Energy Efficient Acceleration of Floating Point Applications onto CGRA

Satyajit Das, Rohit Prasad, Kevin Martin, Philippe Coussy

Energy Efficient Acceleration of Floating Point Applications onto CGRA

Satyajit Das*§, Rohit Prasad*, Kevin J. M. Martin*, and Philippe Coussy*
*Univ. Bretagne-Sud, UMR 6285, Lab-STICC, F-56100 Lorient, France, [firstname].[lastname]@univ-ubs.fr,
§Department of Computer Science and Engineering, IIT Palakkad, India, satyajitdas@iitkpd.ac.in

***Abstract*—In this paper, we propose a novel CGRA architecture and associated compilation flow supporting both integer and floating-point computations for energy efficient acceleration of DSP applications. Experimental results show that the proposed accelerator achieves a maximum of 4.61× speed-up compared to a DSP optimized, ultra low power RISC-V based CPU while executing seizure detection, a representative of wide range of EEG signal processing applications with an area overhead of 1.9×. The proposed CGRA achieves a maximum of 6.5× energy efficiency compared to the CPU.**

I. Introduction

The simplicity of Coarse Grained Reconfigurable Architectures (CGRA) with regular organization of Processing Elements (PE) helps to exploit the instruction level parallelism making them serious candidate for energy efficient accelerators [5]. Modern applications rely on many floating point operations, which are performed in several cycles to not penalize the frequency of the whole architecture. Hence, supporting only integer operations does not cover all the computational needs of an advanced and subsequent applications.

Due to the area overhead induced by floating point capability, typical CGRAs become less attractive if all the PEs contain floating point computational units, which is stressed by the results shown in this paper. The heterogeneity in the PEs often fails to transport data synchronously to the PEs containing floating point units, disturbing parallel float computations resulting in a performance bottleneck. Transporting data synchronously to the processing elements can be guaranteed by decoupling address generation of the data structures from the computation flow. In this context, the contributions of this paper are: noitemsep

- a novel CGRA architecture employing IEEE 754 compliant floating point operations units. The PEs of the CGRA contain a Flexible Address Generation Unit (FAGU) which decouples addresses generation from the computation flow for increasing effective parallelism between the floating point operations at instruction level.
- the associated compilation flow to efficiently exploit parallelism between the floating point operations at instruction level leveraging static scheduling.
- study on EEG application running in the proposed CGRA. Results are compared with the execution in DSP optimized, ultra low-power RISC-V CPU [6] for performance and energy efficiency.

The rest of this paper is organized as follows. Section II discusses the state of the art in the context of floating point acceleration in CGRAs. It also motivates the need of new architecture and compilation flow to support energy efficient floating point application acceleration leveraging CGRA based architecture. Section III presents the proposed architecture and associated compilation flow. The experimental results are presented in the Section IV. Finally, the paper is concluded in Section V.

II. Background and Motivation

Due to significant area overhead for the floating point computations ($\approx 25\%$ of a PE area in our case), employing floating point unit (FPU) in every PE of a CGRA loses its interest. While employing dedicated floating point computational units, CGRAs like [11] use heterogeneous PEs. However, heterogeneity in the PEs results increasing the minimum schedule length where multi-cycle operations like FP computations are involved. This limitation of floating point execution is incurred by *address generation* in the computation flow. Inconsistency in address generations penalizes the effective spatial parallelism in multiple floating-point computations when temporal parallelism (pipelining) cannot be exploited. Typical CGRAs like ADRES [1], Morphosys [8], IMAGINE [7] which support integer-based applications only do not possess dedicated address generation unit (AGU). In these cases, the addresses are part of the instruction and they are stored into the instruction memory as look up tables. This scheme results in increased number of instructions which adds to the energy consumption. For better area and energy efficiency, CGRA like IPA [4] performs address calculation in software (i.e. address generation is a part of the computation flow). Operations for address generation are mapped onto the PEs. For integer and fixed point based compute and control intensive applications, such approach has been proven to be quite efficient. However, for applications with multi-cycle floating point operations, the software based address calculation causes serious performance bottleneck due to asynchronous data-arrival to the PEs with FPU. Asynchronous data access increases the minimum schedule length of a DFG resulting in longer execution time.

