Designing CGRAs with Deep Reinforcement Learning

Jackson Woodruff
University of Edinburgh
J.C.Woodruff@sms.ed.ac.uk

Chris Cummins
Meta AI
cummins@fb.com

***Abstract*—CGRAs promise the performance benefits of ASICs while retaining the flexibility of FPGAs. Heterogeneous CGRAs present a key design-point that is closer to the performance on an ASIC by eliminating unnecessary hardware components. For any particular set of applications, exploring the design-space of CGRAs is a complex task: requiring time-consuming estimates of power consumption, area use, and performance. This is costly for a single architecture, but when we wish to explore multiple sets of architectural parameters for a CGRA, the cost balloons — many architectures must be evaluated for each possible CGRA specification. We introduce RL-CGRA, which uses reinforcement learning to enable fast placements. RL-CGRA enables fast exploration of a wide design-space of CGRA parameters by eliminating the cost of simulated annealing for each architecture.**

***Index Terms*—regular expressions, accelerator, compiler**

I. Introduction

With the end of Denard scaling, extracting more performance requires overcoming the accelerator wall [1]. This wall, and the projections for more dark silicon [2], present the perfect window for specialized hardware accelerators [3]. However, over-specialization of hardware accelerators limits their use-cases [4], [5] which is costly [6], [7]. ASICs are extremely high performance [1], but present significant flexibility challenges [8], [9].

Domain-specific Coarse-Grained Reconfigurable Architectuers (CGRAs) promise to provide near-ASIC performance [10] and resolve flexibility challenges [11]. The greatest performance comes from the most specialized designs [12], forming a critical part of the most low-power accelerators [5], [13]. Hundreds of domain-specific CGRAs [11] have been designed for usecases ranging from low-power [5], [14] to high-performance [15].

Fleets of domain-specific CGRAs have the potential to vastly increase applicability of FPGAs in the cloud [16]. However, the exploration of the CGRA design-space is complex, time-consuming, and limited [17]. Work to enable high-level CGRA-design exploration has focused on fast estimation of clock frequencies [18], energy, and area [19]. However, generating performant architectures from the high-level specifications, a critical aspect for finding high-performance architectures [17], is still a slow process; designing PEs involves complex tradeoffs [20] and simulated annealers used by existing frameworks [16], [21] are slow, limiting high-level parameter exploration to tens of designs [12] or out-of-context analyses for single PEs [20]. Soling this problem requires fast evaluation of potential CGRA architectures.

We propose RL-CGRA, a placer for design-space exploration of heterogeneous CGRAs. RL-CGRA uses experience from placing previous designs to enable it to place new designs quickly, out-preforming simulated annealing-based techniques that are not able to learn from past experience, and enabling far greater sample efficiency. Using learned experience allows RL-CGRA to efficiently explore large spaces of architectural parameters.

Although existing equation-based techniques [12] can be used to give near-optimal designs with a single set of applications, they do not work when the compiler can rewrite the application for the hardware at-hand. When the compiler is aware of the heterogeneity of the underlying hardware, and selects different instructions depending on the available hardware, these simple equations are no longer applicable, as they change depending on what hardware is provisioned. RL-CGRA enables design-space exploration that uses feedback from the compiler to determine optimal placements.

Enabling this design-space exploration for CGRAs critically raises the level of abstraction that can be used to design CGRAs. Using RL-CGRA, architects can ignore the challenging decisions around picking which PEs they wish to include in their designs, and rely on distribution-based generation of plausible architectures and quantitative selection of the most suitable architecture for the task.

In summary, this paper introduces:

- RL-CGRA, the first CGRA placement tool designed to enable large-scale design-space exploration.
- The first RL-placement strategy that incorporates application-performance feedback.

II. Background

Exploring a wide range of architectural parameters on CGRAs has been a key limiter in CGRA design, accounting for a large fraction of the design time in such architectures.

A. Heterogeneous CGRA Design Pattern

Heterogeneous CGRAs are designed to accelerate sets of loops in applications. Their fast reconfiguration time, orders of magnitude faster than FPGAs [22], gives them a significant

Appeared in DoSSA@ISCA 2023.

