Benchmark Analysis With GEM5

Overview

This analysis collects and analyzes data to guide improvements in the design of a custom processor. By leveraging GEM5 simulations, we aim to evaluate how key architectural features influence performance and identify areas for optimization. This report focuses on understanding the effects of branch prediction, prefetching, and frequency scaling on processor behavior by averaging data collected from the provided benchmarks.

The analysis examines metrics such as instructions per cycle (IPC), cycles per instruction (CPI), cache miss rates, memory latency, and execution efficiency. We investigate how well the processor handles instruction fetching, memory accesses, and branching by correlating these metrics with design choices. Special attention is given to identifying bottlenecks in the cache, the impact of read-heavy memory workloads, and performance scalability when the frequency is increased (doubled in this analysis).

This report provides insights into the processor's limitations and proposes design enhancements that improve IPC, reduce latency, and enhance overall efficiency. This approach will ensure that the processor design aligns with performance goals and effectively handles different benchmarks.

Baseline CPU Performance

1. **Instruction Composition**:
    - Integer ALU instructions (1,628,343) dominate the workload, with negligible integer multiplication (25,573) and division (1,216) instructions.
    - Load instructions (717,576) are significantly higher than store instructions (230,797), reflecting a read-heavy workload.
2. **Cache Performance**:
    - The instruction cache (I-cache) has a high miss rate of 23.08%, suggesting significant inefficiencies in instruction fetching.
    - The data cache (D-cache) miss rate is low at 0.75%, which suggests effective utilization of data caching mechanisms.
3. **Fetch and Branch Statistics**:
    - The fetch rate (2.9033), branch rate (0.10851), and branch mispredict count (9,336) suggest a workload that is moderately dependent on fetching with relatively few branches. The low branch rate indicates a low level of branching instructions relative to the total number of instructions.
4. **Read vs Write Requests**:
    - The memory system sees more read requests (42,990) than writes (1,026), which aligns with the earlier observation of a read-heavy workload.
5. **Average Memory Access Latency**:
    - The average memory access latency of 28,802 cycles is high. This may indicate high latency in the DRAM or an issue with memory access, such as poor locality.

Connecting the Stats to Optimizations

1. Prefetching:

- The high instruction cache miss rate and the high number of memory references suggest inefficiencies in fetching both instructions and data. The data cache performs well (0.75% miss rate), but prefetching could improve this further.
- For the instruction cache, prefetching could significantly reduce the high miss rate, improving instruction fetch efficiency.
- For data caches, prefetching could minimize latency due to the high number of memory references.
- **Results of Prefetching**:
    - Improves fetch rate by 10.53% by proactively keeping the pipeline filled with the correct instructions and data.
    - Minimizes memory stalls by 8.01%, especially critical, given the high DRAM latency.
    - Decreases the cache miss rate by about 16% on average for the I-cache and D-cache.

2. Branch Predictor:

- The branch rate is 0.10851, meaning branches are a small fraction of the overall instruction mix.
- While the branch rate is low, mispredictions can still cause pipeline stalls, leading to performance degradation.
- Given the high instruction cache miss rate, a poor branch prediction could worsen the delays, as the CPU might prefetch incorrect instructions or miss opportunities to keep the pipeline fed.
- **Results of Branch Predictor**:
    - A better branch predictor would ensure that the correct execution path is predicted more often (Improved prediction accuracy by 63.60% in analysis).
    - This would minimize pipeline flushes and stalls, improve the fetch rate, and reduce the time spent waiting for instructions by decreasing the number of cycles by 8.58% in analysis tests.
    - One downside is that both I-cache and D-cache showed an increase in miss rate by an average of about 8%; however, this should be offset by prefetch implementation.

