Rodrigo Pimenta
ENEE446, Section 0101

Programming Assignment 1: Experience with the SimpleScalar Simulator

Benchmark: GCC

Total number of dynamic instructions executed	281473209 instructions
Percentage of dynamic instructions that are loads	25.26%
Percentage of dynamic instructions that are stores	13.53%
Percentage of dynamic instructions that are unconditional branches	4.2%
Percentage of dynamic instructions that are taken conditional branches	8.24%
Percentage of dynamic instructions that are not-taken conditional branches	7.58%
Percentage of dynamic instructions that are floating-point operations	0.01%
Percentage of dynamic instructions that are system calls	0.0000016%
Percentage of dynamic instructions that have a register-based RAW data dependency with the preceding instruction	31.939%
Percentage of dynamic instructions that have a register-based RAW data dependency with the previous-to-preceding instruction	17.017%

11. Is the program computation-intensive, memory-intensive, or I/O-intensive?

The program is computation intensive because the percentage of dynamic instructions that are integer operations and floating point operations is 41.91%.

12. How many dynamic instructions, on the average, exist between two branches?

4.995 instructions.

Summary GCC:

First I set up Simple Scalar and its environment accordingly. Then I ran the command, sim-profile -segprof -brprof -iclass -iprof ../benchmarks/cc1.pisa-big -O ../benchmarks/1stmt.i, to get the statistics to questions 1-4, 7, and 8. I then ran the command, sim-bpred -bpred taken ../benchmarks/cc1.pisa-big -O ../benchmarks/1stmt.i, to get statistics for questions 5 and 6. This gave me statistics for bpred_taken.dir_hits, the number of unconditional branches plus the number of taken conditional branches, and bpred_taken.misses, the number of not-taken conditional branches. I used the formula (bpred_taken.dir_hits/number of instructions) * 100 - (percent of unconditional branches) to get the percent of taken conditional branches. I then used the formula (bpred_taken.misses/number of instructions) * 100 to get the percent of not-taken conditional branches. I repeated this process to get statistics for the other benchmark programs. Finally, I ran the command, sim-safe -max:inst 281473208 ../benchmarks/cc1.pisa-big -O ../benchmarks/1stmt.i, to get statistics for questions 9 and 10.

GCC is a very computation-intensive program due to its very large amount of integer and floating-point operations. Integer and floating-point operations account for 41.91% of dynamic instructions. The program also uses a lot of memory, as there is a large amount of loads and store. Both loads and stores account for 37.79% of dynamic instructions. Finally, the program is not I/O intensive at all because it has a very small number of system calls, which account for less than 1% of dynamic instructions. Therefore, the program is computation-intensive as computation operations accounts for the highest number of operations.

GCC has more taken conditional branches than not-taken conditional branches. This means that the program passes more of its conditions for branching, in conditional branches, than it fails its conditions for branching. The program also has more taken conditional branches than unconditional branches, meaning most of its branching originate from passing conditionals for branching.

Benchmark: ANAGRAM

Total number of dynamic instructions executed	17859167 instructions
Percentage of dynamic instructions that are loads	23.84%
Percentage of dynamic instructions that are stores	9.52%
Percentage of dynamic instructions that are unconditional branches	5.91%
Percentage of dynamic instructions that are taken conditional branches	9.96%
Percentage of dynamic instructions that are not-taken conditional branches	8.68%
Percentage of dynamic instructions that are floating-point operations	0%
Percentage of dynamic instructions that are system calls	0.000004423498588%
Percentage of dynamic instructions that have a register-based RAW data dependency with the preceding instruction	36.086%
Percentage of dynamic instructions that have a register-based RAW data dependency with the previous-to-preceding instruction	22.781%

11. Is the program computation-intensive, memory-intensive, or I/O-intensive?

The program is computation intensive because the percentage of dynamic instructions that are integer operations and floating point operations is 41.94%.

12. How many dynamic instructions, on the average, exist between two branches?

4.078 instructions.

Summary ANAGRAM:

First I ran the command, sim-profile -segprof -brprof -iclass -iprof ../benchmarks/anagram.pisa-big ../benchmarks/words < ../benchmarks/anagram.in > ! OUT, to get the statistics to questions 1-4, 7, and 8. Then I ran the command, sim-bpred -bpred taken ../benchmarks/anagram.pisa-big ../benchmarks/words < ../benchmarks/anagram.in > ! OUT, to get statistics for questions 5 and 6. Finally, I ran the command, sim-safe -max:inst 17859166 ../benchmarks/anagram.pisa-big ../benchmarks/words < ../benchmarks/anagram.in, to get statistics for questions 9 and 10.

Anagram is a computation-intensive program due to having 41.94% of all dynamic instructions being integer and floating-point operations. This is larger than the 33.36% of instructions, which are either stores or loads. So it does more computational operations than memory operations. Finally, less than one percent of all operations are system calls, meaning it does not have a lot of I/O operations.

