Performance Impact of OoO Execution and Speculation in gem5

Sai Kaushik S (2025CSZ8470)
Yosep Ro (2025ANZ8223)
Indian Institute of Technology Delhi

October 6, 2025

1 Stats and Metrics

We conducted our experiments using the **gem5 simulator (v25.0)** [1] [2] [3] under a wide range of cache and system configurations. The study considered variations in key architectural components such as cache hierarchy (L1/L2 size, associativity, and replacement policy), prefetcher design, reorder buffer and pipeline width, and branch prediction mechanisms. This extensive configuration enabled an in-depth analysis of how cache and speculative execution behaviors affect overall processor performance.

From each simulation run, we collected a comprehensive set of metrics from the `stats.txt` output, including *Instructions per Cycle (IPC)*, *Average Memory Access Time (AMAT)* for L1 and L2 caches, *cache hit rates*, *Misses Per Kilo-Instruction (MPKI)*, *branch misprediction rate*, and the number of *pipeline flushes*. Additional metrics such as *effective stall cycles* and *energy proxies* were also gathered to capture performance and energy trends across the cache hierarchy. These metrics illustrate how architectural parameters affect performance under a memory-dependent workload. Metrics related to *memory-dependence speculation success/failure* were excluded, as the memory-dependence predictor was not implemented in this work.

Table 1: Pointer Kernel: Effect of L1 Cache Size (L2=256KiB, Prefetcher=None)

L1 (KiB)	IPC	L1AMAT	L2AMAT	Hit Rate (%)	Stalls (M)	Simulation (s)
16	0.1559	6.3841	129.0945	95.8293	0.0002	0.0042
32	0.1558	7.5847	157.9266	95.8305	0.0002	0.0042
64	0.1558	7.5828	157.9526	95.8324	0.0002	0.0042

Table 2: Pointer Kernel: Effect of L2 Cache Size (L1=32KiB, Prefetcher=None)

L2 (KiB)	IPC	L1AMAT	L2AMAT	Hit Rate (%)	Stalls (M)	Simulation (s)
256	0.1558	7.5847	157.9266	95.8305	0.0002	0.0042
1024	0.1526	7.5856	157.9571	95.8307	0.0002	0.0043
4096	0.5614	1.4169	10.0024	95.8322	0.0000	0.0012

Tables 1–3 summarize representative results for the **Pointer Kernel**, focusing on L1/L2 cache size and prefetcher configuration (Regarding kernel for the experiment, you can reference

Table 3: Pointer Kernel: Effect of L2 Prefetcher Type (L1=32KiB, L2=256KiB)

Prefetcher	IPC	L1AMAT	L2AMAT	Hit Rate (%)	Stalls (M)	Simulation (s)
none	0.1558	7.5847	157.9266	95.8305	0.0002	0.0042
tagged	0.2427	8.0100	192.0167	96.3493	0.0002	0.0027
ampm	0.2079	5.9775	124.3867	95.9984	0.0002	0.0032
signature	0.2405	5.6341	106.9135	95.6656	0.0002	0.0027
stride	0.1558	7.5847	157.9266	95.8305	0.0002	0.0042

our Design Report). From the above tables, we can notice that increasing the L1 cache size has no effect on the overall IPC of the runs. But, increasing the size of the L2 cache gives a boost to the IPC of the run. Despite the irregular locations in the memory, using a prefetcher does improve the performance of the pointer-chasing kernel.

More detailed results across other benchmarks (*compute* and *stream* kernels) and extended configurations are available in the public dataset at: https://github.com/sai-kaushik-s/gem5-cache or the rest of this report.

