][] below or c[][])
- Element x[i][j] is sum of all products of all elements in row a[i][*] and all elements in column b[*][i]
- Size of columns(a[][]), rows(b[][]) must match
- Rows of a[][] and columns of b[][] define c[][] (here x[][])

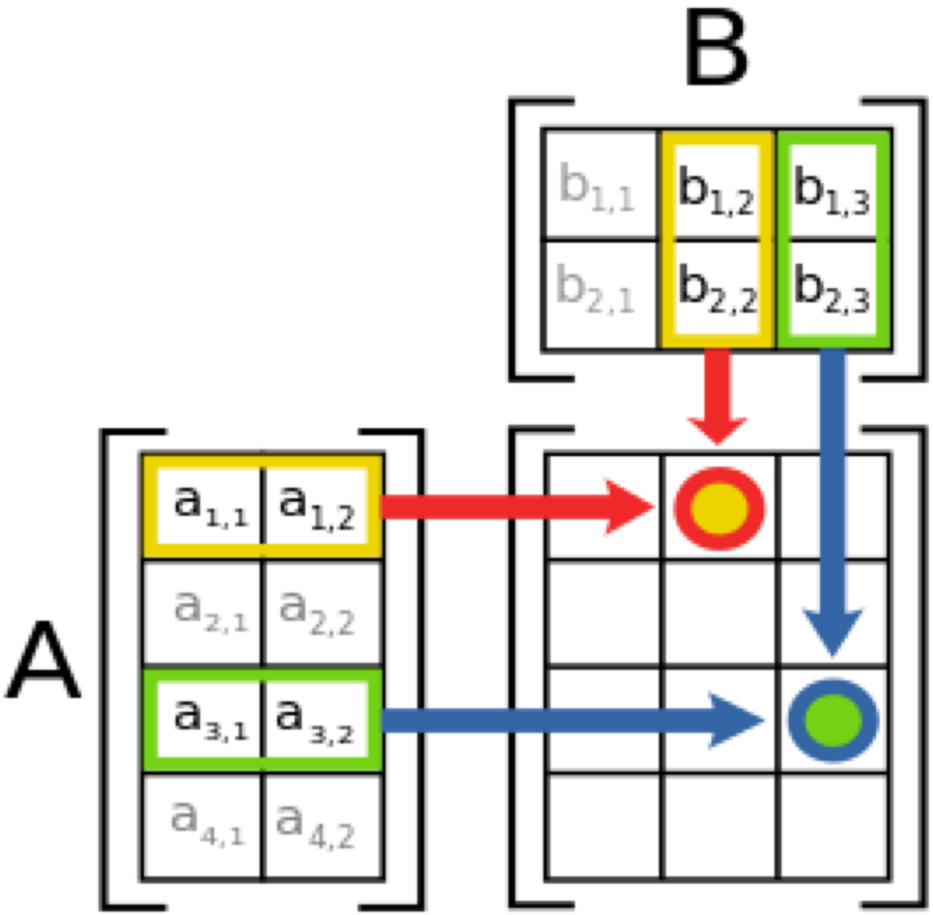
$$\overset{4\times 2 \text{ matrix}}{\begin{bmatrix} a_{11} & a_{12} \\ \cdot & \cdot \\ a_{31} & a_{32} \\ \cdot & \cdot \end{bmatrix}} \overset{2\times 3 \text{ matrix}}{\begin{bmatrix} \cdot & b_{12} & b_{13} \\ \cdot & b_{22} & b_{23} \end{bmatrix}} = \overset{4\times 3 \text{ matrix}}{\begin{bmatrix} \cdot & x_{12} & x_{13} \\ \cdot & \cdot & \cdot \\ \cdot & x_{32} & x_{33} \\ \cdot & \cdot & \cdot \end{bmatrix}}$$

Matrix Multiply

- **In mathematics and Computer Science, matrix multiplication (or matrix product, or MatMult) is an operation that produces a result matrix from two source matrices**
- **Matrix multiplication is basic tool of linear algebra**
- **Respective sizes do not have to be identical, but must match alone two dimensions each:**
- **If a[][] is an $n \times m$ matrix and b[][] is an $m \times p$ matrix, their matrix product a[][] × b[][] is an $n \times p$ matrix, in which the m entries across a row of a[][] are multiplied with the m entries down a column of b[][] and summed**
- **To produce one single matrix element each for one entry of a[][] × b[][]**

