Middle East Technical University Department of Computer Engineering

CENG 331

Computer Organization

Fall '2018-2019

Performance HW

Due date: 30 December 2018, Sunday, 23:55

1 Objectives

Many algorithms, such as the ones that are used in deep learning and image processing, require the application of a basic operation repeatedly. Hence, their performance depend on the performance of those basic operations. In this homework we will consider two such operations, namely **Matrix Multiplication** and **Convolution**.

Your objective in this homework is to optimize these functions as much as possible by using the concepts you have learned in class.

2 Specifications

Start by copying `Optimization.tar` to a protected directory in which you plan to do your work. Then give the command: `tar xvf Optimization.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

Matrix Multiplication

The naive approach for `matrix multiplication` is given in kernels.c as

```
void naive_matrix_multiplication(int dim, int *src, int *src2, int *dst) {
    int i,j,k;

    for(i = 0; i < dim; i++)
        for(j = 0; j < dim; j++) {
            dst[j*dim+i]=0;
            for(k = 0; k < dim; k++)
                dst[j*dim+i] = dst[j*dim+i] + src[j*dim+k]*src2[i+k*dim];
        }
}
```

where the arguments to the procedure are pointers to the destination (dst) and two sources (src,$src2$) matrices, as well as the matrix size N (dim). Overall what this function does is multiplying src with $src2$ and obtaining dst ($dst = src * src2$)

Your task is to rewrite this code and minimize its CPE, by using techniques like code motion, loop unrolling and blocking.

Convolution

Convolution is used to modify and capture the spatial characteristics in both image processing and deep learning. A convolution is done by selecting a region on your main matrix and multiplying the values in that region by another smaller matrix, which is called as kernel.

For example, lets say you have a source matrix (src) and kernel matrix (ker)

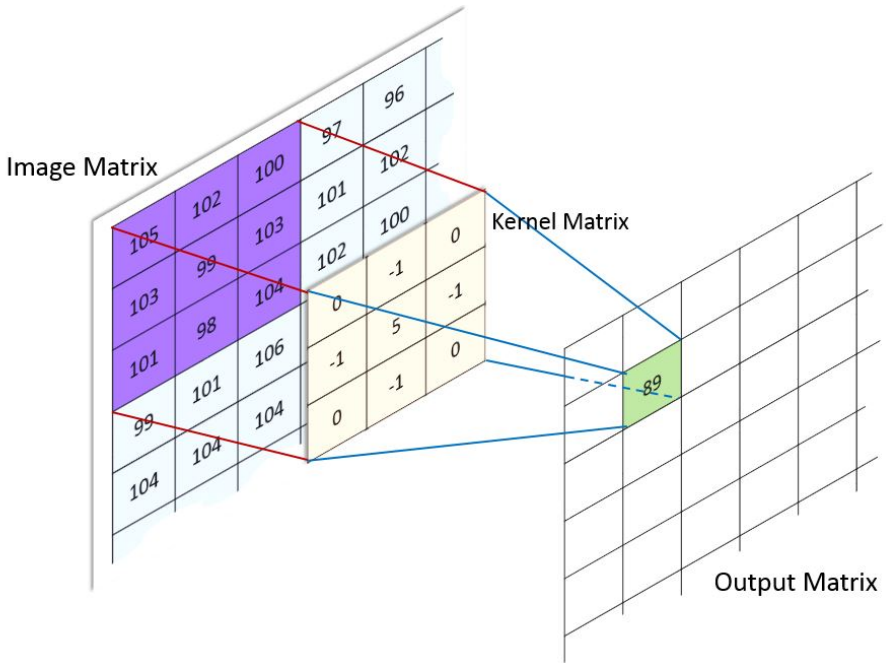
$$src = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

$$ker = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

Your output matrix (dst) will look like

$$dst = \begin{bmatrix} 1*a+2*b+4*c+5*d & 2*a+3*b+5*c+6*d \\ 4*a+5*b+7*c+8*d & 5*a+6*b+8*c+9*d \end{bmatrix}$$

In normal Image processing convolution kernel first rotated, and after that it applied on the source image. However, you are not required to do that in this homework. For additional information and examples about convolution operation click on here.



