International Journal of Networking and Computing – www.ijnc.org
ISSN 2185-2839 (print) ISSN 2185-2847 (online)
Volume 5, Number 1, pages 239–251, January 2015

A Co-Processor Design for an Energy Efficient Reconfigurable Accelerator CMA

Mai Izawa, Nobuaki Ozaki, Yusuke Koizumi, Rie Uno, Hideharu Amano
Graduate School of Science and Technology
Keio University
Yokohama-shi 223-8522 Japan

Received: February 12, 2014
Revised: April 28, 2014
Revised: July 19, 2014
Accepted: October 2, 2014
Communicated by Hidetsugu Irie

Abstract

Cool Mega Array (CMA) is an energy efficient reconfigurable accelerator consisting of a large PE array with combinatorial circuits and a small microcontroller. In order to enhance the energy efficiency of the total system, a co-processor design of CMA called CMA-Geyser is proposed. By partly replacing the programmable microcontroller by the host processor Geyser with a dedicated hardware controller, the setting up for the CMA and data transfer can be efficiently done.

The design using 65nm CMOS process is compared with an off-loading style multicore system Cube-1. By eliminating the data memory required in Cube-1, CMA-Geyser reduced 21.3% of semiconductor area. Also, it achieved about 2.7 times performance of Cube-1 by the efficient data communication between host and the accelerator.

1 Introduction

Coarse-Grained Reconfigurable processor Arrays (CGRA) [5, 2, 6] have received attention as energy efficient accelerators for various types of battery driven mobile devices, and some of them have been utilized in commercial products[8, 7]. CMA (Cool Mega Array)[10] is a CGRA architecture especially designed focusing on energy efficiency.

It provides a large PE (Processing Element) array consisting of combinatorial logic. Data-flow graphs for target application programs are mapped directly on the array, and computation is done without storing intermediate results. The energy for storing intermediate results into registers in PE and clock distribution can be saved. In order to keep the flexibility, a small microcontroller manages data distribution and collection between data memory and input/output of the PE array. The supply voltage of the PE array is scaled so that the computation delay in the PE array is well balanced to the time for data management by the microcontroller.

The first prototype CMA-1 using 65nm CMOS technology achieved 2.7GOPS/11.2 mW sustained performance, however through the analysis, it appeared that the energy consumption of the microcontroller which manages data transfer between PE array and data memory occupies a large part of the total energy as well as data memory itself. Sometimes they become larger than that consumed with the PE array on which the computation is performed. The microcontroller has another problem on its programming. Although the micro-code is just for a management of simple data transfer control between PE array and data memory, the setup of configuration data and initialization of constant registers and mapping registers are cumbersome job for the programmer.

Since CMA is designed as an off-loading type accelerator, a host processor is required when it is embedded in the system. In Cube-1[9], the first scalable heterogeneous multicore system using multiple CMAs, a MIPS R3000 compatible processor with fine-grained power gating Geyser[3] is used as a host processor. Although Cube-1 achieved a certain performance improvement with the use of multiple CMAs, the data transfer time between Geyser and CMAs sometimes limited the performance.

In order to address all these problems, we propose a co-processor type CMA in which a part of microcontroller is replaced by the host processor. By tightly coupling with the cache of the host processor, the total amount of data transfer in the system is much reduced. The programming for setting up the CMA can be done by using co-processor instructions added to the host processor. Since the data memory and a part of microcontroller are omitted, the energy consumption of the total system can be reduced.

The contributions of the paper are as follows:

- A co-processor design of CMA, which is more power-efficient than off-loading style is proposed and designed considering a real chip implementation.

- Two types of host/accelerator collaboration methods: co-processor style and off-loading style are quantitatively compared by using the exactly same host and accelerator design.

The rest of this paper is organized as follows: in Section 2, the architecture of CMA and a multicore system Cube-1 are introduced. After checking current co-processor style CGRAs, the co-processor style CMA called CMA-Geyser is proposed in Section 3. The area, performance and power of CMA-Geyser are evaluated and compared with Cube-1 in Section 4. Section 5 concludes the paper with discussion of future work.

2 CMA architecture

2.1 The concept of CMA architecture

The CMA architecture is mainly targeted at multi-media processing in battery-driven embedded systems like current commercial CGRA systems (SRP[7] and STP-engine[8]). The target applications are image filters and static or dynamic image coders including JPEG, MPEG and H264. In such processing, a fixed amount of data must be processed in a certain time, but there is no advantage in reducing the processing time to less than that required. Minimizing the amount of energy used to process a fixed amount of data is important as it affects battery lifetime. The objective of CMA design is thus to design an architecture for executing a fixed number of computations in the required time with the minimum amount of energy.

