W4112 Database Systems Implementation
Spring 2015
Project 2, Stage 2 & 3 Specifications

April 7, 2015

1 Overview

Now that you understand the branch prediction plan optimizer which you'll be implementing, it's time to implement and test it. You may use Java, C or C++. Your program will accept as input machine parameters and multiple sets of selectivities. For each set of selectivities, your program must output a short snippet of C code which executes a query. For Stage 3, you will copy and paste these snippets into the provided template. This template generates random data (which adheres to some selectivies given as command arguments) and executes your query. During execution, it uses Intel's hardware performance counters to count branches and branch mispredictions. When your query is done executing, it displays various run time statistics including branch misprediction rates.

2 Command line specification

Your program should be compiled and run correctly on the CLIC machines. For compiling, you must use a Makefile. The TAs must be able to compile your program by simply typing "make". To standardize the execution for all three languages, we require that you provide a shell script that correctly executes your program. The TAs must be able to run your program as such:

```
make
./stage2.sh query.txt config.txt
```

Your program should print results to standard out.

3 Query file specification

A query file (query.txt) includes a list of selectivity for basic terms (see definition 4.1 in Section 4.1 in the paper). In other words, it is the list of $p_1, p_2, \ldots, p_n$ in Section 4.2 in the paper. Your program should process multiple sets of selectivities at a time.

A query file is in this format:

```
0.8 0.5 0.3 0.2
0.2 0.1 0.9
0.6 0.75 0.8 1 0.9
0.8 0.8 0.9 0.7 0.7 0.7
```

In this example, the first line represents that f_1's selectivity $p_1 = 0.8$, f_2's selectivity $p_2 = 0.5$, $f3$'s selectivity $p_3 = 0.3$, and f_4's selectivity $p_4 = 0.2$. The second line is another case wherein f_1's selectivity $p_1 = 0.2$, f_2's selectivity $p_2 = 0.1$, and f_3's selectivity $p_3 = 0.9$. (f_i is a selection function for a basic term.) Each line is separated by a newline, and each selectivity value is separated by a single white space. All values

in this file are $0 \leq x \leq 1$. You don't need exhaustive error checks for a query file. You can assume that any query input file adheres to this format.

Note: We will not test your project on large numbers of expressions, so you don't have to worry about cases of overflow and enormous memory/computational usage. We probably won't have the patience to try more than about 9 terms. Remember, your algorithm's runtime should be about $O(4^n)$, so for $n = 9$, your program shouldn't take *too* long to run. (You won't be graded on this unless it takes way too long, indicating that you are doing something wrong.)

4 Config file specification

A config file (config.txt) includes values of estimated costs. Those values are estimated from CPU specification. All of the CLIC machines are Intel Xeon 5550 machines, so you don't need to change this file. We have provided the config file together with this document.

In case you do some experiments on your own machine, you may need to change the estimated values in the config file by following your CPU's specification. (This would probably be unnecessary work, however, so we recommend using the CLIC machines.) For Stage 2, please use the provided config file.

The config file is in the following format. Hint: it follows Java's properties file format.

```
r = 1
t = 2
l = 1
m = 16
a = 2
f = 4
```

All values shown in this file represent parameters for the cost estimation explained in Section 4.2 of the paper. We assume that the costs of applying functions are equal for all functions f_i, and therefore "f" represents all of them. Also, for Stage 3, it is the cost of searching the data in L1 cache, so we applied L1 cache latency for this cost (`http://www.hardware.fr/articles/imprimer/733`). "m" means the cost of a branch misprediction. It corresponds to the length of a stage pipeline (`http://www.realworldtech.com/nehalem`).

5 Output specification

Your standard output represents an optimal plan for the branching. It should include the selectivities you are optimizing, some C code to do the selection, and the estimated cost. Use the following format:

```
=================================================================
0.7 0.4 0.2 0.3 0.6
-----------------------------------------------------------------
if((t1[o1[i]] & t2[o2[i]]) && t3[o3[i]]) {
    answer[j] = i;
    j += (t4[o4[i]] & t5[o5[i]]);
}
-----------------------------------------------------------------
cost: 10.5
=================================================================
0.7 0.8 0.8 0.9
-----------------------------------------------------------------
if(t1[o1[i]] && (t2[o2[i]] && (t3[o3[i]] && t4[o4[i]]))) {
    answer[j++] = i;
}
-----------------------------------------------------------------
cost: 28.3
=================================================================
```

Note: These plans might not be the optimal plan for the lists of selectivities shown. It is only an output format example.

