Exploring iGPU Memory Interference Response to L2 Cache Locking

Alfonso Mascareñas González, Jean-Baptiste Chaudron, Régine Leconte, Youcef Bouchebaba, David Doose, Alfonso Mascareñas González

Exploring iGPU Memory Interference Response to L2 Cache Locking

Alfonso Mascareñas González
ISAE-SUPAERO, Université de Toulouse, France

Jean-Baptiste Chaudron
ISAE-SUPAERO, Université de Toulouse, France

Régine Leconte
ISAE-SUPAERO, Université de Toulouse, France

Youcef Bouchebaba
ONERA, Université de Toulouse, France

David Doose
ONERA, Université de Toulouse, France

Abstract

The demand of parallel execution in real-time embedded applications has motivated the integration of GPUs as processing accelerators on SoCs (System-on-Chip) embedded architectures, often leading to CPU-iGPU architectures. In the safety-critical domain, it is paramount to ensure that the execution deadlines of critical tasks are not exceeded. To ease the analysis of this kind of tasks, we can make their worst-case execution time more predictable. One way to achieve this is by mitigating or controlling the memory interference generated by the concurrent execution of tasks through the application of a series of techniques (e.g., cache partitioning, bank partitioning, cache locking, bandwidth regulation). Originally, these were applied to CPUs, and more recently, to GPUs as well. In this work, we focus on the hardware-based L2 cache locking on iGPUs as memory interference mitigation mechanism. We are interested in evaluating its capacity for reducing the worst-case and the average-case execution time in different scenarios. Our measurement-based analysis has been carried out on the NVIDIA's Jetson AGX Orin 64 GB MPSoC, making use of four representative benchmarks (data resetting, 2D convolution, 3D convolution and matrix upsampling).

2012 ACM Subject Classification Computer systems organization → Embedded systems; Computer systems organization → Real-time systems

Keywords and phrases iGPU, cache locking, real-time, memory interference

Digital Object Identifier 10.4230/OASIcs.WCET.2023.3

Supplementary Material *Software (Source Code)*: `https://github.com/ISAE-PRISE/gcalasy` archived at `swh:1:dir:2a28897e208903f22dfa9e8fc6ba0eba569ff2ce`

Funding This work was supported by the Defense Innovation Agency (AID) of the French Ministry of Defense (research project CONCORDE N° 2019 65 0090004707501).

1 Introduction

Since the last decade, we have been observing an increase in popularity of MPSoCs where CPUs work together with integrated Graphic Processing Units (iGPUs). This is the solution that manufacturers have devised to address the need for higher execution parallelism in real-time embedded systems for autonomous machines. These MPSoCs might be used for safety-critical real-time applications, meaning that the execution deadline of the tasks must be satisfied to avoid fatal outcomes. Therefore, it is of extreme importance to perform Worst-Case Execution Time (WCET) analysis of all tasks making up the overall application. The memory interference among the tasks concurrently executing is one of the main reasons

for missing deadlines. Traditionally, this type of interference is avoided or mitigated through spatial-temporal isolation (e.g., bandwidth regulation) and memory partitioning techniques (e.g., cache partitioning, cache locking, bank partitioning) [5].

In this work, we focus on the iGPU, more specifically, on its on-chip shared L2 cache memory. This cache is shared across all the multiple Streaming Multiprocessors (SMs) composing the GPU, and hence, being susceptible to data evictions. To avoid undesired data removal, specially of the data we consider critical or persistent, we apply the cache locking technique. Our objective is to assess the capacity of cache locking for: (1) mitigating the inter-SM interference within one GPU application, (2) mitigating the inter-SM interference produced by a variable number of non-critical GPU applications on one critical GPU application and (3) mitigating the Low-Power Double Data Rate 5 (LPDDR5) memory interference on one GPU application. To do so, we make use of realistic benchmarks (data resetting, 2D matrix convolution, 3D matrix convolution and matrix upsampling), from which we study their WCET and Average Case Execution Time (ACET). The measurement-based analysis of the L2 cache locking technique is carried out on the Jetson AGX Orin 64 GB [11], a heterogeneous MPSoC by NVIDIA which allows performing cache locking using hardware-specific features.

