Performance Analysis and Optimization of Automotive GPUs

Fabio Mazzocchetti*†, Pedro Benedicte*†, Hamid Tabani*,
Leonidas Kosmidis*, Jaume Abella* and Francisco J. Cazorla*
* Barcelona Supercomputing Center † Universitat Politècnica de Catalunya

***Abstract*—Advanced Driver Assistance Systems (ADAS) and Autonomous Driving (AD) have drastically increased the performance demands of automotive systems. Suitable high-performance platforms building upon Graphic Processing Units (GPUs) have been developed to respond to this demand, being NVIDIA Jetson TX2 a relevant representative. However, whether high-performance GPU configurations are appropriate for automotive setups remains as an open question.**

This paper aims at providing light on this question by modelling an automotive GPU (Jetson TX2), analyzing its microarchitectural parameters against relevant benchmarks, and identifying specific configurations able to meaningfully increase performance within similar cost envelopes, or to decrease costs preserving original performance levels. Overall, our analysis opens the door to the optimization of automotive GPUs for further system efficiency.

I. Introduction

Critical real-time embedded systems (CRTES), such as those managing safety-related functionalities in avionics, space, automotive and railway, have built during decades on simple and low-performance microcontrollers. The increasing software complexity, inherent to the increase in the number and sophistication of delivered functionalities in those systems, has lead towards a slow adoption of multicore microcontrollers. For instance, the Infineon AURIX processor family [21] in the automotive domain, or Gaisler's LEON4 family [13] in the space domain deliver few cores (i.e. in the range 3 to 6) with the aim of providing a moderate performance scale up.

Such designs have already found difficulties to match the increasing complexity of software in those systems, which has increased at a rate of 10x every 10 years and, for the automotive domain reached up to 100 million lines of code for some cars in 2009 [11]. Moreover, as pointed out by ARM prospects, the advent of driver assistance systems and autonomous driving in the automotive domain will lead to a performance demand increase of 100x in the timeframe 2016-2024 [6], thus further exacerbating the performance needs for CRTES.

The answer to this performance demand has been the deployment of accelerators along with the microcontrollers, being Graphic Processing Units (GPUs) the main representative of those [17], [28], [24] despite the existing challenges using GPUs in this domain [29], [5]. In particular, several products such as Renesas R-Car H3 [1], NVIDIA Jetson TX2 [18] and NVIDIA Jetson Xavier [26] have already reached the market building upon GPU technology inherited from the high-performance domain. Automotive GPUs have inherited designs devised for the high-performance domain with the aim of reducing costs in the design, verification and validation process for chip manufacturers.

Unfortunately, reusability of high-performance hardware does not consider GPUs efficiency in the automotive domain and, to the best of our knowledge, the design space for GPUs, where resources are sized with the aim of optimizing specific goals such as performance, has not been yet thoroughly performed for the automotive domain.

This paper covers this gap by providing a GPU design exploration for the automotive domain by analyzing the influence that different microarchitectural hardware parameters, such as cache sizes, number of streaming multiprocessors (SMs), and the like have on performance for an automotive SoC representative, the NVIDIA Jetson TX2 platform [18]. In particular, the main contributions of this work are as follows:

1) An adaptation of a cycle-level CPU-GPU simulator, gem5-gpu [23], to match Jetson TX2 configuration as much as possible. This is the basis of our exploratory study.
2) A systematic performance analysis for the main hardware parameters, assessing to what extent they influence performance for a relevant set of benchmarks. We first analyze the effect of each parameter individually, and then the effect of two more parameters coordinately.
3) Finally, we propose two configurations which deliver: a) similar performance to the baseline design at a decreased hardware costs, and b) higher performance than the baseline design with comparable hardware costs.

Overall, our analysis shows that opportunities for optimization exist and can be exploited in two different axes depending on the needs of end-users.

The rest of the paper is organized as follows. Section II provides some background on automotive systems needing GPUs and on GPU architecture. Section III presents our methodology to model the NVIDIA Jetson TX2 GPU and the benchmarks we use to evaluate our proposals. Section IV provides the design space exploration for the Jetson TX2 GPU. Section V identifies and evaluates our two improved setups. Section VI reviews some related work, and Finally, Section VII summarizes the main conclusions of this paper.

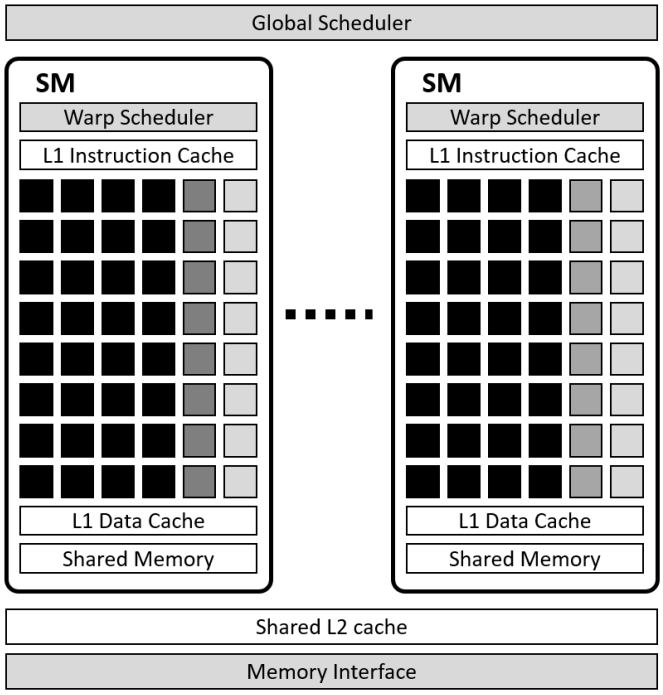


Fig. 1: Schematic of the architecture of a GPU.

