Inorder CPU Modification: Gem5 Hands-on Skills

Yuhui Zheng
Department of Electrical and Computer Engineering,
Purdue University, West Lafayette, IN

Abstraction – Gem5[1] is a modular discrete event driven computer system simulator platform. Gem5 modules are written primarily in C++, with system configured mostly in python. In this report, 5 stage inorder CPU model is modified, and the corresponding performance variations are measured with spec2k6 benchmark.

Keywords – gem5, inorder CPU

1 – Experimental Method

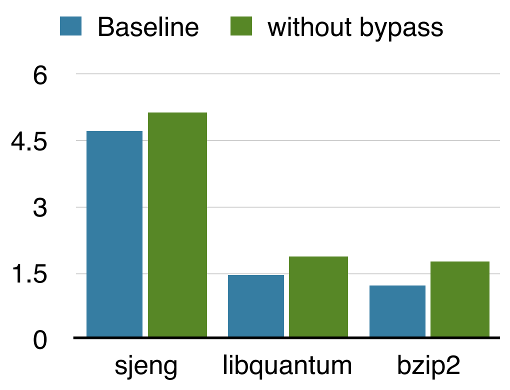
In this report, modifications on gem5 inorder cpu are done in path "/src/cpu/inorder". Meanwhile, all the simulations are done with "—maxinsts=100000000 —cpu-type=inorder —caches" specification. As for data collection, since the actual part we care about is CPI, which largely contributes to performance, the chart and table below are drawn with system.cpu.cpi_total (from output file stat.txt), rather than other parameters. Easy to see, any implementation below will degrade IPC, thus increasing CPI.

2 - Implementation & Results

Part 1: disable bypass in all cases

Analysis:
In order to bypass, a younger instruction looks for older instructions with the same register index number. If the specific older instruction executed, younger instruction could get bypassed value from that older instruction. This is done in inorder_dyn_inst.cc. By commenting out returning old instruction list, bypass is disabled.

Result:



	baseline	without bypass
sjeng	4.720671	5.142619
libquantum	1.423677	1.849413
bzip2	1.202066	1.728845

Fig. 1 – Baseline vs. disable bypassing

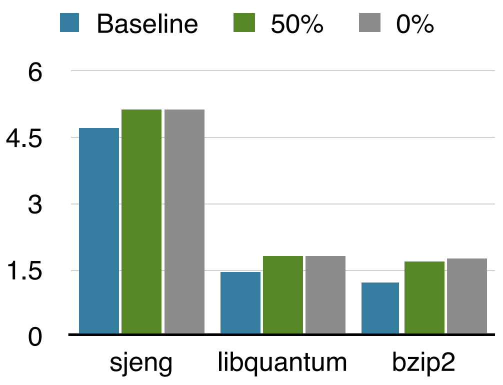
Part 2: degrade branch predictor

Analysis:
Generally, if branch target PC does not match actual next PC, evaluate this prediction as mispredicted. In this case, squash the pending instructions.

In this part, there are ways to degrade branch predictors:
1, modify mispredicted() function in inorder_dyn_inst.hh, to arbitrarily treat correctly predicted instructions as mispredicted ones.
2, modify exicution_unit.cc, to arbitrarily squash correctly predicted instructions.
3, modify pred_taken in bpred_unit.cc, to flip pred_taken with some probability, which results prefetching wrong target PC. In this case, we should assume that the original prediction accuracy is close to 100%.

1st & 2nd are causing segfault with unmapped address for some reason. Thus I've implemented in the 3rd way. It may be more accurate with base/random.cc, but using rand() is also functionally correct.

Result:



	100% (baseline)	50% accuracy	0% accuracy
sjeng	4.720671	5.141504	5.141006
libquantum	1.423677	1.779797	1.796934
bzip2	1.202066	1.686826	1.732603

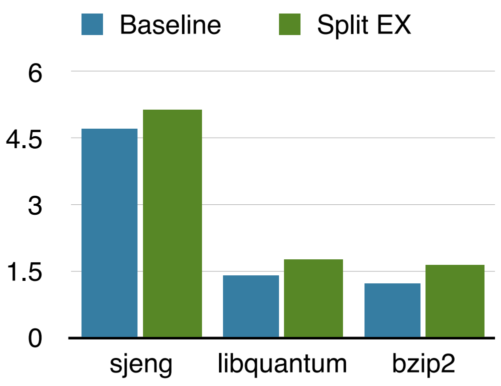
Fig. 2 – Baseline vs. degrade branch prediction

Part 3: split EX stage

Analysis:
The EX stage of original 5 stage pipeline calls:
1, address generator
2, multiply/divide unit
3, general execution
Then, set executed flag by successfully scheduling any one of them. After that, register dependency checking could detect executed flag, and bypass the ready values.

Thus, whenever splitting EX stage regarding register dependencies, make sure that set executed flag in EX2, rather than EX1.

Result:



	baseline	split EX
sjeng	4.720671	5.141501
libquantum	1.423677	1.759381
bzip2	1.202066	1.623705

Fig. 3 – Baseline vs. split execution stage

3 – Reference:

[1] http://www.m5sim.org/