G-GPU: A Fully-Automated Generator of GPU-like ASIC Accelerators

Tiago D. Perez*, Márcio M. Gonçalves†, Leonardo Gobatto†, Marcelo Brandalero‡, José Rodrigo Azambuja†, Samuel Pagliarini*

* Department of Computer Systems, Tallinn University of Technology (TalTech), Estonia
† Institute of Informatics, Federal University of Rio Grande do Sul (UFRGS), Brazil
‡ Brandenburg University of Technology (B-TU), Germany

Emails:{tiago.perez,samuel.pagliarini}@taltech.ee,{marcio.goncalves,leonardo.gobato,jose.azambuja}@inf.ufrgs.br,marcelo.brandalero@b-tu.de

***Abstract*—Modern Systems on Chip (SoC), almost as a rule, require accelerators for achieving energy efficiency and high performance for specific tasks that are not necessarily well suited for execution in standard processing units. Considering the broad range of applications and necessity for specialization, the design of SoCs has thus become expressively more challenging. In this paper, we put forward the concept of G-GPU, a general-purpose GPU-like accelerator that is not application-specific but still gives benefits in energy efficiency and throughput. Furthermore, we have identified an existing gap for these accelerators in ASIC, for which no known automated generation platform/tool exists. Our solution, called GPUPlanner, is an open-source generator of accelerators, from RTL to GDSII, that addresses this gap. Our analysis results show that our automatically generated G-GPU designs are remarkably efficient when compared against the popular CPU architecture RISC-V, presenting speed-ups of up to 223 times in raw performance and up to 11 times when the metric is performance derated by area. These results are achieved by executing a design space exploration of the GPU-like accelerators, where the memory hierarchy is broken in a smart fashion and the logic is pipelined on demand. Finally, tapeout-ready layouts of the G-GPU in 65nm CMOS are presented.**

***Index Terms*—ASIC generator, domain-specific accelerators, general-purpose gpu architectures, integrated circuits**

I. Introduction

New computer applications, especially in the field of Artificial Intelligence (AI), keep pushing the need for more energy-efficient hardware architectures [1]. For many years, application- and domain-specific accelerators, designed by specializing to the task at hand, have been the standard choice for achieving high energy efficiency. Canonical examples are crypto cores [2] and Graphics Processing Unit (GPUs) for which even specialized programming languages and paradigms have been proposed [3]. GPU architectures focus on specialized massively parallel many-core processors that take advantage of Thread-Level Parallelism (TLP) to handle highly parallelizable applications in a Single-Instruction Multiple Threads (SIMT) paradigm. GPUs have been traditionally designed for graphics applications but have recently evolved into efficient general-purpose accelerators for High-Performance Computing (HPC). HPC applications have a wide range, including oil exploration, bioinformatics, and the thriving AI and Machine Learning (ML) domains [4]. NVIDIA GPUs, for instance, are used as accelerators in several top500 supercomputers.

However, despite its widespread use as accelerators, research in GPU architectures is limited due to the lack of open-source models at a sufficiently low level of abstraction and that are representative of modern architectures. To the best of our knowledge, the only configurable open-source GPU architectures available in the literature are FlexGripPlus [5] and FGPU [6]. The first is based on the NVIDIA G80 decade-old architecture and has never been deployed to an FPGA board. The second was designed specifically for FGPA platforms. Therefore, the literature has not yet tackled the challenges in designing, configuring, and implementing modern GPU architectures for ASICs – a platform that presents challenges that are far from those in FPGA design. Still, all commercial GPUs are designed as ASICs.

This work proposes to **bridge this gap** with GPUPlanner, an automated and open-source framework for generating ASIC-specific GPU-like accelerators as IP. We term these general-purpose accelerators G-GPUs. GPUPlanner helps designers in generating GPU-like accelerators through user-driven customization and automated physical implementation. Customization is performed according to a given GPU architecture through a series of parameters that define computation characteristics (e.g., number of processing units) and memory access (e.g., cache sizes), thus providing designers a high degree of scalability to better fit the generated IP into their systems. Implementation strategies explore the use of *smart memories* and on-demand pipeline insertion.

We evaluate our proposed framework by implementing four flavors of G-GPU architectures in terms of performance, power, and area (PPA). Additionally, we provide a reasonable comparison with the popular CPU architecture RISC-V [7], [8] in terms of raw performance speed-up and performance speed-up derated by area. Our main contributions is an open-source framework for automated generation of GPU-like accelerators, from RTL to GDSII – the GPUPlanner.

