Project #2: Cache access optimization

Project #2: Cache access optimization

> This page is out of date. This page shows the semester project description from 2022/2023. The valid semester project description will be published in the 8th week of the semester.

Short tutorial: https://youtu.be/jc1HzQLP6GM

Introduction

You are an employee of an IT firm and you are a member of a team that develops an image editor (Photoshop, Gimp …). In order to gain customers, the editor must be sufficiently fast and easy to use, otherwise it will lose against its competition. You were assigned the task to implement the convolution (https://en.wikipedia.org/wiki/Convolution) operation into the core of the program. Convolution is one of the most important operations when processing digital images. In a nutshell, it computes resulting pixels as weighted sums of their neighboring pixels. The computation is done for each color component separately. Your next task is to compute the histogram.

How convolution works, see here: http://setosa.io/ev/image-kernels/ (http://setosa.io/ev/image-kernels/)

Task

Your homework will be to implement only the below mentioned convolution mask for image sharpening, while optimizing the memory accesses in whatever way is needed – focus on efficient use of cache when using this particular mask.

```
 0  -1 0
-1 5  -1
 0  -1 0
```

If a value lies outside the limits of the interval <0, 255>, use the appropriate limit. The pixels on the image's edges shall be only copied from the original image (the situation where the convolution core overruns the original image).

Furthermore, compute and write into a file the histogram of the focused image in shades of grey. For the conversion from RGB to grayscale, use the luminance model:
Y = round(0.2126*R + 0.7152*G + 0.0722*B)

The histogram shall be computed for left-closed, right-open intervals (bins) < i*L/N; (i+1)*L/N) except the last bin, which is also right-closed; for i=0…N-1, where L=255 and N=5 is the number of bins. In other words, divide the interval <0; 255> into the following bins:

```
Bin:     0 to 50 51 to 101 102 to 152 153 to 203 204 to 255
```

Count:

Input data

The input image will be in the binary encoded portable pixmap format (PPM) (http://en.wikipedia.org/wiki/Netpbm_format) . An image in this format always begins with P6, followed by the image width and height. Next, the constant 255 (maximum intensity of a given pixel component), followed by the main image data - individual RGB components for each pixel. Every pixel component (R, G, B) occupies one byte.

vit-small.zip (https://courses.fit.cvut.cz/BI-APS/@master/media/tutorials/08/vit-small.zip) vit-normal.zip (https://courses.fit.cvut.cz/BI-APS/@master/media/tutorials/08/vit-normal.zip)

The input image file name will be passed on the command line – see the parameters of the main function: main(int argc, char * *argv).

Output of the program

Output of your program will be the focused image in the file named **output.ppm**, and the histogram (luminance counts separated by spaces) in the file **output.txt**.

Homework evaluation criteria

Your program will be evaluated by the total accesses of data L1, instruction L1, shared L2 caches, and foremost, by the number of misses in L1 data, instruction and L2 cache levels. Assume separated L1 instruction and data caches, 32 KB each, 8-way set associative, block size 64B. **Shared L2 cache is 1MB, 16-way set associative, block size 64B.** Replacement strategy is LRU. L2 cache is inclusive (meaning, the contents of L1 caches are also contained in the L2 cache). Your program should be able to detect these parameters and adapt to them, however, for the sake of this homework, it is fully sufficient to optimize your program just for the aforementioned values.

Scoring:

- 5 points for this homework is for a functioning program (pts_function)
- further points will be awarded according to the efficiency of cache usage, provided and only if the program is functioning (pts_efficiency)
- **Total point score for this homework: Points = pts_function + pts_efficiency**

Efficiency of cache usage will be evaluated based on the performance of your program.

Cost = AMAT_i*I_refs + AMAT_d*D_refs,

where

- AMAT_i is the average memory access time for instruction cache requests: AMAT_i = HT_L1i + MR_L1i*(HT_L2 + MR_L2i*HT_RAM)
- I_refs is the number of references to instruction cache
- AMAT_d = HT_L1d + MR_L1d*(HT_L2 + MR_L2d*HT_RAM)

- D_refs the number of references to data cache
- Assumed CPU clock frequency: 3.2 GHz
- Latency of L1: 4 cycles ⇒ HT_L1i = HT_L1d = 1.25 ns
- Latency of L2: 65 cycles ⇒ HT_L2 = 20.3 ns
- Latency of RAM: 140 cycles + 100 ns ⇒ HT_RAM = 144 ns

Consequently, the points for efficiency will be determined by the equation:

pts_efficiency = 10*(Cost_Max - Cost)/(Cost_Max - Cost_Min),

where
Cost_Min = 2.0
Cost_Max = 9.0.