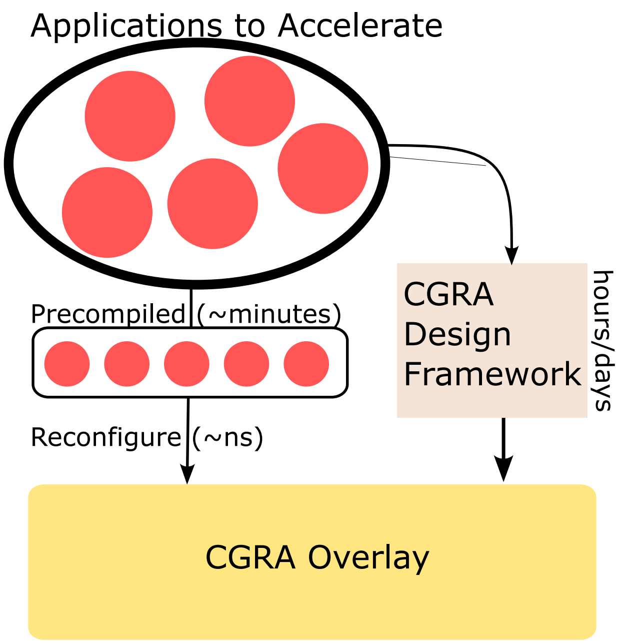

Fig. 1. CGRA-as-overlay designflow.

advantage over FPGA-based systems which cannot context-switch at the same rate as CPUs [23].

A typcial CGRA design sequence invovles two steps: generating the CGRA design, which can take days due to the cost of synthesis [24], and reconfiguring for each application, which takes ns. Figure 1 shows this sequence.

B. The Limits of CGRA Design

High-level CGRA-design is bottlenecked by several key stages. Generally, the set of operations that a CGRA can support is derived from the applications it is intended to support. However, with hardware-aware compilers, simple estimates of the hardware do not suffice as the compiler changes it's behaviour based on the hardware available. Figure 2 shows an example of this, where we wish to design a CGRA to support two different functions, `3 * x + y` and `x << 1`. A naive equation-based hardware-design would create the CGRA in figure (a), but the CGRA in (b) will be more efficient for these functions.

C. The Promise of Design-Space Exploration

Design-space exploration can be used to solve this problem by exploring a large space of combinations of possible designs. The limits of this design-space exploration come in the placement step of operations on the CGRA.

D. Addressing the Challenges of Placement with Reinforcement Learning

To evaluate possible CGRAs with different operations, we must place the operations on the CGRA. However, placing operations on a CGRA efficiently typically uses simulated annealing [25] are slow, and do not learn from previous failures. Further, CGRA compilations are slow — meaning that running many design-space explorations takes significantly longer than would otherwise be required. By leveraging reinforcement learning, we can effectively *learn* to efficiently generate placements for generic architectural parameters.

1) Simulated Annealing Approach: An Overview: In simulated annealing, placements are randomly initialized. The simulated annealer makes random changes to the architecture, re-evaluating the quality of the new placement with these changes — if the changes not too detrimental, they are kept. To determine whether changes are too detrimental, a parameter called the *temperature* is used. The temperature dictates how big of a local minimum the algorithm can overcome. With each iteration, the temperature is reduced, and when the placement stops improving, the simulated annealer terminates.

This process produces good results, but must start from scratch every time.

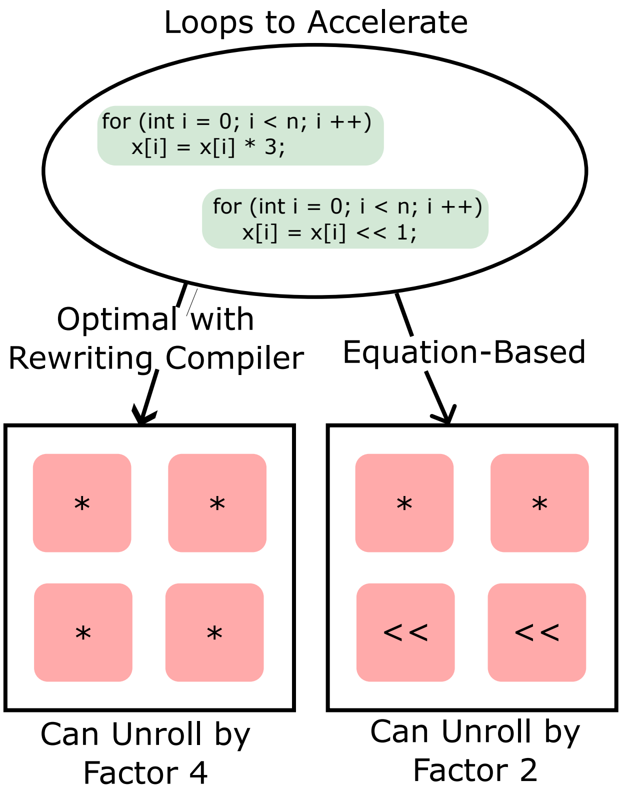

Fig. 2. An example where design-space exploration produces a more optimal CGRA than simple equations for the operations on a CGRA.

