Middle East Technical University | Department of Computer Engineering

CENG 331

Computer Organization

Fall '2016-2017

Performance Lab

Due date: 25 January 2017, Wednesday, 23:55

1 Objectives

This assignment deals with optimizing memory intensive code. Consider a procedure to copy and transpose the elements of an $N \times N$ matrix of type *int*. That is, for source matrix S and destination matrix D, we want to copy each element $s_{i,j}$ to $d_{j,i}$. For the first problem you will optimize Transpose function which is described in Section 3.

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}^{T} = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$$

Transpose of a matrix

Second problem is an intriguing variation of previous problem. Consider the problem of converting a directed graph g into its undirected counterpart g'. The graph g' has an edge from vertex u to vertex v if and only if there is an edge from u to v or v to u in the original graph g. The graph g is represented by its *adjacency matrix* G as follows. If N is the number of vertices in g, then G is an $N \times N$ matrix and its entries are all either 0 or 1. Suppose the vertices of g are named $v_0, v_1, v_2, ..., v_{N-1}$. Then $G[i][j]$ is 1 if there is an edge from v_i to v_j and is 0 otherwise. Observe that the adjacency matrix of an undirected graph is symmetric. Check figure 1 and 2 for more detailed undirected and directed graph representations. For the second problem you will optimize Convert function which is described in Section 3.

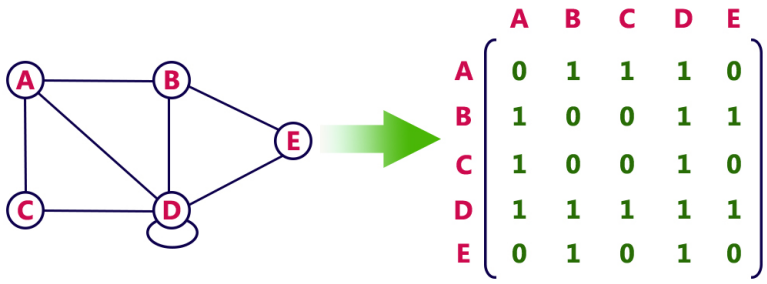

Figure 1: Undirected Graph Representation

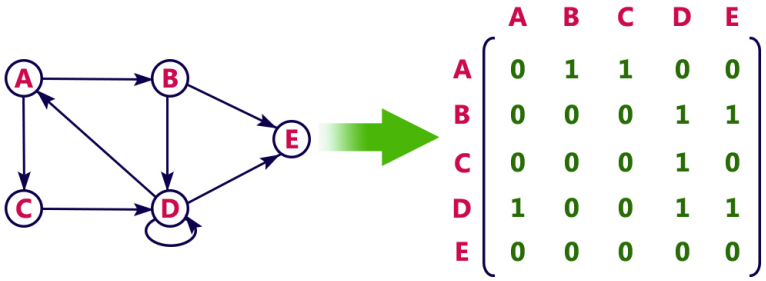

Figure 2: Directed Graph Representation

2 Specifications

Start by copying `perflab-handout.tar` to a protected directory in which you plan to do your work. Then give the command: `tar xvf perflab-handout.tar`. This will cause a number of files to be unpacked into the directory. The only file you will be modifying and handing in is `kernels.c`. The `driver.c` program is a driver program that allows you to evaluate the performance of your solutions. Use the command `make driver` to generate the driver code and run it with the command `./driver`.

Looking at the file `kernels.c` you'll notice a C structure `team` into which you should insert the requested identifying information about you. **Do this right away so you don't forget.**

3 Implementation Overview

Transpose

`Transpose` function can be written with a simple nested loops,

```
void naive_transpose(int dim, int *src, int *dst) {
  int i, j;

  for(i=0; i < dim; i++)
    for(j=0; j < dim; j++)
      dst[j*dim+i] = src[i*dim+j];

}
```

where the arguments to the procedure are pointers to the destination (*dst*) and source (*src*) matrices, as well as the matrix size N (*dim*). Your task is to rewrite this code to make it run as fast as possible using techniques like code motion, loop unrolling and blocking.

See the file `kernels.c` for this code.