II. Hardware Accelerators and our Baseline GPU

In a nutshell, domain- or application-specific accelerators cost too much. Recent developments in High-Level Synthesis (HLS) [9] are encouraging and have helped in accelerate the development of domain-specific hardware accelerators. Yet, for ASIC designs, the performance for flexibility trade-off is not interesting, or the performance is insufficient [10]. This scenario presents itself as an opportunity where general-purpose accelerators have gained ground. Our proposed GPUPlanner framework combines the efficiency from domain-specific accelerators and the ease of use from general-purpose architectures into G-GPU. The result is an automatically

978-3-9819263-6-1/DATE22/©2022 EDAA

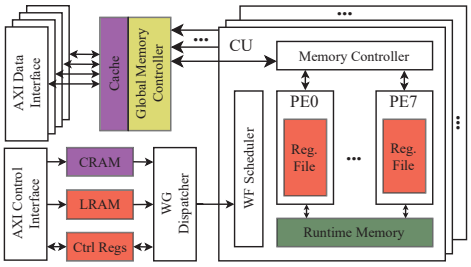

Fig. 1: FGPU architecture colored according to Fig 3.

generated domain-specific ASIC accelerator based on GPU architectures that can be easily programmed with modern programming languages.

FGPU is a configurable open-source GPU-like soft processor designed to accelerate workloads that fit in the SIMT paradigm [6]. Fig. 1 presents an overview of FGPU's architecture. Its main component is the Compute Unit (CU), a SIMD machine of 8 identical Processing Elements (PE0 - PE7) that can be spatially replicated up to eight times. A single CU can run up to 512 work-items (a computational kernel in OpenCL) and supports full thread-divergence, i.e., each work-item is allowed to take a different path in the control flow graph. Work-items are grouped into Wavefronts (WFs) that execute concurrently in a CU, and WFs are combined into Workgroups (WGs), which share a program counter and are assigned to a CU. FGPU is also deeply pipelined. On the software side, only standard OpenCL-API procedures are needed. Most importantly, FGPU can be artlessly scaled up to 64 processing units and is deeply configurable in terms of operations, instructions, and memory access.

Several past works have modified the FGPU to adapt it to different application domains. In [11], the authors have included new instructions along with micro-architecture and compiler enhancements to specialize FPGU for persistent deep learning, achieving 56–693x speed-up in PDL applications. MIAOW [12] is GPU-like implementation based on the AMD Southern Islands architecture and supporting its ISA. Scratch [13] extended MIAOW with automatic identification of the specific requirements of each application kernel and a tool that allows for the generation of application-specific and FPGA-implementable trimmed-down GPU-inspired architectures. Our work is the first to propose a tool that automatically generates tapeout-ready domain-specific accelerators based on GPU-like architectures and to make it publicly available.

III. GPUPlanner Framework

Our experimental investigation started from migrating the FGPU, originally designed for FGPA, to ASIC. To this end, a few changes in the architecture were necessary. As compilers for FGPA have a feature to infer memory from RTL automatically, all the memory blocks in the FGPU code were described as regular FFs. In ASIC, memory IPs are hand-instantiated instead of inferred. Thus, the first task was to clearly define intended behavior from the code and instantiate memory modules, utilizing a 65nm commercial technology.

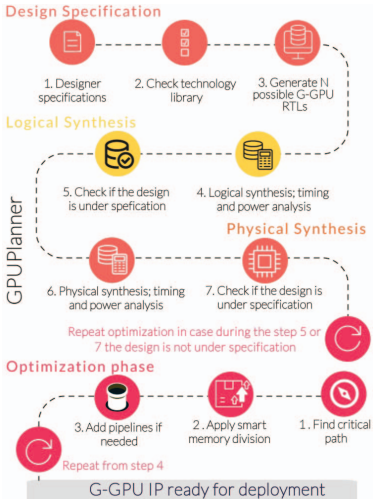

Fig. 2: GPUPlanner's G-GPU generation flow.

One of our main goals is to achieve the best PPA ratio possible from the G-GPU, exercising the maximum possible design space. The first aspect analyzed was the performance. This is done by finding the maximum operating frequency, which does not violate timing. For the logical synthesis, the value found for the standard version (without any of the optimizations done in this work) is 500MHz. The G-GPU has a similar performance across versions with different numbers of CUs because the CU itself is the bottleneck for performance in this architecture. As expected, the critical path for the version without any optimization has its starting point at a memory block. Also, the critical path was found inside the CU partition.