III. Reinforcement Learning Placement

We use a placement strategy based on circuit placement techniques [26], where we ask the agent to place different operations on the CGRA. Compared to traditional circuit placement strategies [27], CGRAs offer an opportunity to use more meaningful rewards: while traditional circuit placement strategies typically rely on write-length as reward to make evaluation fast, applications can be benchmarked quickly using a CGRA design. Further, placing operations on a CGRA is more closely tied to the applications that must run on that

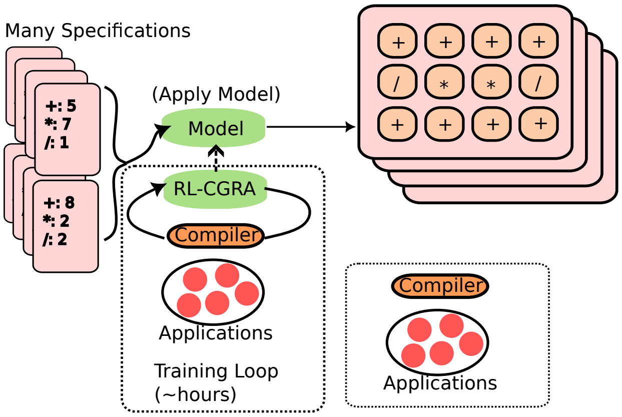

Fig. 3. Workflow using RL-CGRA to enable design-space exploration. The top half of this diagram shows a traditional simulated-annealing-based placement strategy. The bottom half of this diagram shows how RL-CGRA enables exploration of a large heterogeneous design-space.

CGRA, and the parallelism (or not) inherent in those — rather than on the wire lengths.

We instead design an agent that uses application performance as a feedback mechanism. For a single architecture, we do the following:

1) Produce an architecture using RL-CGRA.
2) Evaluate that architecture using the OpenCGRA compiler to determine how good (or not) it was.

This flow is shown in Figure 3.

IV. RL-CGRA

RL-CGRA places operations with a CGRA-like grid. To enable efficient design-space exploration, RL-CGRA uses reinforcement learning to learn where operations should be placed.

A. Environment Design

We ask out agent to learn placement strategies for individual operations on a CGRA. These placement strategies are learned for a particular set of applications, but are independent of the architecture parameters.

For each available operation, we generate a map of probabilities for that is used to place each operation. This agent is shown in figure 4. We invoke the agent for each operator that should be provided in the hardware.

1) Action Space: For an MxN CGRA, the action-space of RL-CGRA is MxN placement candidates. Each action identifies a placement location on the CGRA. We use action *masking* to reduce the action-space by avoiding invalid placements (e.g. where the operation in question has already been placed).

2) Observation Space: The observation-space uses the CGRA compiler to provide feedback on the architectural alternatives it presents. For each candidate-generated architecture, we run the OpenCGRA compiler [21]. This allows us to compute the performance (taken through the initialization-interval) of each loop on the candidate accelerator architecture. The observation is computed from this as the mean performance of each loop on the CGRA.

3) Rewards: For the reward, we use the average initialization interval of each loop — this represents an accurate the performance characteristics of each loop on each architecture.

B. Agent Design

RL-CGRA uses a MLP, with two hidden layers with 256 elements, trained using PPO with $\gamma = 1.0$ and lr $= 0.001$. The choice of $\gamma = 1.0$ reflects the stateless nature of the problem, where each step is equally likely to contribute to the final reward. Similarly, we have used the highest learning rate for which we found RL-CGRA exhibited stability in order to minimize the number of training steps required before feasibility.

C. Training Strategy

During training, we ask RL-CGRA to generate placements for many architectures. We evaluate those placements on their high-level performance characteristics of the applications running on the CGRA.

This, in addition to the resources required for the placement selected, is used to enable design-space exploration.

