Cache Architecture: To be Inclusive or Not To Be Inclusive?

CSCI 530: Computer Architecture and Design Project

Submitted By:
Md Main Uddin Rony
First Year PhD Student
The University of Mississippi

Introduction

The gap between processor and memory speeds hinders the performance of Chip Multiprocessors. An efficient and high performing cache hierarchy can reduce the gap and hence ensure better processor performance. There are multiple interesting questions arise pertaining to the design of memory hierarchy. One of them is whether it should consider cache as inclusive or exclusive? Both designs have their own advantages and disadvantages with performance tradeoff. While exclusion offers bigger space for the data, simplifies the cache coherence protocol, it limits performance when the size of the largest cache is not significantly larger than the sum of the smaller caches. On the other hand, the main advantage of inclusive cache is that when external devices or other processors in a multiprocessor in a multiprocessor system wish to remove a cache line from the processor, they need only have the processor check the L2 cache. Another advantage of inclusive caches is that the larger cache can use larger cache lines, which reduces the size of the secondary cache tags. (Exclusive caches require both caches to have the same size cache lines, so that cache lines can be swapped on a L1 miss, L2 hit).

Processors in modern days commonly follow inclusive design (Natalie, n.d.). IBM carried this tradition into the realm of CMPs by implementing it in the IBM Power4 CMP **(**Barosso, 2000**)**. The general approach used by the Intel processors is for the L3 cache to be inclusive of all of the L1 and L2 caches on the chip. AMD Athlon follows exclusive cache designs. Conversely, however, Compaq's CMP design, the Piranha (Tendler et al, 2001), chose a policy of exclusion. It seems that there is no obvious choice between the exclusive and inclusive cache design.

The goal of this project is to evaluate both exclusive and inclusive cache architecture to see which performed best in terms of miss average latency, execution time, cache hit rate for four different benchmark programs. A simulation tool gem5, has been used to simulate their behavior. The main focus of this project is to examine the performance of the cache architectures with different ratio of L1 and L2 caches size. The hypothesis I am considering for this project is with the increase of the L2 cache size, the extra effort to maintain exclusivity may not be worth it given a very minimal or no increase in L2 hit rate.

Background Study

Exclusive and Inclusive Cache Architecture

Exclusive cache and inclusive cache are two choices for designing cache hierarchies. In Inclusive architecture, data in L1 cache can also be found somewhere in the L2 cache. For understanding how this architecture works we can think of a simple example. The example and figures are borrowed from (ARM, 2015). Let we have a simple memory read, for example, *LDR X0, [X1]* in a single core processor. If X1 points to a location in memory, which is marked as cacheable, then there is a cache lookup in the L1 data cache. If the address is found within the L1 cache, then data is read from the L1 cache and returned to the core. **Figure 1** depicts this scenario.

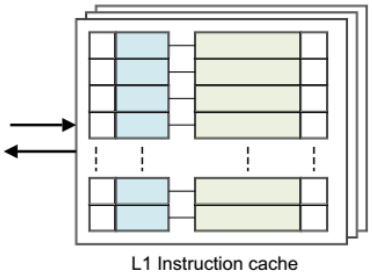

Figure 1: Data Found in L1 cache

But if the address is not found in the L1 cache, but it is in the L2 cache, then the cache line is loaded into the L1 cache from the L2 cache and the data is returned to the core. This can cause a line to be evicted from L1 to make room, but it might still be present in the larger L2 cache. **Figure 2** will help to understand this

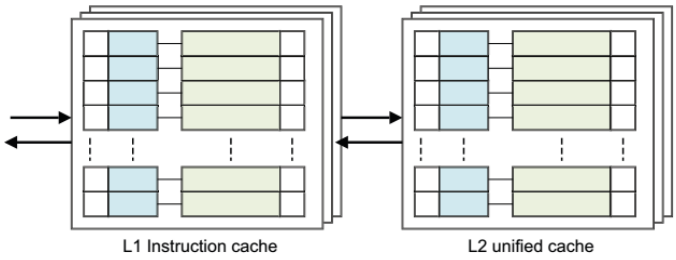

Figure 2: Data Found in L2 cache, absent in L1

If the address is not in either L1 or L2 caches, data is loaded into both the L1 and L2 caches from external memory and supplied to the core. This can cause lines to be evicted. **Figure 3** shows this step.