Larger memories display a higher delay for accessing the stored data when compared with smaller memories. This observation guides our design space exploration: dividing the memory blocks in the critical path is a valid strategy for increasing the performance of a design [14]. Memory division can be applied by diving the number of words, the size of the word, or both. This strategy requires a few alterations in the RTL code. First, the new modules have to be instantiated properly, substituting the target memories for the optimization. Second, the address or the input/output data have to be concatenated accordingly. To attain faster results, this task was fully automated in our framework.

The area of the memory blocks is not linear w.r.t. their size. In fact, two blocks of size $M \times N$ are larger and more power-hungry than a single block of size $2M \times N$ or $M \times 2N$. From the memory division alone, we are increasing the area and power. Also, a small extra logic is necessary to accommodate the addressing control. When exercising the memory division to enhance the design performance, we found cases where the critical path was not in memory blocks. For solving such timing issues, pipelines were introduced in those paths. As a result, we created an open-source tool to automatically generate G-GPU IPs, from RTL to GDSII. The flow of GPUPlanner is highlighted in Fig. 2. For starters, the designer has to define the specifications required from the G-GPU. Our architecture can be configured for CUs ranging from 1 to 8. Also, the designer

has to specify the operating frequency of the G-GPU.

After surveying the possible versions of the G-GPU for desired application scenarios, the designer can generate a specification for each scenario. Then, these specifications are contrasted with the characteristics of the technology intended to be used to create a first-order estimation of the G-GPU PPA. In this phase, there is a possibility to find several versions suitable for the given specification. Still, it also might happen that a configuration that suits the designer's requirements does not exist. However, our framework is not a static input generator. Instead, we provide a map on how to achieve a realistic PPA that might be close enough to the designer's requirements. This map is a dynamic spreadsheet, where the user input the delay of the memory blocks required for the non-optimized version of the G-GPU. Our map gives the maximum performance and which memory has to be divided or where to introduce pipelines to enhance the performance. This is an iterative process and can be repeated until the designer finds the desired performance. Thus, using our map, the designer can rapidly adapt his specification or create new versions of G-GPU. The only hard constraint in our framework is that many of the G-GPU memories have to be dual-port. Further development for single-port memories is scheduled as future work.

From a single push of a button, our framework can perform logic and physical synthesis of the list of designs. After the logic and physical synthesis, the resulting PPA is checked to guarantee it is under the initial specification. If the resulting G-GPU is out of the specifications, the designer should modify it and restart the process. In any case, the resulting layouts are ready to be integrated in a system as a tapeout-ready IP.

IV. Results and Discussion

From the exercise of the GPUPlanner, we found 12 versions worth the PPA trade-off in a general manner. These versions have 1, 2, 4, and 8 CUs. Their variants run at 500MHz, 590MHz, and 667MHz. The characteristics of each version are shown in Table I. In terms of area, the G-GPU size grows linearly with the number of CUs. The optimizations done for augmenting the performance increased the area by an average of 10%, from 500MHz to 590MHz, and 2%, from 590MHz to 667MHz. Thus, if the power consumption is not a priority, the 667MHz is a good fit for having a negligible increase in area in trade-off a better performance. These results demonstrate the potential scalability of the G-GPU architecture.

We chose four versions to perform the physical synthesis. Those are the 1CU@500MHz, 1CU@667MHz, 8CU@500MHz, and 8CU@677MHz. During this phase, the G-GPU is broken into three partitions during implementation: the CU, the general memory controller (MCTRL), and the top. The density of the CU and the MCTRL was set to 70%. Because of our floorplan strategy of breaking the design into partitions, the top has a low density of 30%. Nevertheless, breaking the design in partitions allows the designer to scale G-GPU without any extra effort. Once a CU partition is fully placed and routed, it can be implemented in versions with more than 1 CU by cloning the partition in the final floorplan of the design. Moreover, the user can create a collection of different CU layout blocks and scale the floorplan regarding the number of CUs for different application scenarios easily.

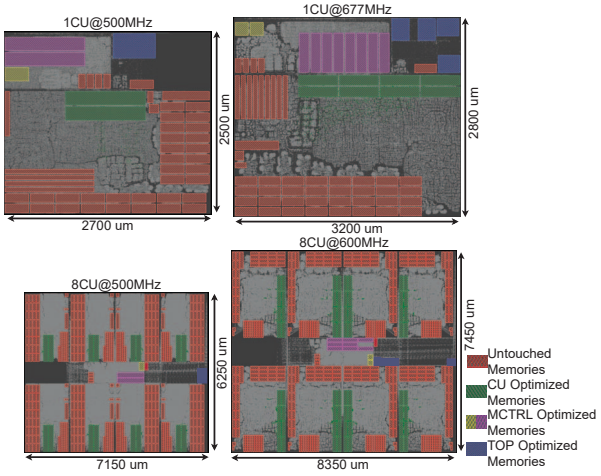

Fig. 3: Layout comparison between minimum and maximum performance of G-GPUs with 1 CU (top) and 8 CUs (bottom).