2 Analysis

2.1 Analysis of Corellation between different parameters.

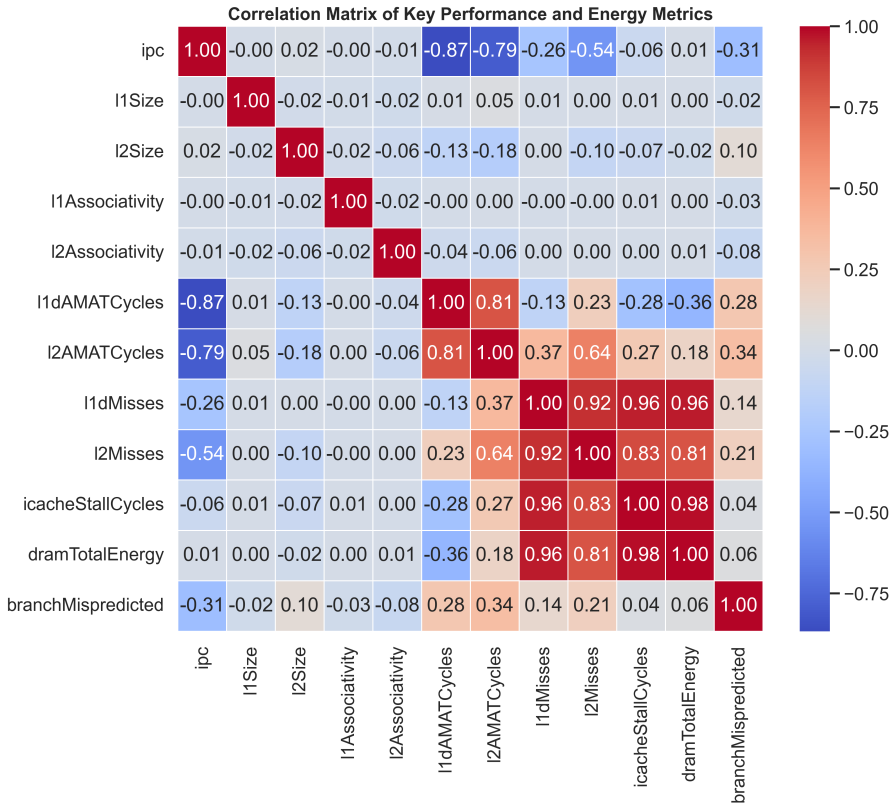

Figure 1: Correlation map of the different metrics from the benchmarks

Figure 1 shows the correlation matrix of key performance and energy-related metrics obtained from multiple benchmark configurations. The correlation coefficients highlight how changes in cache hierarchy and speculation behavior jointly influence processor performance.

A strong negative correlation is observed between **IPC** and both **L1D AMAT** ($r = -0.87$) and **L2 AMAT** ($r = -0.79$), indicating that higher cache access latencies directly reduce instruction throughput. This confirms that memory latency is a dominant bottleneck in most workloads, especially for pointer-intensive and memory-bound kernels. Similarly, **L1D misses** and **L2 misses** exhibit a strong positive correlation ($r = 0.92$), showing that miss penalties propagate through the hierarchy rather than being fully hidden by prefetching or buffering. **DRAM energy** and **stall cycles** also show high correlation values ($r > 0.95$), reflecting that longer memory access paths not only degrade performance but also increase total dynamic energy due to prolonged DRAM activity. Meanwhile, **branch misprediction rate** has a moderate negative correlation with IPC ($r = -0.31$), consistent with the intuition that frequent pipeline flushes disrupt instruction flow but do not dominate the overall performance loss compared to memory stalls.

Interestingly, cache size and associativity parameters show almost negligible direct correlation with IPC or AMAT metrics. This suggests that, within the explored configuration space, *cache latency and miss behavior*—not the nominal cache capacity—primarily govern performance. This trend can also be explained by the nature of the kernels we implemented. In the **compute kernel**, the working set of variables is relatively small, causing performance to remain mostly unaffected by cache size. In contrast, the **stream kernel** exhibits high spatial and temporal locality, which results in consistently high hit rates regardless of cache capacity. However, as shown in Table 2, the **pointer kernel** displays noticeable performance variation with changing L2 cache size, since its irregular access pattern generates higher miss penalties and exposes memory latency more prominently.

Overall, this correlation analysis quantitatively validates the architectural trends observed in Section 1. Memory access latency and miss behavior remain the key performance determinants, while improvements in speculation or cache capacity yield comparatively smaller gains.

2.2 Performance Analysis of IPC and AMAT.

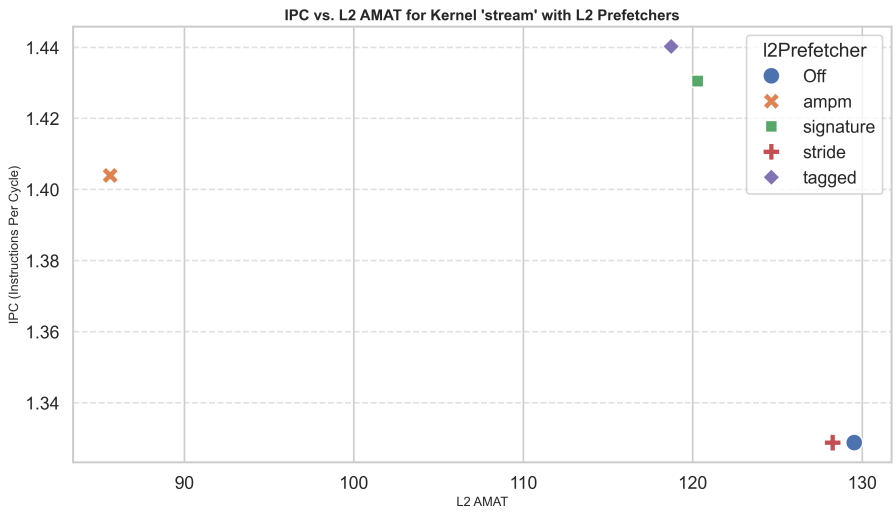

Figure 2: IPC vs L2 AMAT for the 'stream' kernel with the various prefetchers

As seen in the Fig. 2, we can notice that a prefetcher improves the IPC of the benchmark run. From the multiple runs, it is noted that the *tagged* prefetcher returns the highest IPC gains compared to the baseline run with the remaining parameters kept constant. It is followed by *signature* and then *AMPM* prefetchers. The *stride* prefetcher performs as well as without a prefetcher as well.

For an application that accesses multiple data streams at once, as used in our 'stream', *tagged* prefetcher works the best as a simple stride prefetcher would get confused. This is reinforced by the above plot.