Test case	1	2	3	4	5	
Method N	64	128	256	512	1024	Geom. Mean
Naive `Transpose` (CPE)	1.5	5.3	8.2	9.4	10.0	
Optimized `Transpose` (CPE)	1.4	2.5	3.4	4.4	4.6	
Speedup (naive/opt)	1.0	2.1	2.4	2.1	2.2	1.9
Method N	64	128	256	512	1024	Geom. Mean
Naive `col_convert` (CPE)	2.6	5.3	9.1	12.7	19.3	
Optimized `col_convert` (CPE)	2.6	2.8	3.4	4.4	4.5	
Speedup (naive/opt)	1.0	1.9	2.7	2.9	4.3	2.3

Table 1: CPEs and Ratios for Optimized vs. Naive Implementations

Convert

`col_convert` function can be written with a simple nested loops,

```
void naive_col_convert(int dim, int *G) {
  int i, j;

  for(i=0; i < dim; i++)
    for(j=0; j < dim; j++)
      G[j*dim+i] = G[j*dim+i] || G[i*dim+j];

}
```

Your task is to rewrite this code to make it run as fast as possible using techniques like code motion, loop unrolling and blocking.

See the file `kernels.c` for this code.

Performance measures

Our main performance measure is *CPE* or *Cycles per Element*. If a function takes C cycles to run for an matrix of size $N \times N$, the CPE value is C/N^2. Table 1 summarizes the performance of the naive implementations shown above and compares it against an optimized implementation. Performance is shown for for 5 different values of N. All measurements were made on the Intel(R) Core(TM) i7-4770S CPU @ 3.1GHz.

The ratios (speedups) of the optimized implementation over the naive one will constitute a *score* of your implementation. To summarize the overall effect over different values of N, we will compute the *geometric mean* of the results for these 5 values. That is, if the measured speedups for $N = \{64, 128, 256, 512, 1024\}$ are R_{64}, R_{128}, R_{256}, R_{512}, and R_{1024} then we compute the overall performance as

$$R \quad = \quad \sqrt[5]{R_{64} \times R_{128} \times R_{256} \times R_{512} \times R_{1024}}$$

Assumptions

To make life easier, you can assume that N is a multiple of 32. Your code must run correctly for all such values of N, but we will measure its performance only for the 5 values shown in Table 1.

4 Infrastructure

We have provided support code to help you test the correctness of your implementations and measure their performance. This section describes how to use this infrastructure. The exact details of each part of the assignment is described in the following section.

Note: The only source file you will be modifying is `kernels.c`.

Versioning

You will be writing many versions of the `transpose` and `col_convert` routines. To help you compare the performance of all the different versions you've written, we provide a way of "registering" functions.

For example, the file `kernels.c` that we have provided you contains the following function:

```
void register_transpose_functions() {
   add_transpose_function(&transpose, transpose_descr);
}
```

This function contains one or more calls to `add_transpose_function`. In the above example, `add_transpose_function` registers the function `transpose` along with a string `transpose_descr` which is an ASCII description of what the function does. See the file `kernels.c` to see how to create the string descriptions. This string can be at most 256 characters long.

A similar function for your `col_convert` kernels is provided in the file `kernels.c`.

Driver

The source code you will write will be linked with object code that we supply into a `driver` binary. To create this binary, you will need to execute the command

```
unix> make driver
```

You will need to re-make driver each time you change the code in `kernels.c`. To test your implementations, you can then run the command:

```
unix> ./driver
```

The `driver` can be run in four different modes:

- *Default mode*, in which all versions of your implementation are run.
- *Autograder mode*, in which only the `transpose()` and `col_convert()` functions are run. This is the mode we will run in when we use the driver to grade your handin.
- *File mode*, in which only versions that are mentioned in an input file are run.
- *Dump mode*, in which a one-line description of each version is dumped to a text file. You can then edit this text file to keep only those versions that you'd like to test using the *file mode*. You can specify whether to quit after dumping the file or if your implementations are to be run.

If run without any arguments, `driver` will run all of your versions (*default mode*). Other modes and options can be specified by command-line arguments to `driver`, as listed below:

`-g` : Run only `transpose()` and `col_convert()` functions (*autograder mode*).

`-f <funcfile>` : Execute only those versions specified in `<funcfile>` (*file mode*).

`-d <dumpfile>` : Dump the names of all versions to a dump file called `<dumpfile>`, *one line* to a version (*dump mode*).

