Analysis of cache hierarchy in IBM POWER architectures

Saurabh Jacob Kuruvila
Virginia Tech

Shubham Jogprakash Ghosh
Virginia Tech

Abstract

With the plateauing of Moore's law, we have seen a gradual shift in the computation industry towards multi-core systems which are highly dependent on low memory access time to exploit parallel compute capabilities. To achieve this, modern systems use intermediate memory caches - that are often set up in a hierarchical structure. These caches have low memory access latency and help alleviate bottlenecks in data-intensive workloads. In this work, we review the effect of adding three and four-level caches on the IBM POWER architecture to evaluate cryptographic (MD5), search (Breadth-First Search), and sort (Bucket Sort) algorithms. Finally, we then analyze the execution time and miss rate from the Gem5 simulation on the two, three, and four-level configurations, and highlight the pros and cons of deeper and larger caches.

1 Introduction

A cache is an intermediate storage component that is placed in close proximity to the CPU to decrease the latency of memory operations. The cache memory is typically built on high-speed static random access memory (SRAM) modules that are much faster (10 to 100x) than the dynamic random access memory (DRAM) modules used in RAM - primary memory. In order for these caches to be placed so close to the CPU as well as operate at higher speeds, they are much smaller in size (memory capacity) compared to primary memory.

Caches work by storing copies of frequently used data from the main memory closer to the processing element thereby decreasing access time when there is a cache hit. That being said there is also a quantifiable overhead in traversing through the various cache levels and a cache miss can result in higher latency than a system without a cache. In light of this, the workloads that are run on a system equipped with cache must be aware of the spatial and temporal locality of their data to maximize the performance gains.

The performance improvements that caches bring to a system are further exemplified by the slowing down of Moore's law which has pushed the industry away from uni-core single thread focused architectures to multi-core multi-threaded systems. This kind of parallelization has been the primary source of performance gains in the last decade but comes at the price of high memory pressure. As discussed throughout the course the memory wall can be alleviated by caches as a latency-reducing technique, given the right workload. In light of this, we must understand and optimize the cache hierarchy to extract the maximum performance of a system. Generally, caches are divided into several levels - which are defined by their proximity to the processor as shown in Figure 1.

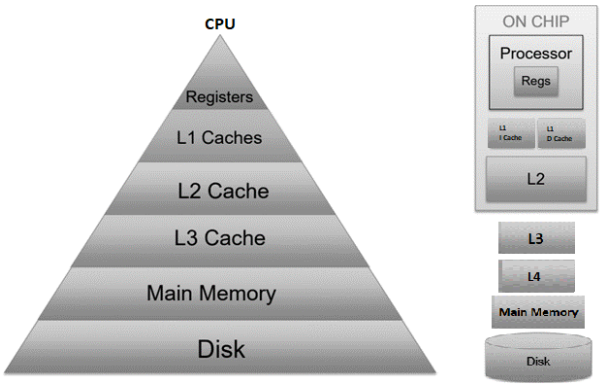

Figure 1: Example of memory hierarchy on a single core system

While discussing the various levels of cache it is also important to address how memory is mapped in cache with respect to main memory, namely:

Direct mapped cache has a contiguous set of bits/bytes (block) mapped to exactly one location of the cache.

Fully associative cache mapping has a contiguous set of blocks mapped to any location of the cache.

Set associative cache mapping is a hybrid mapping technique that maps a block to a subset of locations on the cache.

Since caches are arranged hierarchically, each level of the cache may be mapped uniquely. We see that having higher

associativity results in lower conflict misses but comes at the cost of greater hit time. Overall the cache hierarchy is structured such that the closest L1 cache is designed with an emphasis on hit time (lower associativity) whereas the L3 cache is designed to reduce miss rate. This will be an important factor while determining cache associativity for each level.

In traditional multi-core setups, L1 and L2 cache are bound to their closest CPU along with a shared L3 and L4 cache. IBM first introduced the L3 cache in 2001 on the POWER4 architecture and later the L4 cache on the POWER8 architecture. For our study we will focus on the IMB POWER 7 and POWER 8 platforms [14], [15].