Since most stream processing involves a large amount of parallelism, the required performance can be achieved by parallel processing using many PEs. One key concept of the CMA architecture is reducing the supply voltage of the PE array while still achieving the required computation time.

Another key concept is reducing any energy usage other than that required for computation. The data to be computed must be transferred from the data memory to the computational module, where it is computed. The computation results must then be written back into the data memory. The computation process is indispensable, so it consumes a certain amount of energy. However, all other energy consumption should be eliminated.

Although the CMA architecture is a type of CGRA, the PE array consists of combinatorial circuits without registers and context memory unlike other CGRAs. The supply voltage of the PE array can be scaled without worrying about the effects on the setup time or on the clock skew of the registers. There are no clock tree for distributing the clock in the PE array. As a result, the power consumption in the PE array can be much reduced.

Only the input and output data for the PE array are stored in registers. The microcontroller reads data from the data memory (DMEM) and distributes it to the register attached to the input of the PE array. It also collects the results from the register attached to the output of the PE array, and writes them back it to the data memory. It flexibly manages the data transfer between the memory and registers by using mapping registers and vector operations. With the above structure, it enables to implement various application programs without power hungry dynamic reconfiguration in the PE array.

Since the computation in the PE array and data management by the micro controller are done in a pipelined manner, their execution speeds must be balanced. If the computation delay is shorter than the data management delay, the voltage supplied to the PE array can be reduced. The total power required for computation can thus be reduced without degrading computing performance. On the other hand, if the data management delay is shorter than the computation delay, wave pipelining in the PE array can be used. The delay time for achieving wave pipelining can be also controlled by changing the voltage supplied to the PE array.

2.2 Prototype chip CMA-1

The first prototype, CMA-1[10] with 8×8 PE array was fabricated in $2.1 \times 4.2mm^2$ 65-nm CMOS technology, and achieved 2.7-GOPS/11.2-mW sustained performance. Figure 1 shows the block diagram of CMA-1. It consists of PE array, microcontroller, data memory (DMEM) and registers.

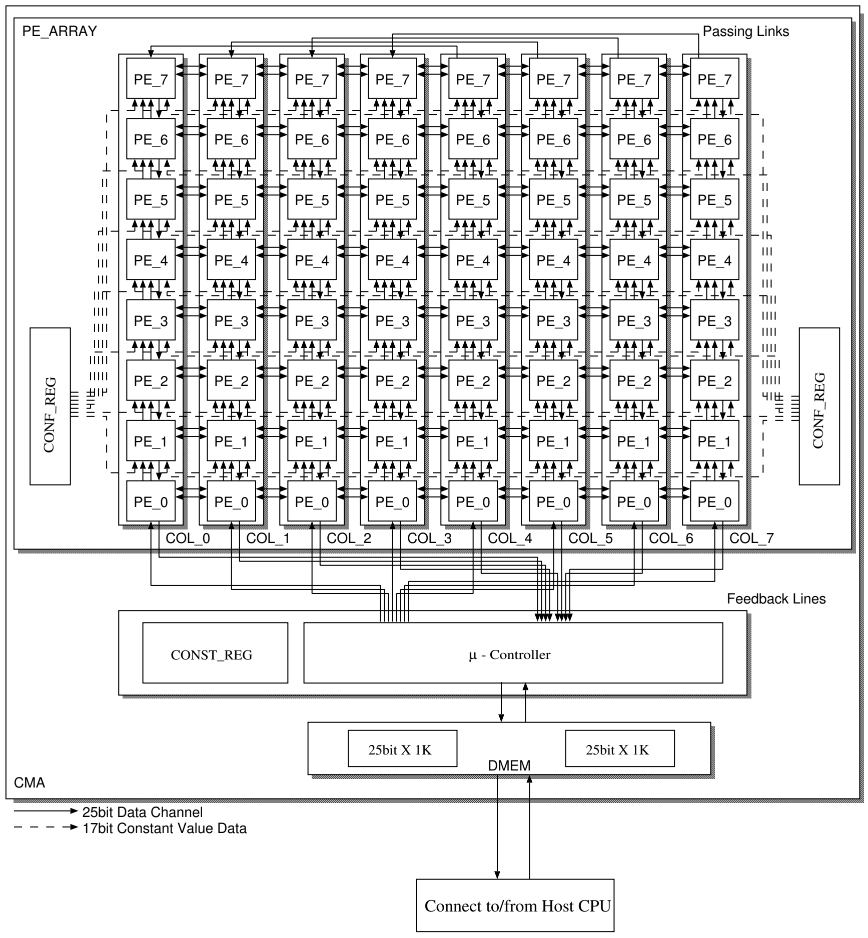


Figure 1: Block diagram of CMA-1

Here, the operation of microcontroller is described in detail.

