CS-350 - Fundamentals of Computing Systems
Homework Assignment #7 - EVAL

Due on November 7, 2024 — Late deadline: November 9, 2024 EoD at 11:59 pm

Prof. Renato Mancuso and Prof. Anna Arpaci-Dusseau

Renato Mancuso, Anna Arpaci-Dusseau

EVAL Problem 1

In this EVAL problem, we will study the impact of the workload created by the client on our monitored hardware counters in our image server.

a) For this part, we are interested in the observing distribution of the instruction count for various requests and how that relates to request length.

Here, it is immediately important to remember that your results might wildly vary when you compare them with those obtained by your colleagues, because you will be running your code on very different machines. The only good point is that now you can compete with your friends about who has the best machine!

To carry out these experiments, run the following command:

```
./build/server_img_perf -q 1000 -h INSTR 2222 & ./build/client -a 30 -I images/ -n 1000 2222
```

(NOTE: if your machine is too weak—#noshame—and starts rejecting requests with the parameters above, you can reduce the arrival rate of requests sent by the client, but keep the same total number `-n 500`).

Now, let's create a plot comparing the length of the request and the collected instruction count for each of the six different image operations. First, split the collected data by the requested image operation (excluding REGISTER). Each operation's data will be plotted in it's own subplot.

In each subplot, create a scatter plot where instruction count is on the x-axis and request length on the y-axis. Keep each plot small enough so that you can fit all of them in a single page, maybe 3 plots per row. Then, create a line of best fit for each of the data plots (i.e. via linear regression).

Again, as in the hw6 eval, I strongly encourage you not to generate each plot manually, but rather use some form of scripting.

Now look at the plots, and answer the following: (1) What is the relationship between instruction count and request length as reported by your line of best fit? (2) Do different image operations behave differently in terms of their instruction counts and time? Answer this question by comparing the characteristics of your linear regression. (3) If the observed relationship is not linear, what could cause any noise, or discrepency between instruction count and request length?

b) For this part, we now want to observe LLC Cache Misses as we either perform random requests, or perform operations on the same image in succession.

To do this, we will use the -L functionality of the client in order to specify a very specific request script. In this request script, we dictate to the client exactly which image operations we want it to request.

Run the following experiment for RUN1:

```
./server_img_perf -q 1500 -w 1 -h LLCMISS 2222 & ./client 2222 -I ./images/ \
-L <SCRIPT>
```

The `<SCRIPT>` will be comprised of two parts: (1) Intro, (2) Ops. The Intro will not change, but we will re-order the Ops section of the script across two runs. Be careful about not omitting the comma ","at the end of these pieces when concatenating Intro and Ops.

(1) Intro (register 5 images):

```
0:R:1:0,0:R:1:1,0:R:1:2,0:R:1:3,0:R:1:4,
```

(2) Ops (repeating pattern):

Problem 1 continued on next page. . .

```
    0:r:1:0,0:r:1:0,0:r:1:0,0:r:1:0,0:r:1:0,0:b:1:0,0:b:1:0,0:b:1:0,0:b:1:0,\
    0:b:1:0,0:s:1:0,0:s:1:0,0:s:1:0,0:s:1:0,0:s:1:0,0:v:1:0,0:v:1:0,0:v:1:0,\
    0:v:1:0,0:v:1:0,0:h:1:0,0:h:1:0,0:h:1:0,0:h:1:0,0:h:1:0,0:r:1:1,0:r:1:1,\
    0:r:1:1,0:r:1:1,0:r:1:1,0:b:1:1,0:b:1:1,0:b:1:1,0:b:1:1,0:b:1:1,0:s:1:1,\
    0:s:1:1,0:s:1:1,0:s:1:1,0:s:1:1,0:v:1:1,0:v:1:1,0:v:1:1,0:v:1:1,0:v:1:1,\
    0:h:1:1,0:h:1:1,0:h:1:1,0:h:1:1,0:h:1:1,0:r:1:2,0:r:1:2,0:r:1:2,0:r:1:2,\
    0:r:1:2,0:b:1:2,0:b:1:2,0:b:1:2,0:b:1:2,0:b:1:2,0:s:1:2,0:s:1:2,0:s:1:2,\
    0:s:1:2,0:s:1:2,0:v:1:2,0:v:1:2,0:v:1:2,0:v:1:2,0:v:1:2,0:h:1:2,0:h:1:2,\
    0:h:1:2,0:h:1:2,0:h:1:2,0:r:1:3,0:r:1:3,0:r:1:3,0:r:1:3,0:r:1:3,0:b:1:3,\
    0:b:1:3,0:b:1:3,0:b:1:3,0:b:1:3,0:s:1:3,0:s:1:3,0:s:1:3,0:s:1:3,0:s:1:3,\
    0:v:1:3,0:v:1:3,0:v:1:3,0:v:1:3,0:v:1:3,0:h:1:3,0:h:1:3,0:h:1:3,0:h:1:3,\
    0:h:1:3,0:r:1:4,0:r:1:4,0:r:1:4,0:r:1:4,0:r:1:4,0:b:1:4,0:b:1:4,0:b:1:4,\
    0:b:1:4,0:b:1:4,0:s:1:4,0:s:1:4,0:s:1:4,0:s:1:4,0:s:1:4,0:v:1:4,0:v:1:4,\
    0:v:1:4,0:v:1:4,0:v:1:4,0:h:1:4,0:h:1:4,0:h:1:4,0:h:1:4,0:h:1:4
```

Notice that RUN1 will perform a variety of operations on different images, but grouped so that all operations of the same type/image are performed one after the other.

