CIS 786: Programming Component : Homework 2/Problem 1 (20 points)

1 Cluster Machines

The base machine for the cluster is `pcc16.njit.edu` whose IP address is `128.235.32.116`. The other three nodes of the cluster are `pcc13, pcc14, pcc15` with similar IP addresses (i.e. add to 100 the two-digit suffix number of the name of the machine). You can connect to the base machine through a secure shell program such as OPENSSH available for free under Red Hat/Fedora Linux. For Windows, NJIT offers a secure shell client at `http://csd.njit.edu/software/` Make sure that if you don't have such a program/client on your machine, you download, install and run it to connect to the cluster.

What login name to use. After Homework 1 is submitted, we will record the email you provided to us through Problem 1. We are going to use that email address to send you an email with login and password information. As soon as you receive the email, you can login to the base machine, and change the supplied password. User accounts will be of the form `userX`, where X is a single digit number 1-9. We are going to use `user0` account as an example for the remainder. Substitute your actual account for `user0` in the remainder.

How long will the accounts be active? The accounts will be deactivated shortly after December 20, 2004.

Backup policies. None. You are responsible for backing up your own files. One simple way is to create a `.tar` file with all your relevant files and upload them to an afs machine or a machine of your preference.

How to use ssh from within pcc16. Use the commane `ssh` to create a telnet equivalent connection, or `sftp` to create an ftp-like connection for file transfer.

Sample code. User account `user0` will be open to all of you (you can read but not write). Sample code, bsplib library tar files can and will be found here at directory `/home/user0` on machine `pcc16.njit.edu`.

How to untar a tar file.

```
% tar xvf testing.tar
```

if the file name is `testing.tar`.

How to create a tar file

```
% tar cvf testing.tar file1 dir1 file2
```

to tar `file1 , file2` and the contents of directory `dir1`. Sometimes you might need to write `dir1/` instead of just `dir1`.

2 Programming Guidelines

A number of programming guidelines that need to be followed for the completion of the programming requirements of this course are listed below.

How do you submit program files? Create a directory with name `homework` at the top-level of your home directory `/home/user0`. Create within it 6 subdirectories labeled `hw1` through `hw6`. Make sure that permissions of `homework` and all six subdirectories are `700` (i.e. readable/writeable/executable only by you). If you don't know how to turn on permissions do a `man chmod` and follow the instructions.

A successful completion of this task will earn you **4 points** towards the completion of Problem 1, Homework 2.

In the future a homework submission will mean copying files into the corresponding hwZ directory. There will be no files in directory hw1 as there was no programming component in Homework 1. The work in `hwZ` should be frozen before midnight of the day the homework is due.

For every source code file (or result file that you may generate) clearly identify yourself in the first line of that file. The first few lines could look like the following ones.

```
 /* Alex. Gerbessiotis: user0  HW2 Programming  */
 // Code follows; Try to avoid C++ oriented comment lines.
 /* by using C oriented comment lines like this one */
int main()
{
 return(0);
}

 /* Experimental results are included here. This line
    may extend on multiple
    lines such as this
    example
  */
```

If for some reason you lose access to the base machine and you have the relevant files stored elsewhere, you can send us an email `alg786@cs.njit.edu` but the Subject: line of you email must clearly state HW number and last three digits of your id, for example

```
Subject:  hw2s345
```

If you decide not to follow the standard guidelines (per homework) for file-names you must provide a `README` file that tells us what is going on.

It is your responsibility that your code is ANSI C++ or ANSI C compatible and compilable. The supplied compiler on pcc16.njit.edu is gcc. There are exntesive manual pages on how to use gcc at `www.gnu.org`

In order to compile and run the files, each one of you will have to create a Makefile and use program make. Information about make is also available at the site mentioned above.

No partial credit will be given to submitted code that it is not compilable. No partial credit will be given for code that does not fully list its bugs. The grader will decide testing instance(s) and grade your submission based on whether it passes successfully or not these testing instances.

C versus C++ Some of you may complain that malloc/free are C and not C++. They are there in any C++ compiler. If you don't know how to use them, then you will learn a little more about C++ that you didn't know. If you already know how to use them, then you can practice more with their usage. On an AFS account `man malloc` may provide some extra documentation. We intentionally use malloc/free as opposed to new/delete to make the assignments more interesting, and also incompatible with published web available code.