2 Related Work

Memory interference has been a constant problem for the real-time community since the introduction of the first multicore platforms. Since then, researchers have proposed a series of techniques to deal with them [5]. Initially, these techniques were aimed to CPUs but adapted variations have been also implemented on GPUs. To begin with, let us consider the bandwidth regulators, which control the amount of requests that other cores can issue. Work [1] presents a software mechanism that implements bandwidth regulators and GPU protection for CPU-GPU architectures (see [16] for CPU approach). We can continue with the cache and bank partitioning techniques which consist in dividing these two shared resources so that each division is private to a processing core. The former technique offers protection against forced cache evictions by concurrent tasks while the latter partially or totally removes the costly intra-bank interference on the main memory. These two techniques are applied on GPUs in work [4] (see [15] for CPU), which makes use of reverse-engineering and page coloring for performing these partitions. Furthermore, work [4] makes use of SM partitioning, i.e., compute kernels belonging to the same application execute on dedicated SMs. Task and memory mapping can also be used for reducing the memory interference. For instance, on a CPU and DSP context, work [7] performs task-core and core-bank mapping for minimizing interference while considering other optimization objectives. Finally, we have the cache locking technique which consists in keeping loaded data in the cache, preventing it from being evicted. Caches can be fully locked (i.e., the whole cache is used for locked data) or partially locked (i.e., a part of the cache is for locked data and another for cacheable data). The locking can be made statically (i.e., locking is done at boot time and remains unchanged) or dynamically (i.e., locking changes during run-time as function of the needs). Generally, static locking offers more predictability while dynamic locking, which can also be predictable, is more performing [6]. The cache locking technique has been commonly applied for increasing the predictability, performance and energy efficiency of CPUs (see [8]). On GPUs, work [13] uses a simulator to evaluate the impact of cache locking. A comparison between cache partitioning and cache locking for CPU-GPU architectures is made by work [14], also via simulations. In contrast to these previous works, ours aims to study this technique on a real MPSoC incorporating an iGPU with L2 cache locking capabilities.

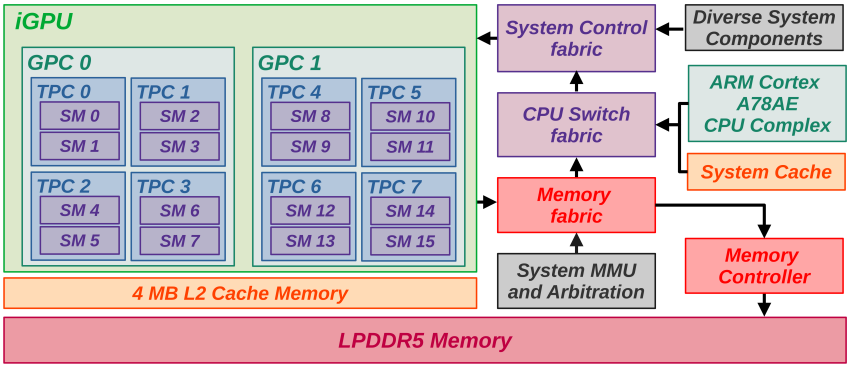


Figure 1 Ampere architecture GPU in Jetson AGX Orin 64 GB.

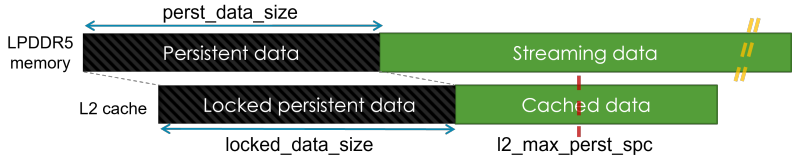
3 Target Platform

3.1 Introduction

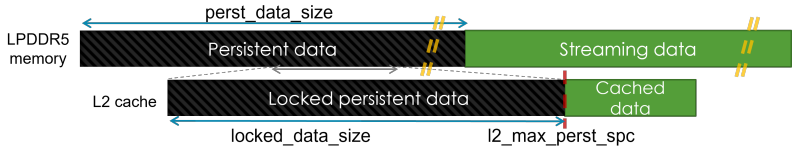
The heterogeneous multicore platform used in this work is the Jetson AGX Orin 64 GB by NVIDIA [11]. It is made up of 12 Cortex-A78AE grouped in clusters of 4 and an Ampere architecture iGPU. The iGPU is composed of 2048 cores distributed among 16 SMs. Each SM has a dedicated L1 cache memory of 192 KB and, all of them, share an L2 cache memory of 4 MB. Internally, the SMs are subdivided in four processing blocks, each of them with their own registers (64 KB in total), an instruction cache and 32 cores (32 threads/clk). As seen in Figure 1, a pair of SMs makes up a Texture Processing Cluster (TPC), and 4 TPCs compose a Graphic Processing Cluster (GPC). There are 2 GPCs in the Orin MPSoC. Computing kernels are executed as grids of blocks of threads, each block being composed of threads. On Jetson AGX Orin, a thread block supports up to 1024 threads. As long as there are enough resources in a SM, multiple blocks of threads can be run by the same SM. We point out two important constraints, the number of threads and blocks supported by a SM. On this platform, the former is limited to 1536 and the latter to 16 [12].

