HAL
archives-ouvertes.fr

A 142MOPS/mW Integrated Programmable Array accelerator for Smart Visual Processing

Satyajit Das, Davide Rossi, Kevin Martin, Philippe Coussy, Luca Benini

▶ **To cite this version:**

Satyajit Das, Davide Rossi, Kevin Martin, Philippe Coussy, Luca Benini. A 142MOPS/mW Integrated Programmable Array accelerator for Smart Visual Processing. IEEE International Symposium on Circuits

Systems, May 2017, Baltimore, United States. 2017 IEEE International Symposium on Circuits and Systems (ISCAS). <hal-01534574>

HAL Id: hal-01534574
https://hal.archives-ouvertes.fr/hal-01534574

Submitted on 5 Oct 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A 142MOPS/mW Integrated Programmable Array accelerator for Smart Visual Processing

Satyajit Das*†, Davide Rossi†, Kevin J. M. Martin*, Philippe Coussy*, and Luca Benini†‡

*Univ. Bretagne-Sud, UMR 6285, Lab-STICC, F-56100 Lorient, France, [firstname].[lastname]@univ-ubs.fr,
†DEIS, University of Bologna, Italy, [firstname].[lastname]@unibo.it
‡Integrated Systems Laboratory, ETH Zurich, Switzerland, [first-initial][last name]@iis.ee.ethz.ch

***Abstract*—Due to increasing demand of low power computing, and diminishing returns from technology scaling, industry and academia are turning with renewed interest toward energy-efficient programmable accelerators. This paper proposes an Integrated Programmable-Array accelerator (IPA) architecture based on an innovative execution model, targeted to accelerate both data and control-flow parts of deeply embedded vision applications typical of edge-nodes of the Internet of Things (IoT). In this paper we demonstrate the performance and energy efficiency of IPA implementing a smart visual trigger application. Experimental results show that the proposed accelerator delivers 507 MOPS and 142 MOPS/mW on the target application, surpassing a low-power processor optimized for DSP applications by 6x in performance and by 10x in energy efficiency. Moreover, it surpasses performance of state of the art CGRAs only capable of implementing data-flow portion of applications by 1.6x, demonstrating the effectiveness of the proposed architecture and computational model.**

I. Introduction

Recent years have seen an explosive growth of small and autonomous devices that sense the environment and wirelessly communicate the sensed data. Collectively referred to as the end-nodes of the Internet of Things (IoT), these smart and connected devices, often powered by coin form factor batteries or energy harvesters share the need for extreme energy-efficiency for operation within power envelopes of only a few milliwatts. Among the others, distributed low-power visual senors are very attractive for several IoT applications, thanks to the richness of the visual content and to their capability to operate within a milliwatt-range power envelope [10]. However, the share of wireless communication in the overall power budget of most visual sensor nodes remains dominant due to the significant bandwidth generated even by low-resolution cameras [12].

In this scenario, a high-potential approach to reduce the energy of visual sensor nodes is to increase the capabilities of near-sensor data analysis and filtering by providing computational power to the processing sub-system. This approach, also referred to as *smart visual sensing* can dramatically shrink the amount of wireless data transmitted, by reducing it to a class, a signature, or even just a simple event. This "semantic compression" has also beneficial effect in terms of aggregate network bandwidth reduction. For this reason, the availability of powerful, flexible, and energy-efficient digital processing hardware in close proximity to visual sensors will play a key role in the IoT revolution [1].

While traditional smart visual sensors rely on inflexible Application Specific Integrated Circuits (ASIC) [11], parallel near-threshold computing is emerging as a trend to exploit the energy boost given by low-voltage operation while recovering the related performance degradation through execution over multiple programmable processors [9]. This approach joins the benefits for performance, energy efficiency, and flexibility. In this work we make a significant step forward toward parallel near-threshold computing, by exploiting an Integrated Programmable Array (IPA) as an accelerator for near-threshold parallel programmable platforms.

Similarly to Field Programmable Gate Arrays (FPGAs), Coarse Grained Reconfigurable Arrays (CGRAs) belong to the class of reconfigurable devices, but they provide significantly smaller re-configuration time and much higher efficiency for execution of data processing algorithms [4]. Point-to-point data communication between PEs represents one of main advantages over energy-expensive data sharing through multi-banked SRAMs in near-threshold processor clusters [9]. However, most traditional CGRA architectures [2] [13] [6] are only suitable for implementing data-flow portions of algorithms, relying on general purpose processors for the control-flow portions.

