CS251A Homework 1

Zhaoning Kong - 005223528

*1) What metric (and mean) should you use to compare the performance between different system configurations? Why?*

A: Depending on where the differences between two systems, we should use different metrics.

|  |  |
| --- | --- |
| Differences | Metrics |
| CPU Model | **1) Branch prediction correct rate:** CPU performance depends on branch prediction schemes, for example, size of BTB, trace cache, etc.  **2) IPC and total execution time:** These are general metrics to evaluate CPU performance. |
| Compiler | **1) Number of branches:** A smarter compiler can reduce the number of branches. For example, using predicates instead of branches.  **2) Branch prediction correct rate:** A smarter compiler results in higher branch prediction correct rates by, for example, using branch prediction hints.  **3) IPC and total execution time:** A smarter compiler reduces data dependencies. For example, the compiler can separate two dependent instructions as far as possible, to prevent data-dependent stalls, which will affect IPC and total execution time. |
| Clock Frequency | **1) IPC and total execution time:** Although IPC and total execution time doesn’t scale linearly with frequency due to memory access latencies, it nevertheless serves as a metric for overall performance. |
| Memory controller | **1) IPC and total execution time:** Different memory controller have different bandwidth and latency, which will affect the overall performance. |
| L2 Cache | **1) Memory access burst patterns:** Without L2 cache, there will be much more accesses to memory.  **2) IPC and total execution time:** Overall metric. |

*2) For these three program properties – a) memory regularity, b) control regularity, and c) locality – name one statistic (or combination of statistics) from stats.txt that might help you categorize a workload as having that property. (e.g. for control regularity, it is inversely proportional to the number of branch instructions … but you can think of a better one).*

A:

|  |  |
| --- | --- |
| Properties | Statistic(s) |
| Memory Regularity | **1) system.cpu.dcache.overall\_hits::total**  **2) system.cpu.dcache.overall\_misses::total**  We can use the ratio of the two metrics above to get d-cache hit ratio. (Although it uses same statictics as locality, they are because of different reasons. For example, for memory accesses in strides, if a smarter memory fetch strategy can detect memory patterns and prefetch successive streams, we will see higher cache hit rates.) |
| Control Regularity | **1) system.cpu.fetch.Branches**  **2) system.cpu.branchPred.condPredicted**  **3) system.cpu.branchPred.condIncorrect**  We can use the ratio of (2) and (3) to get branch prediction accuracy. |
| Locality | **1) system.cpu.dcache.overall\_hits::total**  **2) system.cpu.dcache.overall\_misses::total**  We can use the ratio of the two metrics above to get d-cache hit rates. |

*3) For each microbenchmark, how would you describe/characterize its a) memory regularity, b) control regularity, c) locality?*

A:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Memory regularity | Control regularity | Locality | Explanation |
| mm.c | High | High | High | This program is tiled-matrix multiply. It accesses memory sequentially most of the time. All of the branches are data-independent. |
| lfsr.c | Low | High | Low | This program implements a linear-feedback shift register. It has data-independent loops. However, it accesses data randomly within the allocated array. |
| merge.c | High | Low | High | This program implements merge-sort. It accesses memory sequentially. It contains data-dependent branches that are hard to predict. |
| sieve.c | High | Low | Low | This program prints the number of prime numbers smaller than 1000000. It accesses memory in strides, and contain data-dependent branches. |

*4) For each microbenchmark, explain which microarchitecture parameter is it most sensitive to (in other words, what do you think the “bottleneck” is), and justify using reasoning or statistics.  
You might need to run other experiments, or look at the stats to help identify.*

A:

|  |  |
| --- | --- |
| Program | Most sensitive parameter |
| mm.c | **1) D-cache size:** This program allocated 3 matrices, each of 64kB, which wouldn’t fit in our 64kB d-cache. If we have larger d-cache, there will be less d-cache replacements.  **2) L2-cache latency:** Because it has to access l2-cache quite frequently, if l2-cache latency is lower, we will see better performance. |
| lfsr.c | **1) D-cache latency:** This program allocates a 64kB array, which ideally could fit in the 64 kB d-cache. |
| merge.c | **1) D-cache latency:** This program allocates a 64kB array, which ideally could fit in the 64 kB d-cache.  **2) Branch prediction accuracy:** This program has many data-dependent branches, which relies on accurate branch predictions. |
| sieve.c | **1) L2-cache latency:** This program allocates a 976kB char array, which can’t fit in d-cache, but would fit in l2 cache. If l2-cache latency is lower, we will see better performance.  **2) Branch prediction accuracy:** This program contains many data-dependent branches. |

*5) Pick a microbenchmark; propose one application enhancement, ISA enhancement, and microarchitecture enhancement that you think would be very effective for that microbenchmark.*

A: For sieve.c:

* Application: Programmer could do loop unrolling to reduce data dependencies. For example:

for (int p = 2; p < n; ++p)

total += !notprime[p];

can be optimized to:

for (int p = 2; p < n; p += 2)

total += (!notprime[p] + !notprime[p + 1]);

* ISA enhancement: Support longer SIMD operations. For example, the following code does not have data dependency, thus could be optimized using SIMD:

for (int p = 2; p < n; ++p)

total += !notprime[p];

Current x86 SSE instructions uses 128-bit XMM registers, which can manipulate 4 32-bit integers at a time. If SIMD operations support more bits, performance will improve on this microbenchmark.

* Microarchitecture enhancement:

This microbenchmark can benefit from lower l2-cache latency, because it allocates a char array of length 1000000 (976kB) would mostly reside in l2 cache. Lower l2 cache latency will improve performance.

*6) Which CPU model is more sensitive to changing the memory technology? Why?*

A: DerivO3CPU is more sensitive to change in memory controller. For example, for the benchmark sim\_seconds (Number of seconds simulated) and IPC, the results are as follows:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | | DerivO3CPU | | | MinorCPU | | |
| DDR3 | HBM | Diff | DDR3 | HBM | Diff |
| Sim\_seconds | mm | 0.007333 | 0.007613 | +3.82% | 0.031961 | 0.032021 | +0.18% |
| lfsr | 0.004179 | 0.004297 | +2.82% | 0.026545 | 0.026978 | +1.63% |
| merge | 0.002714 | 0.002736 | +0.81% | 0.006044 | 0.006085 | +0.67% |
| sieve | 0.025626 | 0.025892 | +1.03% | 0.037240 | 0.037245 | +0.01% |
| IPC | mm | 1.963531 | 1.891367 | -3.67% | 0.450526 | 0.449684 | -0.18% |
| lfsr | 1.334604 | 1.298024 | -2.74% | 0.210127 | 0.206757 | -1.60% |
| merge | 1.102896 | 1.093901 | -0.81% | 0.495230 | 0.491880 | -0.67% |
| sieve | 0.483480 | 0.478497 | -1.03% | 0.332692 | 0.332644 | -0.01% |

By analyzing the statistics, it can be seen that DerivO3CPU is more sensitive to memory controller types. This is probably because DerivO3CPU is out-of-order, which have higher costs for flushing the entire pipeline, thus more sensitive to memory latency.