As shown in Figure 2, the microcontroller is consisting of a controller, Fetch register, Launch register and Gather register. First, it reads from DMEM and distributes them to entries of Fetch

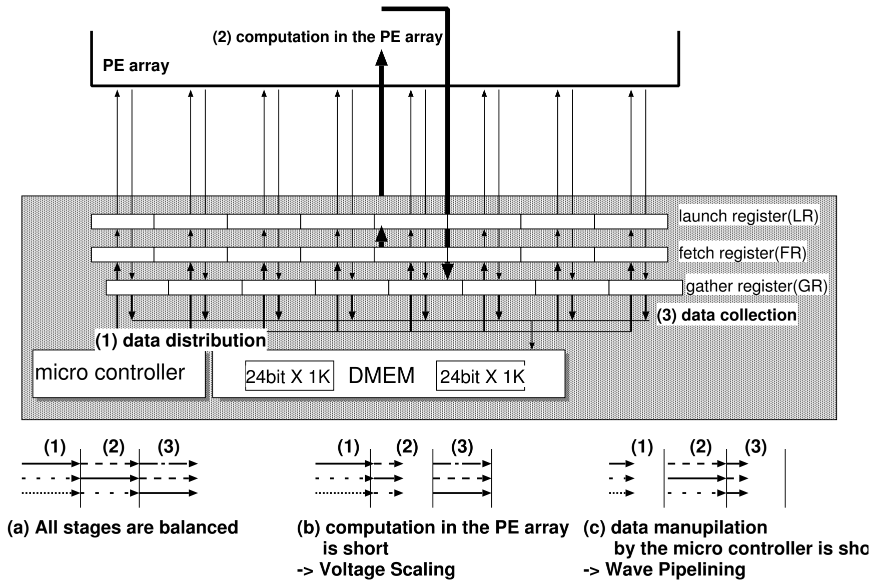


Figure 2: Operations of the microcontroller of CMA-1

register. The data distribution and collection by the micro controller was designed to be flexible to enable arbitrary mapping between the address of the data memory and the input of the PE array using address mapping registers. Stride vector access operations are also supported. When the input data in Fetch register is ready, it is transferred to Launch register and the computation starts in the PE array. After a certain time interval, the result of PE array is stored into Gather register, and the results in the entries of Gather register are written back into DMEM. (1) Distribution from DMEM to Fetch register, (2)computation in the PE array and (3) written back of the results to DMEM from Gather register are done in the pipelined manner. Supply voltage scaling to PE array is used to balance the time for stage (2) with other two stages.

The data distribution and collection time by the microcontroller is balanced with the execution time in the PE array when the supply voltage is 0.8V. In this case, the ratio of the power of microcontoller and data memory is about 40% of the total power. Since the energy consumed in the datapath on the PE array is difficult to reduce, the target for further energy reduction must be the microcontroller.

The second problem is in the programming. In CMA, we used BlackDiamond compiler[12] to map, place and route the application into PE array. C-like source code with some restriction can be used for programming. The micro-code for the microcotroller is used only for controlling data transfer between data memory and Fetch/Gather register. Similar codes can be used for various applications just by changing the time interval of data collection according to the complexity of the data flow graph on the PE array. However, the setting up the configuration data for PE array, and initialization of the constant registers and map registers are cumbersome job for programmers.

2.3 Cube-1

Cube-1[9] is the first prototype of a heterogeneous multicore system using a host processor Geyser and multiple CMA accelerators. Cube-1 uses a 3-D SiP (System in Package) structure to connect multiple chips. Unlike other SiPs, it uses a flexible inter-chip network with inductive coupling wireless interconnects. The number of CMAs can be scalable according to the performance requirement. Now, a system with two chips (1 Geyser + 1 CMA) is available, and one with four chips (1 Geyser + 3 CMAs) shown in Figure 3 is under debugging. The host processor, Geyser[3], is a MIPS R3000 compatible microprocessor with standard 5-stage in-order pipeline. 4Kb 2-way set associative instruction cache and data cache with the same structure are provided as well as a shared TLB with

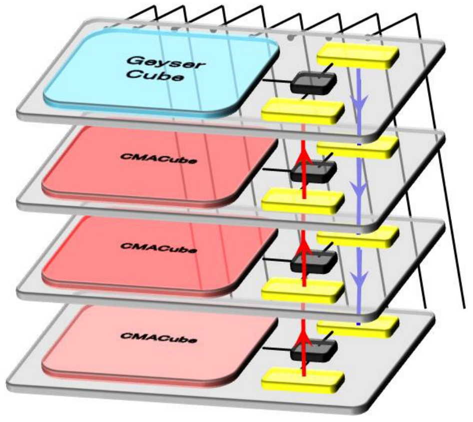


Figure 3: Cube-1 with 1 Geyser-Cube and 3 CMA-Cubes

32 entries. The most interesting feature of Geyser is its fine-grained power gating mechanism, but it is out of scope of this paper.