Fig. 1 elaborates the minimum schedule length problem due to asynchronous data arrival. In the Data Flow Graphs ($DFGs$) operations and data dependencies are represented by nodes and edges respectively. Weights of the edges correspond to the execution latency of source operation nodes. Two

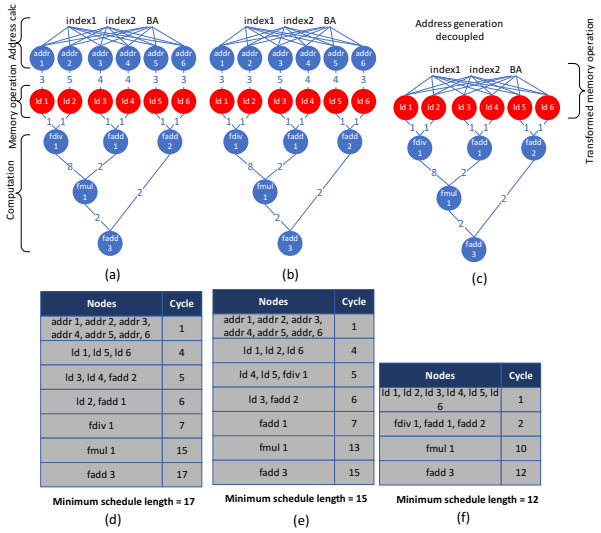

Fig. 1: (a) Sample DFG with software based address generation; (b) The same DFG with different schedule length; (c) DFG with decoupled address generation reducing the schedule length; (d), (e), (f) ASAP schedule of the DFGs (a), (b), (c) respectively

different scenarios of software based address calculation are presented in the first two sub-figures. As Soon As Possible (ASAP) scheduling of the two DFGs in Fig. 1 (a) and (b) reveals that the address calculations are fully part of the critical path of the DFGs. Two different minimum schedule length suggests that software based address generation may take different numbers of cycles depending on the location of the index variable. For instance, early scheduling of the node $load2$ in the critical path helps to achieve better minimum schedule length of 15 in Fig. 1 (e) compared to 17 in (d). Scheduling of *load* nodes determines the data arrival to the computational units. To assure synchronous data arrival to the PEs, we propose to decouple address calculation from the computation flow as shown in the Fig. 1 (c) & (f), which achieves the best schedule length.

Hardware address generation is a well defined problem in modern DSP processors. Shami et al [10] propose a versatile address generation scheme for stream data processing in CGRAs. In this paper, we propose to use a flexible address generation unit to complement the decoupling of address generation from the computation flow to increase effective parallelism in FP computations.

III. Proposed Architecture and Compilation Flow

A. Architecture

The proposed CGRA is loosely coupled with a host CPU (see Fig. 2). The CGRA subsystem consists of a Heterogeneous Processing Element Array (HPEA), a context memory and a DMA controller. The CGRA shares data with the CPU through a tightly coupled data memory (TCDM) with multiple banks and a logarithmic interconnect.

The Context Memory stores configuration data (instruction and constants) for each PE in the PE Array which are broadcast

