

**COS 335 Spring 2009**  
**Assignment 1 Due Tuesday Jan 27**

**Problems 1-3.** Text Ch. 3 problems 3.1, 3.4 (exclude c), 3.6 1 point each. (Remember that grades are 0-10)

The purpose of the last two problems in this assignment is to provide a demonstration that both organization and architecture of a computer can impact even relatively simple programs. The first program involves creating large arrays in virtual memory, then accessing them in row-major and column-major order. The slowdown in column-major access is an artifact of the way that C and Java compilers store arrays in memory, combined with cache assumptions about memory access and virtual memory being paged out to disk. The second program demonstrates (a) the inaccuracy of floating point arithmetic; (b) that the associative law of arithmetic does not hold for computers and (c) a peculiar effect due to the fact that Intel x87 processors do floating point arithmetic internally using 80 bits.

**Problem 4. (4 pts)** The first two programs at the end of the assignment (matrix.cpp and Matrix.java) declare a large 2 dimensional array (64 MB in this case) and then initialize the array as an identity matrix, which has 1's in all cells where the row index is equal to the column index and 0's elsewhere. The program does this three times. The first pass accesses the entire array and makes sure that all memory required by the program is physically allocated by the operating system. (This is called "touching" memory). The second pass redoes the first pass, but times the program to see how long it takes and then displays the results. Both first and second passes use "row-major" order, accessing the array row by row. The third pass uses "column-major" order, accessing the array column by column. Surely this should not make a difference - but it does.

For this problem, do the following, in one language or the other:

- a. Find out how much physical memory is present in the computer on which you are running this program.
- b. Change the MATSIZE constant for each run (note that an int is 4 bytes on a 32-bit machine) so that the array occupies the following amounts of memory  
64MB, 128MB, 256MB, 512MB, 1024MB
- c. Run the program once for each array size noted above. **Submit a report showing the output for each run and note the amount of physical memory and the operating system used on the report. Calculate memory access/sec and graph these against array size for row and column major orders.**

**The program will require several to many seconds to run for the larger array sizes, depending on machine speed. Don't panic and think your computer has crashed.**

The source code is not designed for Visual Studio IDE. It should be compiled and run the executable from the command line. I also recommend that you close ALL other applications while running the program so that the maximum amount of physical memory is available to your program.

**Notes:**

1. Array memory requirements in megabytes =  $((\text{MATSIZE} \wedge 2) * 4) / 1048576$

2. When running in a command prompt you can redirect output to a file use the output redirection operator >. For example, with the C++ exe you can give the following command:

**HW3 > run1.txt**

to place the output in a file called run1.txt

3. The C++ program will compile as is in the Visual Studio 6 IDE, from the command line in Visual Studio 6 and from the command line in Visual Studio.NET 2003/2005. To compile from the command line in VS.NET use Start > Programs > Microsoft Visual Studio .NET 2005 > Visual Studio .NET Tools > Visual Studio .NET 2005 Command Prompt to open a command prompt, then issue the command:

cl matrix.cpp

4. If you must compile in the .NET IDE, change the part of the program BEFORE int main() to

```
#include "stdafx.h"
#include <iostream>
#include <iomanip>
#include <time.h>
#define MATSIZE 4096          //modify as per assignment
using namespace std;
```

You can run this program on any type of OS that you like; Windows, Linux, Mac, Sun etc. You may also translate the program into another language and run the program using that language. If so make a note of the language used in your report. **Please submit: a) your report and b) your program if you did it in another language.**

5. To run the java program, first compile with javac (e.g., javac Matrix.java) then run with java (e.g., java -Xms100M -Xmx100M Matrix). The command line switch -Xms specifies starting heap size, -Xmx specifies max heap. These should be at least 10-20M more than the matrix size. If using a Java IDE, see your IDE documentation.

6. You can also send Java output to a file: java -Xms280M -Xmx280M Matrix > hw3.txt

**Problem 5. (2 pts).** The associative rule of addition states that for any a, b, and c;  $(a+b)+c = a+(b+c)$ . This is not necessarily true for computer addition of floating point numbers. The following program is designed to demonstrate this as well as another peculiarity of floating point arithmetic executed on Intel processors.

The C++ program float.cpp prompts for a,b and c. The code has a set of values for a, b and c in comments that will produce incorrect results for some addition sequences.