3.2 L2 cache locking hardware mechanism

In NVIDIA GPUs with a compute capability of 8.0 or above, L2 cache locking can be done through hardware [9, 10]. This is achieved by: (1) tagging a contiguous region of global memory as persistent (NVIDIA term for lockable) and (2) setting the amount of space from the L2 cache capacity to use for this persistent memory. If the amount of data tagged as persistent is lower than the reserved persistent data space in the L2 cache (i.e., $perst\_data\_size < l2\_perst\_spc$), then normal data (termed "streaming data" by NVIDIA) can occupy this space. We must avoid having more persistent data than space reserved for it in the L2 cache (i.e., $perst\_data\_size > l2\_perst\_spc$), as L2 cache lines thrashing will take place [9], and therefore, no effective cache locking. To avoid this issue, there are two possibilities. The first one consists in tuning the L2 cache *hitRatio* parameter, which is used to determine the percentage of persistent data to be locked in the cache. The second option is to limit the amount of persistent data to the L2 cache reserved space (i.e., $perst\_data\_size \leq l2\_perst\_spc$) even if this means that some persistent data is left unlocked. We make use of the latter option as the first one implies an non deterministic factor (i.e., the portion of memory to be locked in is unknown), and therefore, is not suitable for predictability analysis. Another constraint to respect has to do with the maximum persistent



(a) $perst\_data\_size \leq l2\_perst\_spc$.



(b) $perst\_data\_size > l2\_perst\_spc$.

Figure 2 Mapping persistent data to L2 cache.

L2 cache data space that we are allowed to set (i.e., $l2\_perst\_spc \leq l2\_max\_perst\_spc$). $l2\_max\_perst\_spc$ is 3 MB for the Orin MPSoC, meaning that there is at least a minimum of 1 MB of cacheable space to be used for any kind of data. Our L2 cache locking space is computed according to Equation 1, where $perst\_data\_size$ is the size of persistent data to lock and $l2\_perst\_spc$ the amount of L2 cache space reserved for locking.

$$locked\_data\_size = min(perst\_data\_size, min(l2\_perst\_spc, l2\_max\_perst\_spc)) \quad (1)$$

Figure 2 shows two examples which depict how persistent and non-persistent (streaming) data is placed on the L2 cache. In both cases (Figures 2a and 2b), we assume that the reserved L2 cache space for locking ($l2\_perst\_spc$) is set to the maximum space allowed ($l2\_perst\_spc = l2\_max\_perst\_spc = 3\,MB$). In Figure 2a, the streaming data in the LPDDR is bigger than the one shown in the L2 cache. In this example, not all the reserved lockable space is used as the persistent data ($perst\_data\_size$) fits inside the cache with less space ($perst\_data\_size < l2\_perst\_spc$). Therefore, the locked data size is $perst\_data\_size$. In contrast, Figure 2b supposes the case where the persistent data exceeds the reserved lockable space ($perst\_data\_size > l2\_perst\_spc$). Note that the persistent and streaming data in the LPDDR is bigger than the one shown in the L2 cache. In this case, the locked data size is $l2\_perst\_spc$.

4 Evaluation

The L2 cache locking is evaluated in three scenarios described in Sections 4.1, 4.2 and 4.3. The customized benchmarks used for the evaluation are based on the data resetting (Equation 2), 2D convolution (Equation 3), 3D convolution (Equation 4) and upsampling operations (Equation 5):

$$B[j] = A[j \,\%\, m] \quad (2)$$

where $A = \forall(a_i) \in \mathbb{R}^m$ and $B = \forall(b_j) \in \mathbb{R}^p$. Vector A is the data to protect with the cache locking mechanism.

$$C[i,j]^{(k)} = \Sigma_{u=0}^{p}\Sigma_{v=0}^{q} A\left[i + u - \lfloor\frac{p}{2}\rfloor, \lfloor j + v - \lfloor\frac{p}{2}\rfloor\right] \cdot B[u,v]^{(k)} \quad (3)$$

with $A = \forall(a_{ij}) \in \mathbb{R}^{m \times n}$, $B^{(k)} = \forall(b_{ij}) \in \mathbb{R}^{p \times q}$ and $C^{(k)} = \forall(c_{ij}) \in \mathbb{R}^{m \times n}$. The superindex k is the matrices identification for each convolution compute kernel launch ($k \in \{0, 1, ..., K-1\}$). We consider matrix A as persistent data.

$$C[i,j,k]^{(k)} = \Sigma_{u=0}^{p} \Sigma_{v=0}^{q} \Sigma_{w=0}^{l} A \left[i + u - \lfloor \frac{p}{2} \rfloor, \lfloor j + v - \lfloor \frac{p}{2} \rfloor, w \right] \cdot B[u,v,w]^{(k)} \tag{4}$$

with $A = \forall(a_{ijk}) \in \mathbb{R}^{m \times n \times l}$, $B = \forall(b_{ijk}) \in \mathbb{R}^{p \times q \times l}$ and $C = \forall(c_{ijk}) \in \mathbb{R}^{m \times n \times l}$. The superindex k is the matrices identification for each convolution compute kernel launch ($k \in \{0, 1, ..., K-1\}$). We consider tensor A as persistent data.

$$B = (A)_{\uparrow L} \tag{5}$$

where $A = \forall(a_{ij}) \in \mathbb{R}^{m \times n}$ and $B = \forall(b_{ij}) \in \mathbb{R}^{p \times q}$, L is the upsampling value and A is the data to lock in the L2 cache.