`-q` : Quit after dumping version names to a dump file. To be used in tandem with `-d`. For example, to quit immediately after printing the dump file, type `./driver -qd dumpfile`.

`-h` : Print the command line usage.

Student Information

Important: Before you start, you should fill in the struct in `kernels.c` with information about you(student name, student id, student email).

5 Assignment Details

Optimizing Transpose (40 points)

In this part, you will optimize `transpose` to achieve as low a CPE as possible. You should compile `driver` and then run it with the appropriate arguments to test your implementations.

For example, running driver with the supplied naive version (for `Transpose`) generates the output shown below:

```
unix> ./driver
ID: eXXXXXXX
Name: Alperen Dalkıran
Email: eXXXXXXX@ceng.metu.edu.tr


Transpose: Version = Naive_transpose: Naive baseline implementation:
Dim             64      128     256     512     1024    Mean
Your CPEs       1.6     5.2     8.2     9.4     10.4
Baseline CPEs   1.5     5.3     8.2     9.4     10.0
Speedup         0.9     1.0     1.0     1.0     1.0     1.0
```

Optimizing Convert (60 points)

In this part, you will optimize `col_convert` to achieve as low a CPE as possible. You should compile `driver` and then run it with the appropriate arguments to test your implementations.

For example, running driver with the supplied naive version (for `col_convert`) generates the output shown below:

```
unix> ./driver
ID: eXXXXXXX
Name: Alperen Dalkıran
Email: eXXXXXXX@ceng.metu.edu.tr

Col_convert: Version = Naive_con_convert: Naive baseline implementation:
Dim             64      128     256     512     1024    Mean
Your CPEs       2.6     5.3     9.2     13.1    19.2
Baseline CPEs   2.6     5.3     9.1     12.7    19.3
Speedup         1.0     1.0     1.0     1.0     1.0     1.0
```

Some advice. Look at the assembly code generated for the `transpose` and `col_convert`. Focus on optimizing the inner loop (the code that gets repeatedly executed in a loop) using the optimization tricks covered in class.

Coding Rules

You may write any code you want, as long as it satisfies the following:

- It must be in ANSI C. You may not use any embedded assembly language statements.
- It must not interfere with the time measurement mechanism. You will also be penalized if your code prints any extraneous information.
- **Important:** Please, work on department inek machines. Because, all *CPE* values has been configured according to inek's CPU and we will evaluate your codes on inek machines.

You can only modify code in `kernels.c`. You are allowed to define macros, additional global variables, and other procedures in this file.

Evaluation

Your solutions for `transpose` and `col_convert` will each count for 40% and 60% of your grade respectively. The score for each will be based on the following:

- Correctness: You will get NO CREDIT for buggy code that causes the driver to complain! This includes code that correctly operates on the test sizes, but incorrectly on matrices of other sizes. As mentioned earlier, you may assume that the matrix dimension is a multiple of 32.
- Speed-up: You will get 20 for implementation of `tranpose` if it is correct and achieve mean speed-up threshold of 2.0. One who achieve the biggest speed-up will get 40 points. Other grades will be scaled between 20 and 40 according to your speed-up. You will not get partial credit for a correct implementation that does below the threshold. Similarly, You will get 30 for implementation of `col_convert` if it is correct and achieve mean speed-up threshold of 2.4. One who achieve the biggest speed-up will get 60 points. Other grades will be scaled between 30 and 60 according to your speed-up. You will not get partial credit for a correct implementation that does below the threshold.
- Since there might be changes of performance regarding to CPU status, test the same code many times and take only the best into consideration. When your codes are evaluated, your codes will be tested in a closed environment many times and only your best speed-up of `transpose` and `col_convert` functions will be taken into account.

6 Regulations

1. **Programming Language**: You must write program in ANSI C.
2. **Submission**: Submission will be done via COW. Submit a single c source file named `kernels.c` which will be modified version of supplied `kernels.c` source file.
3. **Late Submission**: Late submission is not allowed.
4. **Cheating: We have zero tolerance policy for cheating.** People involved in cheating will be punished according to the university regulations.
5. **Newsgroup**: You must follow the newsgroup (news.ceng.metu.edu.tr) for discussions and possible updates on a daily basis.