Georgia Institute of Technology

ECE4100/ECE6100/CS4290/CS6290 Fall 2012

Programming assignment #3

Due: Friday (10/19) 6:00 pm (Regular late penalties apply for late submission. For example, late penalty of 10% after 11:55pm on due date.)

This is an individual assignment. You can discuss this assignment with other classmates but you should do your assignment individually. Please follow the submission instructions. If you do not follow the submission file names, you will not receive full credit. Please check the class homepage to see the latest update. Your code must run on the shuttle cluster with g++4.1

Overview

This assignment is composed of two parts: Part 1: Integrating a gshare branch predictor Part 2: Implementing virtual memory

Part 1: Integrating a Branch Predictor (40 points)

You will extend your Lab #2 pipeline design. Download lab3files.tar.gz , which contains files for branch prediction, virtal memory management, and associated knob files. You can copy these files into your directory from lab2. The tarball also contains a file called "addme_to_sim.cpp" that contains the code snippets that you need to add to your existing sim.cpp file.

We have provided the skeleton files for simulating a branch predictor (bpred.h and bpred.cpp) that contains code for AlwaysTaken predictor, NotTaken predictor, and Bimodal predictor.   Your job is to simulate Gshare branch predictor in this simulation framework. You need to implement three functions bpred_gshare_init, bpred_gshared_access, and bpred_gshare_update.   You also need to integrate the branch prediction structures into your exising pipeline, so that you can analyze the performance impact of having a branch predictor (as opposed to stalling on each branch, until the branch resolves). You will need to change the FE stage and EX stage to incorporate the branch prediction structure.

When a processor fetches a conditional branch instruction (cf_type must be CF_CBR), it should access the branch predictor. All other branch types, we assume that they are correctly predicted. There are no pipeline bubbles for other branches. If a branch is mispredicted, the processor should not fetch instructions from the correct path. i.e., the processor should not call the get_op function. Instead, it should stall the pipeline until the branch instruction is resolved. After the branch is resolved in the EX stage, in the following cycle the correct PC address is forwarded to the PC latch. From the next cycle the processor can start to fetch instructions from the correct path. Note that in real hardware, the processor will keep fetching wrong-path instructions until a branch is resolved. Once a misprediction is discovered, the processor flushes the pipeline and starts to fetch instructions from the correct path. Since we are building a trace-driven simulator, we cannot send wrong-path instructions into the pipeline. That's why we are stalling the FE stage to model the time delay between a branch prediction and branch resolution. To simplify the problem, **after** a branch is predicted in the **FE** stage, you also update the branch predictor including BHR with the **correct value** at the same cycle. In real hardware, the branch predictor is speculatively updated with predicted values. When a branch misprediction is detected, the BHR value is recovered. However, since we do not fetch wrong-path instructions, this step is unnecessary. The 2bit counters in the PHT table are initialized with "10b (2)", weakly taken. In the op field, actually_taken field provides the correct branch prediction. So you can check whether a branch prediction is correct or not by comparing actually_taken value. Note that, in a real hardware, when a branch misprediction is detected, the processor cannot start to rename until all correct path instructions are retired. Again, to simplify the simulator, the processor starts to resume as soon as the correct-path PC value is available.

Part 2: Implement virtual memory support with TLB (40 points)

Thus far we have assumed that the addresses generated by the program are directly used for accessing memory. You will extend your simulator to support virtual memory. As we do not have an operating system simulated in our framework, we have provided the functions that do the virtual to physical translation. Your job would be to simulate the associated timing for performing virtual to physical translation. In particular, you are required to integrate a TLB in the design. If there is a TLB hit, then use the virtual to physical translation provided by our function (supplied via the TLB entry). Otherwise, figure out the page table address associated with the virtual page number and access main memory to obtain the PTE. The TLB entry is then updated with the PFN obtained form the PTE entry (provided in our code). You will also need to change the dcache to store physical tags instead of virtual tags (we will assume that the cache is sized such that we can still index with virtual index). We have provided thread id support in the simulator with the view of future assignments. Ignore the value of the thread_id variable for this experiment.

The files vmem.h and vmem.cpp provided the support for implementing TLB and virtual memory translation. We have already implemented tlb_init and tlb_access, you will need to implement tlb_install. Note that you access TLB with the virtual page number (VPN) and if there is a hit you get the Page Frame Number (PFN). Thus, you will need to convert the cache line address into VPN, depending on the page size (set by KNOB_VPN_PAGE_SIZE). On a TLB miss, the system should acess the Page Table Entry (PTE) from the main memory, the location for which is provided by the function vmem_vpn_to_pteaddr. When we get the PTE address, access the function vmem_vpn_to_pfn to get the actual translation and install it in the TLB. You will need to incorporate the memory access incurred by the TLB miss and service this through MSHR.

To reduce the coding effort, we will continute to assume that the system has a perfect ICACHE, and we will assume that the Instruction Fetch does not do any virtual to phsyical translation. Thus, you have to implement the virtual memory translation only the DCACHE side.

KNOBS

New KNOBS (All the knobs from Lab 2 must still be obeyed)

KNOB_USE_BPRED: Use a branch predictor instead of stalling the fetch stage on each conditional branch

KNOB_BPRED_TYPE: Specify the type of branch predictor (0: NotTaken 1:AlwaysTaken 2:LastTime 3:Gshare)

KNOB_BPRED_HIST_LEN: It decides the length of history register. The number of gshare predictor entry is 2^(KNOB_BPRED_HIST_LEN). The maximum value this KNOB is 20.

KNOB_ENABLE_VMEM: Enable virtual memory (0: TLB access not done, 1: virtual memory translation enabled)

KNOB_TLB_ENTRIES: Number of entries in the TLB

KNOB_VPN_PAGE_SIZE: The page size for your virtual memory system (default 4KB)

Grading

- We will check branch predictor accuracy as we vary the HIST_LEN.

- We will check IPC values to grade your homework. Most of gradings are checking the IPC value trend.

- We will also check the hit rate for TLB and memory accesses to check the correctness of virtual memory implementation

Acceptable Error Bounds

- IPC values can be within + or - 2%

- Branch prediction misprediction count must be within +/- 5% of the correct value (note, we will analyze the misprediction count, and not the misprediction rate)

- TLB miss count must be within +/- 5% of the correct value (note, we will analyze the raw miss count of the TLB, and not the TLB miss rate)

Submission Guide:

Please do not turn in pzip files(trace files). Trace file sizes are so huge so they will cause a lot of problems.
(Tar the lab3 directory. Gzip the tarfile and submit lab3.tar.gz file at T-square)

```
cd lab3
make clean
rm *.pzip
cd ..
tar cvf lab3.tar lab3
gzip lab3.tar
```

You should also submit a report file called **report.pdf** (more details provided below)

Testing and Traces :

For this assignment, test sample outputs will be provided. The information about test harness and sample traces will be provided shortly.

Report (20 points) : Simulation and analysis

Using your simulator, you will do performance analysis for the traces we provide .

- Vary the g-share history length (2, 4, 8, 10, 12, 16) and measure the branch predictor accuracy. Do you see the increases in the branch predictor accuracy? If not discuss why.
- Vary the size of the TLB (1,4,16 entries) and report the TLB hit rate, IPC, and number of memory accesses.
- Repeat the above by varying the page size to 8KB.

Include your simulation results in a report. Please note that there are many simulation cases so it may take several hours to simulate all of them. Your submission file name MUST be report.pdf and you must include your name and T-square account name . If you fail to put your name and your file name is different than "report.pdf", your report won't be graded .