Anagram has more taken conditional branches than not-taken conditional branches. This means that the program passes more of its conditions for branching, in conditional branches, than it fails its conditions for branching. The program also has more taken conditional branches than unconditional branches, meaning most of its branching originate from passing conditionals for branching.

Benchmark: COMPRESS95

Total number of dynamic instructions executed	80434022 instructions
Percentage of dynamic instructions that are loads	21.28%
Percentage of dynamic instructions that are stores	14.48%
Percentage of dynamic instructions that are unconditional branches	6.69%
Percentage of dynamic instructions that are taken conditional branches	6.09%
Percentage of dynamic instructions that are not-taken conditional branches	5.07%
Percentage of dynamic instructions that are floating-point operations	0.48%
Percentage of dynamic instructions that are system calls	0%
Percentage of dynamic instructions that have a register-based RAW data dependency with the preceding instruction	36.199%
Percentage of dynamic instructions that have a register-based RAW data dependency with the previous-to-preceding instruction	16.954%

11. Is the program computation-intensive, memory-intensive, or I/O-intensive?

The program is computation intensive because the percentage of dynamic instructions that are integer operations and floating point operations is 46.37%.

12. How many dynamic instructions, on the average, exist between two branches?

5.597 instructions.

Summary COMPRESS95:

I ran the command, sim-profile -segprof -brprof -iclass -iprof ../benchmarks/compress95.pisa-big < ../benchmarks/compress95.in > ! OUT, to get the statistics to questions 1-4, 7, and 8. I then ran the command, sim-bpred -bpred taken ../benchmarks/compress95.pisa-big < ../benchmarks/compress95.in > ! OUT, to get statistics for questions 5 and 6. I finally ran the command, sim-safe -max:inst 80434021 ../benchmarks/compress95.pisa-big < ../benchmarks/compress95.in, to get statistics for questions 9 and 10.

COMPRESS95 is more computation-intensive than memory or I/O. This is because 46.37% of all instructions are integer or floating point operations, as opposed to the 35.76% of all instructions which are loads or stores. This shows it is more computation heavy than memory heavy. Finally, it is not I/O heavy as there are very few system calls.

Compress95 has more taken conditional branches than not-taken conditional branches. This means that the program passes more of its conditions for branching, in conditional branches, than it fails its conditions for branching. The program also has more unconditional branches than taken conditional branches, meaning most of its branching originate from branches that will execute either way, not based on any condition. This is the only benchmark that had this result. The other three had more taken conditional branches than unconditional branches.

Benchmark: GO

Total number of dynamic instructions executed	548131612 instructions
Percentage of dynamic instructions that are loads	21.12%
Percentage of dynamic instructions that are stores	7.55%
Percentage of dynamic instructions that are unconditional branches	3.32%
Percentage of dynamic instructions that are taken conditional branches	6.17%
Percentage of dynamic instructions that are not-taken conditional branches	5.16%
Percentage of dynamic instructions that are floating-point operations	0%
Percentage of dynamic instructions that are system calls	0.000013%
Percentage of dynamic instructions that have a register-based RAW data dependency with the preceding instruction	47.0829%
Percentage of dynamic instructions that have a register-based RAW data dependency with the previous-to-preceding instruction	21.715%

11. Is the program computation-intensive, memory-intensive, or I/O-intensive?

The program is computation intensive because the percentage of dynamic instructions that are integer operations and floating point operations is 56.67%.

12. How many dynamic instructions, on the average, exist between two branches?

6.827 instructions.

Summary GO:

I initially ran the command, sim-profile -segprof -brprof -iclass -iprof ../benchmarks/go.pisa-big 50 9 ../benchmarks/2stone9.in > ! OUT, to get the statistics to questions 1-4, 7, and 8. I then ran the command, sim-bpred -bpred taken ../benchmarks/go.pisa-big 50 9 ../benchmarks/2stone9.in > ! OUT, to get statistics for questions 5 and 6. I finally ran the command, sim-safe -max:inst 548131611 ../benchmarks/go.pisa-big 50 9 ../benchmarks/2stone9.in, to get statistics for questions 9 and 10.

Go is computation-intensive, with 56.67% of all operations being either integer or floating point operations. Only 28.67% of all operations are either load or store, showing it is much more computation-intensive, than memory-intensive. And it has very few system calls.

Go has more taken conditional branches than not-taken conditional branches. This means that the program passes more of its conditions for branching, in conditional branches, than it fails its conditions for branching. The program also has more taken conditional branches than unconditional branches, meaning most of its branching originate from passing conditionals for branching.

The SimpleScalar tool and its collection of microarchitecture simulators is a computer architecture simulator, it can model virtual computer systems and components. This allows the users to collect outputs and performance metrics. Ultimately it is a powerful computer architecture tool and can be used to make many different determinations on computer devices and its components without actually building them.