The steps taken by RL-CGRA are as follows:

1) While Training:
   a) Generate a candidate within the design-space that should be evaluated.
   b) Use RL-CGRA to place that candidate
   c) Evaluate that architecture using OpenCGRA.
   d) Store training data, every 100 iterations, update the RL model.

This training strategy uses the randomness in the RL model to explore the space of placements.

D. Exploration-Exploitation Tradeoff

Critically, it is easy to determine a baseline using a simulated annealer for a single architecture — we can easily tell whether RL-CGRA requires more training. However, as we will see, pathological cases exist, where the training continues for a long period of time, but does not produce outputs equalling the performance of a simulated annealer. In these cases, we propose a secondary termination methodology, by terminating training when enough epochs have been trained over.

In this work, we use the break-even point with the simulated annealer, and 100 epochs as tradeoff points, and 100 epochs represents a significant amount of training time.

E. Simulated Annealer Baseline

We write a simulated annealer to address the same problem. This is the technique employed by existing CGRA design-space exploration techniques. The simulated annealer is written as follows. For a typical architecture, it requires 100 evaluations to termiante.

1) Produce a random initial placement
2) While the architecture has improved recently:
   a) Randomly swap two operations

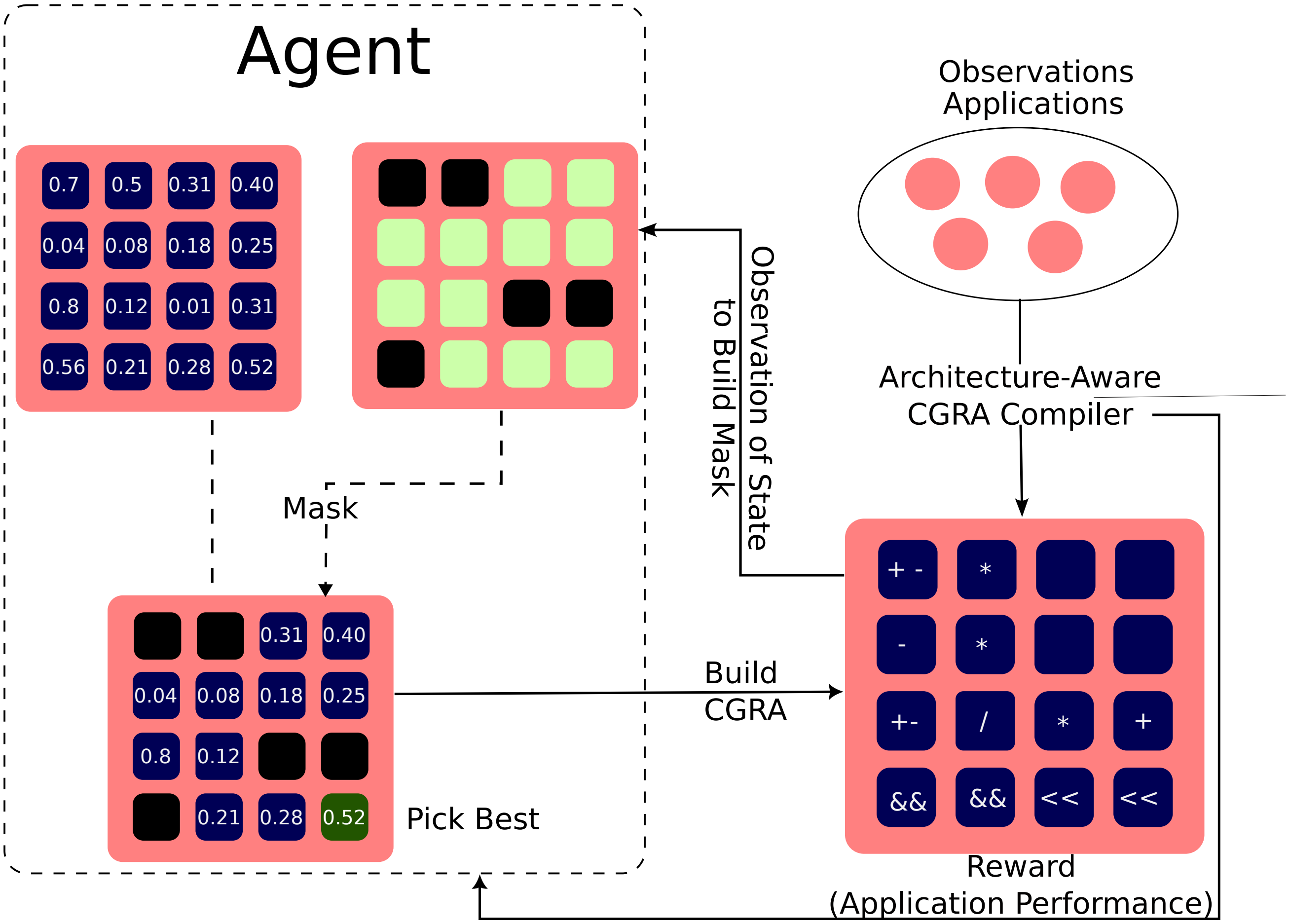
Agent
0.7
0.5
0.31
0.40
0.04
0.08
0.18
0.25
0.8
0.12
0.01
0.31
0.56
0.21
0.28
0.52
Mask
Pick Best
Observation of State to Build Mask
Observations Applications
Architecture-Aware CGRA Compiler
Build CGRA
+ -
*
-
*
+-
/
*
+
&&
&&
<<
<<
Reward (Application Performance)