Matrix Multiply

- **Columns of a[][] match rows of b[][]**
- **Rows of a[][] match c[][]; columns of b[][] match c[][]**



P2 Specify Matrix Multiply

- Size defined via symbolic literal SZ, here 5 (small ☹)
- Square integer matrices a[][], b[][], and c[][] are used to simplify implementation of *multiply function*:

```
c[ row ][ col ] += a[ row ][k] * b[k][ col ];
```

- a[SZ][SZ] is initialized at the point of declaration
- b[row][col] = row * SZ + col initialized for all elements
- c[SZ][SZ] initialized to all elements 0 (really needed?)
- Then c[ ][ ] is recomputed via *matrix multiply*
- Matrices printed before & after matmult ☺ operation

Implement Matrix Multiply

```
// matrix multiply of square integer matrix
// all data pre-computed, not read from console

#include <iostream.h>
#define SZ 5

typedef int m_tp[ SZ ][ SZ ];

// Matrix object declarations, a[][] being initialized
m_tp a =
   {
      { 1,2,3,4,5 },
      { 8,7,6,5,4 },
      { 6,5,4,3,2 },
      { 4,5,6,7,8 },
      { 6,5,4,3,2 }
   };
  // b[][] and c[][] initialized later
  m_tp b, c;
```

Implement Matrix Multiply

```
// a[][] already set; now set b[][] and c[][]
// all 3 matrices are global: caveat!

void init( void )
{ // init
   for( int row = 0; row < SZ; row++ ) {
       for( int col = 0; col < SZ; col++ ) {
           b[ row ][ col ] = row * SZ + col;
           // since c[][] is summed up, needs init!
           // else OK to leave uninitialized!
           c[ row ][ col ] = 0;
       } //end for col
   } //end for row
} //end init
```

Implement Matrix Multiply

```
// output a[][], b[][], and c[][], define C++ width
void print_m( char * msg, m_tp m )
{ // print_m
   cout.width( 6 );
   cout << "Printing matrix " << msg << endl;
   for( int row = 0; row < SZ; row++ ) {
       for( int col = 0; col < SZ; col++ ) {
          cout << m[ row ][ col ] << " ";
       } //end for col
       cout << endl;
   } //end for row
   cout << endl;
} //end print_m

void print( void )
{ // print
       print_m( "a", a );
       print_m( "b", b );
       print_m( "c", c );
} //end print
```

Implement Matrix Multiply

```
void mat_mult( void )
{ // mat_mult
   for( int row = 0; row < SZ; row++ ) {
       for( int col = 0; col < SZ; col++ ) {
           for( int k = 0; k < SZ; k++ ) {
               c[ row ][ col ] +=
                   a[ row ][ k ] * b[ k ][ col ];
           } //end for k
       } //end for col
   } //end for row
}  //end mat_mult

int main( void )
{ // main
    print();     // before multiplication
    mat_mult(); // do work
    print();     // after multiplication
   return 0;
} //end main
```

Discuss Matrix Multiply

- Key operation is:

```
c[ row ][ col ] =
    c[ row ][ col ] +
    a[ row ][ k ] * b[ k ][ col ];
```

- Same as actually written:

```
c[ row ][ col ] +=
    a[ row ][ k ] * b[ k ][ col ];
```

- Notice C++ specification for output width

```
cout.width( 6 ); . . . cout << m[row][col]
```

- similar to printf() of traditional C:

```
printf( "%6d", m[row][col] );
```

Summary

- Multi-dimensional data common in SW Engineering
- Matrix Multiply and related operations typical
- Matrices don't need to be square, but dimensions of a[][] and b[][] in MatMult do defines size of c[][]

References

1. Matrix multiply on Wiki: https://en.wikipedia.org/wiki/Matrix_multiplication
2. How to multiply 2 matrices: https://www.mathwarehouse.com/algebra/matrix/multiply-matrix.php