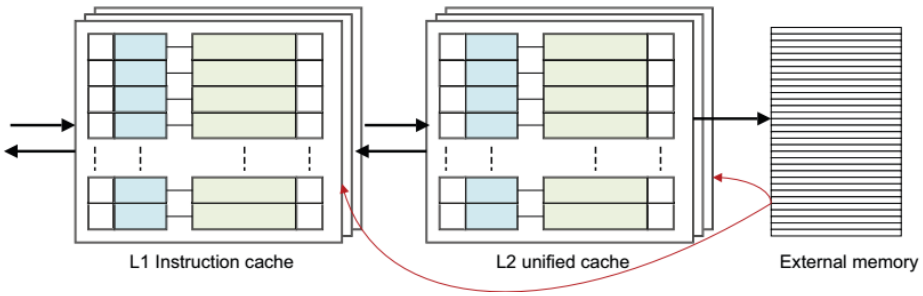

Figure 3: Data Found in external memory, absent in L1, L2

This is how a simple inclusive model works, where the same data can be present in both L1 and L2 caches. But in an exclusive cache, data can be present in only one cache either in L1 or in

L2 but not in the both catches at the same time. So, now question is how this policy is maintained?

Suppose processor requests for a data X, which is found in L1 cache. Then the data is read from L1 and returned it to the processor. If the data can not be found in L1, but present in L2, the data is moved from L2 to L1. Here data is moved, not copied. Hence the possibility of duplicates has been removed. This may cause a line to be evicted from L1, which takes place in L2. Here L2 cache is behaving like a victim cache. If data can not be found in both L1 and L2 cache, then it is fetched from main memory and placed just in L1 but not in L2. This how exclusivity is maintained.

Literature Review

For understanding the performance tradeoff of exclusive and inclusive cache with the increase of cache size I went through couple of research papers. Ying et al (Ying, 2004) did the similar type of work I presented here. They simulated the two-level cache memory and examined the impact of exclusive caching on system performance. They showed that significant performance advantages can be gained for some benchmarks through the use of an exclusive organization. They also tried equal sized victim buffer and victim cache for both architectures and found higher execution time for exclusive caches in some cases. Considering the complexity involved in an exclusive cache hierarchy and the current silicon technology, the recommended exclusive cache hierarchy for server applications that perform a large amount of memory accesses and embedded systems that have limited silicon space for cache and memory. For simulation they built a parameterized exclusive cache hierarchy and integrated it into sim-outorder from the SimpleScalar toolset. They used 10 benchmarks from SPEC2000 (SPEC, 2003)

Aamer Jaleel et al (Aamer, 2010) showed that limited performance of inclusive caches is mostly due to inclusion victims-lines that are evicted from the core caches to satisfy the inclusion property. Their goal was to bridge the performance gap between inclusion and non inclusion by improving the management of an inclusive Last Level Cache. Their main contributions were- 1. they showed a better managed inclusive cache provides equal or better performance than a non-inclusive cache 2. Proposed Temporal Locality Hints mechanism to reduce inclusion victims 3. Proposed Early Core Invalidation technique to derive a line's temporal locality 4. Proposed an alternative approach of Early core invalidation. They used CMP$im (A. Jaleel, 2008), a pin based trace-driven x86 simulator for their experiment. The used SPEC CPU2006 benchmarks.

In another work, Aamer Jaleel et al (Aaemer, 2015) showed that server workloads benefit tremendously from an exclusive hierarchy with large private caches. Their experiment showed an improvements by 5-12% for exclusive cache hierarchy compared to inclusive cache using 16-core Chip Multiprocessors. Another notable contribution of this work is establishing that changing the baseline inclusive hierarchy to an exclusive cache hierarchy improves performance. They used the same simulator and benchmark like the previous mentioned work of them.

I also went through some other papers like (Mohamed Zahran, 2007), where he presented some design alternatives for non-inclusive cache systems and also showed that the main advantage of a non-inclusive cache design arises from its relatively high level hit rate which enhances the the overall average memory system access time. (Natalie, n.d.) presented a hypothesis regarding the ratio of L1 to L2 cache size and the performance of cache architectures.