**For this assignment run the C++ code and find two additional sets of values for a, b, and c that cause the computer to show incorrect results for at least one of the three additions., and find two non-integer sets of values where all additions are correct.**

The "other peculiarity" of Intel floating point arithmetic is that results from floating point arithmetic may vary depending on whether or not intermediate values have been stored to memory during the computation. The reason for this is that the Intel x87 registers where floating point arithmetic is performed are all 80 bits wide. An IEEE 32-bit float (type float in C), however, is only 32 bits wide. When a float is loaded into an x87 register in preparation for an arithmetic operation, the processor converts it to 80 bits and all computations are performed using 80-bit arithmetic.

The java code in float.java will demonstrate the non-associativity of floating point arithmetic, but does not have the other peculiarity relating to storage of intermediate results. This is because the Java instructions are executed against a virtual machine that does NOT do 80-bit arithmetic internally.

In the following example, code is generated by the Microsoft Visual C++ 6 compiler. The mnemonics mean the following:

```
fld      Floating point load from mem, converts values to 80 bits internally
fadd     Floating point add from mem, values are converted to 80 bits before the addition
fstp    Float point store and pop. Value is saved to mem from x87 top-of-stack and the value is
        removed from the register stack. Values are converted to memory size on store.
```

We have not covered enough assembly language for you to make sense of the following instructions yet, but just read an expression such as DWORD PTR \_a\$[ebp] as a reference to the variable a. Thus the first instruction below means "load the variable a from memory."

In the assignment expression  $d = (a+b)+c$  all computation takes place in the 87 in 80 bits:

```
fld  DWORD PTR _a$[ebp]
fadd  DWORD PTR _b$[ebp]
fadd  DWORD PTR _c$[ebp]
fstp  DWORD PTR _d$[ebp]
```

As a, b, and c are loaded from memory into the x87 registers they are converted to 80 bits. The expression is effectively evaluated using the type long double for all operations. When the result is stored back to memory in d it is converted back to a 32-bit IEEE float from an 80-bit IEEE float.

However, in the following sequence:

```
d = a + b;
cout << d;
d += c;
```

We have

```
fld  DWORD PTR _a$[ebp]
fadd  DWORD PTR _b$[ebp]
fstp  DWORD PTR _d$[ebp]
; d is stored in memory in prep for call to cout
fld  DWORD PTR _d$[ebp]
fadd  DWORD PTR _c$[ebp]
fstp  DWORD PTR _d$[ebp]
```

Here the intermediate value  $a+b$  is stored to memory and converted to a 32-bit float. After the call to cout it is again loaded into the 87 and converted to 80 bits prior to the add of c. However, the intermediate result has lost its 80 bits of precision.

**Problem 6. Extra Credit: 2 points.** Compare the Java and C++ versions of the matrix code. Graph your results.

## matrix.cpp

```
#include <iostream.h>
#include <iomanip.h>
#include <time.h>

#define MATSIZE 4096          //modify as per assignment

int main() {
    // ALLOCATE HEAP SPACE
    /**IMPORTANT TO USE THE DELETE OPERATOR WHEN DONE WITH THIS MATRIX**
    int (*mat)[MATSIZE];
    mat = new int[MATSIZE][MATSIZE];
    int i,j;
    clock_t start, finish;

    cout.setf(ios::fixed | ios::showpoint);

    cout << "Starting" << endl;
    // create identity matrix. Do one pass untimed just so that we
    // "touch" and therefore physically allocate all required memory
    for (i=0; i<MATSIZE; i++)
        for (j=0; j<MATSIZE; j++)
            if (i == j)
                mat[i][j]=1;
            else
                mat[i][j]=0;

    // do it again and time it
    cout << "Timing row order access..." << endl;
    start = clock();
    for (i=0; i<MATSIZE; i++)
        for (j=0; j<MATSIZE; j++)
            if (i == j)
                mat[i][j]=1;
            else
                mat[i][j]=0;
    finish = clock();
    cout << "MATSIZE: " << MATSIZE << " Row First TIME " << setprecision(3)
    << (finish - start) / (double) CLOCKS_PER_SEC << " seconds." << endl;

    cout << "Timing column order access..." << endl;
    start = clock();
    for (j=0; j<MATSIZE; j++)
        for (i=0; i<MATSIZE; i++)
            if (i == j)
                mat[i][j]=1;
            else
                mat[i][j]=0;
    finish = clock();
    cout << "MATSIZE: " << MATSIZE << " Col First TIME " << setprecision(3)
    << (finish - start) / (double) CLOCKS_PER_SEC << " seconds." << endl;
    cout << "Array is " << (int)(MATSIZE * MATSIZE * sizeof(int)) / 1048576 <<
    "MB";

    // if running from IDE add some code to pause here
    delete [] mat; // DEALLOCATE SPACE
    return 0;
}
```