The layouts for the versions with 1 and 8 CUs are depicted in Fig. 3. Size scale in the figure only applies to the ones with the same number of CUs. The block memories divided for augmenting the performance are highlighted in green for the CU partition, yellow and pink for the MCTRL, and blue for the top. Note how different the floorplan is between the version with optimizations running at 600MHz and without optimizations running at 500MHz. Block memories have to be strategically placed in order to extract the maximum performance, hence, the differences in the floorplan. The layout of the versions 1CU@500MHz, 1CU@667MHz, 8CU@500MHz have the same performance expected from the logical synthesis (i.e., they can run at the specified clock frequency without any timing violation). However, the layout of version 8CU@667MHz can only run at 600MHz. This is explained by analyzing the floorplan of its layout (see Fig. 3). The connecting routing wires introduce a significant capacitance because of the long distance between the peripheral CUs and the general memory controller.

To fully evaluate the G-GPU as an ASIC accelerator, we compared its performance with the popular RISC-V architecture. We synthesized both architectures using the same technology used before with an operating frequency of 667MHz, the RISC-V having 32Kb memory and the G-GPU with 1/2/4/8 CUs. We chose seven micro-benchmarks from the AMD OpenCL SDK and increased their inputs up until crashing the RISC-V compiler. We further increased the input size of the G-GPU applications to make its computing units fully utilized. To compare the performance of the different-input size applications, we took a pessimistic approach for G-GPU and considered that one could increase RISC-V application input sizes by multiplying its cycle count by the G-GPU/RISC-V input size ratio. These results are shown in Fig. 4.

Our first evaluation compares raw performance between G-GPU and RISC-V for the same input sizes. For applications

TABLE I: Characteristics of 12 different GGPU solutions generated by our tool after logic synthesis in Cadence Genus.

#CU & Freq.	Total Area (mm^2)	Memory Area (mm^2)	#FF	#Comb.	#Memory	Leakage (mW)	Dynamic (W)	Total (W)
1@500MHz	4.19	2.68	119778	127826	51	4.62	1.97	2.055
2@500MHz	7.45	4.64	229171	214243	93	8.54	3.63	3.77
4@500MHz	13.84	8.56	437318	387246	177	16.07	6.88	7.14
8@500MHz	26.51	16.39	852094	714256	345	30.79	13.33	13.86
1@590MHz	4.66	3.15	120035	128894	68	4.73	2.57	2.66
2@590MHz	8.16	5.34	229172	221946	120	8.73	4.63	4.81
4@590MHz	15.03	9.72	436807	397995	224	16.41	8.70	9.02
8@590MHz	28.65	18.49	850559	737232	432	31.25	16.81	17.40
1@667MHz	4.77	3.26	120035	130802	71	4.65	2.62	2.72
2@667MHz	8.27	5.45	229172	222028	123	8.72	4.69	4.87
4@667MHz	15.15	9.83	436807	398124	227	16.43	8.75	9.07
8@667MHz	28.69	18.60	848511	730506	435	30.21	19.10	19.76

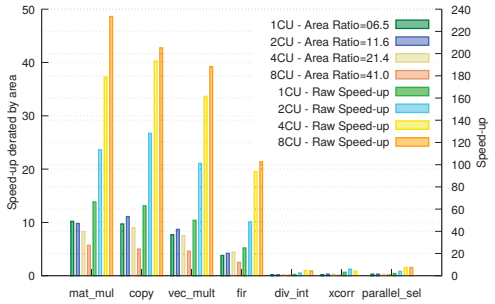

Fig. 4: Speed-up over RISC-V.

with low to no parallelism, G-GPU can be as low as only 1.2 times faster than RISC-V. As G-GPU is a domain-specific ASIC accelerator, such results are expected, once it will not be the best option for general-purpose applications. Therefore, a user interested in implementing a G-GPU as an accelerator can utilize these provided data to ponder if this type of architecture is a good fit for his system, considering only the raw speed-up.