Fig. 4. The structure of the RL-CGRA Agent. We use the agent to place each operation in the set of available operations, and from the constructed CGRA can determine a reward to train the agent with.

b) Evaluate the performance of this architecture to see if it has improved

Simulated annealing can be used to generate CGRA designs. As input, it requires a single set of PE allocations, prohibiting design-space exploration on this scale due to the costs of running a single simulated annealer.

1) The REVAMP Equations: Using a simulated annealer requires a very small set of architectures to explore. The state of the art for choosing which architectures should be explored is given in REVAMP [12], a set of of equations that can be used to give an optimal CGRA architecture for a set of applications. For some operation c that occupies a fraction f of the operations within the applications to be accelerated, they use the equation:

$$\frac{N_c}{N} \approx f$$

Where N_c is the number of PEs supporting operation c, and N is the total number of PEs. These equations can be used as inputs to a simulated annealer — we will evaluate these in section VI

V. Design-Space Exploration

RL-CGRA enables architectural exploration. In contrast to traditional frameworks, which generate architectures from high-level descriptions of the ALUs that should be used, RL-CGRA enables *exploration* of these architecture parameter efficiently.

A. Distribution-Based Generation

For any given set of loops we wish to accelerate on a CGRA, $\mathcal{A}$, there is some minimum set of operations O which must be supported to actually *run* any of the applications.

1) Exponential Distribution: The exponential distribution is characterized by a parameter λ, which controls the mean and variance. The probability density function is given by

$$f(x) = 1 - e^{-\lambda x} x \geq 0$$

Given a target number of operations, we can pick λ so that the mean (given by $1/\lambda$) results in architectures with right scale of resource usage.

Benchmark Suite	Category
LivermoreC	Scientific Benchmark Suite
DarkNet	Machine Learning Library
freeimage	Image Processing Library
ffmpeg	Image Processing Library
BZip2	Compression Program

TABLE I
BENCHMARKS ARE 10 LOOPS RANDOMLY SELECTED FROM THESE PROGRAMS AND LIBRARIES.

2) Manual Selection: We use distributions to lower the load on the CGRA-designer, as automated parameter selection is particularly load-reducing. However, RL-CGRA can just as easily support a human-generated list of architectures. The key here is that RL-CGRA does not have to be trained blind — during training it can see all the architectures it will need to be evaluated on.

VI. EVALUATION

We evaluate the ability of RL-CGRA to learn relevant patters for design-placement on CGRAs. We explore the quality of these placements compared to those generated by simulated annealing approaches, and we explore the number of compiler evaluations required to achieve these placements.

A. Setup

We construct CGRAs for five sets of example loops, taken from the benchmark suites shown in table I. This style of library-function acceleration suits CGRAs well, as they can be designed for a set of libraries that are relevant to a program, and than quickly reconfigured to support each loop in question.

1) Training Setup: We use a learning rate of 0.005, and an episode size of 150 architectures. We use RLlib to distribute the training across a single machine. The training is bottlenecked by the CPU time required to obtain samples, so was run on an 80-core machine without using a GPU.

