RT-ADAS Study Report

Devon Lewis, Melody Chan

Supervised by Renato Mancuso

Abstract—The goal of this directed study was to evaluate the effectiveness of a novel proposed technique for reducing cache contention. In this paper we provide an overview of the technique, as well as our experimental analysis. We find the experiments do not validate our hypothesis, and consider future work to understand why.

1 Introduction

Cache contention plagues real time system designers, creating another source of unpredictability for them to manage. Contention mitigation strategies have been explored intensively, and in this paper we offer another approach. We modified open source benchmark suites to test our hypothesis. While one experiment did seem to look promising, the second did not validate our hypothesis.

2 Jetson Xavier NX

We performed all experiments on the NVIDIA Jetson Xavier NX. The NVIDIA Jetson Xavier NX is an embedded system-on-module (SOM) designed for AI, robotics, and edge computing applications.

2.1 Memory Architecture Topology

Memory architectures for most systems can be broadly classified into a hierarchical structure consisting of several levels, with the top level being the fastest and smallest, and the bottom level being the slowest and largest. For the purpose of this report, we will only focus on the cache level hierarchy, which is divided into levels (usually L1, L2, and L3), with L1 being the fastest and smallest and L3 being the slowest and largest.

For Jetson Xavier NX, the CPU consists of 6 cores, with each core possessing its individual L1 cache. The L1 instruction cache and data cache both have a size of 32KB. Every pair of adjacent cores share an L2 cache of size 2MB, while a 4MB-L3 cache is distributed among all cores.

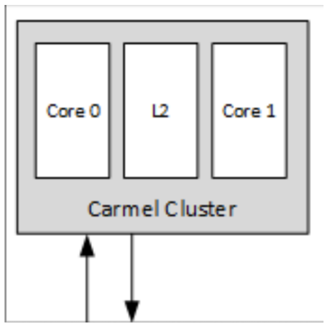

Fig. 1: Xavier series SoC L2 cache architecture

3 Intercore Interference

Intercore interference occurs when multiple cores in a multicore processor access shared memory resources simultaneously. Our research concentrates on investigating interference in shared memory caches and explores the effects of contention on overall system performance. The interference often results in reduced performance due to increased access latency and diminished system efficiency. However, our objective is to discover an approach that could potentially achieve the opposite effect.

4 Constructive Interference

The concept of constructive interference proposes the hypothesis that if we were to constantly access some pages required by a main application in a helper thread, which in this report will be called a santa thread, on one of the cores, we can keep the pages in cache and therefore speed up the runtime of the main application by reducing its memory access time. In this way, the santa is interfering with the main application, but in a helpful way.

The end goal is to target only those pages that are most important to the application, by reading profiles generated by the Black Box Profiler, but the implementation must be improved first.

5 Experiments

We chose to conduct our experiments using pre-existing benchmarks in Isolbench [1] and the Black-Box Profiler [2]. The last two cores out of the 6 cores are deactivated by default, subsequently we are examining interference occurring among the remaining four cores. In both application experiments, we initiated an additional thread within the main application code, designed to continuously access the same memory pages as the main application. Specifically, the main application always runs on core 0, while the santa thread operates on core 1. We selected these two cores because, according to the memory architecture of the Xavier NX, they share the same L2 cache. Consequently, running the santa thread on core 1 has the greatest potential to impact the main application.

5.1 Implementation

The implementation is quite simple for both bandwidth and the staircase access. For bandwidth, we create a thread that repeatedly reads the buffer written to in main.

For the staircase, we do the same, but we modified the thread's behavior to only read some portion of the buffer read by the main. In the main access loop, the reads are done repeatedly on one-fifth of the buffer. Then after some amount of iterations, the next fifth is added to the total amount of buffer read. Our implementation allows the thread to repeatedly read some section of the buffer. For example, while the benchmark main access loop runs, the santa thread can be configured to only read the first-fifth of the buffer. The rationale was to test the following part of our hypothesis. If the santa thread keeps the most used pages in cache by accessing those, it would improve performance more than if it the thread accessed arbitrary pages. The nature of the benchmark is such that the first fifth of the buffer will correspond to the most used pages, since they are read the most. Therefore, setting the santa thread to access that section should increase performance. We timed each run in clock cycles using per-core clock counters.

5.2 Evaluating Constructive Interference

To evaluate our technique, we used Isolbench to generate contention on different cores. This approach allowed us to define the contention size, select the core on which to run the contention, specify the memory access method (either read or write), and set the duration of contention. We will somtimes refer contention as memory bombs in this report.

Below is an example command to run contention of size 4096KB on core 2 with WRITE commands infinitely:

```
$ ./bandwidth -m 4096 -a write -c 2 -t 0
```

5.3 Test Cases

We conducted our experiment using four test cases, including one base case. The test cases are as follows:

1) **Base Case:** In the base case, main application runs on core 0 while the other three cores are left unstressed. It is labeled as No Contention on the graphs in this report.
2) **Test Case 2:** In this test case, the main application runs on core 0 while the all other cores(core 1, 2, 3) run contention. It is labeled as Contention (3 cores) on the graphs in this report.
3) **Test Case 3:** In this test case, the main application runs on core 0 while only two other cores(core 2, 3) run contention. Core 1 remains unstressed. It is labeled as Contention (2 cores) on the graphs in this report.
4) **Test Case 4:** In this test case, the main application runs on core 0 while only two other cores(core 2, 3) run contention. Core 1 runs the santa thread. It is labeled as Santa on the graphs in this report.