Simulation Techniques

To evaluate the performance of the inclusive and exclusive cache systems, I used gem5 simulator (Gem5, 2011). It is a modular platform for computer-system architecture research, encompassing system-level architecture as well as processor architecture. The simulator is the result of the combined efforts of many academic and industrial institutions and industrial institutions, including AMD, ARM, HP, MIPS, Princeton, MIT, and the University of Michigan, Texas and Wisconsin.

Configuration Script

I used the basic X86 environment which I set during the building process of gem5. Gem5 has 5 different binaries. I chose “opt” because it is built with most optimizations on, but with debug symbols included and also faster.

The gem5 binary takes, as a parameter, a python script which sets up and executes the simulation. In this script, we have to create a system to stimulate, create all of the components of the systems, and specify all of the parameters for the system components. Although gem5 is shipped with some default script files, to change the behavior of the default clusivity nature of the cache which is “mostly_inclusive”, I needed to write a separate gem5 configuration script.

At first I wrote a simple script. This script modeled a simple system with one simple CPU core. The CPU core is connected to a system-wide memory bus and also integrates a single DDR3 memory channel that is connected to the memory bus. gem5’s modular design is built around the SimObject type. Most of the components in the simulated system are SimObjects: CPUs, caches, memory controllers, busses, etc. I integrated this SimObject to change the behavior of the scripts according to my need. My configuration script can be found here.

Cache Configurations

To integrate a cache hierarchy which I can be modified as my requirement, I need to override the default cache configuration. So that I wrote my own cache configuration script that extend the *Cache SimObject*. The *Cache SimObject* inherits from the *BaseCache* object.

Within the Base cache class, some parameters have a default value. Many of the parameter doesn’t have any default value. I have to set these parameters. Moreover, to override the default configuration of some values like cache size, I had to arrange the code accordingly. The

cache configuration file I have used can be found here. The cache configuration values that I have used in this project are following:

Parameters	Description	Default Value
assoc	Associativity	2
tag_latency	Tag Lookup Latency	2
data_latency	Data Access Latency	2
response_latency	Latency for the return path on a miss	2
mshrs	Number of MHRS (max outstanding requests)	4
tgts_per_mshr	Max number of access per MSHR	20
size	Capacity	64kB for L1, 256kB for L2
clusivity	Clusivity with upstream cache	'mostly_incl'

Table1 : Parameters value in Gem5 configuration file

A variety of cache configurations are used for the simulation. For simplicity, I didn't change the L1 cache size and type. I took 64kB fixed for the both L1 data cache and instruction cache. As, there is no upstream cache over L1, performance doesn't vary much for its exclusiveness or inclusiveness nature. That's why I took the L1 cache type as mostly inclusive. I kept the maximum default configuration as gem5 has for the simplicity.

To show the effect of exclusiveness and inclusiveness of the cache configuration with the change of cache size, I need to vary the size and nature of L2 cache. As, I already mentioned I have two choice for cache nature- mostly exclusive and mostly inclusive. For varying the cache size, I took 5 different sizes of L2 caches. They are 128kB, 256 kB, 512 kB, 1024 kB and 2048 kB. Remember, for maintaining the cache configuration policy, I can't choose 64kB for L2 cache as I fixed the L1 cache size at 64 kB.

Benchmarks

I have used the four benchmarks that was provided in the class for the first project. The reason behind choosing this four benchmarks was they explore a range of demands on the simulated processor. Understanding the benchmark's nature is important in terms of performance for my project. Hence, I am going to mention the benchmarks in details, which has been collected from this source.

matmul: a 2x2 register-blocked matrix multiplication of two 84x84 matrices. The matrices are pseudorandomly generated and all sizes are hard-coded. This program was selected because it should have a mix of cache accesses and floating point operations.

BFS: computes a breadth-first search problem. This program was selected because it should have poor data cache locality.

SHA: computes the SHA-1 cryptographic hash of its input. This program was selected because it should be integer-operation intensive and very friendly for branch prediction and cache.

Queens: solves the N queens problem for an N specified as an argument. This program was selected because it should be very friendly to the cache, but very challenging for branch prediction.

Statistics Parser and Report Visualization

Gem5 generates some output files and one of them is statistics file which contains all of the gem5 statistics registered for the simulation. I wrote a python script which collects my required stats from this file and processes for further analysis. To generate the visual analysis, I used bokeh, an interactive visualization tool for python. All the generated stats files, the script and graphs can be found here.