II. Background

This section describes some background on the need for GPUs in automotive systems, as well as the basic organization of a GPU.

A. GPU-based Automotive Systems

The advent of Advanced Driver Assistance Systems and Autonomous Driving (AD) imposes a higher level of system autonomy to take decisions on behalf of the driver, or even to fully replace the driver, as expected for the systems with Autonomy Level 5 – the highest autonomy level according to SAE International [25]. To make these systems real, a number of processes related to *Perception* and *Prediction* modules of autonomous driving systems need to be automated to process large amounts of data from sensors (camera, LiDAR, radar) in real-time to deliver system responses timely [4], [3], [2]. Therefore, object detection, trajectory prediction, and collision avoidance algorithms, among others, need extremely high performance to take driving decisions in very short timeframes.

B. GPU Architecture

While different GPUs from different vendors may have significant differences, and differences may also be relevant across different GPU generations for the same vendor, some elements are mostly common to all GPUs. Next, we describe those, as they are the basis of the study provided later on.

Figure 1 depicts the main elements of the architecture of the GPU. First, it has a global scheduler that dispatches work to the different Streaming Multiprocessors (SMs) of the GPU. Each SM, to some extent, is a cluster of computing resources orchestrated coordinately, whereas different SMs may lack any coordination and work highly independently. Each SM has a set of storage resources that include a first level (L1) instruction cache and an L1 data cache, as well as some shared memory to manage shared data. Each SM also includes a *warp* scheduler, where warp refers, in NVIDIA nomenclature, to the set of identical computations that are dispatched to the parallel computing elements atomically. Thus, it is the smallest scheduling unit.

Computing elements include CUDA[1] cores (indicated with black squares in the figure), able to process a warp entirely, as long as the particular operation requested is part of the CUDA cores, which typically include most integer and single-precision floating-point arithmetic operations. Along with the CUDA cores, some other units (indicated with different gray squares) perform other types of operations, typically with lower bandwidth than CUDA cores, such as load/store operations of data, and some area-costly operations that whose hardware cannot be replicated as many times as CUDA cores (e.g., double precision or highly-complex floating point operations).

SMs also share one or more levels of cache (e.g. L2), thus competing for cache space across SMs, as well as the memory interface. Note, however, that different GPU architectures may be different. For instance, it is not uncommon having clusters of SMs, so that each cluster shares a cluster-local shared cache, and then all clusters share a global L2 cache. Still, the concept of highly parallel computing resources and hierarchical storage organization holds in the general case for GPUs.

III. Methodology

A. Simulation Infrastructure

In this paper, we have used an in-house version of the gem5-gpu simulator [23]. Gem5-gpu is a cycle-accurate and heterogeneous CPU-GPU simulator incorporating gem5 [8] and gpgpu-sim [7] simulators. These are the most accurate and widely-used cycle-level simulators in the computer architecture community. In our version of gem5-gpu, we applied major modifications to include latest versions of gem5 and gpgpu-sim. This was a fundamental requirement before being able to model latest GPU architectures such as Pascal since in the baseline simulator only Fermi was supported.

B. Modelling NVIDIA Tegra X2

We have modeled an SoC, similar to NVIDIA TX2, in our simulator. We have done a comprehensive study to extract the available and public architectural parameters of TX2 in order to tune the simulator to closely model the chosen platform. We have extracted as much information as we could from public sources about our NVIDIA GPU and we included them in the configuration of the gpgpu-sim. However, we still do not have access to part of the detailed parameters since they are not provided by the manufacturer. Alternatively, we have tried to estimate the missing parameters according to the available information and also by fine-tuning those parameters with the

[1]CUDA is the programming paradigm for NVIDIA GPUs.

	Configuration
GPU	NVIDIA Pascal: 256 CUDA cores 2 SMs with 4 SMBs each. 32 CUDA cores per SMB
CPU	2-core Denver2 (128KB 4-way IL1, 64KB, 4way dL1) 4-core ARM A57 (48KB 3-way IL1, 32KB 2-way dL1) 2MB 16-way L2 per cluster
DRAM	8 GB, 256-bit LPDDR4x, 59.7 GB/s

TABLE I: System Configurations of NVIDIA TX2 SoC.

help of synthetic experiments. Table II shows the detailed parameters that we have employed in our simulator.

To out knowledge, the latest gpgpu-sim version (released in late 2018) is the most accurate and open-source simulator to model a Pascal architecture, which is the architecture used in the TX2 GPU. In addition, we have designed several synthetic benchmarks to validate our configuration against the real GPU.