B. Comparison to Simulated Annealing: Quality of Placements

We demonstrate that RL-CGRA generates high-quality placements, of comparable quality to simulated annealing placements. Figure 5 shows the training curves for each of the benchmarks. We can see that over 100 epochs, all but one of the benchmarks achieves within 10% the performance of the simulated annealer placements, using a single-shot placement.

We can also see that in the case of two of the benchmarks (DarkNet and LivermoreC), RL-CGRA achieves better performance than the simulated annealing placement.

C. Comparison to Simulated Annealing: Exploration Time

The key advantage of RL-CGRA is that it enables design-space exploration for sets of applications without costly re-evaluation using a simulated annealer over-and-over again. This section explores two key aspects of this: the training curves for RL-CGRA that show how quickly it is able to learn placements (section VI-C1) and the required size of design-spaces that are to be explored to take advantage of this (section VI-C2).

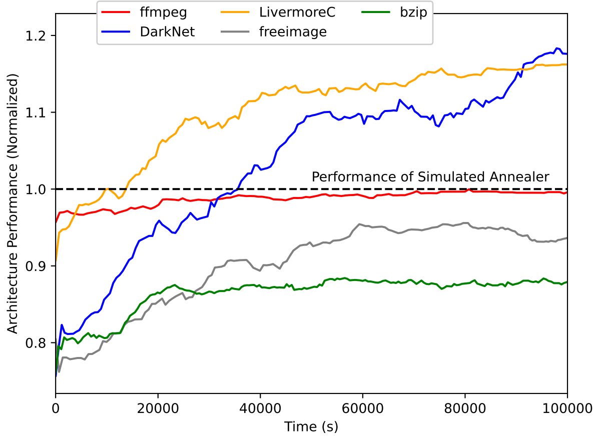

Fig. 5. Training curves for each benchmark. For each benchmark, we show the training curve of RL-CGRA and the results of a simulated annealer on the architecture generated using the REVAMP equations. Performance is measured relative to the performance of the baseline simulated annealer.

1) Training Curves for RL-CGRA: As discussed in section IV-D, we use the training curve from RL-CGRA to determine where to stop exploring and start exploiting our model for fast placements.

Where performance plateaus is dependent on several factors, including the applications we wish to map in-particular. Most critically, it is related to the size of the action space (size of the CGRA) and the size of the observation space (number of operations). Figure 5 shows this.

2) Time Taken to Explore Design Spaces: In this section, we analyse the time taken by RL-CGRA to explore various design spaces. RL-CGRA is most useful for large-scale design-space explorations, where the costs of running many simulated annealers would be prohibitive. Table II shows this — comparing the number of evaluations in the simulated annealer and to train RL-CGRA. From this, we can see that for LivermoreC and DarkNet, RL-CGRA learned the environment effectively — enabling efficient design-space explorations with little overhead.

In the cases of FreeImage and Bzip2, RL-CGRA was efficient, learning nearly enough information to reach the standard of simulated annealing placement. In these cases, adjustments to the termination conditions could yield less optimal, but more efficient agents.

In the case of BZip2, RL-CGRA fails to learn an effective placement strategy. The lack of progress in this case suggests that RL-CGRA is not a good tool to use to enable a design-space exploration of the BZip2 benchmark.

VII. RELATED WORK

A huge number of CGRA design frameworks have been developed [12], [16], [21], [24], [28]–[32]. These frameworks provide (to varying degrees) the required infrastructure around CGRA design, taking applications and producing verilog and compilers for the generated architectures.

Benchmark	Number of Evaluations Simulated Annealer	Number of Evaluations RL-CGRA
LivermoreC	3500	35
FreeImage	15,000	1500
FFMpeg	15,000	1500
DarkNet	450	5
BZip2	15,000	1500

TABLE II

Number of evaluations required by RL-CGRA compared to simulated annealing for various sizes of inputs, and size of DSE required to break-even compared to simulated annealing approaches, which take 100 evaluations on average.

A. Hetergeneous CGRAs

Heterogeneous CGRAs are common in a wide range of applications: neural networks [33], [34], scientific computing [5], [35], and ultra-low power computing [13].

A number of studes have explored PE design: all from the perspective of a single PE [20]. Various projects emply PE heterogeneity but do not explore how best to obtain it [5], [13].

Tools such as Radish [36] explore the use of design-space exploration algorithms for CGRA generation. Melchert et al. [20] use compiler-generated repeated sub-graphs to enable PE DSE.