Simulation Result Analysis

The simulation compare the performance of a mostly inclusive cache with a mostly exclusive cache, where the cache configurations vary in: L2 cache size and L2 cache type(Mostly exclusive or Mostly inclusive). Each simulation gathers a lot of statistics but I should consider here only some fixed stats like dcache miss latency, dcache miss rate, L2 cache miss latency, L2 cache miss rate, Number of seconds simulated etc. **Table 1** in Appendix shows the stats fields I have collected from stats file generated by simulation for this project.

The behavior of the data cache would be more interesting with the the clusivity of the cache than the instruction cache. That's why I have considered only the data cache stats for this project. Overall miss latency is always proportional to average miss latency, that's why I considered only average miss latency time for analyzing the latency. I showed number of cache hit as a metrics. For showing the comparison in execution time, I took number of seconds simulated into account.

Dcache Miss Latency and L2 Data Hit

Figure 4 shows the average miss latency of mostly inclusive L1 Dcache to the average miss latency of mostly exclusive L1 Dcache varying L2 cache size from 128 kB to 2048 kB. For the benchmarks "Queens" and "SHA", increasing of L2 size seems to have no effect on latency but in all the cases exclusive cache performs slightly better. But for the "BFS", inclusive cache

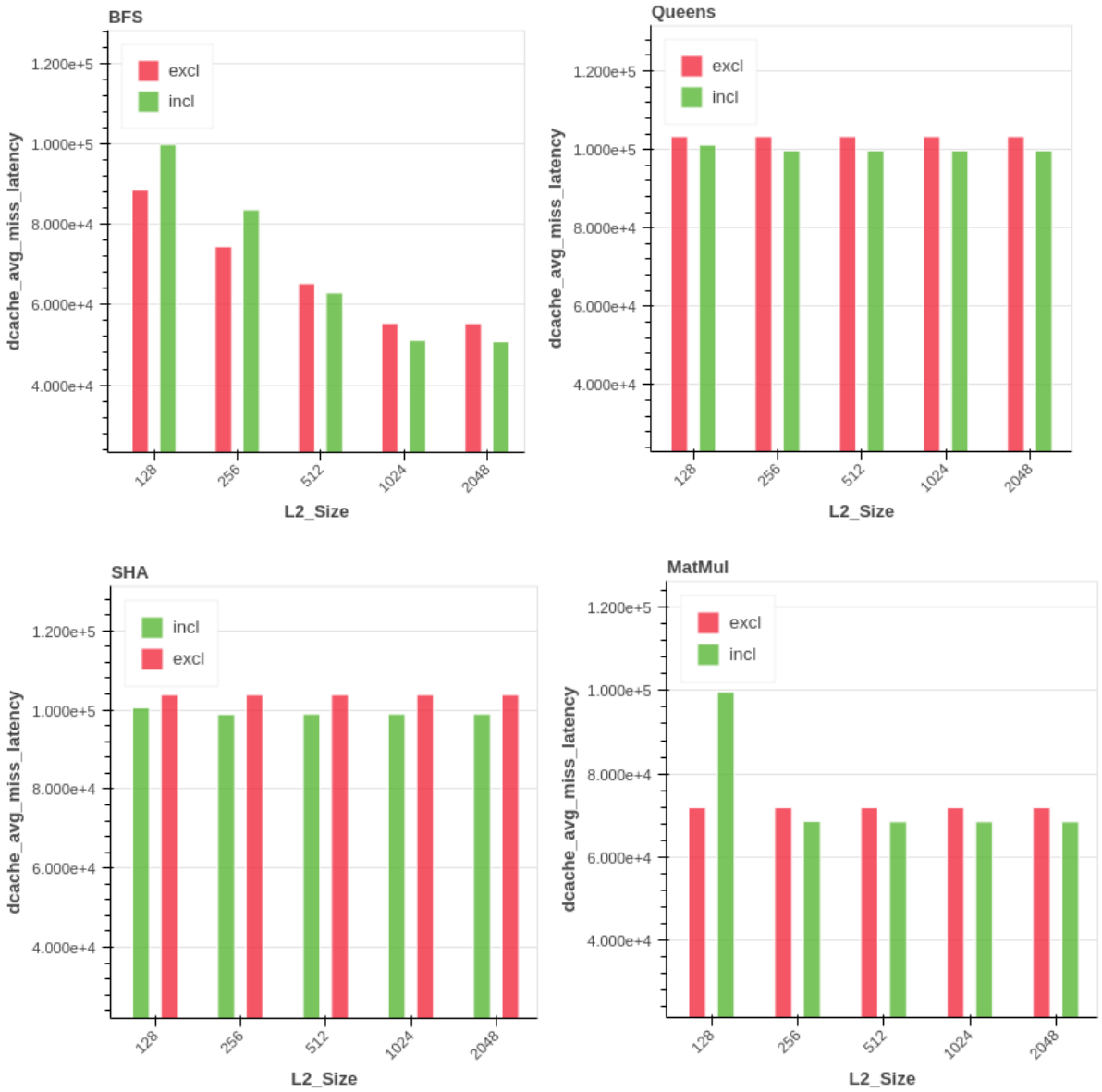

Figure 4: Comparison of average dcache miss latency for different benchmarks

shows better performance with the increase of L2 cache size. “MatMul” shows an improvement for exclusiveness at lower cache size, but becomes saturated with increase of L2 size.

Possible explanation of different behavior of the benchmarks can be - both “SHA” and “Queens” are cache friendly program and need less cache to use. That’s why increasing of cache size has literally no impact on the miss latency. But “BFS” has poor data cache locality which improves with the increasing of L2 cache size. “MatMul” has a mix of cache accesses and floating point operations and low level inclusive cache (128kB) fails to support these operations. But with the increase of cache size inclusive cache improves the performance even better than the exclusive one which is saturated in all the cases.

For better understanding the latency scenario, let's look at the L2 data hit which a great factor on latency of L1 cache as processor looks for data in the L2 cache if it fails to find it in L1. **Figure 5** shows the L2 data hit comparison of the benchmarks.