## matrix.java

```
class Matrix
// to run with 280 megabytes of heap: java -Xms280M -Xmx280M Matrix
{
public static final int MATSIZE = 4096;          //modify as per assignment
public static int[][] mat = new int[MATSIZE][MATSIZE];

// memory occupied = 4096 * 4096 * 4 bytes = 64MB

public static void main(String args[]) {
    int i,j;
    long start, finish;
    System.out.println("Starting\n");
    // create identity matrix. Do one pass untimed just so that we
    // "touch" and therefore physically allocate all required memory
    for (i=0; i<MATSIZE; i++)
        for (j=0; j<MATSIZE; j++)
            if (i == j)
                mat[i][j]=1;
            else
                mat[i][j]=0;

    // do it again and time it
    System.out.println("Timing row order access...\n");
    start = System.currentTimeMillis();
    for (i=0; i<MATSIZE; i++)
        for (j=0; j<MATSIZE; j++)
            if (i == j)
                mat[i][j]=1;
            else
                mat[i][j]=0;
    finish = System.currentTimeMillis();
    System.out.println("MATSIZE: " + MATSIZE + " Row First TIME " +
+ (double)(finish - start)/1000.0 + " seconds.\n");

    System.out.println("Timing column order access...\n");
    start = System.currentTimeMillis();
    for (j=0; j<MATSIZE; j++)
        for (i=0; i<MATSIZE; i++)
            if (i == j)
                mat[i][j]=1;
            else
                mat[i][j]=0;
    finish = System.currentTimeMillis();
    System.out.println("MATSIZE: " + MATSIZE + " Col First TIME " +
+ (double)(finish - start)/1000.0 + " seconds.\n");

    System.out.println("Array is " + (int)(MATSIZE * MATSIZE * 4) / 1048576 +
"MB\n");
}
}
```

## float.cpp

```
#include <iostream.h>
#include <iomanip.h>

int main(int argc, char* argv[])
{
    float a, b, c, d, e;
    cout.setf(ios::fixed | ios::showpoint);
    cout << "Enter three floating point numbers a, b and c: ";
    cin >> a >> b >> c;

    //echo inputs
    cout << "a=" << a << '\t' << "b=" << b << '\t' << " c=" << c << '\n';

    // try for example a = 0.1  b = 5.812  c = 78.876;

    // evaluate addition without intermediate storage of results
    cout << "Additions without intermediate storage of results\n";
    d = (a+b)+c;
    e = a+(b+c);
    cout << "(a+b)+c = " << setprecision(8) << d << '\n';
    cout << "a+(b+c) = " << setprecision(8) << e << '\n';

    // Intermediate storage and output
    cout << "Additions with intermediate storage AND output of intermediate
results\n";
    d = (a + b);
    cout << "(a+b) = " << setprecision(8) << d << '\n';
    d += c;
    cout << "(a+b)+c = " << setprecision(8) << d << '\n';
    e = (b+c);
    cout << "(b+c) = " << setprecision(8) << e << '\n';
    e += a;
    cout << "a+(b+c) = " << setprecision(8) << e << '\n';

    // Intermediate storage but not output
    cout << "Additions with intermediate storage but no output of intermediate
results\n";
    d = (a + b);
    d += c;
    cout << "(a+b)+c = " << setprecision(8) << d << '\n';
    e = (b+c);
    e += a;
    cout << "a+(b+c) = " << setprecision(8) << e << '\n';
    return 0;
}
```

## FloatTest.java

```
class FloatTest
{
// not part of the homework assignment; this is included just in case you want
// to try it out. Results will usually vary from C++!

    public static void main(String args[])
    {
        float a, b, c, d, e;
        a = (float) 0.1;
        b = (float) 5.812;
        c = (float) 78.876;

        // evaluate addition without intermediate storage of results
        System.out.println ("Additions without intermediate storage of
results\n");
        d = (a+b)+c;
        e = a+(b+c);
        System.out.println("(a+b)+c = " + d + "\n");
        System.out.println("a+(b+c) = " + e + "\n");

        // Intermediate storage and output
        System.out.println("Additions with intermediate storage AND output of
intermediate results\n");
        d = (a + b);
        System.out.println("(a+b) = " + d + "\n");
        d += c;
        System.out.println("(a+b)+c = " + d + "\n");
        e = (b+c);
        System.out.println("(b+c) = " + e + "\n");
        e += a;
        System.out.println("a+(b+c) = " + e + "\n");

        // Intermediate storage but not output
        System.out.println("Additions with intermediate storage but no output of
intermediate results\n");
        d = (a + b);
        d += c;
        System.out.println("(a+b)+c = " + d + "\n");
        e = (b+c);
        e += a;
        System.out.println("a+(b+c) = " + e + "\n");
    }
}
```