NVIDIA TX2 is based on the 16nm NVIDIA Tegra Parker system on chip (SoC). The TX2 has a Pascal GPU with 2 SMs, each of them with 4 SM Blocks (SMBs). Each SMB comprises of 32 cores and in total the GPU has 256 cores. The TX2 SoC also comprises of two different clusters of dual- and quad-core CPUs, whose L2 cache is shared inside each cluster. The first CPU cluster, *Denver2*, has 2 cores each with its own private first level instruction and data caches (referred to as iL1 and dL1 respectively). The other CPU cluster has 4 ARM Cortex-A57 cores also with private iL1 and dL1 caches. Table I presents the architectural parameters of both CPUs and the GPU in NVIDIA TX2. In this work, we focus on the development board of the TX2 processor (Jetson), which has one SoC. It is worth pointing out that commercial versions of the TX2 can have up to 2 SoCs like the one described and even one discrete GPU.

C. Benchmarks

In this paper, we use Rodinia [12] benchmark suite for our experiments. Rodinia benchmark suite is targeting heterogeneous computing and in order to study emerging platforms such as GPUs, Rodinia suite includes applications and kernels that target multi-core CPU and GPU platforms.

The EEMBC (Embedded Microprocessor Benchmarks Consortium) recently released ADASMark [14], an ADAS Benchmark suite that would be highly relevant for our study. However, in the moment of writing this paper we still have not been able to access this benchmark suite. Therefore, we used some of the most suitable benchmarks for GPU microarchitecture such as Rodinia. In fact, Rodinia includes some key kernels in autonomous driving systems that have similarities with ADASMark such as image processing and pattern recognition.

IV. Design Space Exploration

The objective of the design space exploration is to know which parameters of the processor design could be increased/decreased and what would be their impact on performance. Since there are many parameters that can influence

	CPU Configuration
Core	ARMv8 ISA, 2.0 GHz, 128-entry ROB 40-entry Issue Queue, Full Out-of-Order 3-Width Decoder, 3-Width Instruction Dispatch 48 KB, 3-Way TLB 48-Entry Fully-Associative L1 TLB
Caches	32 KB L1-D Cache, 2-Way, 1 Cycle 48 KB L1-I Cache, 3-Way, 1 Cycle 2 MB L2 Cache, 16-Way, 12 Cycles 64 Bytes Cache Line Size
Prefetcher	Stride Prefetcher (Degree 1) 2K Branch Target Buffer (BTB) 32-Instruction Fetch Queue 15 Cycles Misprediction Penalty
DRAM	DDR4 1866 MHz, 2 Ranks/Channel 8 Banks/Rank, 8 KB Row Size. t_{CAS} = t_{RCD} = t_{RP} = CL = 13.75ns t_{REFI} = 7.8 us
	GPU Configuration
SMs	1.1 GHz, 32 Warps, 65536 shader registers 32 Thread blocks, 2048 threads per core 4 scheduler per core
Memory	48KB 4-way, 512 KB 4-way L2 64KB shared memory, 8 GB total memory size 16 sub-partition per memory channel

TABLE II: System Configurations employed in the gem5-gpu simulator.

performance, making all the possible combinations is unfeasible. Thus, we will first change one parameter at a time, and then changing more than one parameter together (for instance, size and way of caches).

A. Changing one parameter

In Figure 2, we see the results for the design space exploration when changing only one parameter. Please note that the plots have 2 different scales: 0-2 (a, b, d, e, h) and 0-5 (c, f, g, i, j, k). Y-axis shows the slowdown in comparison with the baseline configuration. In the first row, we show the results for the variation in L1 and L2 associativity and L2 size. Increasing the L1 size provides a small performance improvement, while reducing the size degrades the performance. When reducing the L2 cache size, we observe performance degradation, however, further increasing the L2 cache size does not provide further performance improvement.

Regarding the associativity, neither changes in L1 nor in L2 result in significant changes when increasing or decreasing it moderately (1 to 8 ways). This parameters are again tested in the next part since changing both size and associativity at the same time may have different effects that are not seen when changing them one at a time.

Figure 2 e) shows that doubling the number of CUDA cores does not increase performance, but reducing it can result in significant performance degradation, specially in compute-intensive applications like hotspot or particlefilter_naive. In the next two Figures, f) and g), we show the effects on changing the sizes of the register file and the shared memory. Both components do not show significant differences when reducing

Name	Short name	Problem type	Domain
Back Propagation	backprop	Unstructured Grid	Pattern Recognition
Breadth-First Search	bfs	Graph Traversal	Graph Algorithms
3D Stencil	cell	Structured Grid	Cellular Automation
Gaussian Elimination	gaussian	Dense Linear Algebra	Linear Algebra
Hotspot3D	hotspot	Structured Grid	Physics Simulation
Myocyte	myocyte	Structured Grid	Biological Simulation
Needleman-Wunsch	needle	Dynamic Programming	Bioinformatics
k-Nearest Neighbors	nn	Dense Linear Algebra	Data Mining
Particle Filter	pf_float	Structured Grid	Medical Imaging
Particle Filter	pf_naive	Structured Grid	Medical Imaging
SRAD	srad	Structured Grid	Image Processing

TABLE III: Rodinia benchmarks used in the experiments.

its size by half or increasing it to double. Figure 2 h) changes the number of warp schedulers in each SM. While increasing (from 4 to 8 or 16) it does not show any performance benefits, and reducing it to just 1 incurs in significant penalty (10% more execution time), reducing it by half (from 4 to 2) results in the same average performance, with almost no variability in the different benchmarks.