The execution time of these benchmarks is computed through CUDA events recording, process that consists is retrieving a timestamp of the start and stop events before and after the execution of the benchmark respectively. For each observed WCET and ACET result provided in Sections 4.1, 4.2 and 4.3, 10^5 measurements were considered. Initial measurements affected by the load of the CUDA driver are considered statistical outliers, and hence, discarded.

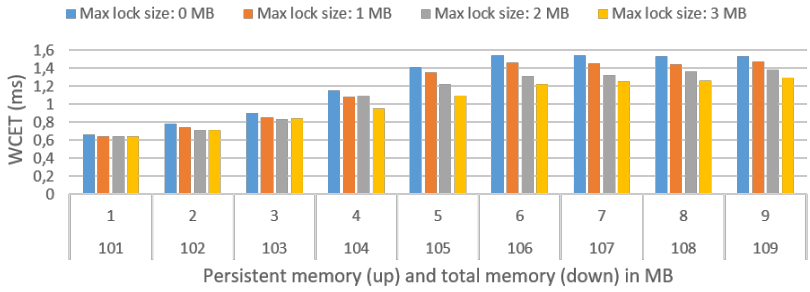
4.1 Scenario 1: SM interference of a single compute kernel

The objective of this scenario is to test the capabilities of the L2 cache locking for mitigating the inter-SM interference when executing a single application, i.e., how the application interferes with itself by making use of all the available SMs. In this scenario, the graphs with the results show the response of a benchmark using the measured WCET and ACET metrics when fixing the lockable space (*l2_perst_spc*) to 0 MB (no locking), 1 MB, 2 MB and 3 MB (*l2_max_pesrt_spc*) represented using blue, orange, grey and amber colors respectively. The X axis indicates the amount of persistent and total data (persistent plus streaming) resulting from varying the size of the data set. The Y axis represents the metric value under analysis in milliseconds. The benchmarks used for testing are the ones described in Equations 2, 3, and 5. The results are illustrated in Figures 3, 4 and 5 respectively.

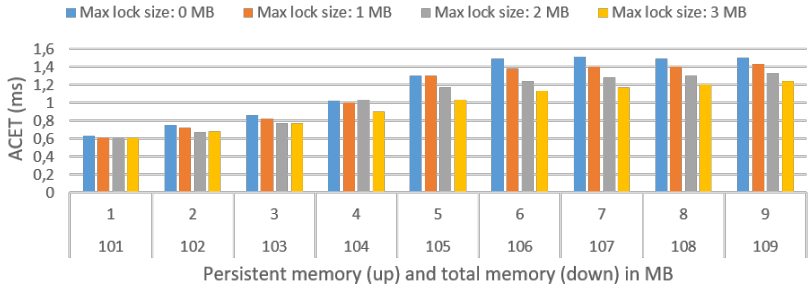
The data resetting benchmark yields great results. On average, the WCET is reduced by 14.64% with a maximum reduction of 23% (Figure 3a) and the ACET is reduced by 15.45% with a maximum reduction of 24.54% (Figure 3b) for a 3 MB cache lock. For a single application inter-SM interference, the previous gains can be considered high. This is due to the high reuse of specific data (persistent data). By locking this data, the number of loads to the LPDDR are reduced, mainly performing stores.

More typical operations are offered by the 2D convolution and upsampling benchmarks. On average, the 2D convolution has its WCET reduced a 5.55% with a maximum reduction of 10.24% (Figure 4a) and the ACET is reduced a 6.7% with a maximum reduction of 24.54% (Figure 4b) for a 3 MB cache lock. The upsampling sees an average WCET reduction of 15.67% with a maximum reduction of 29.15% (Figure 5a) and an average ACET reduction of 13.66% with a maximum reduction of 28.35% (Figure 5b) for a 3 MB cache lock.

Overall, we can see that the L2 cache locking manages to reduce the WCET and ACET. In this sense, the 2D convolution is the one benefiting the least and the data reset and upsampling the most. For the three benchmarks, note that significant execution time drops are achieved even when the persistent data of the benchmark exceeds the L2 lockable space.