In the context of smart visual applications this shortcoming of traditional CGRAs is quite severe, since after brute-force morphological filtering (e.g. erosion, dilatation, Sobel convolution), these algorithms usually require the execution of highly control intensive code for high-level feature extraction, classification, labeling, triggering. In this work we exploit a mapping flow [3], which allows to map complete Control Data Flow Graphs (CDFG) onto CGRAs. In the context of smart vision, the proposed architecture allows to implement both the filtering portions and control oriented portions of applications on the IPA, providing significant performance and energy efficiency improvement with respect to both general-purpose processors and traditional CGRA architectures.

In this paper we present the IPA architecture, computation model and the implementation of a smart visual trigger algorithm on the accelerator, including the triggering function dominated by highly control intensive code. Experimental results show that the performance achieved by the IPA during the execution of the target application is 507 MOPS with an energy efficiency of 142 MOPS/mW at the supply voltage of 0.6V. The IPA, relying on a compilation flow which starts from plain ANSI C code (no vectorization, no intrinsics, no assembly) achieves a maximum speed up of 9x compared to a an general-purpose processor highly optimized for DSP applications when programmed in ANSI C code, and by 6x with code optimized with vectorization and intrinsics, with an area overhead of 1.4x. With respect to other state of the art CGRAs highly optimized for data flow dominated applications, the proposed architecture delivers 1.6x better performance thanks to its capability to implement both data- and control-flow dominated kernels.

The paper proceeds as follows. Section II presents the detailed architecture and fine-grained power management. Section III presents the implementation results, the application and comparison results with the state of the art. Finally the paper concludes in section IV.

II. Architecture

A. Integrated Programmable-Array Accelerator (IPA)

The architecture comprises a PE array, a global context memory, a DMA controller (DMAC), a tightly coupled data memory (TCDM) with multiple banks and a logarithmic interconnect. Figure 1 shows the organization of the IPA.

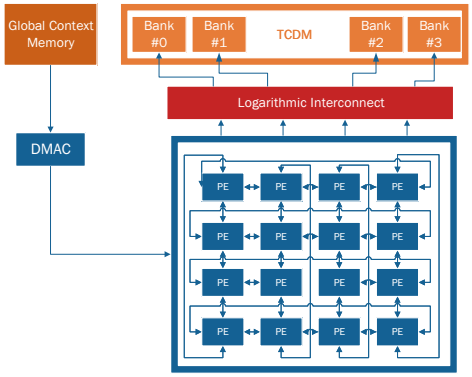

Fig. 1: Integrated CGRA computing system

PE Array: The array consists of a parametric number of PEs, connected with mesh torus network. Figure 2 describes the components of a PE. Two input muxes select two operands (OpA and OpB). The sources are the neighbouring PEs and the register file. A 32-bits ALU and a 16-bit x 16-bit = 32-bit multiplier are employed in this block. The Load Store Unit (LSU) is optional for the PEs (the optimal number of LSU is a parameter studied in this section). The Control unit is responsible for fetching the instruction from the corresponding address of the instruction memory and managing program flow. The Regular Register File (RRF) and Output Register (OR) store the temporary variables, while constants are stored in Constant Register File (CRF). The Condition Register (CR) contains 0 for all the normal operations and true conditions, and 1 for false conditions. The boolean OR of all the control bits from all PEs gives the indication that one PE has executed false condition in the previous cycles. So next, the offset address of the false path must be fetched. The Jump Register (JR) contains the address to be jumped.

Global Context Memory (GCM): The Global Context Memory stores configuration data (instruction and constants) for each PE in the PE Array.

DMA controller (DMAC): The DMA controller identifies configuration data for the corresponding PE and transfers it in the *load context* stage. It also initiates the execution phase after loading all the contexts.

TCDM and logarithmic interconnect: The TCDM has a number of ports equal to the number of memory banks providing concurrent access to different memory locations. Load store operations in the PEs are based on a high bandwidth low-latency interconnect, implementing a word-level interleaving scheme to reduce access contention.