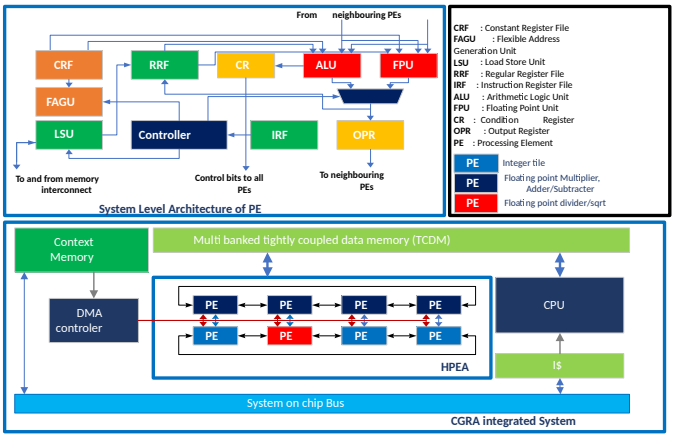

Fig. 2: CGRA integrated System and PE architecture

by the DMA controller. The configuration data is stored in the respective memory segments of the PE prior to starting of the execution. The HPEA consists of heterogeneous PEs, connected with a mesh torus network for data sharing and a bus network for context broadcast. To support floating point operations, we have employed floating point multiplier, adder/subtractor and divider/square-root units inside the PEs as shown in the Fig. 2. In addition to the 32-bit ALU and a Load-Store Unit (LSU), each PE consists of a Constant Register File (CRF) which stores the immediate values of the instructions, a Regular Register File (RRF) and Output Register (OPR) to store temporary variables. The Controller in the PE fetches instructions from the Instruction Register File (IRF). A Flexible Address Generation Unit (FAGU) is implemented in each PE, which facilitates random access to the tightly coupled data memory for the LSUs.

B. Compilation flow

In the compilation flow, the CGRA is modeled by a bipartite directed graph with two types of nodes: operator and register, in which temporal aspect is implicitly represented by connections from registers to operators. Applications are modeled as CDFGs. CDFG is composed of a Control Flow Graph (CFG) and a set of basic blocks (BB) represented by DFGs. The compilation flow finds the mapping of DFGs onto the CGRA graph. Steps involved in the proposed compilation flow are discussed chronologically below.

1) BB selection and multi-cycle operation serialization: This step selects one basic block from the CDFG using forward traversal-Breadth First Search (BFS) [2], and identifies multi-cycle operations (floating point computations). The operation nodes are then transformed by adding dummy nodes equal to the number of the total cycles needed to perform the operation. As an example, in the sample DFG of the Fig. 3 (a) node 4 is a float operation that takes 3 cycles to compute. This step transforms the DFG according to the Fig. 3 (b) (adding $4'$ and $4''$ sequentially). The transformed DFG is then passed to the scheduling and binding step.

2) Scheduling and binding: The proposed approach uses a backward traversal list scheduling algorithm to schedule the

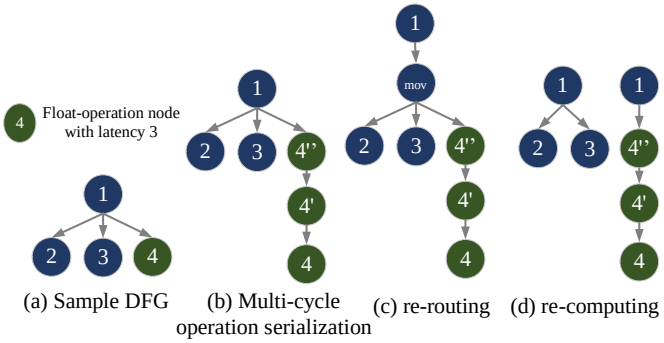

Fig. 3: Graph transformations used in the compilation flow

DFG nodes of each basic block. It relies on a heuristic in which the schedulable operations are listed by priority order. Priority of the nodes depends on their weight followed by mobility and number of fan-outs. The weight of any node is the execution latency in number of cycles.

The binding algorithm follows an incremental version of the Levi's [9] subgraph matching algorithm. Since this step is exact in nature, the algorithm we propose adds the newly scheduled operation node except the float operations and its associated data node to the sub-graph composed of already scheduled and bound nodes. Only the previous set of solutions that have been kept are used to find every possibility to add this couple of nodes without considering the non-yet scheduled nodes.