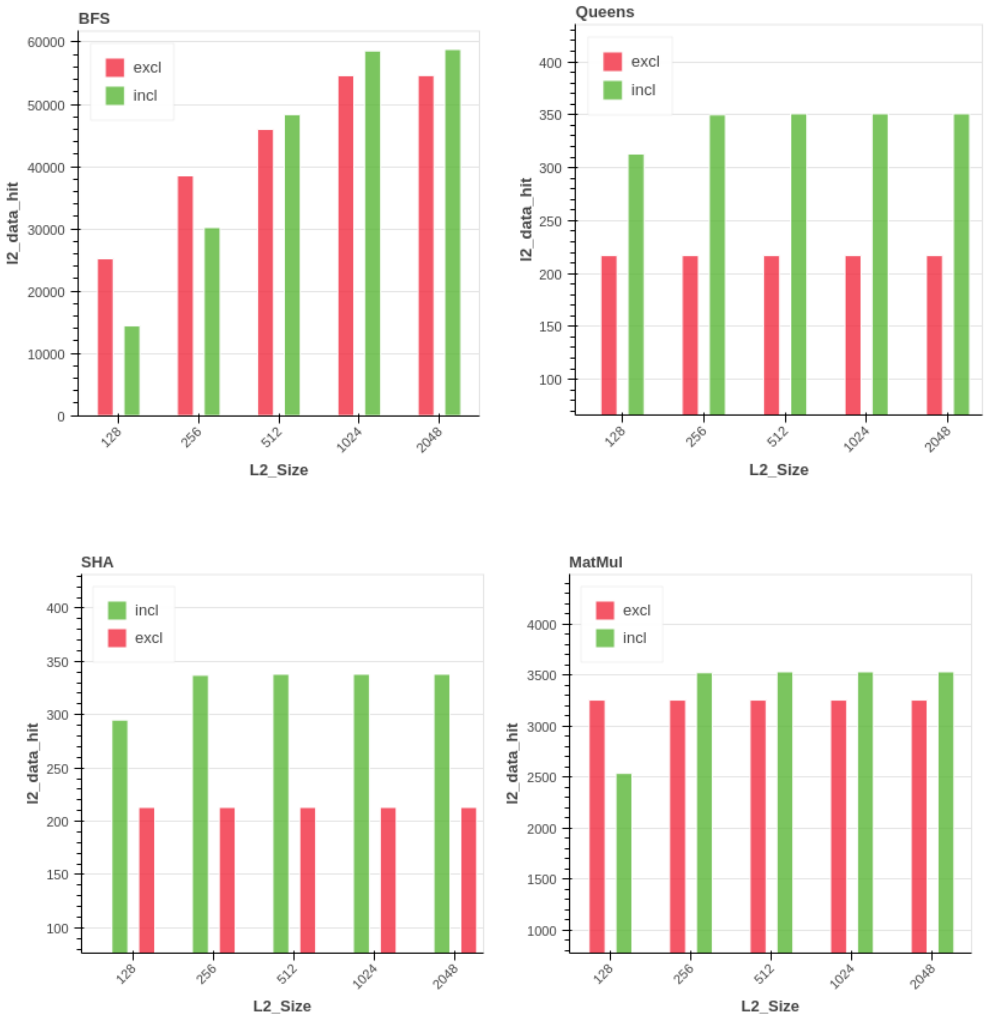

Figure 5: Comparison of Data Hit in L2 cache for different benchmarks

If we look at the number of data hit in L2 cache for BFS, we can see that although the number of hits is bigger for exclusive cache at the first stage, exclusive cache outperforms it with the larger cache size. The number of hits for both exclusive and inclusive cache get saturated at higher stage of memory size. The poor data locality of BFS can explain the lower data hit at lower cache size. The number of hits for “MatMul” also explains the the higher latency for the inclusiveness at low size cache. But hit numbers increase over increasing of the L2 size.

For “BFS”, the possible explanation of increasing hit number for inclusiveness at higher size of the cache is, for maintaining the exclusiveness of the cache, when processor copies a data from L2 to L1, it erases the copy from L2 and keeps the single copy in L1. So, at some stage, the

existing copy in L1 can be victim of cache replacement policy. As there is no copy of the data in both L1 and L2 cache, so whenever processor wants it again, it has to be cache miss in both stage. But in inclusive cache, although the data evicted from L1, it has a copy in L2. So it will be a cache hit in L2 if processor wants it again. As BSF shows poor locality, it shows the better performance with inclusiveness in higher size cache. The scenario of cache hit and cache miss in both exclusive and inclusive cache can be applied for other benchmarks as well.