Although the wireless inter-chip links support enough bandwidth (8Gbps) and DMA block data transfer mechanisms are provided in the router, it appeared that the data transfer time between Geyser and CMAs sometimes occupies more than 20% of total execution time[9].

3 Related Work

All problems around the CMA can be solved by replacing the microcontroller of CMA by the host processor itself. By tightly coupling the host and the CMA, the total amount of data transfer in the system can be reduced. Since the host processor can manage the CMA setup by introducing co-processor instructions, the programming environment is much improved. Since the data memory is not required any more, the total amount of hardware is reduced resulting in the total energy saving.

Before the design, some co-processor style CGRAs are reviewed. S5/S6 engine by Stretch[1] is a typical example of co-processor style CGRA. Tensilica's configurable processor is used as an extensible host processor, and CGRA unit is attached by sharing the general purpose registers of the host. Programmers can specify parts of the program to be accelerated and define instructions executed on the CGRA. Although tightly coupling is achieved by using shared registers, the bandwidth to and from CGRA unit is limited in this approach. CHIMAERA[15] also connects a reconfigurable function unit to the processor core by sharing registers. Shadow register file, a copy of the register file in the processor core is used for data transfer. Although multiple registers can be read by the reconfigurable function unit, only a register can be used for writing the result.

In Garp[11], a main processor and reconfigurable array are relatively loosely coupled. The data transfer between cache and reconfigurable array is explicitly controlled by the main processor. In ADRES[5] and SRP[7], a row of PE array can be used as a VLIW processor. The operations which does not require a large degree of parallelism are executed on a VLIW processor rather than the PE array. The sharing data between VLIW processor and PE array can be naturally done, since the VLIW processor is a part of PE array. However, since VLIW processor itself is specialized for computation, another host processor is needed for operating system, I/O and user interface.

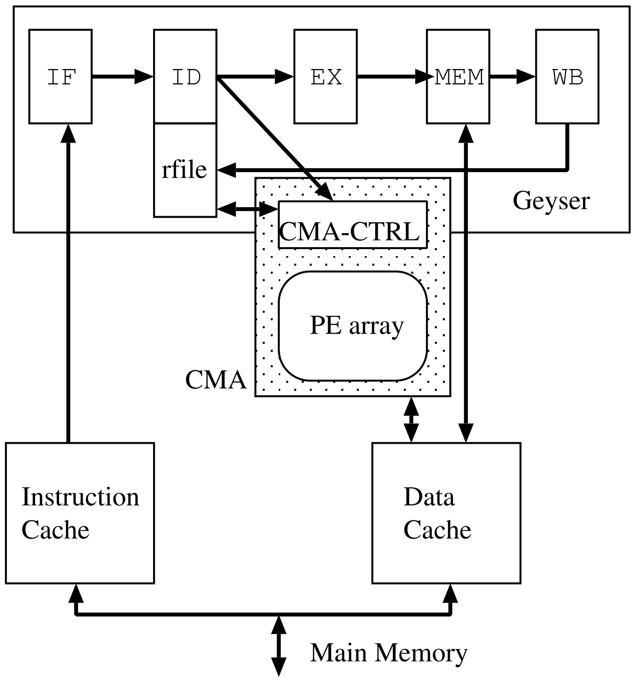


Figure 4: The pipeline structure of CMA-Geyser

4 CMA-Geyser

4.1 The structure of CMA-Geyser

4.1.1 Design policy

The straight forward design of co-processor style CMA is just replacing the microcontroller by the host CPU. However, if the data transfer to Fetch register or from Gather register is completely controlled by the host CPU, the host cannot operate any other jobs during the execution of CMA. So, we separated functions of the current microcontroller into two parts: the data management during CMA operation and configuration management before CMA execution. The former part is implemented with a hardware module CMA-CTRL, and the latter part is done with newly added co-processor instructions.

Figure 4 shows the diagram of CMA-Geyser. The PE array and CMA-CTRL are provided in parallel with the EX stage of the 5-stage pipeline.

