**CS420/CSE402/ECE492 Parallel Programming for Scientists and Engineers**

**Fall 2012**

**Machine Problem 2: Vector Operations**

**Due: Wednesday, September 26, 2012 at 11:59:59 p.m**

This MP is about programming microprocessor vector extensions. It is divided into three parts. The third part is for students enrolled for 4 credits. In the first two parts, you are asked to enhance the matrix transposition and matrix-matrix multiplication kernels from MP1 using vector extensions so that you end up with three versions for each part:

1. Simple naïve version (from MP1)
2. Vectorized version
3. Tiled and vectorized version (you can use the tiled version from MP1 and vectorize it)

You have to use SIMD vector operations discussed in the `Vectorization’ lecture of 10th September 2012. Other than that, you will need to use use `\_mm\_unpacklo\_ps’ and `\_mm\_unpackhi\_ps’ for matrix transpose. You can read more about these opertions at:

[http://msdn.microsoft.com/en-us/library/25st103b(v=vs.80).aspx](http://msdn.microsoft.com/en-us/library/25st103b(v=vs.80).aspx" \t "_blank)

and

[http://msdn.microsoft.com/en-us/library/25st103b(v=vs.80).aspx](http://msdn.microsoft.com/en-us/library/25st103b(v=vs.80).aspx" \t "_blank)

**Part A**

Calculate the transpose of a square matrix A and store it back in A as follows:

*A = AT*

You have to compute the transpose using two of the three methods mentioned above: Simple and vectorized. There is no *tiled and vectorized* version since the vectorized version is already tiled.

In order to vectorize the transpose, you implement a tiled version in which each tile is manipulated using the following scheme. The idea is to to transpose 4x4 matrix tiles using \_mm\_unpacklo\_ps and \_mm\_unpackhi\_ps as follows:

Matrix1:

row1:  1, 2, 3, 4

row2:  5, 6, 7, 8

row3:  9,10,11,12

row4: 13,14,15,16

Step 1 -> Matrix2:

row1: \_mm\_unpacklo\_ps(row1,row3):  1, 9, 2,10

row2: \_mm\_unpacklo\_ps(row2,row4):  5,13, 6,14

row3: \_mm\_unpackhi\_ps(row1,row3):  3,11, 4,12

row4: \_mm\_unpackhi\_ps(row2,row4):  7,15, 8,16

Step 2 -> Matrix3:

row1: \_mm\_unpacklo\_ps(row1,row2):  1, 5, 9,13

row2: \_mm\_unpackhi\_ps(row1,row2):  2, 6,10,14

row3: \_mm\_unpacklo\_ps(row3,row4):  3, 7,11,15

row4: \_mm\_unpackhi\_ps(row3,row4):  4, 8,12,16

**Part B**

Calculate the Matrix-Matrix product of square matrices A and B and store it in C. Once again, you need to implement it using all three methods mentioned above.

*C = AB*

*(This A should be the original matrix A and not the transpose)*

**Part C (for students registered in 4 credits)**

This part is a little different from what we did in MP1 (please look at the code). We have unrolled the loop and use the same values for sets of 4 neighboring cells.

Calculate the 5-point stencil over a 2D array (matrix) ‘E’ and store it in the same matrix i.e. E. Given a **square** grid (matrix E) in two dimensions, the 5-point stencil of a point in the grid is made up of the point itself together with its four neighbors. For example, in the figure below, the value of ‘x’ would be the average of the values in cells ‘n’, ‘w’, ‘s’, ‘e’, and ‘x’ itself. The following loop would update all the points in the grid.

**for(i=1;i<n-1;i++)**

**{**

**for(j=1;j<n-1;j+=4)**

**{**

**float E1[4];**

**E1[0] = (E[i\*n+j-1] + E[i\*n+j+1] + E[(i-1)\*n+j] + E[(i+1)\*n+j] + E[i\*n+j])/5;**

**E1[1] = (E[i\*n+j] + E[i\*n+j+2] + E[(i-1)\*n+j+1] + E[(i+1)\*n+j+1] + E[i\*n+j+1])/5;**

**E1[2] = (E[i\*n+j+1] + E[i\*n+j+3] + E[(i-1)\*n+j+2] + E[(i+1)\*n+j+2] + E[i\*n+j+2])/5;**

**E1[3] = (E[i\*n+j+2] + E[i\*n+j+4] + E[(i-1)\*n+j+3] + E[(i+1)\*n+j+3] + E[i\*n+j+3])/5;**

**E[i\*n+j]=E1[0]; E[i\*n+j+1]=E1[1];E[i\*n+j+2]=E1[2];E[i\*n+j+3]=E1[3];**

**}**

**}**

Notice that the first and last columns and the first and last rows remain constant. That is **E[0][:]**, **E[n-1][:]**, **E[:][0]**, and **E[:][n-1]** are not changed by the loop. If E is an *n*×*n* matrix, you can assume that (*n-2*) will be divisible by tile size.

**Output**

Your program should be able to read input matrices from data set files. You will be required to read each matrix from a txt file where the first line specifies the number of rows (It is a square matrix. Therefore, we don’t need to input the number of columns since it is the same as the number of rows). There will be n\*n lines following the first line. Each line will contain the value of one element in row major format i.e. first n lines will contain all elements in row M[0][:] where M is the input matrix. Your program should be able to accept as the first command line argument the name of the data set. Each data set will be stored in three files. Each file will correspond to one of the four input matrices: *A*, *B*, *C* and *E*. Furthermore, your program should also be able to accept the tile size as the second command line parameter.

For example, if my dataset name is **set1**, the following files would contain the matrices.

**set1\_A.txt**

**set1\_B.txt**

**set1\_E.txt**

The following command should be able to perform all three operations using both naïve and tiled versions:

**./mp2 set1 16**

(**set1** is the name for dataset, tile size is **16** and mp1 is your program executable)

You should write the output matrices using the following names in the same format as the input matrices:

**out\_A.txt** (naïve transpose of matrix A)

**out\_A\_v.txt** (vectorized transpose of matrix A)

**out\_A\_vt.txt** (tiled-vectorized transpose of matrix A)

**out\_C.txt** (naïve product of B and C)

**out\_C\_v.txt** (vectorized product of B and C)

**out\_C\_vt.txt** (tiled-vectorized product of B and C)

**out\_E.txt** (naïve 5-point stencil over E)

**out\_E\_v.txt** (vectorization 5-point stencil over E)

**out\_E\_vt.txt** (tiled-vectorization 5-point stencil over E)

Other than writing the matrices, you are also required to output the execution time and speedup in a txt file (results.txt) using the following format.

**Op Naïve Time Vec Time Tile-Vec Time Vec-Sp Tile-Vec-Sp**

Trans a b - a/b -

Matmul a b c a/b c/a

Stencil a b c a/b c/a

**Note: Failure to follow the naming conventions would result in zero points.**

**Testing your Program**

We are providing two datasets on the assignments page (same as MP1 other than the stencil). You can run these sets and match your outputs with the output files that we have provided. You can match the outputs by using **diff** command of linux e.g. **diff my\_output TA\_output**. If the **diff** command does not return anything, this means that the files are identical. Your program will be correct if the six output files are identical to the files provided.

The output for stencil would be change in accordance with the change in the scheme. We will update the datasets accordingly. So please get the new datasets.

**Submission**

Please email your code file i.e. mp2.c, to srungar2@illinois.edu.