1. branch ratios on x86 Haswell Machine

a. Run the bzip2 benchmark and measure instructions:u, branches:u, and r5301c4:u (r5301c4:u, according to libpfm4, corresponds to the BR_INST_RETIRED:COND event which measures only conditional branches).

Performance counter stats for '/opt/ece571/401.bzip2/bzip2 -k -f ./input.source':

169,205 instructions:u

36,378 branches:u

32,485 r5301c4:u

0.001414955 seconds time elapsed

i. What are the total number of instructions, branches, and conditional branches?

The total number of instructions are 169205, branches are 36378, and the conditional branches are 32485.

ii. What is the ratio of branches to total instructions?

36378/169205=21.499%

iii. What is the ratio of conditional branches to total instructions?

32485/169205=19.199%

b. Now do the same test with the equake_l benchmark (note that's an L not a 1).

perf stat -e instructions:u,branches:u,r5301c4:u /opt/ece571/equake_l.specomp/equake_l < /opt/ece571/equake_l.specomp/inp.in

Performance counter stats for '/opt/ece571/equake_l.specomp/equake_l':

1,426,639,369,751 instructions:u
120,052,071,550 branches:u
87,896,758,558 r5301c4:u

134.235077395 seconds time elapsed

i. What are the total number of instructions, branches, and conditional branches?

Total number of instruction: 1426639369751. Numbers for branches: 120052071550. Numbers for conditional branches 87896758558.

ii. What is the ratio of branches to total instructions?

120052071550/1426639369751=8.42%

iii. What is the ratio of conditional branches to total instructions?

87896758558/1426639369751=6.16%

2. Branch miss rate on x86 Haswell Machine

a. For the bzip2 benchmark measure branches:u and branch-misses:u. Calculate the branch miss rate branchmiss/ branchtotal *100

169,198 instructions:u

36,380 branches:u

3,261 branch-misses:u # 8.96% of all branches

0.016157836 seconds time elapsed

b. calculate the branch miss rate for equake_l.

Performance counter stats for '/opt/ece571/equake_l.specomp/equake_l':

1,426,558,844,673 instructions:u

120,029,064,038 branches:u

587,228,666 branch-misses:u # 0.49% of all branches

134.236129106 seconds time elapsed

3. Speculative execution on x86 Haswell Machine

libpfm4 tells us that UOPS_RETIRED is r5301c2:u and UOPS_EXECUTED is r5302b1:u

a. Find out what percentage of instructions were executed but not retired with bzip2.

Performance counter stats for '/opt/ece571/401.bzip2/bzip2 -k -f ./input.source':

199,927 r5301c2:u

280,542 r5302b1:u

0.001392473 seconds time elapsed

(280542-199927)/280542=28.74%

b. Find out what percentage of instructions were executed but not retired with equake_l.

Performance counter stats for '/opt/ece571/equake_l.specomp/equake_l':

1,860,228,735,238 r5301c2:u

3,363,679,962,357 r5302b1:u

140.489058077 seconds time elapsed

(3363679962357-1860228735238)/3363679962357=44.70%

4. Branch ratios on an ARM64 System

a. Run the bzip2 benchmark and measure instructions and branches.

r10:u maps to PC_BRANCH_MIS_PRED and r12:u maps to PC_BRANCH_PRED.

Performance counter stats for '/opt/ece571/401.bzip2/bzip2 -k -f ./input.source':

20,141,620,091 instructions:u

261,330,987 r10:u

3,088,891,512 r12:u

10.490800082 seconds time elapsed

What are the total number of instructions and branches? What is the ratio of branches to total instructions?

Total number of instructions are 20141620091, branches are (261330987+3088891512)=3350222499. Ratio of branches to total instructions: (3350222499/20141620091)=16.63%

5. Branch miss rate on ARM64 System

a. Calculate the branch miss rate for bzip2.

Ratio of miss: 261330987/335022249=78%

6. Short Answer Questions

a. Does the branch to instruction ratio differ between equake and bzip2 on Haswell? Why might this be?

Yes, the branch to instruction ratio differ between equake and bzip2 on haswell. Bzip2 is the integer benchmark has 3 inputs, equake is a float benchmark has 1 input. Equake has prefetch rate of 0.0959%, bzip2 has prefetch rate of 0.0353%. Equake and bzip2 are write differently so that the branch ratio is different.

b. Does the branch to instruction ratio differ between bzip2 on the Haswell machine and bzip2 on the ARM64 machine? Why might this be?

Yes, because different machines has different compiler for the benchmark, for the file that generated after the compilation is different. So the ratio differs.

c. How do the branch miss-prediction rates compare between bzip2 and equake on the Haswell machine? What might be the source of any differences?

Branch miss-prediction rate for bzip2 is 8.96%, for equake is 0.49%. Different benchmarks has different ways of predicting branches, so the miss-prediction ratio is different.

d. How do the branch miss-prediction rates compare between bzip2 on Haswell and bzip2 on the ARM64 machine? What different design decisions might have been made between the two machines that affects these results?

Bzip2 miss-prediction rate on Haswell is 8.96%, on ARM64 is 78%. Different machines has different compiler for the benchmark, so it compiles differently and that is why it works different on Haswell and ARM64. The different design decisions includes the architecture difference would cause the branch in different ways.

e. How do the executed vs retired instruction rates differ between bzip2 and equake on the Haswell machine? What implications might this have about the power efficiency of the two benchmarks?

Bzip2's ratio of executed vs retired instruction is 280542/199927=1.403. equake's ratio is 3,363,679,962,357/1,860,228,735,238=1.808. equake's ratio is higher than the Bzip2's. This would imply that the equake would consume more power than the bzip2, that is to say that equake is having lower power efficiency.

f. Imagine you wanted to write a benchmark to validate the branch prediction performance counters on a system. What kind of short benchmark could you write that would give you a 50% miss-predict rate?

The benchmark should be a FOR loop that keeps doing the function and with a branch to a certain point. And there should also be a counter that counts the numbers that it predict the branch and miss-predict of the branch. The matrix matrix multiply has the branch miss-prediction ratio of 58%, I would choose it to make the benchmark.

g. Optionally write such a small example program, test it out, and report your results. (note if you do this, the Linux C library has two random routines, rand() and random() and one has many extraneous branches while one doesn't).