2 Related Work

[1] demonstrates the performance effects of a shared L3 victim cache coupled with a cache replacement policy with an HPC benchmark tool. Similarly, Article [6] highlights tests run on different cache architectures to obtain a wider view of the performance implications of each one giving us a comparative view of the latency with respect to the size of the cache. Simulations run in [5] assess cache compression on various cache levels and give us insight into performance evaluation in simulation environments. [4], evaluates the performance when a history table is added between the L2 and L3 cache to help the L2 cache determine which lines are available in the L3 cache and the benefits of allowing L2 cache write-backs to the L2 cache itself rather than consuming the L3 cache.

IBM followed a trend of increasing the levels of the caches up to their POWER 8 architecture (L4), [3] but later dropped it and revisited the benefits of larger L3 caches (8MB to 120Mb) [2] [9] [10] in their POWER 9 systems. This was primarily attributed to access latency of an extra cache level [17] and a trend that we expect to see with our workloads with L4 cache equipment setup.

Figure 2 shows the trends in different parameters as the POWER architecture progresses.

POWER7	POWER8	POWER9
1.2 billion transistors	4.2 billion transistors	8 billion transistors
45 nm process	22nm process	14nm process
Peak bandwidth: 40 GB/s	Peak bandwidth: 96 GB/s	Peak bandwidth: 192 GB/s
L1 Cache: 32+32 KB/core	L1 Cache: 64+32 KB/core	L1 Cache: 32+32 KB/core
L2 cache: 256 KB/core	L2 cache: 512 KB/core	L2 cache: 512 KB/core
L3 cache: 4 MB/core	L3 cache: 8 MB per chiplet	L3 cache: 120 MB per chip

Figure 2: Trends in IBM POWER architecture [8]

Additionally, we also referred o material from the coursework that focused on memory hierarchy. Homework 2 gave us practical insights into caching via GEM5 simulations on the POWER architecture, where we see about 24x speedup versus a system without cache. Results from here were further enhanced in homework 3 where we see further performance gains that optimization of loop indices on a matrix multiplication function has on improving locality of data. Homework 4 also shows us that the associativity of each cache level can influence the memory access time, with diminishing results at higher associativity levels due to blocks being under-saturated. Overall, we learned that the average memory access times are dependent on cache miss rates, cache hit rates, and cache miss penalties of each level of cache and are also heavily influenced by the nature of the workload being run.

3 Gem5 Setup

For our study, we will utilize the Gem5 simulator to set up various cache hierarchies on the IBM POWER 7 and POWER 8 architecture.

POWER 7:
L2 cache size: 256KB Associativity: 8-way set associative
L3 cache size: 4MB Associativity: 8-way set associative

POWER 8:
L2 cache size: 512KB Associativity: 8-way set associative
L3 cache size: 8MB Associativity: 8-way set associative
L4 cache size: 128MB Associativity: 16-way set associative

To validate the associativity configuration for two different workloads, viz. 128*128 Matrix Multiplication and MD5 by varying the associativity values and the sizes of each level of cache (20 different settings) to obtain different simulation results. Post running these tests, we compared the outputs to recognize the best settings for the POWER architecture. On observation of the simulation times, the workloads performed best when the associativity value for the L2 cache was 8, the associativity value for the L3 cache was 8 and the associativity value for the L4 cache was 16. This is in accordance with the values set by IBM in their architecture.

To add the two additional layers of shared cache we start by mortifying the caches.py file. Primary changes involved the creation of L3Cache and L4cache classes and Initializing cache parameters based on each architecture. Second, the simulator driver (SimulateTests.py) is modified to create instances of each cache class and their associated memory busses. We initially intended to create individual bus classes (L3Xbar and L4Xbar) for each interconnection by realizing the functionality would remain the same between the L2, L3, and L4 buses. An easier approach was to create new instances of the L2Xbar and bind each instance to a unique level. Depending on the configuration used - POWER 7 or POWER 8, the last cache level would be connected to the members that are connected to a DDR3 1600 Mhz DRAM module. Finally, 3 workloads - MD5, bucket sort, and graph traversal via breadth-first search were implemented that targeted 1Mb,

4MB, and 8MB of memory usage.

4 Workloads

A system that leverages caches for performance gains is heavily dependent on the workload and its ability to exploit the spatial and temporal locality of data. As seen in previous assignments a poorly optimized workload such as matrix multiplication with column-major access cannot make efficient use of the caches making any changes in size and hierarchy irrelevant. In light of this, we attempted to evaluate a diverse set of workloads while staying in line with modern computing algorithms in cryptography and data manipulation (sort and search) which have their memory access patterns.