We adopted an approach to share the data cache between CMA and Geyser CPU. CMA-CTRL and Geyser CPU share a segment of address space, and both are connected with the data cache directly. The priority is given to the Geyser CPU, and CMA-CTRL is stalled during the access by the host. Also, if cache miss occurs, the access from both is stalled. The benefit of using data cache instead of data memory in CMA is that the programmer does not have to make the data block so as to fit the data memory space. The expensive two port data memory can be omitted from CMA architecture. On the other hand, data distribution and write-back of the results cannot be executed simultaneously.

Compared with register sharing used in Stretch and CHIMAERA, the sharing data cache is relatively loose interconnection. However, sharing the cache is suitable for the block data transfer including stride vector transfer operation supported for scatter/gather in the PE array of CMA. A large total throughput of the data transfer can be kept with the method.

4.1.2 Execution on CMA-Geyser

The following co-processor instructions are provided for preparation of CMA operation:

- CMA_SET Rx: transfers the configuration/control data in the address specified by Rx to configuration registers, constant registers, microcode registers and mapping registers in CMA_CTRL.
- CMA_RUN: starts the CMA.
- MF Cx Rx (Move From CMA to Rx) gets the data from a CMA register to a general purpose register in Geyser. This is used for small data transfer between CMA and Geyser.
- MT Cx Rx (Move To CMA from Rx): transfers data to the CMA register from a general purpose register Rx. It is used for small data transfer between CMA and Geyser.

The execution code of CMA; configuration code, constant data, initial data for address map registers and microcode for CMA-CTRL are prepared in a certain address, and transferred by using the CMA_SET instruction. RoMulTiC[13], data multicast by using two dimensional bit-map is used for quick configuration data transfer. Then, Geyser starts the CMA by executing CMA_RUN instruction. Note that during the CMA execution, Geyser can execute its own instructions since the execution of CMA is controlled by microcode in the CMA-CTRL. The data transfer to Fetch register, execution in PE array and written back from Gather register are done independently by CMA-CTRL. When new data are fetched or results are written, the data cache will miss and the data are continuously transferred from the main memory. Unlike the original CMA, the data cache is not a dual port memory, and three stage pipeline operation in CMA is hard to be implemented. Thus, in CMA-Geyser, two stages: data fetch from the data cache and computation in PE array/write-back to the data cache are executed in the pipelined manner. In this implementation, the execution time is stretched because of the bottleneck in the second stage when execution time on the PE array is large. Thus, in CMA-Geyser, the supply voltage of PE array is set to be higher than that in CMA-1 for avoiding the bottleneck. When the execution of CMA is finished, Geyser receives an interrupt signal. Busy waiting is also supported.

If the data set processed by the CMA is enough small, Geyser can avoid the overhead of unnecessary write back or fill in from/to data cache by using MT or MF instruction. However, these instructions are provided mostly for exceptional control or debugging, and are not used in common programs.

5 Evaluation

Here, CMA-Geyser is compared with Cube-1 from the viewpoint of area, execution time and power consumption.

5.1 Implementation of CMA-Geyser

As shown in Table 1, CMA-Geyser is implemented with the same CMOS 65nm process as Cube-1 using the same tools. The size of PE array, instruction cache, and data cache are also the same. The layout of CMA-Geyser is shown in Figure 5. The CPU is distributed around the PE array of CMA.

A major difference of them is that Cube-1 provides inductive coupling links to connect Geyser and CMA. In this evaluation, the area and power for inductive coupling of Cube-1 is excluded from the evaluation for the fair comparison. On the other hand, those for routers and network interface are included, since they are substantially required to connect a host CPU and off-load style accelerators. Another difference is that there is a real chip of Cube-1, while CMA-Geyser has not yet. For fair comparison, both chips are compared with the post-layout simulation and analyzed by Synospsys Primetime.

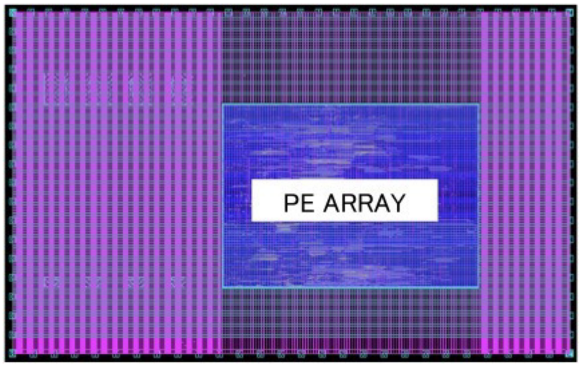


Figure 5: The layout of CMA-Geyser

5.2 Comparison Results

5.2.1 Area

The required number of gates used for CMA-Geyser and Cube-1 is compared in Figure 6. Note that in this evaluation, the router and through-chip wireless interconnect of Cube-1 are excluded. Thus, it is assumed that the host Geyser is connected with a single CMA directly through interface modules in Cube-1. This structure minimizes the hardware requirement at the sacrifice of the flexibility.