The naive approach for `Convolution` is given in kernels.c as

```
void naive_conv(int dim,int *src, int *ker,int *dst) {
    int i,j,k,l;

    for(i = 0; i < dim-8+1; i++)
        for(j = 0; j < dim-8+1; j++) {
            dst[j*dim+i] = 0;
            for(k = 0; k < 8; k++)
                for(l = 0; l < 8; l++) {
                    dst[j*dim+i] = dst[j*dim+i] +src[(j+l)*dim+(i+k)]*ker[l*dim+k];
                }
        }

}
```

where the arguments to the procedure are pointers to the destination (dst), source (src) and kernel(ker) matrices, as well as the matrix size N (dim). You will only be using top left most $8x8$ region of ker matrix for your kernel.

Your task is to rewrite this code and minimize its CPE, by using techniques like code motion, loop unrolling and blocking.

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.

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 $N = \{32, 64, 128, 256, 512\}$ are R_{32}, R_{64}, R_{128}, R_{256}, and R_{512} then we compute the overall performance as

$$R = \sqrt[5]{R_{32} \times R_{64} \times R_{128} \times R_{256} \times R_{512}}$$

Assumptions

Assume that N is a multiple of 32 and the dimensions of kernel as $8x8$. For convolution, dimensions of destination matrix can be determined by $N - K + 1$ where K is equal to dimension size of the kernel. Your code must run correctly for all such values of N , but we will measure its performance only for $N = \{32, 64, 128, 256, 512\}$

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 `matrix_multiplication` and `convolution` functions. 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_conv_functions() {
    add_conv_function(&convolution, convolution_descr);
    /* ... Register additional test functions here */
}
```

This function contains one or more calls to `add_conv_function`. In the above example, `add_conv_function` registers the function `convolution` along with a string `convolution_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 `matrix_multiplication` 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 `matrix_multiplication()` and `convolution()` 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 `matrix_multiplication()` and `convolution()` 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 Convolution (50 points)

In this part, you will optimize `convolution` to achieve as low 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 `Convolution`) generates the output shown below:

```
unix> ./driver
ID: eXXXXXXX
Name: Fatih Can Kurnaz
Email: eXXXXXXX@ceng.metu.edu.tr


conv: Version = naive_conv: Naive baseline implementation:
Dim             32      64      128     256     512    Mean
Your CPEs       187.5   241.8   272.6   295.6   315.8
Baseline CPEs   185.9   241.4   272.4   296.1   315.3
Speedup         1.0     1.0     1.0     1.0     1.0     1.0
```

Optimizing Matrix Multiplication (50 points)

In this part, you will optimize `matrix_multiplication` 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 `matrix_multiplication`) generates the output shown below:

```
unix> ./driver
ID: eXXXXXXX
Name: Fatih Can Kurnaz
Email: eXXXXXXX@ceng.metu.edu.tr

Multip: Version = Naive_matrix_multiplication: Naive baseline implementation:
Dim             32      64      128     256     512    Mean
Your CPEs       141.2   306.6   645.6   1386.4  3376.7
Baseline CPEs   141.7   306.8   642.5   1388.7  3373.7
Speedup         1.0     1.0     1.0     1.0     1.0     1.0
```

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.
- It must solely belong to you.
- **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 `matrix_multiplication` and `convolution` will each count for 50% of your grade. 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: For each part(`Convolution` and `Matrix Multiplication`) the 3 student with the lowest amount of CPE's (highest speedup) will receive 50 points for their implementation. Rest of the grades will be scaled accordingly, based on their standing in between highest cpe score of the top 3 student and base line cpe count.
- 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-ups will be taken into account.

6 Regulations

1. **Programming Language**: You must use 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 piazza and cow for discussions and possible updates on a daily basis. In order to have an idea of what will be your grade, you should know about speedups of other students. Therefore, sharing your highest speedups is highly recommended.