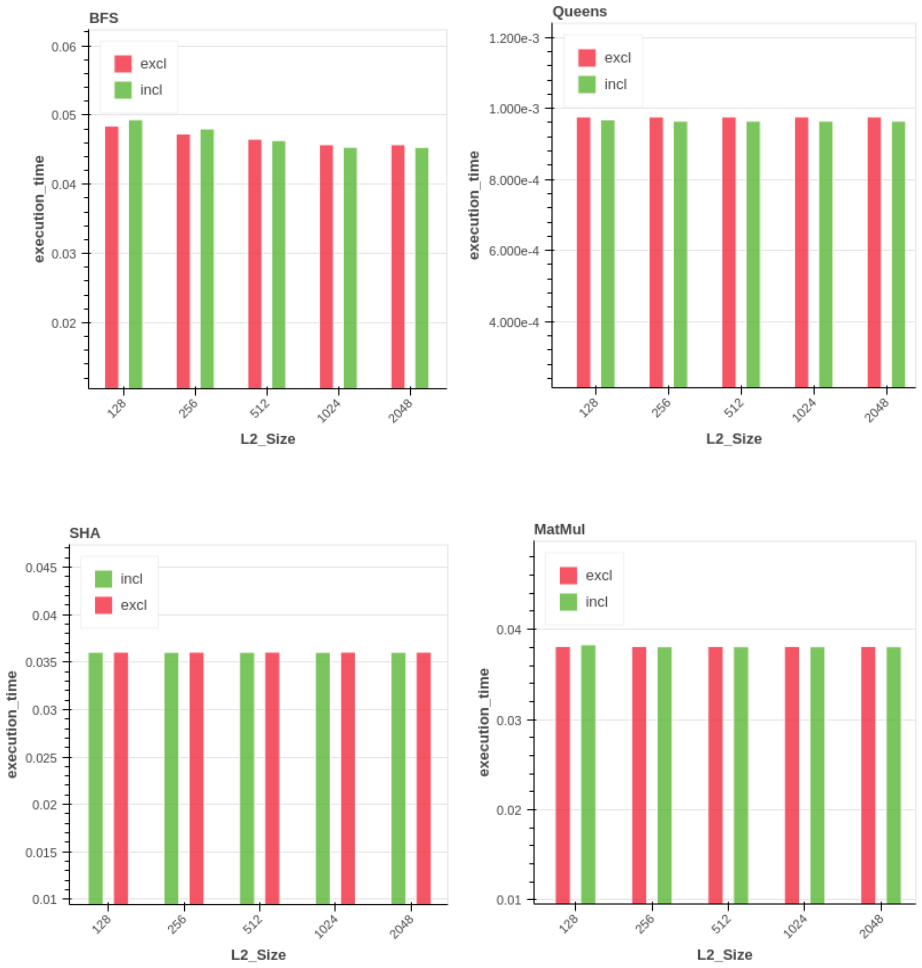

Figure 6: Comparison of execution time

Execution Time

I took number of seconds simulated as the execution time of a program. **Figure 6** shows the comparison of inclusive and exclusive cache in terms of execution time with varying size of L2 cache for the different benchmarks. “SHA” does not show any significant change in execution time for both exclusive and inclusive cache even with the increase of the cache size. For 128 kB cache size, exclusive cache shows a little better performance than inclusive one for “MatMul”. But in later stage, both becomes same and saturated. “Queens” shows better performance for inclusiveness but a little. Moreover, both inclusive and exclusive type do not show much

variance with the increasing cache size for this benchmark. Significant variance in execution time can be found if we look at “BFS”. With the increase of cache size, execution time of inclusive cache becomes more improved even becomes lower than the exclusive cache at one stage. Again poor locality can be the best possible explanation for this benchmark.

The better execution performance for some benchmarks with inclusive cache in some benchmarks is - the exclusive cache has a higher worst-case penalty than the inclusive cache. As L2 size increases, the worst-case latency is incurred more frequently, due to increased L2 hit rate.

Discussion

From the simulation result it is obvious that, if L2 cache size significantly larger than the L1 cache, the performance becomes better. Although exclusive cache offers more space than the inclusive one, we can expect a better performance for the exclusiveness of the cache if the L2 cache is comparatively smaller as the difference of number of hits will not be influential much. Another thing we can observe from above experiment that, if a program shows bad locality, it may struggle in terms of both execution time and cache hit for exclusive cache configuration. So, for handling bad locality code, inclusiveness of the cache is required. I have run the experiment with only two level cache. There are some more parameters that may affect the outcome. For example, if we introduce another level of cache, the performance may change. Cache associativity can be another factor. For going through the full-fledged study of cache configuration in terms of clusivity, we need to consider the other factor as well.

Conclusion

This experiment focuses on comparative study of cache configuration in terms of its exclusiveness and inclusiveness. Which cache configuration is better, that was the main goal of this project. The answer isn’t so cut-and-dry as is often the case in computer architecture, the answer is cop-out “it-depends”. Exploring the simulation result with four benchmarks which covers four different scenario, I found that there is a tradeoff between cache size and its clusivity. We can establish a general rule-of-thumb for choosing the cache configuration type which is - Exclusivity is necessary if the L2 is less than 4 to 8 times the size of the L1; otherwise the duplication of data begins to impact of the L2’s local hit rate. Another interesting fact is a program which lacks good locality may struggle in the ideal exclusive scenario(When L2 is smaller compared to L1 size). But if we want to make the cache inclusive all the time, it may not be wise enough as inclusive waste space and cache is expensive. So, ideal combination of cache configuration in terms of clusivity for different workload can be an interesting field of study.

References

1. Aamer Jaleel, Eric Borch, Malini Bhandaru, Simon C. Steely Jr., and Joel Emer: Achieving Non-Inclusive Cache Performance with Inclusive Caches