Data memory and almost a half of the microcontroller used in CMA in Cube-1 is eliminated in CMA-Geyser. Also, interface modules with DMA function is not needed. As a result, compared with Cube-1, CMA-Geyser reduces the area 25.4% in CMA part, 9% in Geyser part and 21.3% in total.

5.2.2 Performance

We evaluated performance of CMA-Geyser and Cube-1 with two application programs. One is a simple image filter (gray-scale filter) which changes a color RGB image into a gray-scale image. The other is a commonly used JPEG coder for 400 × 400 pixels images.

Figure 7 shows the number of execution cycles when a simple image filter: the gray scale filter for 1024 pixels is executed in CMA-Geyser, Cube-1 with a single CMA and Geyser without any accelerator. CMA-Geyser achieved 3.6 times performance as Geyser and 2.27 times as Cube-1, respectively. In Cube-1, the data transfer between the host and CMA can be overlapped with a processing time in CMA. In Figure 7, ”processing” of Cube-1 shows the part which cannot be overlapped by the data transfer time. Nevertheless, it appears that the data transfer bottlenecks the performance in Cube-1.

The execution time on CMA in Cube-1 itself is faster than that of CMA-Geyser, since the data

Table 1: Specifications of CMA-Geyser and Cube-1

Chip	Technology	Fujitsu e-shuttle 65-nm 12-metal CMOS
	System Clock freq.	100 MHz
CMA	Supply Voltage	0.6-1.2 V for PE array
	PE Array	8 × 8
Geyser	Data/Inst. Cache	4KB 2-way
	TLB	16 entries shared
Tools	Synthesis	Synopsys's Design Compiler
	Layout	Synopsys's IC Compiler

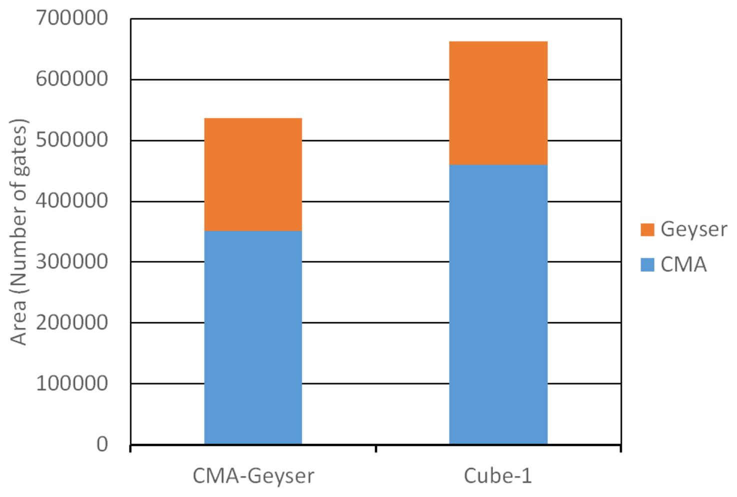


Figure 6: The required area of total gates

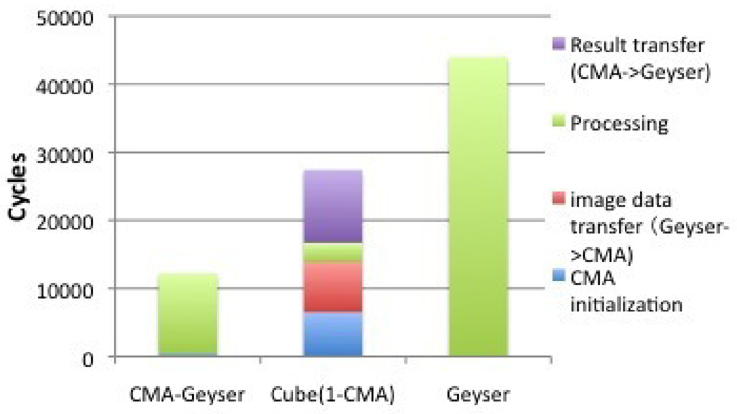


Figure 7: Execution cycles of Gray Scale Filter

distribution and collection can be done simultaneously by using dedicated data memory. Also, the operation time of CMA-Geyser includes the time for data transfer between data cache and main memory, since it is hard to be separated. However, including the overhead of data transfer between the cache memory in the host and data memory of the accelerator, the execution time of CMA-Geyser is much shorter.