Our second evaluation factors previously measured area into performance speed-up. We derated the previously measured speed-up by dividing the area ratio (G-GPU/RISC-V). A G-GPU with 1 CU has an area that is 6.5 times larger than the RISC-V, and it achieves the best increase in performance per area of 10.2 times the RISC-V's. On the other hand, G-GPU with 8 CUs has an area that is 41 times bigger than RISC-V's, thus achieving the best increase in performance per area of 5.7 times faster than RISC-V's. This trend happens mainly because data dependency and global memory communication limit parallelism. Thus, the provided increased processing power of a G-GPU configuration with more CUs.

We are planning to update the GPUPlanner to be able to implement the 8-CU G-GPU without performance loss. The performance problem of the layouts with 8 CUs has the possibility to be solved by replicating the general memory controller, shortening the distance between the peripheral CUs, and reducing the delay introduced by the routing wires. Also, we intend to include support of memory hierarchy and incorporate single-port memories into GPUPlanner.

V. Conclusion

Our results showed that G-GPUs are feasible domain-specific ASIC accelerator. Furthermore, when the G-GPU performance is contrasted with that of a RISC-V, it shows that our architecture has tremendous benefits for applications with high parallelism. Moreover, as GPUPlanner is an open-source framework, it gives the community the opportunity to explore the design space of GPU-like accelerators. Our work goes beyond the analysis of what constitutes a reasonable G-GPU accelerator in 65nm, as our tool can be easily extended to support other baseline GPU architectures and technologies.

Acknowledgments

This work has been partially conducted in the project "ICT programme", supported by the European Union through the European Social Fund. This work was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001, CNPq, and FAPERGS.

References

[1] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proc. of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[2] C. Mucci, L. Vanzolini, A. Lodi, A. Deledda, R. Guerrieri, F. Campi, and M. Toma, "Implementation of aes/rijndael on a dynamically reconfigurable architecture," in *2007 Design, Automation Test in Europe Conference Exhibition*, pp. 1–6, 2007.

[3] T. D. Han and T. S. Abdelrahman, "hicuda: High-level gpgpu programming," *IEEE Trans. on Parallel and Distributed Systems*, vol. 22, no. 1, pp. 78–90, 2011.

[4] P. P. Brahma, D. Wu, and Y. She, "Why deep learning works: A manifold disentanglement perspective," *IEEE Trans. on Neural Networks and Learning Systems*, vol. 27, no. 10, pp. 1997–2008, 2016.

[5] J. E. R. Condia, B. Du, M. Sonza Reorda, and L. Sterpone, "Flexgripplus: An improved GPGPU model to support reliability analysis," *Microelectronics Reliability*, vol. 109, p. 113660, 2020.

[6] M. Al Kadi, B. Janssen, and M. Huebner, "Fgpu: An simt-architecture for fpgas," in *ACM/SIGDA Int. Symp. on Field-Programmable Gate Arrays*, p. 254–263, ACM, 2016.

[7] M. Gautschi *et al.*, "Near-Threshold RISC-V Core With DSP Extensions for Scalable IoT Endpoint Devices," *IEEE Trans. on VLSI Systems*, vol. 25, no. 10, pp. 2700–2713, 2017.

[8] OpenHW Group, "Cv32e40p risc-v ip," 2016. https://github.com/openhwgroup/cv32e40p.

[9] A. Canis *et al.*, "Legup: High-level synthesis for fpga-based processor/accelerator systems," FPGA'11, p. 33–36, ACM, 2011.

[10] J. Weng, S. Liu, V. Dadu, Z. Wang, P. Shah, and T. Nowatzki, "Dsagen: Synthesizing programmable spatial accelerators," in *ACM/IEEE Int. Symp. on Computer Architecture*, pp. 268–281, 2020.

[11] R. Ma *et al.*, "Specializing fgpu for persistent deep learning," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 14, July 2021.

[12] V. Gangadhar *et al.*, "Miaow: An open source gpgpu," in *IEEE Hot Chips Symp.*, pp. 1–43, 2015.

[13] P. Duarte, P. Tomas, and G. Falcao, "Scratch: An end-to-end application-aware soft-gpgpu architecture and trimming tool," in *IEEE/ACM Int. Symp. on Microarchitecture*, p. 165–177, ACM, 2017.

[14] H. E. Sumbul, K. Vaidyanathan, Q. Zhu, F. Franchetti, and L. Pileggi, "A synthesis methodology for application-specific logic-in-memory designs," in *ACM/EDAC/IEEE Design Automation Conference*, pp. 1–6, 2015.