4.1 MD5 encryption

MD5 is a message digest-based cryptographic algorithm that is used as a one-way function hash function. It takes a message of any length to return a hashed value of a fixed length by processing it in blocks of 512 bytes. In modern-day applications, MD5 is used for checksum purposes to detect any unintentional corruption of transmitted data.

We have chosen this algorithm as it is a good representation of a modern cryptographic workload. The hash calculation in MD5 is primarily based on matrix transformations. Since the operations tend to access sequential chunks of data while calculating each hash for an individual test string, we can expect to see good spatial and temporal locality of data.

MD5 Simulation Results

Architecture	Cache level present	Workload size	Simulation Seconds	Cache Overall Miss Rate				
				L1D	L1I	L2	L3	L4
POWER 7	L2 = 256KB	1MB+	0.340088	0.001219	0.003853	0.069311	-	-
		4MB+	1.530014	0.001219	0.003849	0.067455	-	-
		8MB+	2.890127	0.001219	0.003848	0.067223	-	-
POWER 7	L2 = 256KB L3 = 4MB	1MB+	0.340031	0.001219	0.003853	0.069311	0.515159	-
		4MB+	1.536132	0.001219	0.003849	0.067455	0.999960	-
		8MB+	2.901614	0.001219	0.003848	0.067223	0.999857	-
POWER 8	L2 = 512KB	1MB+	0.340085	0.001219	0.003853	0.069299	-	-
		4MB+	1.530010	0.001219	0.003849	0.067452	-	-
		8MB+	2.890127	0.001219	0.003848	0.067212	-	-
POWER 8	L2 = 512KB L3 = 8MB	1MB+	0.340031	0.001219	0.003853	0.069299	0.515250	-
		4MB+	1.529591	0.001219	0.003849	0.067452	0.503485	-
		8MB+	2.895957	0.001219	0.003848	0.067212	0.771497	-
POWER 8	L2 = 512KB L3 = 8MB L4 = 128MB	1MB+	0.340765	0.001219	0.003853	0.069299	0.515250	1.000000
		4MB+	1.532728	0.001219	0.003849	0.067452	0.503485	1.000000
		8MB+	2.898358	0.001219	0.003848	0.067212	0.771497	0.650493

Figure 3: Simulation times for MD5 workload

4.1.1 Analysis of results

From the table 3 and graphs 6,7,8 we can see that the MD5 workload does not gain any performance with an increase in L2 cache size (256KB on POWER7 to 512KB on POWER8) as there no notable difference in execution time for 1MB+, 4MB+ and 8MB+ workloads. i.e. poor scaling with an increase in L2 cache size.

The addition of a 4MB L3 cache level to the POWER 7 configuration shows a mixed trend. Small workloads of 1MB+ that fit within the L3 cache show a marginal improvement vs L2-only configurations but larger workloads of 4MB+ and 8MB+ show a decrease in performance most likely caused by over-saturation of the L3 cache. A similar trend is seen with the POWER 8 configuration whose L3 cache is saturated by the 8MB+ workload but shows a marginal speed up with the 1MB+ and 4MB+ workload.

Power 8 also supports an L4 cache of 128MB which shows a higher execution time across the board when compared to the same system without an L4 cache. This is most like due to the increased overhead of filling and accessing an extra layer of cache. This can be seen in the overall miss rate parameter that shows the L4 cache not being hit with small workloads of 1MB+ and 4MB+.

4.2 Breadth first search

The Breadth-First Search (BFS) algorithm is used to search for a node in a graph-based data structure. Here we traverse from the root of the graph to each level and check all the nodes available at that level before moving on to a lower level in the graph.

We have chosen this workload because we believe that the algorithm will have a poor data locality as cache lines will be flushed out to accommodate new for new nodes during traversal. This would imply that the BFS workload will be cache intensive as blocks are replaced every few iterations.