In Figures 2 i), 2 j), and 2 k), we show the results for the number of SMB per SM, the number of SM per cluster, and the number of clusters with just 1 SM, respectively. Since the simulator does not support more than 4 SMB per SM (the standard in Pascal), we just focus on reducing it to 1 and 2. We observe that this reduction would incur in significant performance penalties. The last two (SM per cluster and number of clusters) show a similar trend, since in the end both of them are increasing the total amount of SMs. We observe that increasing the total number of SMs has positive effects in performance, but just up to a certain point. The increase in SMs has to be tailored to the application implementation and behavior, so some applications may not be able to use all these SMs while others could if they were implemented with more granularity. These two components are the ones that vary most depending on the benchmark, with several improving performance and others decreasing it.

B. Changing two or more parameters

Some parameters that changed on their own have a specific impact on performance, can behave differently if changed together with other parameters. This is because how one parameter performs depends also on how other parameters perform. This is obvious in caches. Changing the associativity of a small cache may have bigger impact on performance than changing the associativity of a big cache, since sets may be big enough anyway to notice any degradation.

The first two parameters that we have changed together are cache size and associativity, both for L1 (Figure 3a) and L2 (Figure 3b). For both we have changed the associativity (first element in the x-axis) and the size (second element) together. Although for the L1 we do not see significant performance changes, for the L2 we see that reducing the size and associativity to small enough numbers we have significant performance degradation. This degradation is bigger than the one observed in the previous experiments when only changing the associativity or the size separately.

In the next experiments, we change the size of both caches at the same time (Figure 3c), and we see similar results to the previous experiment, with a lot of degradation when the L2 size is small. We also changed both L1 and L2 associativities at the same time (Figure 3d) and as in previous experiments we see that it has no noticeable effect. In the last experiment with multiple cache components, we change both L1 and L2 sizes and associativities. The results are similar to b) and c), with big performance penalties in the small setup and no improvement in the big one.

Finally, in Figure 3f, we change both the number of SM as well as the number of CUDA cores per SM, changed from 128 to just 32. Comparing with the previous experiment, Figure 2k, we see that the trends are similar, with small variations depending on the benchmark.

C. Changing the software

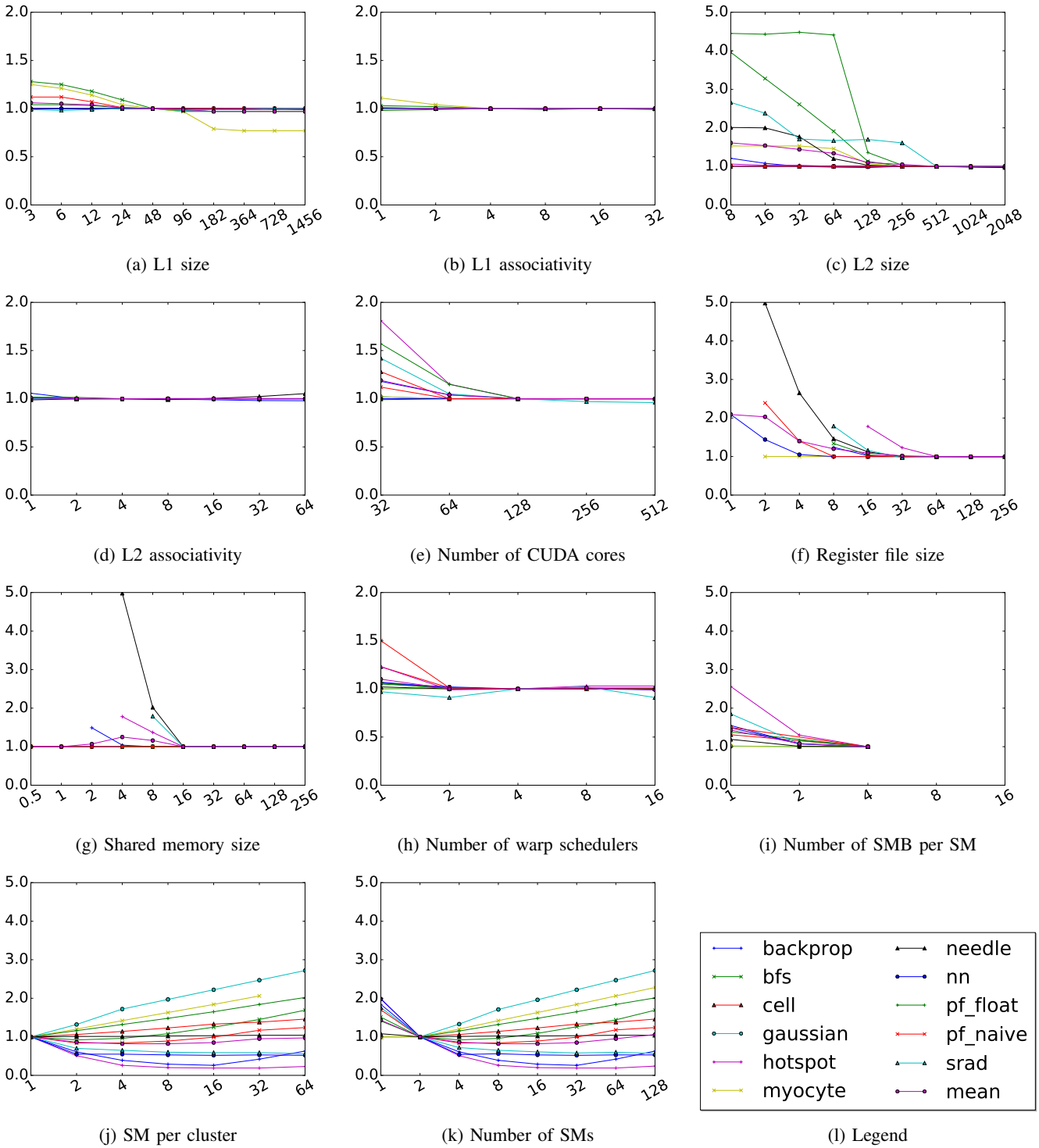
The changes in performance we observe when modifying the different hardware components depend on the type of application we are running (compute intensive, memory intensive etc), but also on the specific CUDA/OpenCL implementation done. Usually, when parallelizing an application for GPUs the computation is divided into grids and thread blocks (using NVIDIA's terminology). A specific grid and thread block division of a program could be optimal for a configuration (number of SM, sizes of caches etc) while being suboptimal for others.