Next, evaluation results of JPEG coder is shown. Figure 8 shows the task flow of JPEG decoder. First, the header of JPEG image is analyzed. Then, the image data are divided into the unit of MCU (Minimum Coded Unit), according to the header information. For each MCU, Huffman decoding, the inverse quantization, the inverse DCT and the YUV to RGB conversion are applied in order. Considering the operation time and parallelism, three tasks: inverse quantization, inverse DCT and convert YUV to RGB are accelerated with CMA. For inverse quantization, four data are processed in the PE array. In the inverse DCT, Chen's algorithm is applied, and eight data are processed in parallel. Format transform from YUV to RGB is executed for two data sets are processed because of the limitation of inputs/outputs of the PE array. C-like Program for each application is compiled, mapped and routed by using Blackdiamond[12] compiler.

When three tasks are executed in Cube-1 with only a CMA in order, the overhead of configuration

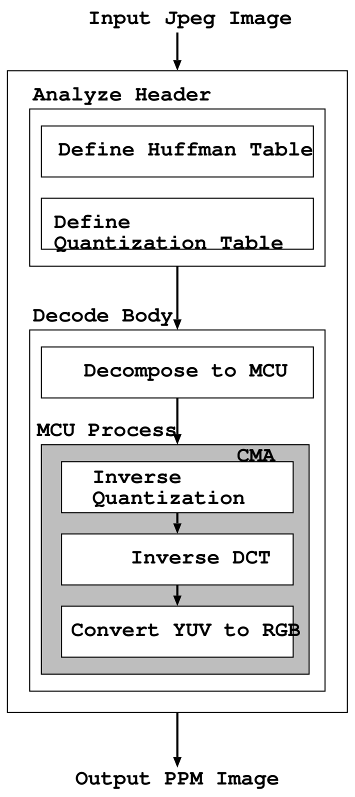


Figure 8: Task flow for JPEG decoder

transfer becomes large and enough acceleration is not achieved. So, in Cube-1, only YUV to RGB data transfer is executed on a single CMA. On the other hand, by using CMA-Geyser, the data setup required for execution can be done quickly by using co-processor instructions. Thus, three tasks can be switched quickly and executed on a single PE array. Figure 9 shows the execution cycles when JPEG decoder is executed.

As shown in Figure 9, the data transfer time of Cube-1 is much reduced compared with the case shown in Figure 7, since it is hidden by computation time in the CMA. DMA transfer mechanism between host processor and CMA is efficiently used in this application. Thus, CMA-Geyser achieved 4.1 times performance as Geyser and 2.7 times as Cube-1, respectively.

5.2.3 Power Consumption

Figure 10 shows the average power consumption of CMA-Geyser and Cube-1 when the gray scale filter is executed. Although the original Cube-1 provides a ring based NoC to connect Geyser and CMA[9], the power consumption of routers and wireless interconnect is eliminated in this evaluation as they are not needed to 1-to-1 interconnection. Here, the power for the interface which can be used to connect host and accelerator directly is included. Thus, the total power consumption of Cube-1 is slightly better than that of CMA-Geyser.

Figure 11 compares the power consumption of CMA-Geyser and Cube-1 when JPEG decoder is executed. In this application, since the computation in Geyser is larger than that in Gray Scale filter, the ratio of Geyser in power becomes larger, but the tendency is almost the same.

Note that since off-loading style accelerators commonly use a network or bus for keeping the scalability. In this case, the power consumption of network must be added.

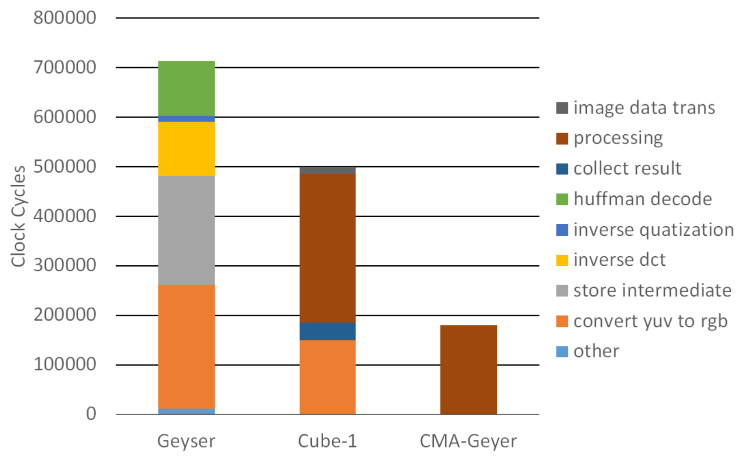
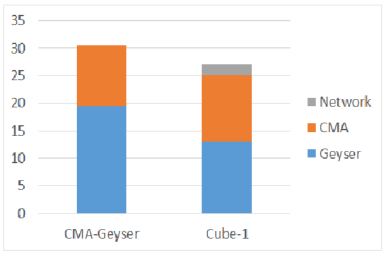


