Assignment 2 - Software Project, Fall 2009

Due: December 15, 2008. Submit in pairs.

In this exercise you will provide several implementations of matrix multiplication algorithms. You will analyze their performance and select the most efficient implementation. You can limit your implementations to square matrices in which the number of rows is equal to the number of columns. The assignments website, `http://www.cs.tau.ac.il/~ozery/courses/soft-project09/Assignments.php`, provides the file framework for this project, which is detailed at the end of this assignment. You are required to strictly follow the provided function prototypes and the file layout. Note that for a $n \times n$ matrix we refer to n as the *size* of the matrix.

Ex 2.1 mlpl (50 pts)

(1) Code requirements:

a. For the simple iterative matrix multiplication algorithm, implement all the six options of loop ordering (ijk, ikj, jik, jki, kij and kji) as explained in class. These functions should appear in a file named `multiply.c` and comply to the prototypes provided in `multiply.h` (found on website).

b. A matrix is kept in a 1D array, where the elements are ordered **by columns (!!!)**: the first n elements correspond to the first column, the next n elements correspond to the second column, etc. Specifically, the entry (i,j) of the matrix, should be accessed as `A[i + j*n]`. The type of the elements is `elem` (defined in `matrix.h`).

c. Use `get_matrix_space` and `free_matrix_space` to allocate and free memory for matrices. These two functions are defined in `allocate_free.h` and implemented in `allocate_free.c`. An example for allocating a $n \times n$ matrix, A:

```
elem *A;
A = get_matrix_space(n);
```

d. The following procedure, defined in `matrix_manipulate.h`, should be implemented in `matrix_manipulate.c`:

```
void   fill_matrix(elem A[], int n);
```

This procedure fills A, a $n \times n$ matrix, with random integers in range $[-50, 50]$. Implement using `rand()` (see `man 3 rand`).

e. Your main program, implemented in a file named `mlpl.c`, should start by reading an integer k from the standard input, and proceed as follows:

Case 1: $k \leq 0$. In this case, for $n = 4, 8, 16, 32, 64, 128, 256, 512, 1024$ (use `for` loop):

1. Allocate two $n \times n$ matrices, A and B. Fill A and B with random integers in range [-50,50] by calling `fill_matrix`.
2. For each possible loop ordering - multiply A by B using the procedures defined on `multiply.h` (i.e. `mult_ijk, mult_jik`, etc.). Measure the running time of each loop ordering using `clock()`. For example,

```
#include <time.h>
  clock_t t1,t2;
  t1 = clock();
  mult_ijk(a,b,c,n);
  t2 = clock()-t1;
  printf("total time in CPU ticks = %ld\n",t2);
```

The output of your program should be the running times of your multiplication procedures. For each matrix size, print a line with the matrix size followed by the running times of the all matrix multiplication procedures. The order of running times should be: ijk , ikj , jik , jki , kij , kji. The running times are the total number of CPU ticks spent in each procedure. The output values should be separated by commas. For example:

```
4 , 10000 , 10000 , 10000 , 10000 , 10000 , 10000
8 , 10000 , 10000 , 10000 , 10000 , 10000 , 10000
...
```

Case 2: $k > 0$. In this case, k, which represents a matrix size, will be followed by a set of $2k^2$ integers (separated by spaces). Read these numbers into two $k \times k$ matrices, A and B. Multiply A by B using the multiplication procedure with the best loop ordering (selected according to the measurements in case 1). The output should be printed in the same format as the input (i.e. the matrix size followed by a sequence of integers, separated by spaces). Examples of input and output files can be found at the web site. You can use the `diff` command to compare them to your results.

(2) Run your program in the Linux lab. Prepare a graph, which plots the running times (in CPU ticks, Y-axis) as a function of the matrix size (X-axis)[1].

(3) Measure the performance of your program using `gprof` (explained at the end of this assignment). Use the obtained statistics to improve the performance of your program and submit the most efficient code that you can write. Submit the obtained statistics for your program performance (print the short version of the `gprof` output, by running `gprof -b`). Compare `gprof` results to the results of your measurements in (a)[2].

(4) What is the most efficient loop ordering? Explain why.

(5) Submit the details of the computer (in the Linux lab) that was used for your measurements. To do so, print the file `/proc/cpuinfo`.

[1] You can save the output of your program as a *.csv file, which can be open by Excel. In Excel, go to the Chart Wizard and plot your data as XY (Scatter).

[2] Look at the performance ratio of the procedures. Ignore absolute running times which may differ for `gprof` and `clock()`.