D. Parameter classification

Depending on the results obtained in the design space exploration, we classify the parameters into four categories:

1) Parameters that (based on Rodinia) are not worth to increase beyond a given point since they produce no gains.
2) Parameters that require SW to be modified (e.g. block-/grid) to make it worth to change them.

In the following table (Table IV), we show a classification of the parameters into the two different types, and the last column shows the limit where an increase of the parameter does not provide significant performance benefits.



(a) L1 size (b) L1 associativity (c) L2 size

(d) L2 associativity (e) Number of CUDA cores (f) Register file size

(g) Shared memory size (h) Number of warp schedulers (i) Number of SMB per SM

(j) SM per cluster (k) Number of SMs (l) Legend

Fig. 2: Design Space Exploration results for one parameter changes (all sizes are in KB).

V. Improved setups

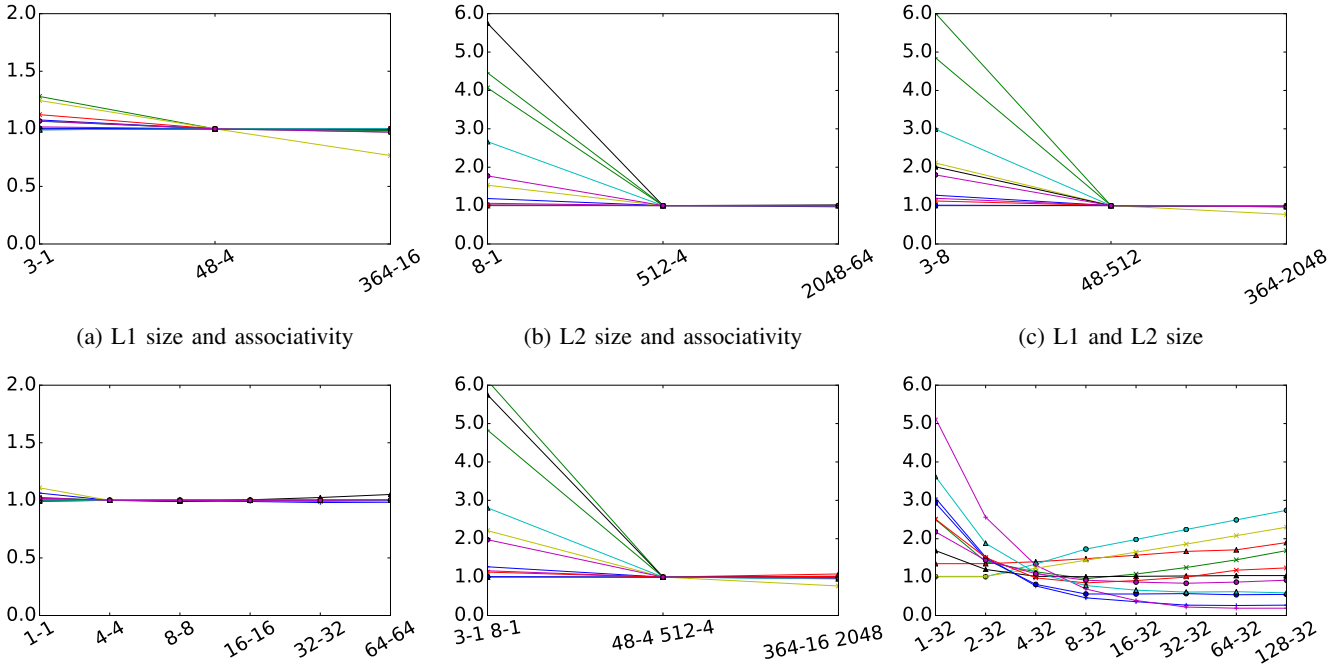
Based on the knowledge obtained in the design space exploration, we want to propose modified hardware designs with some improvement, either in terms of performance or in terms of die area. The two setups that we propose are:

- Same performance with less cost.
- More performance with the same cost.