VIII. Conclusion

RL-CGRA presents the first higher-level CGRA design framework enabling the exploration of large CGRA design-spaces. RL-CGRA uses reinforcement learning to enable the operation-placement phase of CGRA design to happen quickly and efficiently for large-scale design-space explorations in CGRA design.

RL-CGRA is able to integrate application-performance metrics into its model, enabling it to quickly find effective placements by learning from previous placements using metrics that are directly relevant to the end placement.

References

[1] A. Fuchs and D. Wentzlaff, “The accelerator wall: Limits of chip specialization,” in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, IEEE, 2 2019.

[2] “International roadmap for devices and systems 2020 update: More moore,” 2020.

[3] M. B. Taylor, “Is dark silicon useful?,” ACM Press, 6 2012.

[4] G. Mathew, C. Parnin, and K. T. Stolee, “Slacc,” ACM, 6 2020.

[5] Z. Ebrahimi and A. Kumar, “BioCare: An energy-efficient CGRA for bio-signfal processing at the edge,” *ISCASS*, 2021.

[6] M. Khazraee, I. Magaki, L. Vega Gutierrez, and M. Taylor, “ASIC clouds: Specializing the datacenter,” *IEEE Micro*, pp. 1–1, 2017.

[7] E. Brunvand, D. Kline, and A. K. Jones, “Dark silicon considered harmful: A case for truly green computing,” *International Green and Sustainable Computing Conference*, 2018.

[8] T. Nowatzki and K. Sankaralingam, “Analyzing behavior specialized acceleration,” in *the Twenty-First International Conference*, ACM Press, 4 2016.

[9] J. Woodruff, J. Armengol-Estapé, S. Ainsworth, and M. F. P. O’Boyle, “Bind the gap: Compiling real software to hardware FFT accelerators,” *PLDI*, 2022.

[10] B. W. Denkinger, M. Quiros-Peon, M. Konijnenburg, D. Atienza, and F. Catthoor, “VWR2A: A very-wide-register reconfigurable-array architecture for low-power embedded devices,” *DAC*, 2022.

[11] L. Liu, J. Zhu, Z. Li, Y. Lu, Y. Deng, J. Han, S. Yin, and S. Wei, “A survey of coarse-grained reconfigurable architecture and design,” *ACM Computing Surveys*, vol. 52, pp. 1–39, 10 2019.

[12] T. K. Bandara, D. Wijerathne, T. Mitra, and L.-S. Peh, “REVAMP: A systematic framework for heterogeneous CGRA realization,” *ASPLOS*, 2022.

[13] G. Gobieski, A. O. Atli, K. Mai, B. Lucia, and N. Beckmann, “Snafu: An ultra-low-power, energy-minimal CGRA-generation framework and architecture,” *ISCA*, 2021.

[14] W. Byun, M. Je, and J.-H. Kim, “An energy-efficient domain-specific reconfigurable array processor with heterogeneous pes for wearable brain-computer interface socs,” *Transactions on Circuits and Systems*, 2022.

[15] “Accelerated computing with a reconfigurable dataflow architecture,” 2021. Available at https://sambanova.ai/wp-content/uploads/2021/06/SambaNova_RDA_Whitepaper_English.pdf.

[16] S. A. Chin, N. Sakamoto, A. Rui, J. Zhao, J. H. Kim, Y. Hara-Azumi, and J. Anderson, “Cgra-me: A unified framework for cgra modelling and exploration,” in *2017 IEEE 28th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, IEEE, 7 2017.

[17] B. Mei, A. Lambrechts, J.-Y. Mignolet, D. Verkest, and R. Lauwereins, “Architecture exploration for a reconfigurable architecture template,” *IEEE Design and Test of Computers*, 2005.

[18] D. L. Wolf, C. Spang, and C. Hochberger, “Towards purposeful design space exploration of heterogeneous CGRAs: Clock frequency estimation,” *DAC*, 2020.

[19] M. Wijtvliet, H. Corporaal, and A. Kumar, “CGRA-EAM — rapid energy and area estimation for coarse-grained reconfigurable architectures,” *ACM Transactions on Reconfigurable Technology and Systems*, vol. 14, pp. 1–28, 2021.