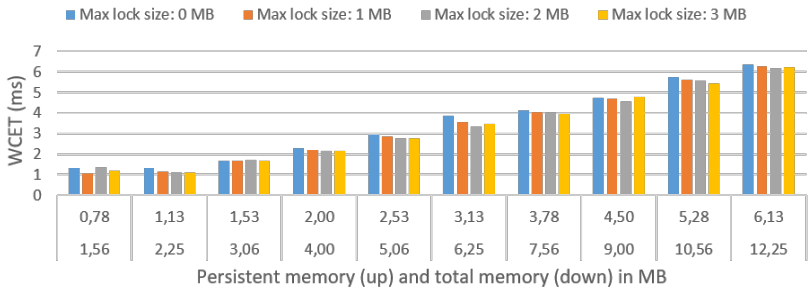


(a) Measured worst-case execution time.

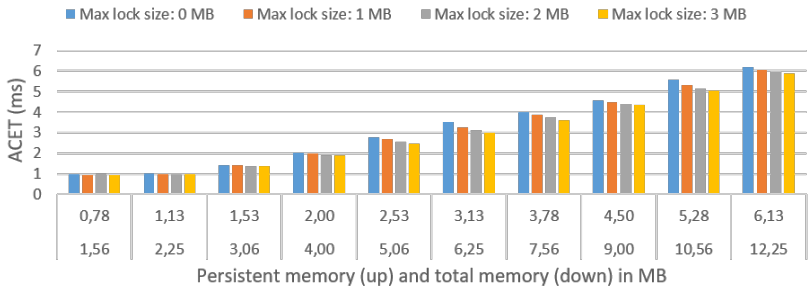


(b) Average-case execution time.

Figure 3 Data reset benchmark behavior.

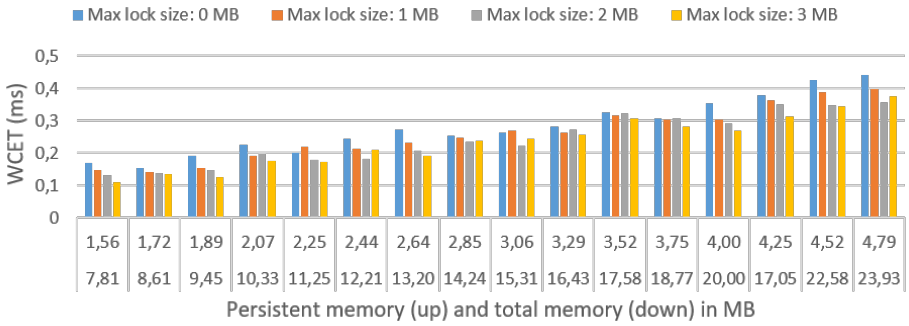


(a) Measured worst-case execution time.

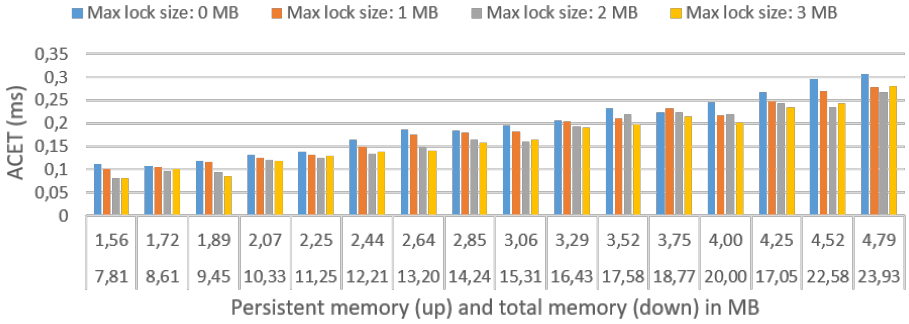


(b) Average-case execution time.

Figure 4 2D convolution benchmark behavior.



(a) Measured worst-case execution time.



(b) Average-case execution time.

Figure 5 Matrix upsampling benchmark behavior.

4.2 Scenario 2: SM interference from non-critical compute kernels

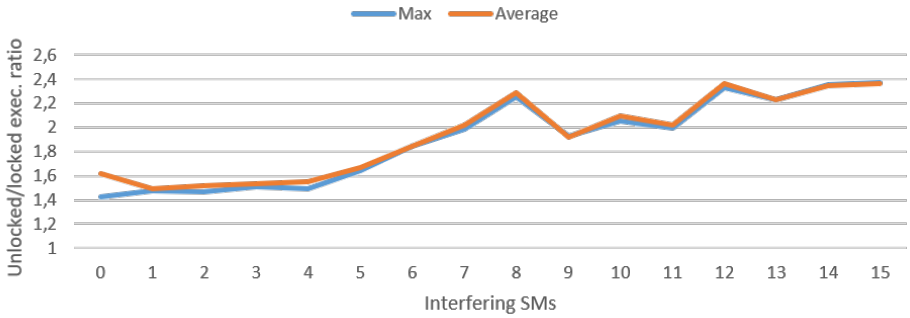
In this scenario, we seek to analyze the capacity of the L2 cache locking for isolating the SMs used by a critical application from SM interference produced by non-critical applications. To perform this study, we make use of SM partitioning in order to place together on a SM the thread blocks belonging to the same application. The partitioning is done by exploiting the threads per SM constraint (alternatively we can use the blocks per SM constraint). The interfering non-critical applications are executed as persistent kernels, i.e., the compute kernel re-execute the non-critical applications without launching from host again. The data resetting benchmark (Equation 2) plays the role of the interfering applications.