Breadth First Search Simulation Results

Architecture	Cache level present	Workload size	Simulation Seconds	Cache Overall Miss Rate				
				L1D	L1I	L2	L3	L4
POWER 7	L2 = 256KB	1MB+	0.086384	0.005672	0.009111	0.115039	-	-
		4MB+	0.375639	0.006846	0.010054	0.100467	-	-
		8MB+	0.750196	0.006014	0.010040	0.102614	-	-
POWER 7	L2 = 256KB L3 = 4MB	1MB+	0.086398	0.005905	0.009111	0.115039	0.491419	-
		4MB+	0.380024	0.007017	0.010054	0.100467	0.838975	-
		8MB+	0.762555	0.006080	0.010040	0.102614	0.980176	-
POWER 8	L2 = 512KB	1MB+	0.086340	0.005672	0.009111	0.113270	-	-
		4MB+	0.375639	0.006846	0.010054	0.100467	-	-
		8MB+	0.750199	0.006014	0.010040	0.102614	-	-
POWER 8	L2 = 512KB L3 = 8MB	1MB+	0.086378	0.005852	0.009111	0.113270	0.499092	-
		4MB+	0.375506	0.007101	0.010054	0.100467	0.478024	-
		8MB+	0.758753	0.006183	0.010040	0.102614	0.831989	-
POWER 8	L2 = 512KB L3 = 8MB L4 = 128MB	1MB+	0.087113	0.005852	0.009111	0.113270	0.499092	1.000000
		4MB+	0.378717	0.007101	0.010054	0.100467	0.478024	1.000000
		8MB+	0.760987	0.006268	0.010040	0.102629	0.831989	0.572745

Figure 4: Simulation times for Breadth First Search workload

4.2.1 Analysis of results

From the table 4 and graphs 6,7,8 we can see that the BFS workload has little to no change in execution time for architectures setup with only L2 caches of 256KB and 512KB respectively. This implies that an increase in the size of the L2 cache is not very effective.

Next, we can see that the addition of a 4MB L3 cache to the POWER 7 configuration does decreases execution time for the 1MB+ workload but shows a significant slowdown for higher workloads of 4MB+ and 8MB+ due to the increased miss rate caused by over-saturating the L3 cache. It can also be seen that a similar slowdown can be seen for the POWER 8 configuration with 8MB of L3 cache for similar reasons. This result shows us that an increase in L3 cache size can benefit the performance of the system on this workload.

Finally, we can see that the POWER 8 architecture has an additional 128MB of L4 cache but show degraded performance in all instances of BFS workloads (1MB+,4MB+ and 8MB+) as seen in the increased execution time. We believe this is most likely due to the increased cost of filling and traversing an extra layer of cache.

4.3 Bucket sort

Bucket sort is a sorting algorithm that breaks down the array of numbers that need to be sorted into smaller chunks called - "buckets" and sorts the buckets individually. These buckets are recursively sorted using the insertion sorting algorithm. In insertion sort, each item is sorted and placed in the final sorted position before moving to the next item.

This algorithm was selected as memory allocation is done for a linked list that is constantly accessed and updated. The insertion sort algorithm that runs on this linked list also follows an in-place replacement that can allude to good spatial locality and can benefit from the addition of a cache.

Bucket Sort Simulation Results								
Architecture	Cache level present	Workload size	Simulation Seconds	Cache Overall Miss Rate				
				L1D	L1I	L2	L3	L4
POWER 7	L2 = 256KB	1MB+	0.100644	0.060221	0.000026	0.172095	-	-
		4MB+	0.952702	0.059644	0.000004	0.216729	-	-
		8MB+	1.515671	0.016252	0.000002	0.800612	-	-
POWER 7	L2 = 256KB L3 = 4MB	1MB+	0.099001	0.065784	0.000026	0.172095	0.08536	-
		4MB+	0.931198	0.066619	0.000004	0.216729	0.06176	-
		8MB+	1.575232	0.017992	0.000002	0.800612	0.75669	-
POWER 8	L2 = 512KB	1MB+	0.093087	0.060221	0.000026	0.014813	-	-
		4MB+	0.950691	0.059644	0.000004	0.212156	-	-
		8MB+	1.514144	0.016252	0.000002	0.793363	-	-
POWER 8	L2 = 512KB L3 = 8MB	1MB+	0.093432	0.060222	0.000026	0.014813	0.99167	-
		4MB+	0.929582	0.066408	0.000004	0.212156	0.06290	-
		8MB+	1.476576	0.023238	0.000002	0.793363	0.06242	-
POWER 8	L2 = 512KB L3 = 8MB L4 = 128MB	1MB+	0.093825	0.060222	0.000026	0.014813	0.99167	1.00000
		4MB+	0.933104	0.066408	0.000004	0.212156	0.06290	1.00000
		8MB+	1.482451	0.023238	0.000002	0.793363	0.06242	0.99876

Figure 5: Simulation times for Bucket Sort workload

4.3.1 Analysis of results

We can see from 5 and graphs 6,7,8 that the L2 cache only configurations we can see that the Bucket Sort algorithm sees a significant improvement in performance in the POWER 8 configuration with 512KB of L2 cache compared to the 256KB present in the POWER 7 configuration. This shows that this workload can benefit from an increase in L2 cache size.