Ex 2.2 block_mlpl (50 pts)

(1) Code requirements:

a. Implement the blocked algorithm for matrix multiplication. Use only the loop ordering that was selected to be the most efficient in 2.1. This procedure should appear in a file named `multiply.c` and comply to the prototype provided in `multiply.h`:

```
void mult_block(elem A[], elem B[], elem C[], int n, int r);
```

A and B are two $n \times n$ matrices and r is the block size. Assume $n = 2^m$, and $r = 2^l$, where $1 \leq l \leq m$. As before, the elements in each matrix are ordered **by columns**.

b. Your main program, implemented in a file named `block_mlpl.c`, is **similar** to `mlpl` (as defined in (1).e in Section 2.1). The program should start by reading an integer k from the standard input, and proceed as follows:

Case 1: $k \leq 0$:

1. For $n = 4, 8, 16, 32, 64, 128, 256, 512$, allocate two $n \times n$ matrices, A and B. Fill A and B with random integers in range [-50,50] by calling `fill_matrix`. Compute $A * B$, using `mult_block` with $r = n/2$.
2. Allocate two 1024×1024 matrices, A and B. Fill A and B with random integers in range [-50,50] by calling `fill_matrix`. For $l = 1, 2, \ldots, 9$, compute $A * B$, using `mult_block` with $r = 2^l$.
3. For each call for matrix multiplication (i.e. computing $A * B$), print a line with: matrix size, block size, and the running time in CPU ticks. Separate the numbers in each line by commas. For example:

```
4 , 2 , 10000
8 , 4 , 10000
...
512, 256 , 10000
1024 , 2 , 10000
...
1024 , 512 , 10000
```

Case 2: $k > 0$. In this case, k is a power of two. After k, which represents a matrix size, will follow a set of $2k^2$ integers (separated by spaces). Read these numbers into two $k \times k$ matrices, A and B. Multiply the matrices A and B using `mult_block` with $r = \min(n/2, r')$, where r' is the block size that leads to the best performance for $n = 1024$.

(2) Use the output of `block_mlpl` (when the input $k \leq 0$) to plot a graph describing the running time as a function of matrix size n, when $r = n/2$. The X-axis should be the matrix size n and the Y-axis the running time in CPU ticks. Analyze the results.

(3) For a matrix size $n = 1024$, which block size leads to the best performance? Explain why.

More Information on the Submission

File framework

Below is the description of the file framework for this assignment which you should strictly follow:

- The code for all the multiplication algorithms should be written in file `multiply.c` and should comply to the prototypes provided in the file `multiply.h`.
- The file `matrix.h` contains the definitions of the data types. Here, you may add general definitions of your project.
- The file `matrix_manipulate.c` should contain the implementation of the functions whose prototypes are defined in the file `matrix_manipulate.h`. Here, you can add additional functions for matrix manipulation that will simplify your code.
- For memory allocation you should use the functions provided in the files `allocate_free.c` and `allocate_free.h`.
- The `main` functions for `mlpl` and `block_mlpl` should appear in files `mlpl.c` and `block_mlpl.c` respectively.

Compilation

Your code should be compilable and executable on the LINUX machines in the Computer Science school. Your code must comply to the ANSI C specification and should be compiled using the standard `gcc` compiler. For your development you should use computers abel-XX (XX is between 01-35) located in the classroom 019. For the Excel application you can use the terminal server (see `http://www.cs.tau.ac.il/faq/index.php/Terminal_Server`) or your windows account.

The makefile for `mlpl` and `block_mlpl` is provided at the website. To compile your programs run:

```
make mlpl
make block_mlpl
```

The first command will create an executable `mlpl` for exercise 2.1. Similarly the second command will create an executable `block_mlpl` for exercise 2.2. To remove the object and executable files you can use the command:

```
make clean
```

gprof

Instructions for using the `gprof` command (see `man gprof` for additional details):

1. Create executable files using the provided makefile. This makefile contains the flag `-pg` both in compilation and linkage commands.
2. Run your programs as usual (e.g. `mlpl`).
3. Run `gprof`. For example, `gprof mlpl > profile.txt`.
4. Open and analyze the output file (e.g. `less profile.txt`). An explanation about the output format will appear at the end of the output file.

Submission

- The source files (i.e. *.c and *.h files), makefile, and "partners.txt" files for this assignment should be submitted under `~/soft-proj09/assign2`.

  Important note: As in exercise 1, make sure that you have correct permissions (755) for all the directories and files.

- Manual submission (one for each pair) should include: *(i)* printouts of source files, *(ii)* solutions to analysis questions (questions (2)-(5) in Section 2.1, (2)-(3) in Section 2.2). The manual submission should also include: ID, user-name, and name - of both partners.

Automatic Testing

Matrix multiplication of two input matrices ($k > 0$, Case 2) will be automatically tested. The output in this case should be the same as the one provided in the input/output examples on the website. The running times and their analysis will be checked manually.

Good Luck!