Figures 6a, 6b,6c show the WCET and ACET gain due to the L2 cache locking (Y axis) as function of the interfering number of SMs (X axis). The benchmarks used for testing are those described in Equations 2, 3, and 4.

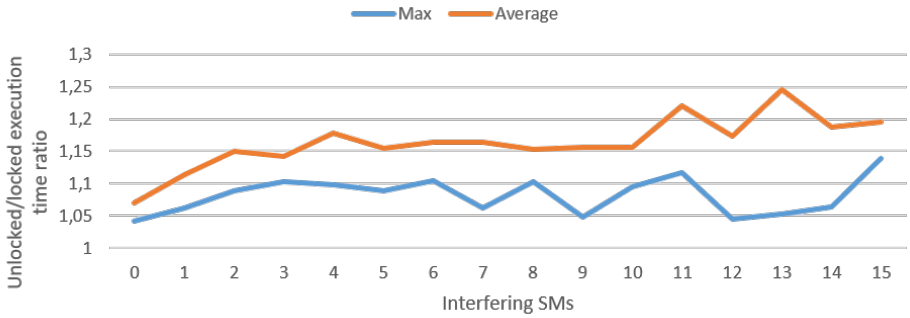
The measures taken from the data resetting benchmark (Figure 6a) indicate a clear benefit for the WCET and ACET metrics. The performance gain is more notable as the number of interfering SMs increases. For example, when half of the SMs are used by the critical application, the WCET and ACET are 2.26 and 2.29 times less respectively with respect to the L2 cache non-locking approach. As commented in Section 4.1, these gains are not the most typical ones as this benchmark specially makes use of persistent data whose protection from being evicted yields remarkable results. In this sense, the 2D and 3D convolution benchmarks (Figures 6b,6c) offer more common responses. The ACET of the former benchmark benefits more than its WCET, having time reductions of 13.24% and

9.38% respectively when using 8 critical SMs. The WCET and ACET of latter benchmark are similar, progressively benefiting of the L2 cache lock as the number of interfering SMs increases (no significant gains with low number of non-critical SMs). With 8 critical SMs, the WCET and ACET have a 10.14% and 6.8% boost respectively.

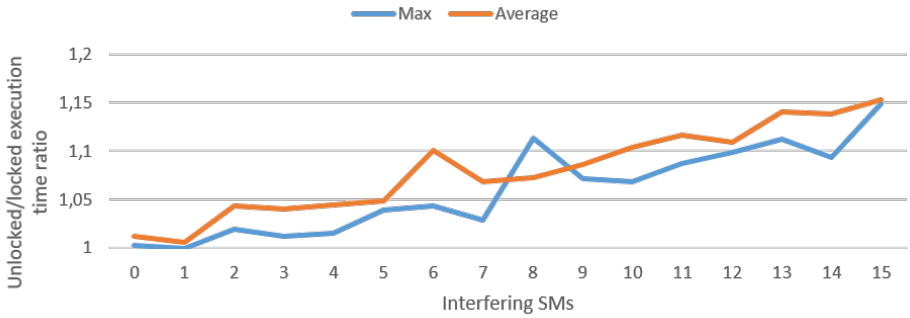
In general, the L2 cache locking serves well for protecting the locked persistent data from other concurrently running applications, leading to performance enhancements of the former. Its effectiveness mainly depends on the design of the applications (critical and non-critical) and the resources used for them (e.g., number of SMs, blocks, threads) but also on the L1 and L2 cache capacity of the iGPU.



(a) Data resetting. Configuration: 3 MB of persistent data and 103 MB of total data.



(b) 2D convolution. Configuration: 2 MB of persistent data and 4 MB of total data.



(c) 3D convolution. Configuration: 4 MB of persistent data and 8.04 MB of total data.

Figure 6 Performance gain under non-critical SM interference led by 3 MB L2 cache locking.