For the experiment, we chose to contend the cores with memory bombs of size 16384KB, which is large enough exceed the cache sizes and to guarantee the occurrence of interference between the cores. In Figure 2, some critical values are labeled on the graph with dotted lines in order to show what happens to the latency if the main application size exceeds certain memory cache size, including L1, L2, and L3 cache labeled from left to right on the x-axis. The main application was repeated with sizes that range from 2^0KB to 2^{20}KB, in all four test cases mentioned in the previous section.

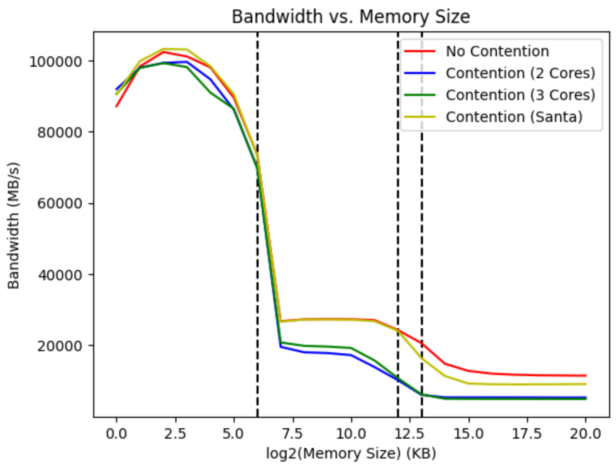

Fig. 2: rt-benchmarks output graph

From the figure, we can see that constructive interference increases bandwidth relative to 2-core and 3-core contention. This is promising, but we were more interested in the performance of the staircase benchmark as we wanted to target the most important pages rather than all of them.

5.4 Black-Box Profiling

The Black-Box Profiler locates and sorts the memory pages accessed by a specific application based on how important the pages are. A page is important if the application speedup when the page is cached is large. The profiler code base includes the staircase benchmark.

We ran many versions of this experiment. For each version, we used a script to take take the average over 10 runs for various buffer sizes corresponding to memory pages that increased fivefold for each run. First, we had bandwidth running with 16384KB of memory. The santa thread was configured to read the first fifth of the buffer. These results showed the santa thread performing worse than 2-core contention. We thought it may have had to do with Linux power management, but after fixing the clock frequency, the results remained the same.

We then retried the experiments with a lower memory size for bandwidth, but the results remained the same. We then modified the thread to read the entire buffer rather than the first fifth, but 2-core contention was still faster without the santa. We then modified the benchmark to read the entire buffer rather than segments, along with the previous modifications. The result of that experiment is in figure 3

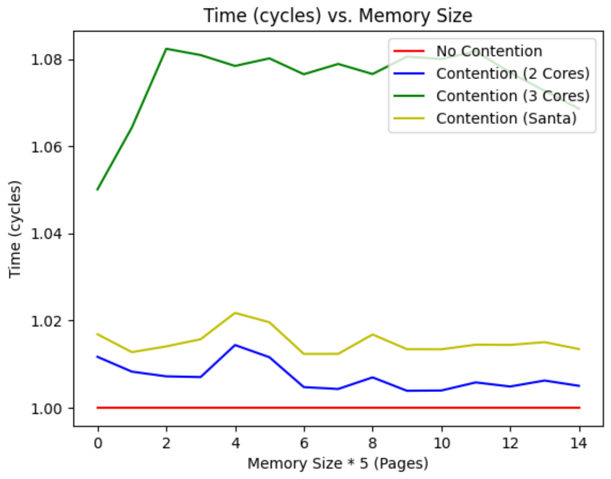

Fig. 3: blackbox output graph

Here the plots are normalized to the baseline, and we see constructive interference taking slightly more cycles than 2-core interference.

6 Conclusion

This paper explained and evaluated a novel technique for cache contention mitigation. Although our hypothesis regarding important pages seems to be invalided, our first tests with bandwidth seem promising. Future work includes digger deeper for the reason why our hypothesis is incorrect. We plan to use ARM's L2 and L3 cache performance counters to how the cache hits are affected by the santa thread, to hopefully gain insight to improve our implementation.

References

[1] P. K. Valsan, H. Yun, and F. Farshchi, "Taming Non-blocking Caches to Improve Isolation in Multicore Real-Time Systems," *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2016. DOI: 10.1109/rtas.2016.7461361.

[2] G. Ghaemi, D. Tarapore, and R. Mancuso, "Governing with Insights: Towards Profile-Driven Cache Management of Black-Box Applications," in *33rd Euromicro Conference on Real-Time Systems (ECRTS 2021)*, B. B. Brandenburg, Ed., ser. Leibniz International Proceedings in Informatics (LIPIcs), vol. 196, Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021, 4:1–4:25, ISBN: 978-3-95977-192-4. DOI: 10.4230/LIPIcs.ECRTS.2021.4. [Online]. Available: https://drops.dagstuhl.de/opus/volltexte/2021/13935.