3. Doubling Frequency:

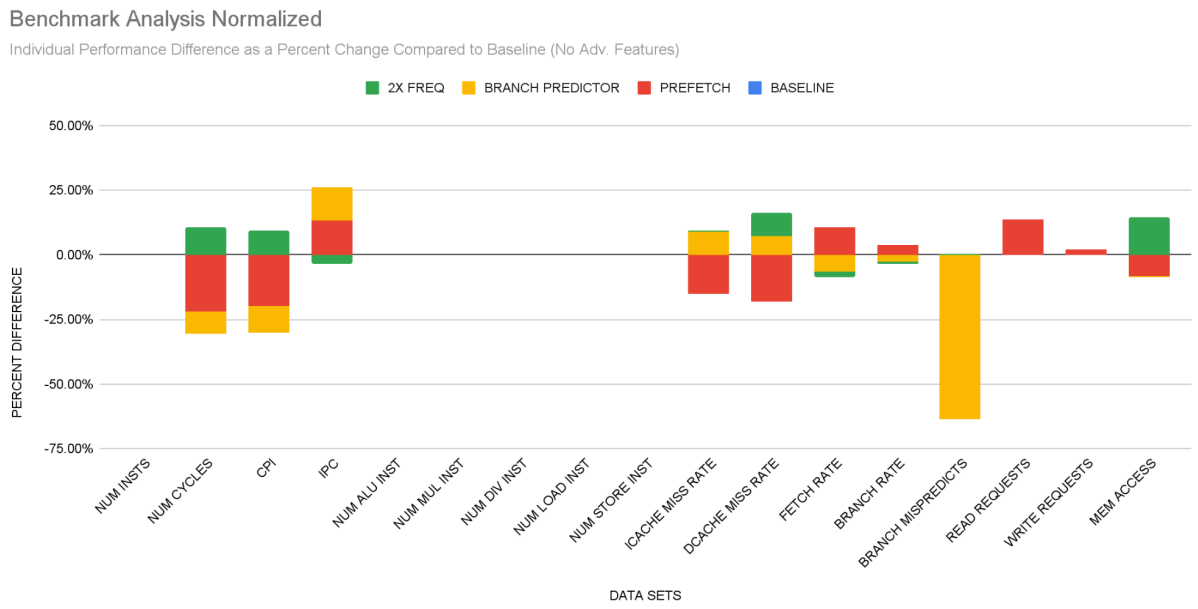
- Faster clock cycles could partially mitigate the high memory latency, as memory operations might be completed in fewer CPU cycles.
- Doubling the frequency would allow each pipeline stage to process instructions faster, effectively improving the throughput for workloads that are not memory-bound.
- For memory-bound workloads, increasing the frequency could reduce the memory latency. For example, a memory operation that takes 28,802 cycles could effectively take half the time in real-world seconds.
- **Results of Doubling Frequency:**

- Doubling the frequency reduced the number of cycles required for execution by 10.58%, leading to a 9.6% improvement in CPI. This indicates that tasks are completed faster due to shorter cycle times.
- The fetch rate improved by 2.43%, and instruction cache miss rates dropped slightly by 0.30%, showing a modest enhancement in fetching efficiency.
- Memory access latency increased by 14.64%, highlighting a potential trade-off that prefetching can mitigate.

Conclusion

This report found that the processor's performance is slowed by high instruction cache misses and long memory access times, even though it runs efficiently overall. Improving branch prediction and adding better prefetching can help keep the pipeline running smoothly and reduce delays. Doubling the frequency showed promise, but better cache and memory management are needed to work well. We will optimize the instruction cache, add prefetching, and make branch prediction more accurate to improve the processor. These changes will help the processor run faster and handle tasks more efficiently. Below is the supporting data for our findings.

Appendix



Benchmark Normalized Average Data Visualization (Image 1)

AVG	SIM INSTS	NUM CYCLES	CPI	IPC	ALU INST	MUL INST	DIV INST	LOAD INST	STORE INST	ICACHE MISS	DCACHE MISS	FETCH RATE	BRANCH RATE	BRANCH MISPREDICTS	READ REQUESTS	WRITE REQUESTS	MEM ACCESS LATENCY	FREQ
BASELINE	2603507.667	1521187.333	0.6135976667	1.990038667	1628343.667	25573.66667	1216.666667	717576.3333	230797.3333	0.230867	0.007500666667	2.903307667	0.1085193333	9336.333333	42990.33333	1026	28801.73333	100
PREFETCH	2603507.667	1190414.333	0.4912703333	2.253455333	1628343.667	25573.66667	1216.666667	717576.3333	230797.3333	0.195856	0.006135	3.208985	0.1127673333	9336.333333	48839.66667	1046	26494.73	100
BP	2603507.667	1390744.333	0.5519053333	2.249906667	1628343.667	25573.66667	1216.666667	717576.3333	230797.3333	0.2518623333	0.008042666667	2.719273	0.1056603333	3398	42995.33333	1027	28668.08667	100
FREQ	2603507.667	1682168	0.6725166667	1.922917	1628343.667	25573.66667	1216.666667	717576.3333	230797.3333	0.2315616667	0.008174	2.832664333	0.1077553333	9379.666667	42988.33333	1025.666667	33018.01667	200

Benchmark Average Data Raw (Image 2)