These setups would be optimized for the Rodinia benchmarks used in the design space exploration and will be based on the TX2 basic design.

A. Proposed setups

The two proposed setups and its objectives are:



(a) L1 size and associativity (b) L2 size and associativity (c) L1 and L2 size

(d) L1 and L2 associativity (e) L1 and L2 size and associativity (f) SM with 32 CUDA cores

Fig. 3: Design Space Exploration results for several parameter changes (all sizes in KB). The legend is the same used in Figure 2.

Parameter	Category	Limit
L1 size	1	48KB
L1 associativity	1	4
L2 size	1	256KB
L2 associativity	1	2
Number of CUDA cores	1	128
Register file size	1	64KB
Shared memory size	1	16KB
Number of warp schedulers	1	2
Number of SMB per SM	2	-
SM per cluster	2	-
Number of SMs	2	-

TABLE IV: GPU parameters classified depending on the potential improvement on hardware or software.

- *Decrease the hardware keeping the same performance.* The first improved setup proposed aims to reduce the amount of die space used while roughly keeping the same performance (within 5% of performance degradation). Looking at the results of the design space exploration, the features that are susceptible to being reduced without too much performance degradation are: register size, warp schedulers per SM and shared memory. In all of these features, the number of units or size can be divided by half without having a significant impact in performance. Thus, our proposal is to use the same configuration but reducing the register size from 64KB to 32KB ($\frac{1}{2}$), the warp schedulers per SM from 4 to 2 ($\frac{1}{2}$) and the shared memory from 64KB to 16KB ($\frac{1}{4}$).
- *Increase performance using the same hardware.*

Parameter changed	Base setup	Reduced die space	Increased perf. a)	Increased perf. b)
Number of SM	2	2	4	4
Warp sched	4	2	4	4
Register file	64KB	32KB	64KB	64KB
Shared mem	64KB	16KB	64KB	64KB
L1 size	48KB	48KB	96KB	96KB
L2 size	512KB	512KB	256KB	128KB

TABLE V: Changes made in the improved setups.

In order to increase the performance using the same amount of die space, we need to increase some resources, which will increase performance and area and decrease others, which will decrease performance but decrease area. The tradeoff between the resources increased and decreased needs to be positively balanced to improve the performance per die area.

Based on the design space exploration, our proposal for this setup is to increase the number of SMs and L1 cache sizes, while decreasing the size of the L2. The difference in die space between the new SMs and larger L1 should be similar to the decrease in L2 size. Furthermore, we will provide two variations of this setup. In one we will double the size of the L1 and the number of SMs, while reducing the L2 size by half. Furthermore, in a more area limited setup, we will also double the L1 and number of SMs, while reducing the L2 to one fourth of its size.

Justifying that the hardware cost of our proposed setups is challenging without having information about the actual

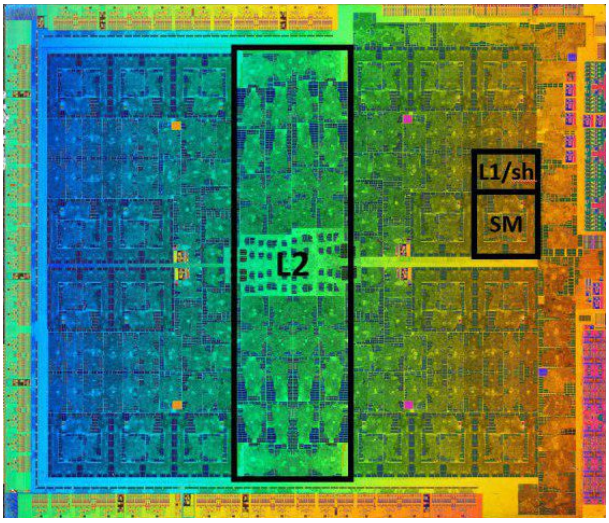


Fig. 4: NVIDIA GTX 1080 die. Same architecture (Pascal) and manufacturing process (16nm) as the TX2.

space occupied by each resource in the real hardware implementation. Since there is no available information about the Tegra X2 die, instead we use, as a reference, the GTX 1080's die information. The GTX 1080 is a discrete graphics card developed by NVIDIA with the same architectural generation (Pascal) and manufacturing process (16nm).

In the GTX 1080 die, we see the area dedicated to the SMs to the per SM caches (including L1 and shared memory), to the shared L2 and the rest to I/O and other features (debugging, performance counters etc). Although these are estimates, they give us a good idea of the overall die space used to each processor part.

B. Evaluation

In the first setup, the improvement is in the die space used. Specifically, we use half the register size, half the schedules and a quarter of the shared memory. Of the three, the shared memory is the one that has a bigger impact in die area reduced. An advantage of reducing these components is that they are private to each SM, so if we increase the number of SMs, the reduced area would be proportional to the number of SMs. In contrast with decreasing shared resources such as the L2 cache, that would have the same impact if we changed the number of SMs.