For the floating point operation binding, the algorithm finds the binding solution only for the first operation node among all the dummy nodes (i.e. node 4 in Fig. 3 (b)). The tool assigns same solution to rest of the transformed nodes (node $4'$, $4''$ in Fig. 3 (b)). This way, the binding of the floating point operation satisfies multi-cycle operations onto the CGRA with static scheduling.

3) Dynamic Transformation: If binding step does not find any solution for the current node, the graph is transformed on the fly. The transformation either reroutes the operation (Fig. 3 (c)) or distributes the fan-outs (Fig. 3 (d)) to satisfy available connectivity constraints of the CGRA node.

4) Stochastic Pruning: Exactness of the placement approach leads to very large number of partial mappings. It grows exponentially if not pruned. Hence, we use the stochastic pruning approach described in [3].

After mapping all the nodes in the current basic block, the flow selects another basic block for DFG transformation and mapping, and apply the flow described in this section. Once, all the basic blocks are mapped, the compiler generates an assembly containing a single mapping for the entire CDFG.

IV. Experiments and Results

This section analyzes the implementation results of the proposed CGRA, providing an estimation of the performance, area, and energy efficiency when running dimensionality reduction of seizure detection algorithm. The dimensionality reduction is based on principal component analysis (PCA) and composed of five kernels where Singular Value Decomposition (SVD) is performed by three of these kernels. SVD is a well-known algorithm that computes eigenvalues and eigenvectors of a co-variance matrix. The method transforms the starting matrix into a bi-diagonal form through successive Householder matrices followed by diagonalizing the matrix through Givens matrices [12]. The output of this procedure is the eigenvectors' matrix, used to find the Principal Components (PCs).

TABLE I: Area in μm^2

	CGRA	RISC-V CPU
DMA controller	593	
interconnect	6,273	
Context memory	9,345	
TCDM	65,164	
HPEA	122,692	
Total Area	204,067	106,759

A. Implementation Results

This section describes the implementation results of the proposed CGRA accelerator, providing a comparison with a low power RISC-V based processor [6]. This core is highly optimized for DSP benchmarks and features SIMD extensions, including dot-product and shuffle instructions, and misaligned load support that greatly reduce the load-store traffic to data memory while maximizing computational efficiency.

Both the CGRA and CPU RTL implementations were synthesized with Synopsys design compiler 2014.09-SP4 in STMicroelectronics 28nm UTBB FD-SOI technology. Synopsys PrimePower 2013.12-SP3 was used for timing and power analysis at the supply voltage of 0.6V, 125°C temperature, in typical process conditions. In this operating condition the CGRA and the RISC-V CPU operates at 50 MHz. The Context Memory of the CGRA was sized at 4 KB, to fit both instructions and constants of the HPEA. The TCDM is sized at 32 KB with 4 memory banks. The implementation considers an HPEA consisting of a 4x2 PE array, each PE including 20x64-bit IRF, a 32x8-bit RRF and 32x16-bit CRF. The PE array is sized to support DSP and bio-signal processing applications. We do not focus on design space exploration, since this is out of the scope of this paper. Each of the PEs in the top row consists of Floating point multiplier and adder/subtractor. To perform the very few divide and square root operations that are present in the targeted EEG kernels, one floating point divider/square-root unit is employed in the HPEA. For area comparison, the RISC-V CPU includes 32 KB of data memory and 1 KB of instruction cache, which is equivalent to the design parameters of the proposed CGRA.

Table I shows the area breakdown of the CGRA and comparison with a RISC-V CPU. A complete area breakdown (Fig. 4) of the PE containing floating point multiplier, adder/subtractor depicts FPU almost acquires 25% area of the PE, whereas the memory components RRF, CRF, IRF takes around 7%, 13% and 28% respectively. ALU and FAGU together covers 21% of the PE area. In this paper for fair comparison and to achieve similar accuracy, we employed the same IEEE compliant FPU that is used by the RISC-V core.