[20] J. Melchert, K. Feng, C. Donovick, R. Daly, C. Barrett, M. Horowitz, P. Hanrahan, and P. Raina, “Automated design space exploration of CGRA processing element architectures using frequent subgraph analysis,” *CoRR*, 2021.

[21] C. Tan, N. B. Agostini, J. Zhang, M. Minutoli, V. G. Castellana, C. Xie, T. Geng, A. Li, K. Barker, and A. Tumeo, “Opencgra: Democratizing coarse-grained reconfigurable arrays,” in *2021 IEEE 32nd International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, IEEE, 7 2021.

[22] S. Liu, J. Weng, D. Kupsh, A. Sohrabizadeh, Z. Wang, L. Guo, J. Liu, M. Zhulin, R. Mani, L. Zhang, J. Cong, and T. Nowatzki, “OverGen: Improving FPGA usability through domain-specific overlay generation,” *Micro*, 2022.

[23] E. Rossi, M. Damschen, L. Bauer, G. Buttazzo, and J. Henkel, “Preemptions of the partial reconfiguration process to enable real-time computing with FPGAs,” *ACM Transactions on Reconfigurable Technology and Systems*, vol. 11, no. 2, 2018.

[24] K. Koul, J. Melchert, K. Sreedhar, L. Truong, G. Nyengele, K. Zhang, Q. Liu, J. Setter, P.-H. Chen, Y. Mei, M. Strange, R. Daly, C. Donovick, A. Carsello, T. Kong, K. Feng, D. Huff, A. Nayak, R. Setaluri, J. Thomas, N. Bhagdikar, D. Durst, Z. Myers, N. Tsiskaridze, S. Richardson, R. Bahr, K. Fatahalian, P. Hanrahan, C. Barrett, M. Horowitz, C. Torng, F. Kjolstad, and P. Raina, “AHA: An agile approach to the design of coarse-grained reconfigurable accelerators and compilers,” *TECS*, 2022.

[25] C. Tan, C. Xie, A. Li, K. J. Barker, and A. Tumeo, “Aurora: Automated refinement of coarse-grained reconfigurable accelerators,” in *2021 Design, Automation and Test in Europe Conference and Exhibition (DATE)*, IEEE, 2 2021.

[26] Z. Jiang, E. Sonhori, S. Wang, A. Goldie, A. Mirhoseini, J. Jiang, Y.-J. Lee, and D. Z. Pan, “Delving into macro placement with reinforcement learning,” *MLCAD*, 2021.

[27] M. Elgammal, K. E. Murray, and V. Betz, “Learn to place: FPGA placement using reinforcement learning and directed moves,” *FPT*, 2020.

[28] R. Liu, S. Canturk, F. Wenkel, S. McGuire, E. Wang, A. Little, L. O'Bray, M. Perlmutter, B. Rieck, M. Hirn, G. Wolf, and L. Rampasek, "Taxonomy of benchmarks in graph representation learning," *CoRR*, 2022.

[29] C. Liu, H.-C. Ng, and H. K.-H. So, "Quickdough: A rapid FPGA loop accelerator design framework using soft CGRA overlay," *FPT*, 2015.

[30] L. Silva, M. Canesche, R. Ferreira, and J. A. Nacif, "Hpcgra - an orthogonal designed cgra generator for high performance spatial accelerators," in *XXI Simpósio em Sistemas Computacionais de Alto Desempenho*, Sociedade Brasileira de Computação, 10 2020.

[31] Y. Guo and G. Luo, "Pillars: An integrated CGRA design framework," *WOSET*, 2020.

[32] A. Podobas, K. Sano, and S. Matsuoka, "A template-based framework for exploring coarse-grained reconfigurable architectures," *ASAP*, 2020.

[33] T. Geng, C. Wu, C. Tan, B. Fang, A. Li, and M. Herbordt, "CQNN: a CGRA-based QNN framework," *HPEC*, 2020.

[34] J. Lee and J. Lee, "NP-CGRA: Extending CGRAs for efficient processing of light-weight deep neural networks," *DATE*, 2021.

[35] G. Charitopoulos, I. Papaefstathiou, and D. N. Pnevmatikatos, "Creating customized CGRAs for scientific applications," *Electronics*, vol. 10, 2021.

[36] M. Willsey, V. T. Lee, A. Cheung, R. Bodik, and L. Ceze, "Iterative search for reconfigurable accelerator blocks with a compiler in the loop," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, pp. 407–418, 3 2019.