Next, the addition of an L3 cache to both POWER 7 and POWER 8 configurations shows a consistent decrease in execution time with the exception of the 8MB+ workload run on the 4MB L3 configuration used the in POWER 7 test bench as it is over-saturated. Overall we can see that the Bucket sort workload does benefit from the addition of an L3 cache with larger sizes allowing for better results with more memory-intensive workloads.

The POWER 8 architecture has an additional 128MB of L4 cache and does show an improvement in performance for 1MB+, 4MB+ and 8MB+ bucket sort workloads when compared to the POWER 7 configuration. That being said the intermediate configuration of POWER 8 without an L4 cache shows even better performance and strengthens the argument that the L4 cache increases memory access overhead.

5 Observations

In this section, we describe the trends we have seen in the results that we obtained while simulating the behavior of POWER 7 and POWER 8 architectures when different workloads are run in the presence of different cache sizes and levels. We focus on the execution times and the miss rates for each cache level as the primary parameters for comparison as they best depict the performance traits of the workloads that were run. It can be seen that a lower miss rate yields lower simulation seconds and faster execution. Figures 6,7 and 8 show a graphical representation of the simulation time based on the size of the three workloads

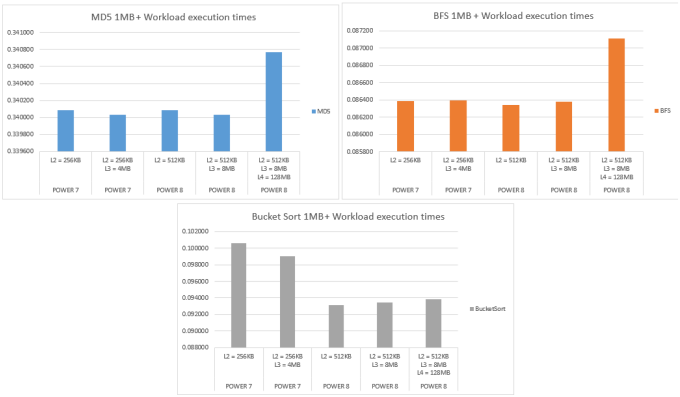

Figure 6: Simulation time graph for 1MB+ workload

5.1 Addition of L3 Cache

In both POWER 7 and POWER 8 architectures, we see a general trend where the addition of an L3 cache helps in the faster execution of the workloads. This is because the time

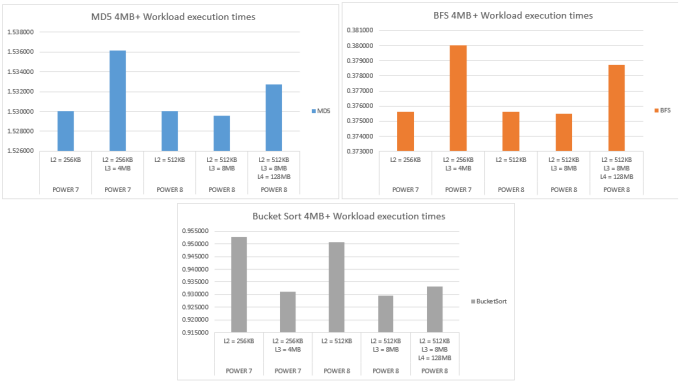

Figure 7: Simulation time graph for 4MB+ workload

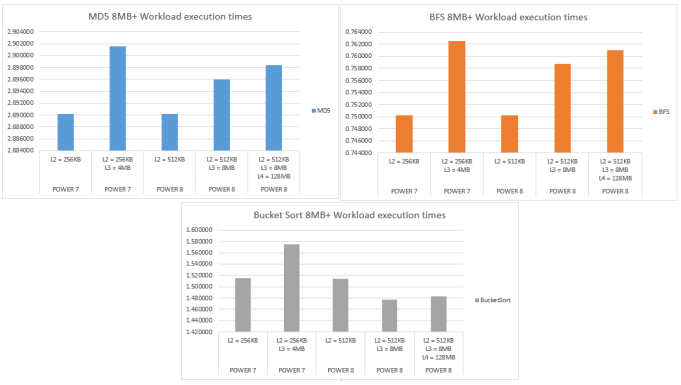

Figure 8: Simulation time graph for 8MB+ workload

to access the primary memory is more than accessing the L3 cache, and since the L3 cache is bigger than the L2 cache, it can store more data relevant to the application, making it easier to retrieve cache-intensive workloads. However, when the workload overflows the L3 cache's capacity, this benefit starts to diminish as there is still access to the primary memory that is required. We can see the same in the 8MB+ workload times as seen in graph 8.