4.3 Scenario 3: LPDDR5 interference from ARM cores

The purpose of the third scenario is to analyze the capacity of the L2 cache locking for mitigating the overheads produced by the contention of the LPDDR memory when stressed by 11 ARM cores. These execute a benchmark based on stores and loads from different array strides, resulting in high level of DDR memory interference. By locking the persistent data, the number of accesses to the LPDDR memory from the iGPU would be reduced, and hence, suffer from less interference. We make use of two locking sizes (none or 3 MB) and different benchmark memory configurations. The benchmarks used are the data resetting (Equation 2), the 2D convolution (Equation 3) and the upsampling (Equation 5). The WCET and ACET response to the L2 cache locking of the two first benchmarks can be seen in Figures 7a and 7b and Figures 8a and 8b respectively.

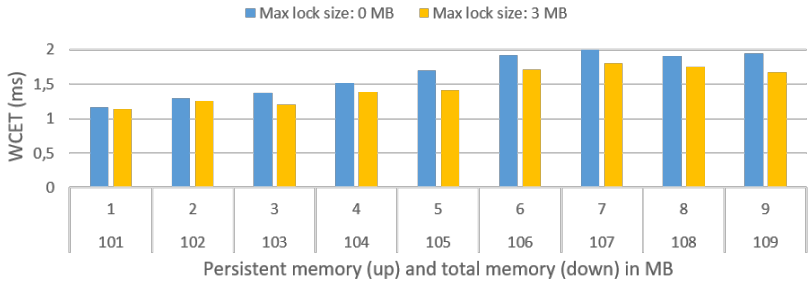
The data resetting benchmark results show a WCET and ACET improvement when applying the L2 cache lock. Nevertheless, by comparing these results with the homologous of Scenario 1 (Figure 3), we can deduce that the improvement comes from the inter-SM interference reduction rather than from avoiding the LPDDR5 interference as it would be expected. The LPDDR5 memory interference represents, on average, 25.07% and 20.21% of the WCET and ACET respectively. In contrast to the previous benchmark, the 2D convolution benefits from the mitigation of the LPDDR5 interference when applying the L2 cache locking. On average, 60.49% and 36.36% of the WCET and ACET reduction comes from this interference mitigation. On average, the LPDDR5 memory interference is responsible of 26.33% and 18.98% of the WCET and ACET respectively. In the same line, we have observed that for the upsampling benchmark, 65% and 63.16% of the WCET and ACET reduction are due to LPDDR memory interference mitigation.

All in all, we have seen that the L2 cache locking is able to reduce the main memory interference. However, as the data resetting benchmark has shown us, this is not always the case, reducing instead the inter-SM interference as in Scenario 1 (Section 4.1).

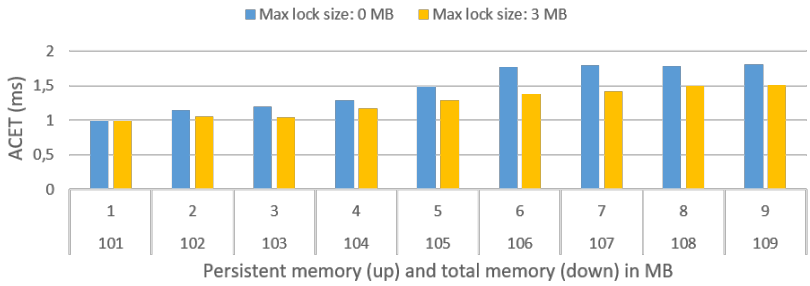
5 Conclusions and Future work

The effectiveness of the L2 cache locking depends on the application to execute, the GPU resources used of it, the amount of persistent and total data used by the application and the GPU L1 and L2 cache capacity. For the used tests and scenarios, we have observed a significant WCET and ACET reduction with many of the configurations we have used. The maximum WCET mitigation of inter-SM interference with one application (Section 4.1) lies between 10.24% and 28.35% depending on the test. For the inter-SM interference with 8 non-critical applications running concurrently (Section 4.2), we observe a maximum WCET reduction that ranges from 9.38% to 55.69%. In the case of the LPDDR interference scenario (Section 4.3), we have have seen maximum performance amelioration between 16.6% and 19.17%.

The benchmarks used in this work are made of a single compute kernel. Therefore, as future work, we would like to extend this study by testing the L2 cache locking for applications composed of chains of compute kernels like in neural network applications (e.g., Resnet [3], Segnet [2]). Besides, there is an aspect that should be studied regarding the L2 cache mechanism. According to NVIDIA, locked data on the L2 cache is automatically unlocked if not used [10]. The conditions under which this reset occurs should be analyzed.