From the 11 Rodinia benchmarks that we analyzed, 8 of them have the same performance (within less than 1% of variation). Based on our previous analysis, from the 3 benchmarks that show a significant increase in execution time, one (hotspot) is mainly due to the decrease in register size and the other 2 (needle and srad) are due to the decrease in shared memory size. On average, less than 5% of performance degradation is shown when making these changes.

In the second setup, we want to improve performance while increasing some resources and decreasing others to maintain die area. Since knowing the exact area gained or lost with each change is challenging, we propose two setups. In both,

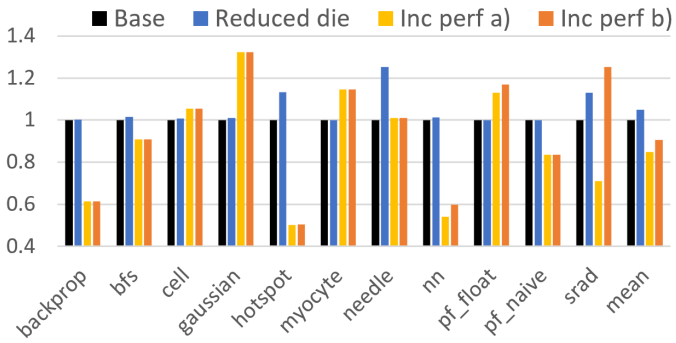


Fig. 5: Execution time of the proposed setups normalized to the baseline.

we double the number of SMs and the L1 size, but in one, we reduce the L2 size by half and in the other, by one quarter.

For both variations, the results change significantly depending on the benchmark. Of the 11 benchmarks, in 4 of them, we observe a performance improvement, in 4 of them we observe a slowdown, and the rest stay the same. Overall, the setup that halves the L2 shows a 10% reduction in execution time while the setup that divides it by 4 shows a 5%.

Between the two setups, some benchmarks (cell, gaussian, hotspot, particlefiler_naive) show no difference at all. Of the ones that show difference, all, except for bfs, perform better in the 256KB L2 than in the 128KB.

VI. Related work

GPU performance analysis for general purpose applications has been a research topic for many years. Two main research lines exist on this topic. The first one focuses on the analysis of Commercial Off-The-Shelf (COTS) GPUs by means of the execution of different types of parallel applications [30], [27], [22]. These studies have allowed determining what the most convenient way is to deploy and run software on those GPUs and, at most, it has been guessed how hardware should be modified to improve performance. However, since COTS GPUs cannot be modified, hypotheses raised cannot be verified. The second research line on this topic considers the use of GPU performance simulators in order to determine how to tune high-performance GPUs for general purpose high-performance applications [19], [20]. However, those GPUs are significantly different from automotive ones since they are not subject to strict power constraints such as those in the automotive domain, which are intended to operate under much lower power envelops. Thus, conclusions cannot be extrapolated across both market segments.

Several works have analyzed the performance of COTS automotive GPUs to optimize the behavior of applications running atop [16]. While conclusions reached by those works are highly valuable for an efficient use of hardware, they do not provide any insight on how to optimize hardware design. Finally, some works target task scheduling on GPUs for an efficient use of hardware resources, minimizing their timespan while respecting their deadlines [15], [10], [9].

VII. Conclusions

Performance requirements of future automotive systems have increased significantly with the promise of Autonomous Driving. GPUs are commonly used to reach the required performance levels. In this work, we focus on the NVIDIA Jetson TX2, a widely-used and well-known platform that focuses on automotive domain.

First, we modelled the TX2 in our CPU-GPU simulator as a representative solution of GPUs for the automotive domain. Then, using Rodinia benchmarks that focus on several heterogeneous computing (CPU+GPU) applications, we analyzed how the different parameters of the TX2's GPU affected performance. This is done with isolated changes (just changing one parameter at the same time) as well as with combined changes (changing several parameters).

Building on the conclusions of this experimentation, we propose three different improved setups: one that reduces die space significantly with less than a 5% impact in performance, and two that maintain approximately the same die space but reduce execution time from 10 to 15%.

Some hardware improvements could only be fully exploited by modifying the software (mainly the block and thread distribution). We leave this tuning of software as future work to continue improving GPU-based system to meet the demands of the automotive industry.

Acknowledgments

This work has been partially supported by the Spanish Ministry of Economy and Competitiveness (MINECO) under grant TIN2015-65316-P, the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 772773) and the HiPEAC Network of Excellence. Pedro Benedicte and Jaume Abella have been partially supported by the MINECO under FPU15/01394 grant and Ramon y Cajal postdoctoral fellowship number RYC-2013-14717 respectively and Leonidas Kosmidis under Juan de la Cierva-Formacin postdoctoral fellowship (FJCI-2017-34095).

References

[1] RENESAS R-Car H3. https://www.renesas.com/en-us/solutions/automotive/products/rcar-h3.html.

[2] Autoware. An open autonomous driving platform. https://github.com/CPFL/Autoware/, 2016.

[3] Udacity. An Open Source Self-Driving Car. https://github.com/udacity/self-driving-car/, 2017.

[4] Apollo, an open autonomous driving platform. http://apollo.auto/, 2018.

[5] Sergi Alcaide, Leonidas Kosmidis, Hamid Tabani, Carles Hernandez, Jaume Abella, and Francisco J Cazorla. Safety-related challenges and opportunities for gpus in the automotive domain. *IEEE Micro*, 38(6):46–55, 2018.