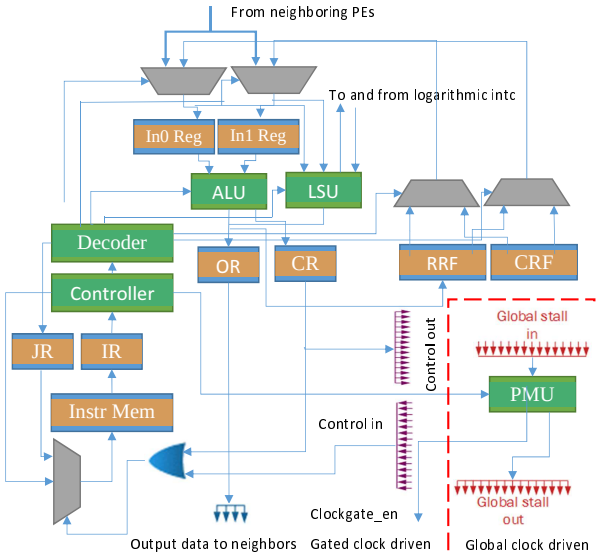

Fig. 2: Components of PE

B. Power Management

In order to reduce dynamic power consumption in idle mode, each PE employs a Power Management Unit (PMU). The idle nature of the PE arises from three situations. (a) Unused PE: when a PE is not used in the mapping the PE is clock gated for the entire execution process. (b) Load Store stall: In case of contentions in the TCDM access, the PMU generates a global stall which is received by all the PEs. Until the global stall is resolved all the PEs are clock gated by their corresponding PMUs. (c) Multiple NOP operations: The NOP instruction contains the number of successive NOPs. The PMU consists of a counter to resolve the number of consecutive NOPs. The *ckgate_en* remains low until the count finishes.

C. Overview of the computation model

After compiling a kernel, the mapping tool generates the context and the addresses for the data in the local shared memory. The context is stored in the *GCM*. The context contains instructions and constants for each PE in the array. Prior to the execution starts, the context is loaded into the corresponding instruction memory and constant register file of the PEs as follows.

1) Load context: Figure 3 shows the components engaged in this process. In each cycle of the load context process the *DMA controller* receives the context word from the context memory and broadcasts to the PEs.

The protocol presented here supports two modes for loading instructions.

Broadcast mode: this mode is used to broadcast the instruction to the set of PEs that execute the same instruction at the same cycle.

Normal addressing mode: This mode is used to load instructions and constants specific to a single PE.

The organization of the global context memory is shown in Figure 4. Here each address contains a 64-bits context word. The GCM is divided into several segments (table I), where each segment contains a set of instructions and constants to be

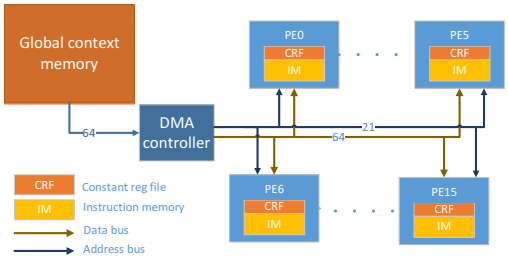

Fig. 3: The memory hierarchy to load context into PEs

broadcast or normally addressed. The first bit in each segment represents whether next set of instructions is for broadcast (0) or normal addressing mode (1). In broadcast mode next 16 bits represents the mask, where the position of the high bits represents the addresses of the PEs to be broadcast. For normal addressing mode, next 4 bits of the first mode bit represents the PE address where the next set of instructions and constants will be fed.

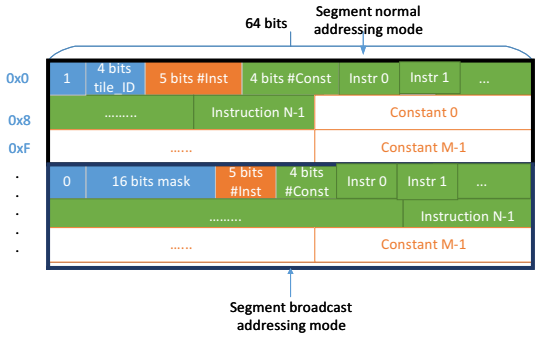

Fig. 4: Organization of the global context memory

TABLE I: Structure of a segment

Bits	Description
1	Addressing modes
4/16	Normal Address/Mask
5	Total number of instructions (N)
4	Total number of constants (M)
20xN	Instructions
32xM	Constants

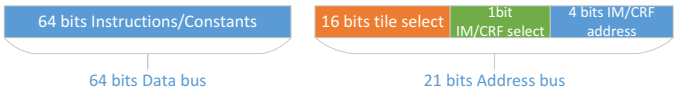

Fig. 5: Format of the address and data bus from DMA

The format of the address and data bus from the DMAC to the PEs is presented in Figure 5. The address bus contains 21 bits of information containing the PE address and the corresponding instruction memory (IM) or constant register file (CRF) addresses. The first 16 bits represent the mask to select one or more PEs. The position of the high bits represents the PE address to be loaded. For broadcast mode more than one high bits will be present and for normal addressing modes only one high bit is present to select the PEs. The next bit to the mask selects whether the data has to be written in IM [1'b0] or in the CRF [1'b0]. The next 4 bits represent the IM address or the address of the CRF to be written. The data bus contains 64 bits of information, where maximum of 3 full instructions (60 bits) and part of 4th instruction (4 bits) or 2 full constants (32 bits each) can be accommodated.