3 Sequential Matrix Multiplication and Benchmarking (12 points)

This is the second task of Problem 1 of HW2.

I ask you to experiment with the programming environment and implement the following two functions that implement the traditional matrix multiplication algorithm known as Cannon's method (the three for loop algorithm that was presented in class in the context of PRAM algorithms). The multiplication will involve square arrays of dimension $n \times n$.

You are asked to implement the following functions

```
void mmulrm (FTYPE *A, FTYPE *B, FTYPE *C, int n)
void mmulcm (FTYPE *A, FTYPE *B, FTYPE *C, int n)
```

where $FTYPE$ might be defined for test purposes as either

```
typedef double FTYPE;
```

or

```
typedef float FTYPE;
```

i.e. your code should be able to do float and double matrix multiplication with only recompilation of the same source. A, B are the input arrays and C the output that stores the product of $A \times B$.

One obvious observation is that an one dimensional array will be used to store a two-dimensional one. So 2-dimensional arrays a, b, c will be stored within A, B, C; all three latter arrays are 1-dimensional. To accommodate this, A for example will have n^2 elements indexed $A[0..n^2 - 1]$.

A **row-major** storage of a 2-dimensional array into a one-dimensional one stores the first row into $A[0..n - 1]$, the second row into $A[n..2n - 1]$, and so on.

A **column-major** storage of a 2-dimensional array into a one-dimensional one stores the first column into $A[0..n - 1]$, the second column into $A[n..2n - 1]$, and so on.

Say a 2-dimensional array a whose elements are addressed as $a(i, j)$ is stored into our $A[0..n^2 - 1]$ and suppose the storage we use is a column major one. Then $a(i, j)$ is mapped to $A[x]$ where x is given in the figure below. The exact value of x depends also on whether i, j start from 1 or 0 (and thus end in n or $n - 1$).

```
Column Major
 a(i,j) is mapped to A[j*n+i]       if i,j = 0 ... n-1
 a(i,j) is mapped to A[(j-1)*n+i-1] if i,j = 1 ... n

Row Major
 a(i,j) is mapped to A[i*n+j]       if i,j = 0 ... n-1
 a(i,j) is mapped to A[(i-1)*n+j-1] if i,j = 1 ... n
```

The two functions previously defined are called `mmulrm, mmulcm` because the former assumes a, b, c, the 2-dim arrays, are stored into A, B, C in row major (the `rm` suffix), and the latter assumes they are stored in column major form (the `cm` suffix).

```
void mmulrm (FTYPE *A, FTYPE *B, FTYPE *C, int n)
void mmulcm (FTYPE *A, FTYPE *B, FTYPE *C, int n)
```

`Side Effects.` Although space for A, B has already been allocated, you are supposed to accommodate space for C (i.e. you need to allocate it) within the body of the two implemented functions.

```
C= (FTYPE *) malloc (n * n * sizeof(FTYPE));
```

You are asked to provide the code for the two functions, given the representation of the inputs/outputs provided.

Testing. It's up to you how to do the testing of the correctness of your functions. I'll test your implementations on problems sizes of size $n = 64, 256, 512, 1024$ possibly 2048.

Benchmarking. You need to time your implementations by using C/C++ facilities for timing. Use wall-clock times only (not just CPU times).

Points. You earn 12 points by completing this assignment; this is in addition to the 4 points already assigned.

4 Install bsplib on your own account (4 points)

You can collect the last 4 points by installing a version of BSPlib on your own account `bsp0`.

Where is the code. Protected area of Web-page, bottom of it.

How to do it. Follow the instructions available as a link in the protected area.

Why. Just to play around. If you follow my instructions faithfully you are going to install a 1-processor version of BSPlib. You can run even a sequential program with it. If you are curious to see how you can do it, go to and grab the first tar file.

`http://web.njit.edu/~alexg/cluster/pris.html`

The whole process on the cluster takes no more than about 5 minutes. On a faster machine the installation takes less than a minute or two. Be patient.

5 Other Information

Check web-page regularly.

Check `/home/user0` for file `README.1st` for information that will be posted there.