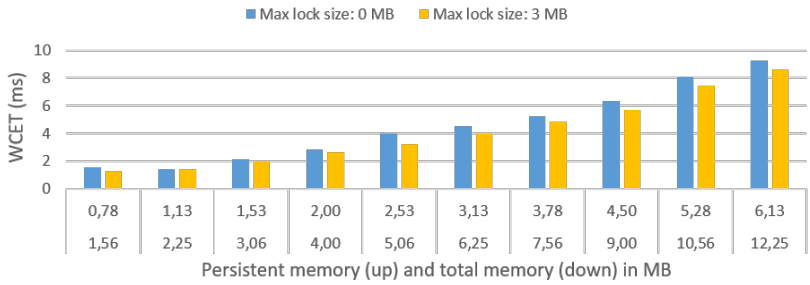


(a) Measured worst-case execution time.

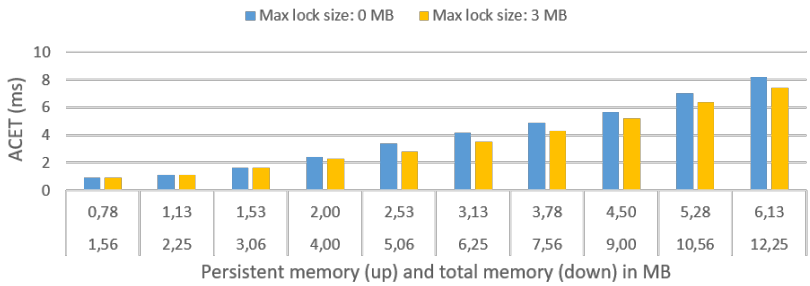


(b) Average-case execution time.

Figure 7 Data resetting behavior under LPDDR5 interference.



(a) Measured worst-case execution time.



(b) Average-case execution time.

Figure 8 2D convolution behavior under LPDDR5 interference.

References

1 Waqar Ali and Heechul Yun. Work-in-progress: Protecting real-time gpu applications on integrated cpu-gpu soc platforms. In *2017 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 141–144, 2017. doi:10.1109/RTAS.2017.26.

2 Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(12):2481–2495, 2017. doi:10.1109/TPAMI.2016.2644615.

3 Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. doi:10.1109/CVPR.2016.90.

4 Saksham Jain, Iljoo Baek, Shige Wang, and Ragunathan Rajkumar. Fractional gpus: Software-based compute and memory bandwidth reservation for gpus. In *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 29–41, 2019. doi:10.1109/RTAS.2019.00011.

5 Tamara Lugo, Santiago Lozano, Javier Fernández, and Jesus Carretero. A survey of techniques for reducing interference in real-time applications on multicore platforms. *IEEE Access*, 10:21853–21882, 2022. doi:10.1109/ACCESS.2022.3151891.

6 Antonio Martí-Campoy, Angel Perles, Francisco Rodríguez-Ballester, and J. Busquets-Mataix. Static use of locking caches vs. dynamic use of locking caches for real-time systems. In *CCECE 2003 - Canadian Conference on Electrical and Computer Engineering. Toward a Caring and Humane Technology*, volume 2, pages 1283–1286 vol.2, June 2003. doi:10.1109/CCECE.2003.1226134.

7 Alfonso Mascareñas González, Jean-Baptiste Chaudron, Frédéric Boniol, Youcef Bouchebaba, and Jean-Loup Bussenot. Task and memory mapping optimization for sdram interference minimization on heterogeneous mpsocs. In *2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–8. IEEE Press, 2022. doi:10.1109/ETFA52439.2022.9921677.

8 Sparsh Mittal. A survey of techniques for cache locking. *ACM Transactions on Design Automation of Electronic Systems*, 21(3), May 2016. doi:10.1145/2858792.

9 NVIDIA. *CUDA C++ Best Practices Guide*, May 2022.

10 NVIDIA. *CUDA C++ Programming Guide*, December 2022.

11 NVIDIA. *NVIDIA Orin Series System-on-Chip - TECHNICAL REFERENCE MANUAL*, March 2022.

12 NVIDIA. *Ampere Tuning Guide*, February 2023.

13 John Picchi and Wei Zhang. Impact of l2 cache locking on gpu performance. In *SoutheastCon 2015*, pages 1–4, 2015. doi:10.1109/SECON.2015.7133036.

14 Xin Wang and Wei Zhang. Cache locking vs. partitioning for real-time computing on integrated cpu-gpu processors. In *2016 IEEE 35th International Performance Computing and Communications Conference (IPCCC)*, pages 1–8, 2016. doi:10.1109/PCCC.2016.7820644.

15 Heechul Yun, Renato Mancuso, Zheng-Pei Wu, and Rodolfo Pellizzoni. Palloc: Dram bank-aware memory allocator for performance isolation on multicore platforms. In *2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 155–166, 2014. doi:10.1109/RTAS.2014.6925999.

16 Heechul Yun, Gang Yao, Rodolfo Pellizzoni, Marco Caccamo, and Lui Sha. Memguard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms. In *2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 55–64, 2013. doi:10.1109/RTAS.2013.6531079.