2) Execution: After loading the context, the DMAC notifies all the PEs to start the execution process. The execution follows the mapping described in [3], where each kernel CDFG is represented as a set of basic blocks, each containing sequence of consecutive statements defining a data flow graph with control entering at the beginning and leaving at the end. The control flow at the end of basic blocks is supported with jump (jmp) and conditional jump (cjmp) instructions. The PEs fetch 20-bit instruction from their local instruction memory at each cycle. Due to power constraints the instruction set architecture is kept quite simple. The immediate data are shifted to constant register file which eases the compression of the instruction. Hence the pressure on the decoder is quite low. Table II describes the format of the instruction.

TABLE II: Instruction format

5 bits	2 bits	3 bits	1 bits	4 bits	1 bit	4 bits
Opcode	Output Reg type	Dest Reg Add	OpA Type	OpA Add	OpB Type	OpB Add
Jmp	Address		unused			
Cjmp	Address of true path		Address of false path		unused	
NOP	Number of consecutive NOPs		unused			

III. Benchmarking

This section analyzes the implementation results of the IPA, providing an estimation of the performance, area, and energy efficiency when running several kernels involved in a smart trigger vision application.

A. Implementation Results

This section describes the implementation results of the proposed IPA accelerator, providing a comparison with the or10n CPU [5] highly optimized to accommodate DSP applications. Both designs 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, 25°C temperature, in typical process conditions. In this operating condition the IPA operates at 100 MHz, delivering a theoretical peak performance of 1.6 GOPS with an energy efficiency of 242 MOPS/mW. The global context memory of the IPA was sized at 4KB, to fit both instructions and constants of the PEs of the array in a worst-case scenario (i.e. all PEs instruction and constant memory are full filled). This approach allows to employ a double-buffering mechanism to swap the configuration context on the global context memory while the previous context is being executed, during execution of complex applications composed of several kernels, like the one presented in this work. The TCDM is sized at 32KB with 4 memory banks. The implementation considers an IPA consisting a 4x4 PE array, each PE including 20x32-bit instruction memory, a 32x8-bit regular register file and 32x16-bit constant register file. For area comparison, the CPU includes 32kB of data memory, 4kB of instruction memory, and 1 kB of instruction cache, which is equivalent to the design parameters of the proposed IPA. Table III shows the area breakdown of the IPA, compared to that of a CPU with 1KB of instruction cache.

B. Performance and energy efficiency

Following the considerations of previous section, this section provides performance comparison of IPA running at 100 MHZ with respect to the CPU running at 45 MHZ

TABLE III: Area [μm^2]

	PE Array	TCDM	Inter connect	DMAC	GCM	Total area
IPA	117173.7	65163.8	6273	593.612	9344.96	198549.10
CPU						139508.76

clock frequency, that are the operating frequency of the two architectures at the operating voltage of 0.6V. The experiment is carried out on a smart visual surveillance application [7] performing on 160x120 resolution of images, consisting 9 different motion detection kernels including morphological filters (e.g. finding minimum and maximum pixel, erosion, dilatation, Sobel convolution), and a smart trigger kernel asserting an alarm if the size of the detected objects surpasses a defined threshold, the latter kernel composed of highly control intensive code. Table IV shows the performance comparison executing the application in the IPA (programmed in plain ANSI C code) and a highly optimized core with the support for vectorization and DSP extensions that can only be exposed optimizing the source code with intrinsics [5]. The IPA can achieve an average performance of 507 MOPS (with a peak of 632 MOPS) and 142 MOPS/mW on average (with a peak of 177 MOPS/mW) allowing to surpass the CPU by to 6x and 10x in performance and energy efficiency, respectively. It is interesting to notice that while DSP instructions do not improve the performance of the core during execution of the smart trigger kernel, its implementation on the IPA provides even more benefits with respect to the data-flow part of the application (motion detection), improving performance by 10x with respect to execution on the processor (Table V).