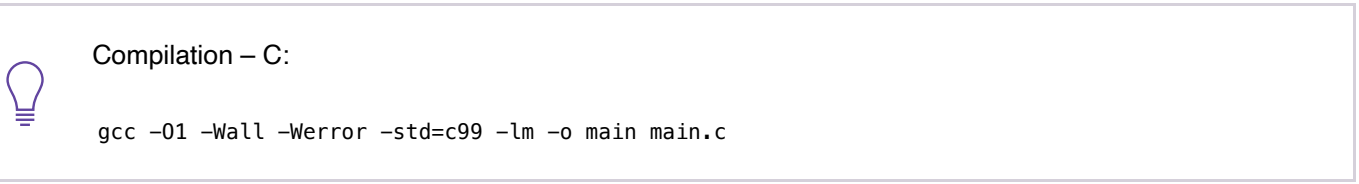
Any negative point score will be truncated to 0 points for efficiency. Any score exceeding 10 points will be truncated to 10 points. Point score will be rounded up to a multiple of 0.25. Your program will be tested with several different image files (with varying sizes). **The values of Cost_Min = 2.0 and Cost_Max = 9.0 apply to images of the size of vit_normal.ppm, 64-bit Ubuntu, gcc 4.8.4.**

Total maximum point score for this project is 15 points.

Homework submission

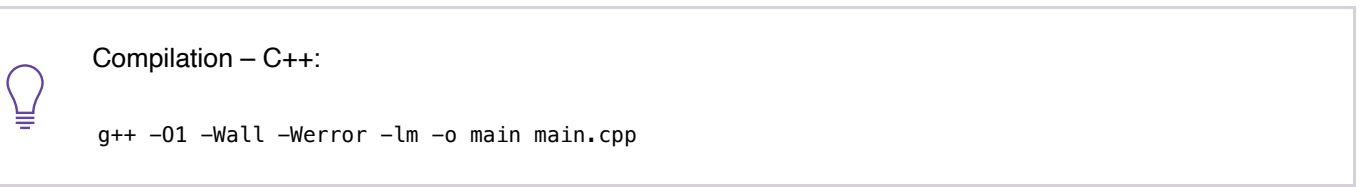
Your program must be written in C or C++ without external libraries. The image format is chosen so that its loading can be easily programmed in a few lines of code. The program can be optimized for compilation on a 32 and 64 bit Intel x86 architecture. Compilation and testing will be done in an environment similar to that in the lab computers (on Linux).

Use http://biaps.fit.cvut.cz/second_semestral_project/index.php to upload your solution in ZIP archive including file `main.c` (if you used C), or `main.cpp` (if C++).



Compilation – C:

```
gcc -O1 -Wall -Werror -std=c99 -lm -o main main.c
```



Compilation – C++:

```
g++ -O1 -Wall -Werror -lm -o main main.cpp
```

All submitted solutions will be checked for plagiarism (all affected people will be punished without discrimination).

Caution: The file output.txt must contain exactly 5 decimal numbers separated by single spaces. The last number must not be followed by anything (not even a newline).

Hint

The Cost used to evaluate your solution is directly coupled to the program's execution speed (faster program means lower cost). Measuring run time is an easy way to assess the efficiency of your approach from both the memory access and algorithmic aspects. You can measure run time the following way:

```
#include <unistd.h>
#include <time.h>
...
struct timespec start, stop;
clock_gettime( CLOCK_REALTIME, &start);
...
clock_gettime( CLOCK_REALTIME, &stop);
double accum = ( stop.tv_sec - start.tv_sec )*1000.0 + ( stop.tv_nsec - start.tv_nsec )/ 1000000
printf( "Time: %.6lf ms\n", accum );
```

Note: When compiling, do not forget -lrt. Example: g + + main.cpp -g -lrt

To evaluate the program's cache utilization, you can use cachegrind. Do not forget to set the cache parameters (–I1=… –D1=… –LL=…), or else some default values will be used (or your processor's values).

```
valgrind --tool=cachegrind --I1=32768,8,64 --D1=32768,8,64 --LL=1048576,16,64 ./a.out
```

This way you can get an idea about the overall behavior of your program. If you want to analyze the program deeper, you can use the cg_annotate tool. Example:

```
cg_annotate <filename> ~/homework/main.cpp
```

where filename is the file generated by cachegrind. Use the absolute path to your source file (main.cpp).

Test image 10 x 8 (https://courses.fit.cvut.cz/BI-APS/@master/media/tutorials/08/test-10x8.zip)

For vit_small.ppm, the otput file md5 sum is 32554ccd9b09af5b660a17b05350959b. The header of the output file is the same as the header of the input file – a whitespace used to separate individual items of the header is \n. Histogram starts with: 24432, 16307, 15192. Next two numbers are not shown here.

Project #2: Cache access optimization
projects/02/index.adoc, last change ac69eac1 (2023-09-24 at 22:58, Ing. Michal Štepanovský, Ph.D.)