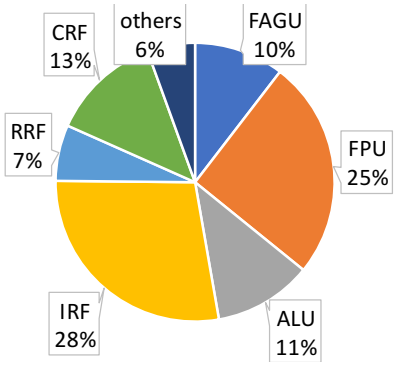

Fig. 4: Area break down of a PE with FPU

TABLE II: Performance comparison in cycles

Kernels	CGRA without FAGU	CGRA with FAGU	RISC-V	Gain in CGRA with FAGU
mean covariance	734,934	300,050	732,377	2.44x
accumulate	94,018	48,877	66,334	1.36x
householder	211,113	125,375	132,870	1.06x
diagonalize	257,488	84,186	175,092	2.08x
PC	268,809	105,993	488,252	4.61x

B. Performance Results

This section provides performance comparison between the proposed CGRA and the RISC-V running PCA kernels. Additionally, we have presented performance results for the CGRA version where address is computed in software. Table II presents latency (in cycles) information executing the kernels computing PCA in the EEG analysis. The CGRA execution achieves a speed-up up to 4.64× with an average of 2.3×. It is to be noted that the performance gain in the kernels like accumulate, householder and diagonalize is less than the gain in the kernels like PC computation and mean-covariance. Due to heavy control intensive nature of these kernels (accumulate, householder and diagonalize), the compiler is unable to exploit instruction level parallelism efficiently. The table also indicates that due to long schedule performance degradation for floating point computations, CGRA without FAGU did not achieve any performance gain.

C. Energy Consumption & Efficiency

Following the considerations of the Section 4.1, here we present energy efficiency of the CGRA compared with the RISC-V CPU. Table III compares the energy consumption in μJoule while running different kernels in the CGRA and CPU. The table shows that the CPU energy consumption surpasses the CGRA by up to 3.3× with an average of 1.86×.

TABLE III: Energy consumption comparison in μJoule

Kernels	CGRA	RISC-V	Energy gain
mean-covariance	3.24	5.71	1.8x
accumulate	0.38	0.52	1.4x
house_holder	1.02	1.04	1x
diagonalize	0.8	1.4	1.8x
PC	1.14	3.8	3.3x

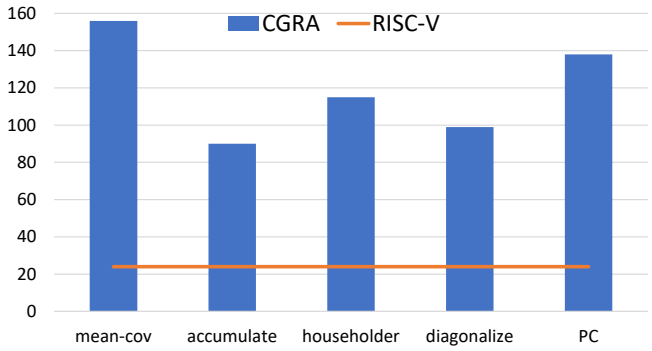

Fig. 5: Energy efficiency compared to the RISC-V CPU

For the execution in CGRA, Million Operations Per Second (MOPS) only considers the active PEs during execution, since a PE may be idle due to TCDM bank access conflicts, consecutive NOPs or unused in the application execution. Executions with high number of active PEs/cycle achieve large MOPS. Fig. 5 shows due to highly control intensive nature of the accumulate and diagonalize, the CGRA achieved efficiency of 90 and 99 MOPS/mW respectively. CGRA achieves a maximum of 156 MOPS/mW energy efficiency for the mean-covariance which is 6.5× higher compared to the RISC-V CPU which has the efficiency of 24 MOPS/mW. For the householder and PC, CGRA achieves an efficiency of 115 and 138 MOPS/mW respectively.