Possible explanations of why the *AMPM* prefetcher gets a lower IPC despite the least AMAT could be:

- **Low Prefetch Accuracy:** The prefetcher coule fetch the data that the processor never uses. This useless data is brought into the L2 cache from the memory and potentially evicting useful data. When the data is accessed, it encounters a cache miss, which imacts the overall IPC of the run.
- **Lack of timeliness:** If the prefetch is too early, the data might never be used by the processor and could be evicted when some other data is to be fetched from the memory, making the prefetch useless.

2.3 Performance Analysis of Cache Miss and Stall cycles.

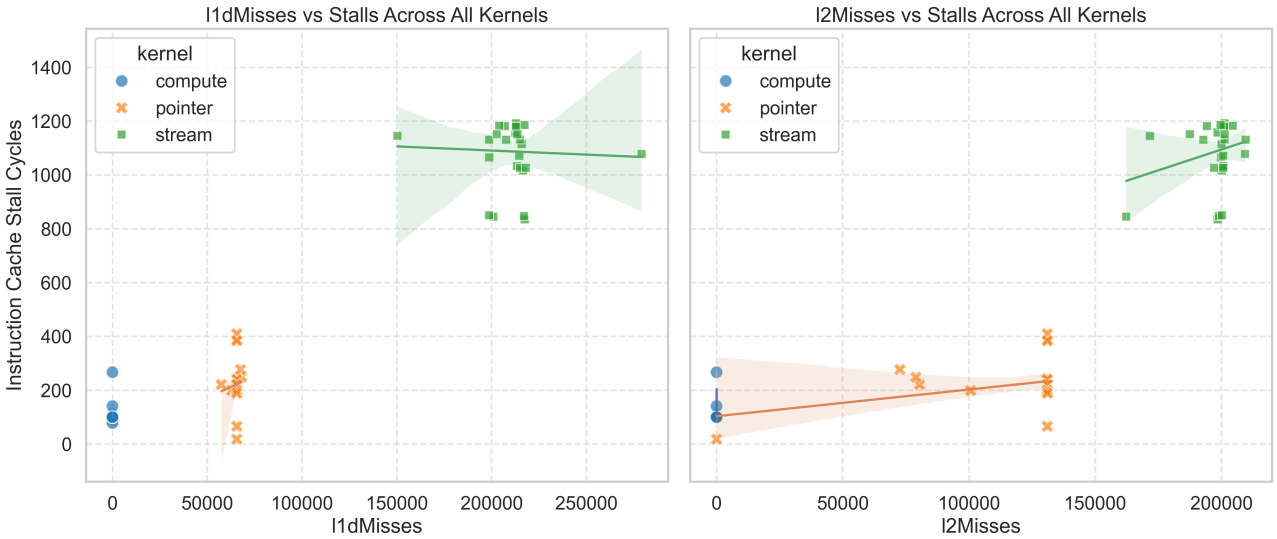

Figure 3: Instruction Stall Cycles vs L1 and L2 Cache Misses

From the Fig. 3,

- The 'stream' kernel is clearly memory-bound. There is a large amount of misses in both the L1 and L2 caches. The continuous stream of data requests likely overwhelms the cache and memory subsystem, causing the instruction fetch unit to wait.
- The 'pointer' kernel demonstrates the impact of an unpredictable access pattern. It has a moderate number of L1 misses but larger number of L2 misses. This results in a moderate number of stall cycles langing inbetween the 'comput' and 'stream' microbenches.
- The 'compute' kernel is compute-heavy and only has the compulsory misses. It has no other misses throughout the run, making its stalls due to the dependecy on the precious instructions. Thus, has the least stalls.

3 Conclusion

The expected behavior of the microbenches, as mentioned in the design report, is validated by the data from the multiple runs, as seen from the tables and the plots.

- To improve the performance of the 'comput' kernel, we need to increae the CPU parameters like the pipeline width. Another way to improve the performance is to introduce parallel computing.
- To improve the performance of the 'pointer' kernel, we notice that increasing the L2 cache size improves the performance as it reduces its miss rate, thus reducing the high latency memory accesses.
- To improve the performance of the 'stream' kernel, we need to use an appropriate prefetcher, which goes a long way by fetching the required data during the compute cycles of the previous indices.

References

[1] N. L. Binkert et al., "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, 2011. DOI: 10.1145/2024716.2024718. [Online]. Available: https://doi.org/10.1145/2024716.2024718.

[2] J. Lowe-Power et al., "The gem5 simulator: Version 20.0+," *CoRR*, vol. abs/2007.03152, 2020. arXiv: 2007.03152. [Online]. Available: https://arxiv.org/abs/2007.03152.

[3] A. Hansson, N. Agarwal, A. Kolli, T. F. Wenisch, and A. N. Udipi, "Simulating DRAM controllers for future system architecture exploration," in *2014 IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS 2014, Monterey, CA, USA, March 23-25, 2014*, IEEE Computer Society, 2014, pp. 201–210. DOI: 10.1109/ISPASS.2014.6844484. [Online]. Available: https://doi.org/10.1109/ISPASS.2014.6844484.