TABLE IV: Performance comparison

Applications	CPU	CPU	CPU (optimized)	IPA
Motion Detection	**cycles**	2 237 124	1 308 036	261 120
	energy[μJ]	10.179	5.952	0.679
	perf gain	9x	5x	
	energy gain	15x	9x	
Smart Trigger	**cycles**	480 000	480 000	48 000
	energy[μJ]	2.184	2.184	0.125
	pref gain	10x	10x	
	energy gain	17x	17x	
Overall	**cycles**	2 707 200	1 785 600	309 120
	energy[μJ]	12.318	8.124	0.804
	preformance gain	9x	6x	
	energy gain	15x	10x	

TABLE V: Energy efficiency in IPA for different kernels

kernel	min img	sum img	sub img	val abs	bin	mul	dilat	eros	Conv	Trig	Avg
MOPS	557	632	593	618	461	443	436	454	371	507	507
MOPS /mW	156	177	166	173	129	124	122	127	104	142	142

C. Comparison with the state of the art architectures

Table VI presents the performance comparison of the smart visual trigger application running on a CPU and two state of the art reconfigurable array architectures [7] [8], chosen due to the availability of the target application, with similar features to other state of the art CGRAs. Results show that, although the two state of the art CGRAs deliver huge performance when dealing with the data-flow portion of the application, thanks to highly optimized and pipelined datapath that allows to implement operations on binary images as boolean operations [7], they are not able to implement the control dominated kernel, which runs on the CPU forming a major bottleneck for performance when considering the whole application. On the other hand, the superior flexibility of the IPA allows to implement the whole application on the accelerator, allowing to surpass performance of other CPU + CGRA systems by 1.6x. It is important to note that in the context of more complex smart vision applications, such as context understanding and scene labeling, it is common that control intensive kernels dominate the overall execution time share, further improving performance with respect to CGRA accelerators only able to map DFGs.

TABLE VI: Performance comparison of smart visual surveillance application [cycles/pixel]

Reference	Motion Detection	Smart Trigger	Total
CPU	116	25	141
CPU (Optimized)	68	25	93
[7]	2.09	25	27.09
[8]	1.27	25	26.27
IPA	13.6	2.5	16.1

IV. Conclusion

This paper presents an integrated programmable-array accelerator (IPA) and its computational model, featuring the capability to map both data flow and control intensive portions of applications, suitable for standalone implementation of a smart visual imaging applications. The experimental results show that, on the target smart visual trigger application, the IPA achieves an average performance of 507 MOPS with average energy efficiency of 142 MOPS/mW at 0.6V surpassing a general purpose processor by 6x in performance and 10x in energy efficiency. The proposed IPA also surpass and state of the art CGRA architectures performance by 1.6x, thanks to the capability of efficiently implementing control intensive code.

References

[1] NVIDIA INTELLIGENT VIDEO ANALYTICS (IVA) PLATFORM. http://www.nvidia.com/object/intelligent-video-analytics-platform.html.

[2] F. Bouwens et al. 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.

[3] S. Das et al. Efficient mapping of CDFG onto coarse-grained reconfigurable array architectures. In *22nd Asia and South Pacific Design Automation Conference*, To be appeared in 2017.

[4] B. De Sutter et al. 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.

[5] M. Gautschi et al. Tailoring instruction-set extensions for an ultra-low power tightly-coupled cluster of openrisc cores. In *2015 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, pages 25–30, Oct 2015.

[6] K. Masuyama et al. A 297mops/0.4 mw ultra low power coarse-grained reconfigurable accelerator cma-sotb-2. In *2015 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, pages 1–6. IEEE, 2015.

[7] C. Mucci et al. Intelligent cameras and embedded reconfigurable computing: a case-study on motion detection. In *System-on-Chip, 2007 International Symposium on*, pages 1–4, Nov 2007.

[8] D. Rossi et al. A heterogeneous digital signal processor implementation for dynamically reconfigurable computing. In *2009 IEEE Custom Integrated Circuits Conference*, pages 641–644, Sept 2009.

[9] D. Rossi et al. A 60 gops/w,- 1.8 v to 0.9 v body bias ulp cluster in 28nm utbb fd-soi technology. *Solid-State Electronics*, 117:170–184, 2016.

[10] M. Rusci et al. An event-driven ultra-low-power smart visual sensor. *IEEE Sensors Journal*, 16(13):5344–5353, July 2016.

[11] Y. Shi and F. D. Real. *Smart Cameras: Fundamentals and Classification*, pages 19–34. Springer US, Boston, MA, 2010.

[12] V. Shnayder et al. Simulating the power consumption of large-scale sensor network applications. In *Proceedings of the 2Nd International Conference on Embedded Networked Sensor Systems*, SenSys '04, pages 188–200, New York, NY, USA, 2004. ACM.

[13] H. Singh et al. Morphosys: an integrated reconfigurable system for data-parallel and computation-intensive applications. *IEEE Transactions on Computers*, 49(5):465–481, May 2000.