Assignment 3 - Matrix Multiplication

Aims

- Revise and develop knowledge of the programming fundamentals using the Parallel Virtual Machine (PVM) platform.
- Implement a complex algorithm using a distributed computing architecture.
- Explore the complexities of task abstraction for computation by multiple machines.

The Problem

For this assignment you should write a program in C and PVM to multiply two square matrices. In order to implement this program, you should review the code from lecture 17 (Also available from the assignment 3 directory):

Lecture 17 Examples

These program store and retrieve matrices stored as files.

Function/Item	Purpose
mkIdentityMatrix	makes an identity matrix of a given file name and size
mkRandomMatrix	makes a random matrix of a given file name and size
getMatrix	display the contents of a matrix of given filename and size
matrix.c	various functions for reading and writing matrix values

For example, you can create an 20 by 20 identity matrix in file I by:

```
$ mkIdentityMatrix I 20
```

These programs deal with matrices numbering from 1 (not 0). That is, they assume the first row is 1, the first column is 1. Another program that will help is seqmm.c This is a sequential version of matrix multiplication that will show you how to use the matrix.c functions.

Your version should have a master PVM task which reads in two matrices, spawns a worker task for each coordinate, then sends each one a row of the first matrix and a column of the second matrix. The task computes the product for that coordinate and sends it back. After receiving all the results, the master program writes the new product matrix. The program command line should have the file names of each of

the two matrices and a name for the product, plus the size of the matrices.

Example usage (Where A, B and C are the matrix files:

```
[mwelch8@turing mwelch8]$ MatrixMul A B C 20

Multiplication of A an B Complete, result stored in file C

[mwelch8@turing mwelch8]$
```

You may use any code examples from this course without reference.

Hints and Tips

- Make sure you submit a makefile and that your program compiles error-free with -Wall set.
- Build your programs on a node of the cluster - not `turing` or `bourbaki` (The libraries are only installed on the cluster).
- Make sure there is a `clean`
- Include error checking on everything (e.g. Command-line args, function returns etc.)

Tentative Marking Guide

item	Marks
The makefile	....
targets	[/1]
uses -Wall	[/1]
compiles without warnings etc.	[/1]
The Program	....
Correct Multiplication Operation	[/8]
Algorithm Distributes Processing	[/8]
Command line error checking	[/2]
Checks for system call failures	[/2]
Doesn't use hardwired constants	[/1]
Consistent Use of Style	[/1]
Total	25 Marks