Now, for RUN2, switch the Ops, section of the client SCRIPT to the following:

(2) Ops (repeating pattern):

```
0:s:1:2,0:r:1:3,0:s:1:4,0:b:1:4,0:s:1:1,0:b:1:0,0:v:1:1,0:b:1:4,0:b:1:1,0:h:1:0,\
0:v:1:2,0:r:1:1,0:s:1:0,0:b:1:3,0:v:1:4,0:r:1:0,0:v:1:1,0:v:1:4,0:r:1:4,0:s:1:4,\
0:s:1:4,0:r:1:0,0:s:1:3,0:h:1:4,0:s:1:1,0:v:1:3,0:v:1:2,0:s:1:1,0:h:1:3,0:h:1:3,\
0:v:1:0,0:h:1:0,0:r:1:2,0:b:1:3,0:v:1:2,0:b:1:1,0:h:1:2,0:v:1:3,0:b:1:4,0:v:1:3,\
0:r:1:3,0:r:1:2,0:v:1:2,0:b:1:2,0:r:1:2,0:h:1:1,0:b:1:3,0:h:1:1,0:r:1:1,0:b:1:0,\
0:h:1:3,0:s:1:0,0:h:1:2,0:v:1:0,0:v:1:3,0:r:1:4,0:v:1:1,0:s:1:0,0:v:1:3,0:r:1:0,\
0:h:1:0,0:s:1:1,0:v:1:4,0:s:1:2,0:v:1:0,0:r:1:4,0:s:1:0,0:v:1:4,0:b:1:0,0:h:1:3,\
0:r:1:1,0:b:1:2,0:h:1:1,0:s:1:3,0:b:1:3,0:h:1:1,0:r:1:4,0:s:1:0,0:h:1:4,0:b:1:3,\
0:b:1:1,0:h:1:4,0:r:1:4,0:b:1:1,0:r:1:1,0:s:1:1,0:b:1:2,0:h:1:2,0:b:1:0,0:r:1:0,\
0:v:1:0,0:b:1:4,0:s:1:3,0:b:1:0,0:b:1:1,0:r:1:3,0:h:1:4,0:h:1:0,0:h:1:1,0:v:1:1,\
0:b:1:2,0:s:1:3,0:r:1:2,0:v:1:4,0:v:1:0,0:r:1:3,0:s:1:2,0:v:1:2,0:r:1:3,0:h:1:3,\
0:h:1:2,0:s:1:2,0:h:1:0,0:h:1:4,0:v:1:1,0:s:1:4,0:s:1:3,0:h:1:2,0:r:1:1,0:s:1:2,\
0:r:1:2,0:r:1:0,0:b:1:4,0:s:1:4,0:b:1:2
```

Notice that RUN2 performs the exact same operations as RUN1 but shuffled randomly.

Post process the outputs from RUN1 and RUN2 and for each run, produce a plot of the CDF (as we did in HW5/HW6) of the number of LL cache misses caused by each operation. Additionally, include a vertical bar marking the average number of LLC misses for each run.

Plot both CDF lines on the same plot for ease of comparison in two different colors.

Also, report and compare the MIN and MAX LL cache miss for each run.

(1) Do you notice any differences between the CDFs of two runs? What about the MIN/MAX LL cache misses? Why might these differences occur? (2) What can you say about the statefullness of the resources in our image server?

c) Now, we wish to observe L1 cache misses across two similar operations. Note: that the code for imglib.c has been modified slightly from the previous part. Be sure to analyze your image server with the latest imglib.c from the hw7 template.

Specifically, we will be isolating the results of the `IMG_BLUR` and `IMG_SHARPEN` image operations.

Run the command:

Problem 1 continued on next page. . .

```
./build/server_img_perf -q 1000 -h L1MISS 2222 & ./build/client -a 30 -I images/ -n 1000 2222
```

Now on one plot, plot two distributions. In one color, plot the distribution of L1MISS counted from `IMG_BLUR` operations and in another color plot the distribution of L1MISS counted from `IMG_SHARPEN` operations.

To do so, isolate the reported counts from these two types of operations and split them into two datasets. With each, produce a plot of the distribution of L1MISS counts you have collected. The distribution plot should have on the x-axis a set of L1MISS count bins, e.g., from 0 (included) to 500 (excluded), from 500 (included) to 1000 (excluded), and so on in steps of 500 increments. However, you will need to find the appropriate bin size to visulize your data.

Given each transaction, look at its reported L1MISS count. On the y-axis, plot how many requests fall in each bin! But do not plot the raw count. Rather, normalize that value by the total number of requests you are plotting. As in HW2, you have produced a distribution plot (however, not of request lengths, now instead, of the result of our counter).

(1) What do you notice about the distribution of L1MISS counts of `IMG_BLUR` and `IMG_SHARPEN`? How do they differ? (2) If you are using the new image library, you should see a significant difference. Go to `imglib.c`, and inspect the implementation of `blurImage()` and `sharpenImage()`. What could be causing these two functions, which perform similar operations, to have vastly different L1 cache behaviours? Provide a suggestion on how to improve the operation with more L1 cache misses.

d) (NOT GRADED! JUST SOME FOOD FOR THOUGHT. No need to answer.) Take a look at the `perf_event_open()` man page. What other performance counters could potentially offer us insights on our image server?