[6] ARM. ARM Expects Vehicle Compute Performance to Increase 100x in Next Decade. https://www.arm.com/about/newsroom/arm-expects-vehicle-compute-performance-to-increase-100x-in-next-decade.php, ARM Press Release, April 2015.

[7] Ali Bakhoda, George L Yuan, Wilson WL Fung, Henry Wong, and Tor M Aamodt. Analyzing cuda workloads using a detailed gpu simulator. In *2009 IEEE International Symposium on Performance Analysis of Systems and Software*, pages 163–174. IEEE, 2009.

[8] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R Hower, Tushar Krishna, Somayeh Sardashti, et al. The gem5 simulator. *ACM SIGARCH Computer Architecture News*, 39(2):1–7, 2011.

[9] N. Capodieci, R. Cavicchioli, M. Bertogna, and A. Paramakuru. Deadline-based scheduling for GPU with preemption support. In *2018 IEEE Real-Time Systems Symposium (RTSS)*, pages 119–130, Dec 2018.

[10] Nicola Capodieci, Roberto Cavicchioli, and Marko Bertogna. NVIDIA GPU scheduling details in virtualized environments: Work-in-progress. In *Proceedings of the International Conference on Embedded Software*, EMSOFT '18, pages 12:1–12:3, Piscataway, NJ, USA, 2018. IEEE Press.

[11] R.N. Charette. This car runs on code. In *IEEE Spectrum online*, 2009.

[12] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W. Sheaffer, Sang-Ha Lee, and Kevin Skadron. Rodinia: A benchmark suite for heterogeneous computing. In *IEEE International Symposium on Workload Characterization*, 2009.

[13] Cobham Gaisler. *Quad Core LEON4 SPARC V8 Processor - LEON4-NGMP-DRAFT - Data Sheet and Users Manual*, 2011.

[14] EEMBC. ADASMark.

[15] Glenn A. Elliott. *Scheduling of GPUs, with applications in advanced automotive systems*. PhD thesis, The University of North Carolina at Chapel Hill, 2015.

[16] J. Fickenscher, O. Reiche, J. Schlumberger, F. Hannig, and J. Teich. Modeling, programming and performance analysis of automotive environment map representations on embedded GPUs. In *2016 IEEE International High Level Design Validation and Test Workshop (HLDVT)*, pages 70–77, Oct 2016.

[17] E. Francis. Autonomous cars: no longer just science fiction. *Automotive Industries*, 193, 2014.

[18] E. Francis. Autonomous cars: no longer just science fiction. *Automotive Industries*, 193, 2014.

[19] Sunpyo Hong and Hyesoon Kim. An analytical model for a GPU architecture with memory-level and thread-level parallelism awareness. In *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ISCA '09, pages 152–163, New York, NY, USA, 2009. ACM.

[20] Sunpyo Hong and Hyesoon Kim. An integrated GPU power and performance model. In *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ISCA '10, pages 280–289, New York, NY, USA, 2010. ACM.

[21] Infineon. AURIX Multicore 32-bit Microcontroller Family to Meet Safety and Powertrain Requirements of Upcoming Vehicle Generations. http://www.infineon.com/cms/en/about-infineon/press/press-releases/2012/INFATV201205-040.html.

[22] Paulius Micikevicius. GPU performance analysis and optimization. In *Proceedings of the 3rd GPU Technology Conference*, GTC '12, 2012.

[23] Jason Power, Joel Hestness, Marc S Orr, Mark D Hill, and David A Wood. gem5-gpu: A heterogeneous cpu-gpu simulator. *IEEE Computer Architecture Letters*, 14(1):34–36, 2015.

[24] Roger Pujol, Hamid Tabani, Leonidas Kosmidis, Enrico Mezzetti, Jaume Abella, and Francisco J Cazorla. Generating and exploiting deep learning variants to increase heterogeneous resource utilization in the nvidia xavier. In *31st Euromicro Conference on Real-Time Systems (ECRTS 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[25] SAE International. *J3016: Taxonomy and Definitions for Terms Related to On-Road Motor Vehicle Automated Driving Systems*, 2014.

[26] Danny Shapiro. Introducing xavier, the nvidia ai supercomputer for the future of autonomous transportation. *NVIDIA blog*, 2016.

[27] Jaewoong Sim, Aniruddha Dasgupta, Hyesoon Kim, and Richard Vuduc. A performance analysis framework for identifying potential benefits in GPGPU applications. In *Proceedings of the 17th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '12, pages 11–22, New York, NY, USA, 2012. ACM.

[28] K. Suleman. Intel paves the road for bmw's inext autonomous cars in 2021. 2017.

[29] Hamid Tabani, Leonidas Kosmidis, Jaume Abella, Francisco J Cazorla, and Guillem Bernat. Assessing the adherence of an industrial autonomous driving framework to iso 26262 software guidelines. In *Proceedings of the 56th Annual Design Automation Conference 2019*, page 9. ACM, 2019.

[30] Y. Zhang and J. D. Owens. A quantitative performance analysis model for GPU architectures. In *2011 IEEE 17th International Symposium on High Performance Computer Architecture*, pages 382–393, Feb 2011.