5.2 Size of L2 and L3 Cache

The size of both the L2 and L3 cache was doubled as we moved from the POWER 7 architecture to the POWER 8 architecture. This meant that more data can be cached at each level for the workloads we were running. Hence, we see better performance numbers in the case of running the 1MB+ and 4MB+ workloads in the POWER 8 architecture. The 4MB+ workload over-saturates the L3 cache in POWER 7 but not in the POWER 8 architecture. This shows that having a larger L3 cache will prove beneficial when executing cache-intensive workloads.

5.3 Addition of L4 Cache

The last set of simulations we ran included an L4 cache of size 128MB similar to the POWER 8 architecture. The addition of the L4 cache is mostly an overhead for all the workloads and adds to the simulation time. Smaller 1MB+ and 4MB+ workloads never access the L4 cache which can be seen in the L4 miss rate of 1 in tables 3, 5, and 4. As the size of the workloads increase, this overhead decreases. This goes to show that if the workload's cache utilization is very high, such that the L4 cache is utilized consistently, the addition of the L4 cache may be beneficial. Other than that, we will always see L4 as additional overhead.

6 Performance with POWER 9

From the trends we observed in the previous section we can see that a bigger L3 cache size is more useful when it comes to executing workloads with bigger sizes. This is the primary reason why IBM moved from a 4-level cache hierarchy in POWER 8 to a 3-level cache hierarchy in POWER 9. To further test our workloads, we simulated the performance of our biggest workloads (8MB+) sizes on a POWER 9 architecture of L2 cache with 512KB and L3 cache of 128MB. Figure 9 shows the execution time comparisons of POWER 9 against POWER 8 both with and without an L4 cache. As noted in the previous sections, the L4 cache is an added overhead to the overall system. The graphs show that increasing the size of the L3 cache is a better solution than adding a separate L4 cache. This graph also emphasizes the reason why IBM dropped the L4 cache from their POWER 8 architecture, resulting in reduced latency and improved execution times. [17]

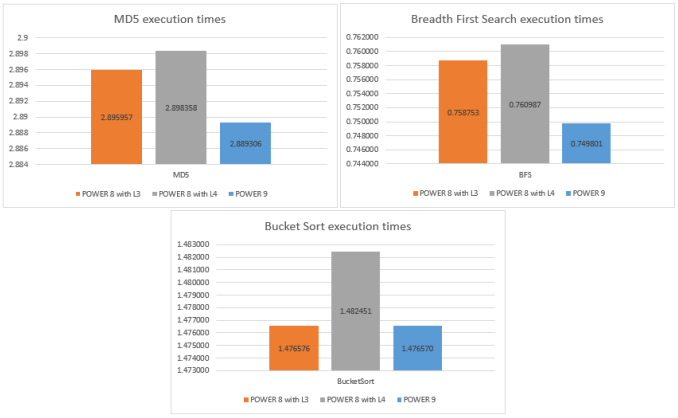

Figure 9: Performance comparison with POWER 9

7 Possible Future Work

The results from this study give us an overview of performance variations caused by the varying size and depth of a cache hierarchy. While diving deeper into the implementation we came across several parameters that could be better explored and tuned to improve overall system performance. A few areas of interest have been highlighted below.

- Testing the setup with a diverse set of cache-friendly and cache-intensive workloads. This will provide a better view of a general-purpose system that deals with cache-optimized and unoptimized applications.
- Testing the setup with larger size workloads to determine the cases where the L3 cache may overflow. This may give us better insight into the question of large L3 cache vs addition of L4 cache.
- Testing the simulation with accurate tag, data, and response latency gives us a more realistic view of how a system actually responds to a workload. A general trend is that as size increases in every level of cache, so does the latency.
- Evaluate different cache replacement policies such as First-In-First-Out (FIFO), Last-In-First-Out (LIFO), and Most Recently Used (MRU).