The first part represents the list of selectivities you read from the query file. The second part is your computed plan corresponding to that list of selectivities. Here, t_i represents the character array, whose values are either 1 or 0, randomly filled but weighted by selectivity. o_i represents the offsets in the character array, which is also randomly created (this value is completely random). (See Section 7.1 of the paper). The third part shows the estimated per-record cost. For example, the first example represents a "mixed algorithm plan" you saw in Section 4 of the paper. You will use this code fragment in Stage 3.

Note: When you write code for the optimal plan, please be careful about these points: Algorithm No-Branch can only be applied to the last &-term (Section 4.1, the last part). The left node of && is always an &-term (Algorithm 4.11, the last sentence "its corresponding plan can be recursively derived by combining the &-conjunction A[S].L to the plan for A[S].R via &&").

6 Stage 2 Submission

Please submit these items:

1. Well-commented code (Java, C, or C++)
2. Makefile (you need to compile and run your program on a CLIC machine)
3. stage2.sh (shell script to run your program)
4. query.txt (sample selectivity lists you tested)
5. config.txt (as provided, no need to change)
6. Execution output file (using the query.txt and config.txt you submit)
7. Comprehensive README file (txt or PDF format)

Put all these files into one folder, create a "stage2_uniA_uniB.tar.gz" file (where uniA and uniB are your UNIs), then submit it on Courseworks.

Note: please follow the standard tarball conventions – your tarball should contain one directory of the same name as the tarball (stage2_uniA_uniB in this case) and all of your files or other directories should reside in that directory. This is commonly done so that we can untar your submissions without making a mess or the extra step of creating a directory ourselves. Many in the Unix world consider this a common courtesy and get very annoyed when the convention isn't followed. At least one of your TAs is included in this group of people.

7 Testing/Experimentation

In order to test the efficacy of your plan optimizer, we will execute your optimized plans and monitor the actual branch misses. Your plan optimizer produces C code. The provided code in "branch_mispred.c" can monitor it for exactly 4 terms. To use it, just copy and paste your code into the "branch_mispred.c" file to the indicated location. Recompile the code with make. Execute it by providing four selectivities as command line arguments. Here's an example:

```
user@jakarta:~/project2/template$ ./branch_mispred 0.5 0.5 0.5 0.5
Loop start!
Loop stop!
Elapsed time: 0.972368956 seconds
CPU Cycles:           2972229769
Instructions:         916655516
IPC:                  0.308407
Branch misses:        96554859
```

```
Branch instructions:  290596236
Branch mispred. rate: 33.226466%

overall selectivity = 0.063624680
theoretical selectivity = 0.062500000

user@jakarta:~/project2/template$ ./branch_mispred 0 0 0 0
Loop start!
Loop stop!
Elapsed time: 0.111856937 seconds
CPU Cycles:           339020229
Instructions:         600107766
IPC:                  1.770124
Branch misses:        5859
Branch instructions:  200018932
Branch mispred. rate: 0.002929%

overall selectivity = 0.000000000
theoretical selectivity = 0.000000000
```

As expected, there is little branch misprediction in some conditions and a good deal with others. We are also giving you CPU cycle counts and instruction counts. From this we also calculate IPC (instructions per cycle), a basic measure of microarchitectural performance.

8 Stage 3 Submission

For stage 3, you only need to submit your report. It should be in PDF format, and we are expecting it to be about 5 pages. In addition to the 5 pages, please include a one page appendix with your raw output from branch_mispred.c. You are encouraged to compile data generated from many branch_mispred.c runs into charts or tables to demonstrate the efficacy of your plan optimizer.

You don't need to submit C code (customized branch_mispred.c) for this stage.

Like in stage 2, please name your file "stage3_uniA_uniB.pdf" and submit it on Courseworks.