W4112 Database Systems Implementation
Spring 2013
Project 2, Stage 2 & 3 Specifications

April 1, 2013

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 be accept as input machine parameters and multiple 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 be copy and pasting 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 Linux perf 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 need to use Makefile. The TAs must be able to compile your program by simply typing "make". To standardize execution for all three languages, we are requiring that you provide a shell script to correctly execute your program. The TAs must be able to run your program as such:

```
./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 paper section 4.1). In other words, it is the list of p1, p2, ... pn in chapter 4.2 in the paper. Your program should process multiple sets of selectivity at a one time.

A query file should be 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
```

It this example, the first line represents that f1's selectivity p1 = 0.8, f2's selectivity p2 = 0.5, f3's selectivity p3 = 0.3, and f4's selectivity p4 = 0.2. The second line is another case wherein f1's selectivity p1 = 0.2, f2's selectivity p2 = 0.1, and f3's selectivity p3 = 0.9. (fi is a selection function for a basic term.) Each line is separated by newline, and each selectivity value is separated by 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 is followed by this format.

Note: We will not be checking your project for large numbers of expressions, so you don't have to worry about cases of overflow and massive memory/computational usage. We likely won't have the patience to try more than about 9 terms. Remember, your algorithms 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 provide the config file together with this document.

In case you do some experiments on your own machine, you may need to change this estimated values by following your CPUs specification. (This just seems like unnecessary work, however, so we recommend using CLIC.) However, for stage 2, please use 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 represents parameters for the cost estimation explained in section 4.2. We assume that the costs of applying function are equal for all functions fi, 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.behardware.com/articles/733-4/report-intel-nehalem-architecture.html`). "m" means the cost of a branch misprediction. It corresponds to the length of stage pipeline (`http://www.realworldtech.com/page.cfm?ArticleID=RWT040208182719&p=1`).

5 Output specification

Your standard output represents an optimal plan for the branching. If 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 plan might not be an optimal plan for those lists of selectivity. It is only an output format example.

The first section represents the list of selectivities you read from the query file. The second section is an optimal plan corresponding to the list of selectivity. ti represents the character array, whose values are either 1 or 0, randomly filled but weighed by selectivity. oi represents the offsets in the character array, which is also randomly created (this value is completely random). (See section 7.1). The third section shows the estimated per-record cost. For example, the first example represents a "mixed algorithm plan" you saw in chapter 4. You will use this code fragment in stage 3.

Note: When you write a 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 CLIC environment)
3. stage2.sh (shell script to run your program)
4. query.txt (sample selectivity lists you tested)
5. config.txt
6. Execution output file (using query.txt and config.txt you submit)
7. Comprehensive README file (txt or PDF format)

Put all files into one folder, create 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 creating a mess and without having to do 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, copy and paste your code into the "branch_mispred.c" file to the indicated location. Recompile the code with using make. Execute it providing four selectivities as command line arguments. Here's an example:

```
jdd@jakarta /home/jdd/coms4112/Project2/Template $ ./branch_mispred 0.5 0.5 0.5 0.5
Loop start!
Loop stop!
Elapsed time: 0.976970911 seconds
CPU Cycles:           2985136826
Instructions:         1021197569
IPC:                  0.342094
Branch misses:        93918620
Branch instructions:  290158085
```

```
Branch mispred. rate: 32.368087%

overall selectivity = 0.065108620
theoretical selectivity = 0.062500000
jdd@jakarta /home/jdd/coms4112/Project2/Template $ ./branch_mispred 0 0 0 0
Loop start!
Loop stop!
Elapsed time: 0.110857010 seconds
CPU Cycles:           336881733
Instructions:         700092650
IPC:                  2.078156
Branch misses:        1914
Branch instructions:  200018124
Branch mispred. rate: 0.000957%

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 needs to be in PDF format and we are expecting it to be about 5 pages. In addition to the 5 pages, please include one page appendix with raw output from branch_mispred.c. You are encouraged to use data from many executions branch_mispred.c and compile it into charts or tables to demonstrate the efficacy of your plan optimizer.

You dont 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.