V. Conclusion

This paper presents an energy efficient CGRA and associated compilation flow to perform floating point application acceleration. Supporting floating point (or more generally multi-cycle operations) must go along with hardware based address generation unit which decouples the address generation from the execution flow to leverage parallel arbitrary data accesses. The novel architectural approach proposed in this paper reduces the complexity and size of the instruction memory in the traditional CGRAs which do not possess a dedicated address generation unit. Thanks to the efficient compilation flow, the architecture exploits instruction level parallelism efficiently while running floating-point computations compared to the software based address calculation scheme. Case study on the EEG application shows that the proposed CGRA achieves an average of 120 MOPS/mW energy efficiency which is around 5× higher than the RISC-V CPU with an area overhead of 1.9× only.

References

[1] F. Bouwens, M. Berekovic, A. Kanstein, and G. Gaydadjiev. Architectural exploration of the adres coarse-grained reconfigurable array. In *Proceedings of the 3rd International Conference on Reconfigurable Computing: Architectures, Tools and Applications*, ARC'07, pages 1–13, Berlin, Heidelberg, 2007. Springer-Verlag.

[2] S. Das, K. J. Martin, D. Rossi, P. Coussy, and L. Benini. An energy-efficient integrated programmable array accelerator and compilation flow for near-sensor ultralow power processing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 38(6):1095–1108, 2018.

[3] S. Das, T. Peyret, K. Martin, G. Corre, M. Thevenin, and P. Coussy. A scalable design approach to efficiently map applications on cgras. In *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 655–660, July 2016.

[4] S. Das, D. Rossi, K. J. M. Martin, P. Coussy, and L. Benini. A 142mops/mw integrated programmable array accelerator for smart visual processing. In *Circuits and Systems (ISCAS), 2017 IEEE International Symposium on*, pages 1–4. IEEE, 2017.

[5] B. De Sutter, P. Raghavan, and A. Lambrechts. Coarse-grained reconfigurable array architectures. In S. S. Bhattacharyya, E. F. Deprettere, R. Leupers, and J. Takala, editors, *Handbook of Signal Processing Systems*, pages 449–484. Springer US, 2010.

[6] M. Gautschi, P. D. Schiavone, A. Traber, I. Loi, A. Pullini, D. Rossi, E. Flamand, F. K. Gürkaynak, and L. Benini. Near-threshold risc-v core with dsp extensions for scalable iot endpoint devices. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(10):2700–2713, Oct 2017.

[7] B. Khailany, W. J. Dally, U. J. Kapasi, P. Mattson, J. Namkoong, J. D. Owens, B. Towles, A. Chang, and S. Rixner. Imagine: Media processing with streams. *IEEE Micro*, 21(2):35–46, 2001.

[8] M.-H. Lee, H. Singh, G. Lu, N. Bagherzadeh, F. J. Kurdahi, M. Eliseu Filho, and V. C. Alves. Design and implementation of the morphosys reconfigurable computing processor. *Journal of VLSI signal processing systems for signal, image and video technology*, 24(2-3):147–164, 2000.

[9] G. Levi. A note on the derivation of maximal common subgraphs of two directed or undirected graphs. *Calcolo*, 9(4):341–352, 1973.

[10] M. A. Shami and A. Hemani. Address generation scheme for a coarse grain reconfigurable architecture. In *Application-Specific Systems, Architectures and Processors (ASAP), 2011 IEEE International Conference on*, pages 17–24. IEEE, 2011.

[11] D. Voitsechov and Y. Etsion. Inter-thread communication in multithreaded, reconfigurable coarse-grain arrays. *arXiv preprint arXiv:1801.05178*, 2018.

[12] J. H. Wilkinson and C. Reinsch. *Handbook for Automatic Computation: Volume II: Linear Algebra*, volume 186. Springer Science & Business Media, 2012.