8 Conclusion

In conclusion, we can see the effect of adding an L3 cache to the architecture enables faster execution of most workloads and is substantiated by the performance results of our simulations. This result is especially true for the POWER 8 configuration where more often than not, the L3-equipped system outperforms one with L4. The outlier to this result can be seen with the POWER 7 configuration that has a small L3 cache that is easily saturated by modern workloads which results in a larger amount of misses in the cache. The overhead of L3 cache traversal followed by high primary memory access latency causes the execution time of larger workloads to suffer. This aligns with IBM's reasoning behind the doubling of L2 and L3 cache size while moving from POWER 7 to POWER 8.

The POWER 8 architecture also introduced an additional cache level, L4, which was much larger than the L3 cache. In our simulations, we observed that this addition proved to be an overhead for small workloads. This is primarily because of the overhead of loading the L4 cache which is eventually not utilized by these workloads. The L4 cache comes into the picture only when the L3 cache is over-saturated. We observed this scenario when we ran the 8MB+ MD5 workload which spilled into the L4 cache. Despite having an L4 cache the results do not show a significant improvement in execution time which is in line with IBM's decision to revert to an L3-based hierarchy with a large capacity. [16]

Through this study, we have evidence that a sufficiently sized L2 and L3 cache will extract the best performance from most workloads as seen in the simulation times of the L3 cache-only configuration of POWER9. While this is true, we cannot move to extremely large capacities for these cache levels due to underlying trade-offs such as increased cache hit time and increased die size. [12]

In general, we can safely conclude that the best performance for any workload is dependent on the underlying architecture which is not the same for different workloads. This means that to extract the best performance results for a specific workload, we need to tune the system's architecture to match its memory access patterns and computational unit requirements.

References

[1] Christie L. Alappat, Johannes Hofmann, Georg Hager, Holger Fehske, Alan R. Bishop and Gerhard Wellein, "Understanding HPC Benchmark Performance on Intel Broadwell and Cascade Lake Processors", High-Performance Computing: 35th International Conference, ISC High Performance 2020

[2] J. Cocke and V. Markstein, "The evolution of RISC technology at IBM," in IBM Journal of Research and Development, vol. 34, no. 1, pp. 4-11, Jan. 1990

[3] Starke, William J., Jeffrey Stuecheli, David Daly, J. Steve Dodson, Florian Auernhammer, Patricia Sagmeister, Guy L. Guthrie, Charles F. Marino, Michael S. Siegel and Bart Blaner. "The cache and memory subsystems of the IBM POWER8 processor." IBM J. Res. Dev. 59 (2015): n. pag.

[4] E. Speight, H. Shafi, Lixin Zhang, and R. Rajamony, "Adaptive mechanisms and policies for managing cache hierarchies in chip multiprocessors," 32nd International Symposium on Computer Architecture (ISCA'05), 2005

[5] Keun Soo Yim, Jihong Kim, and Kern Koh, "Performance Analysis of On-Chip Cache and Main Memory Compression Systems for HighEnd Parallel Computers", Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA '04, June 21-24, 2004

[6] https://www.tomshardware.com/news/amd-3d-v-cache-benchmarks-mixed-results-milan-x-cpus

[7] https://www.gem5.org/documentation/

[8] https://www.midlandinfosys.com/power9_vs_power8

[9] https://www.ciotechoutlook.com/news/what-s-the-difference-between-ibm-s-power8-and-power9-nid-5978-cid-21.html

[10] https://openpower.ic.unicamp.br/post/series-ibm-power/

[11] http://gibsonnet.net/aix/ibm/systems_power_software_i_perfmgmt_underthehood.pdf

[12] https://www.anandtech.com/show/16924/did-ibm-just-preview-the-future-of-caches

[13] https://www.anandtech.com/show/16084/intel-tiger-lake-review-deep-dive-core-11th-gen/4

[14] https://www.redbooks.ibm.com/redbooks/pdfs/sg248079.pdf

[15] https://www.redbooks.ibm.com/redbooks/pdfs/sg248171.pdf

[16] https://en.wikichip.org/wiki/ibm/microarchitectures/power9

[17] https://www.nextplatform.com/2018/08/28/ibm-power-chips-blur-the-lines-to-memory-and-accelerators/