2. A. Jaleel, R. Cohn, C. K. Luk, B. Jacob. CMP$im: A Pin-Based On-TheFly Multi-Core Cache Simulator. In MoBS, 2008
3. Aamer Jaleel, Joseph Nuzman, Adrian Moga, Simon C. Steely Jr., Joel Emer: High Performing Cache Hierarchies for Server Workloads, March 2015
4. ARM Cortex-A Series, version: 1.0 : Programmer's Guide for ARMv8-A, 2015 from: http://infocenter.arm.com/help/topic/com.arm.doc.den0024a/DEN0024A_v8_architecture_PG.pdf
5. Gem5: The gem5 Simulator, May 2011
6. J. M. Tendler, J. S. Dodson, J. J. S. Fields, H. Le, and B. Sinharoy. POWER4 system microarchitecture. IBM Journal of Research and Development, 26(1):5--26, January 2001
7. L. Barroso, K. Gharachorloo, R. McNamara, A. Nowatzyk, S. Qadeer, B. Sano, S. Smith, R. Stets, and B. Verghese. Piranha: A Scalable Architecture Based on Single-Chip Multiprocessing. In Proceedings of the 27th Annual International Symposium on Computer Architecture (ISCA'00), pages 282--293, June 2000.
8. Mohamed Zahran: Non-Inclusion Property in Multi-level Caches Revisited, June 2007
9. Natalie Enright, Dana Vantrease: To Include or Not To Include:The CMP Cache Coherency Question, n.d.
10. SPEC, "SPEC CPU2000 Benchmarks," Standard Performance, Evaluation Corporation, 2000. http://www.spec.org/osg/cpu2000/ (Current June 2003).
11. Ying Zheng, Brian T. Davis, Matthew Jordan: Performance Evaluation of Exclusive Cache Hierarchies, March 2004

Appendix

Stats Field	Description
sim_seconds	Number of seconds simulated
system.cpu.dcache.overall_hits::total	number of overall hits in L1 dcache
system.cpu.dcache.overall_misses::total	number of overall misses in L1 dcache

system.cpu.dcache.overall_miss_latency::total	number of overall miss cycles in L1 dcache
system.cpu.dcache.overall_accesses::total	number of overall (read+write) accesses in L1 dcache
system.cpu.dcache.overall_miss_rate::total	miss rate for overall accesses in L1 dcache
system.cpu.dcache.overall_avg_miss_latency::total	average overall miss latency for L1 dcache
system.cpu.icache.overall_hits::total	number of overall hits in L1 icache
system.cpu.icache.overall_misses::total	number of overall misses in L1 icache
system.cpu.icache.overall_miss_latency::total	number of overall miss cycles in L1 icache
system.cpu.icache.overall_accesses::total	number of overall (read+write) accesses in L1 icache
system.cpu.icache.overall_miss_rate::total	miss rate for overall accesses in L1 icache
system.cpu.icache.overall_avg_miss_latency::total	average overall miss latency for L1 icache
system.l2cache.overall_hits::cpu.inst	number of overall hits in L2 cache for instruction
system.l2cache.overall_hits::cpu.data	number of overall hits in L2 cache for data
system.l2cache.overall_hits::total	number of overall hits in L2 cache
system.l2cache.overall_misses::cpu.inst	number of overall misses in L2 cache for instruction
system.l2cache.overall_misses::cpu.data	number of overall misses in L2 cache for data
system.l2cache.overall_misses::total	number of overall misses in L2 cache
system.l2cache.overall_miss_latency::cpu.inst	number of overall miss cycles in L2 cache for instruction
system.l2cache.overall_miss_latency::cpu.data	number of overall miss cycles in L2 cache for data
system.l2cache.overall_miss_latency::total	number of overall miss cycles in L2 cache
system.l2cache.overall_accesses::cpu.inst	number of overall (read+write) accesses in L2 cache for instruction

system.l2cache.overall_accesses::cpu.data	number of overall (read+write) accesses in L2 cache for data
system.l2cache.overall_accesses::total	number of overall (read+write) accesses in L2 cache
system.l2cache.overall_miss_rate::cpu.inst	miss rate for overall accesses in L2 cache for instruction
system.l2cache.overall_miss_rate::cpu.data	miss rate for overall accesses in L2 cache for data
system.l2cache.overall_miss_rate::total	miss rate for overall accesses in L2 cache
system.l2cache.overall_avg_miss_latency::cpu.inst	average overall miss latency for instruction in L2 cache
system.l2cache.overall_avg_miss_latency::cpu.data	average overall miss latency for data in L2 cache
system.l2cache.overall_avg_miss_latency::total	average overall miss latency in L2 cache

Table 1: Considered Stats Fields from Gem5 output file stats.txt