Figure 9: Execution cycles of JPEG decoder



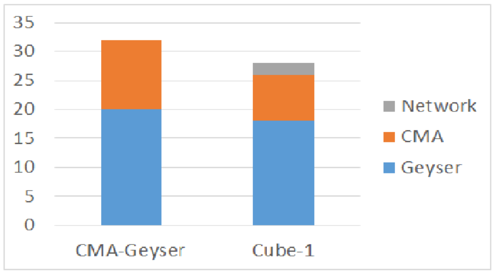


Figure 10: Power Consumption of Gray Scale Execution

Figure 11: Power Consumption of JPEG Decoder Execution

5.2.4 Discussion

As far as we know, there is no previous report on quantitative comparison between co-processor style accelerator and corresponding off-load type. One of reasons is that the fair comparison is difficult since the architecture of accelerators is largely influenced which type is adopted. Fortunately, in CMA, PE array and microcontroller are relatively independent, thus, two types of systems using completely the same PE array can be compared. For generalization of the results, the problem is that Cube-1 uses a specialized through chip network. Although it provides 4Gbps bandwidth which can transfer a word (32bit) per a clock cycle in the block transfer mode, the latency is slightly larger than that of a network in the same chip. Thus, the performance difference between Cube-1 and CMA-Geyser becomes small if the Cube-1 was implemented in a single chip.

The energy efficiency of CMA-1 has been compared with other accelerators[14][4] and it achieved the best energy efficiency in accelerators with 65nm standard CMOS process[10]. Thus, CMA-Geyser can be also the energy efficient system with a CPU and an accelerator if the system size is limited. The drawback of co-processor type accelerator is its poor scalability. While the number of accelerators which can be connected directly to the data cache is strictly limited, Cube-1 can extend its size just stacking chips. Thus, the advantage of CMA-Geyser is limited in a small system. Introducing extendability in the co-processor type is generally difficult but it is our future work.

6 Conclusion

A co-processor design of low power reconfigurable accelerator CMA, called CMA-Geyser is proposed. By introducing co-processor instructions, hardware controller and direct interconnection between data cache, setting up and execution of CMA can be done efficiently.

The design using 65nm CMOS process is compared with an off-loading style multicore system Cube-1. By eliminating the interfaces and routers required in Cube-1, CMA-Geyser reduced 24.8% semiconductor area. When the consuming power of the network is eliminated, the power consumption of Cube-1 is slightly better than that of CMA-Geyser. However, CMA-Geyser achieved about 2.7 times performance of Cube-1 by the efficient data communication between host and the accelerator.

The fine-grained power gating mechanism used in Geyser can be almost directly applied to PE array of CMA-Geyser. Reducing the leakage power of a large PE array by using power gating is another possibility of this work.

Acknowledgment

This work was partially supported by JSPS KAKENHI S Grant Number 25220002. This work is also supported by VLSI Design and Education Center (VDEC), the University of Tokyo in collaboration with Cadence Design Systems, Inc.

References

[1] J. M. Arnold. S5: The Architecture and Development Flow of a Software Configurable Processor. In *Proceedings of the International Conference on Field Programmable Technology (FPT)*, pages 121 – 128, 2005.

[2] C.Ebeling, D.C.Cronquist and P.Franklin. Rapid -Reconfigurable Pipelined Datapath. In Proc. of the FPL 2004, 2004.

[3] D.Ikebuchi, et.al. . Geyser-1: A MIPS R3000 CPU core with fine grain runtime power gating. In *Solid-State Circuits Conference, 2009. A-SSCC 2009. IEEE Asian*, pages 281–284. IEEE, 2009.

[4] F. Clermidy, et al. A 477mW NoC-Based Digital Baseband for MIMO 4G SDR. In *Proc. of ISSCC Dig. Tech. Papers,*, pages 278–279, 2010.

[5] F.J.Veradas, M.Scheppler, W.Moffat, B.Mei. Custom Implementation of the Coarse-Grained Reconfigurable ADRES architecture for multimedia Purposes. In Proc. of International Conference on Field Programmable Logic and Applications (FPL05), pages 106–111, 2005.

[6] H. Amano, Y. Hasegawa, S. Tsutsumi, T. Nakamura, T. Nisimura, V. Tunbunheng, A. Parimala, T. Sano and M. Kato. MuCCRA Chips: Configurable Dynamically-Reconfigurable Processors. In *Proc. of ASSCC*, pages 384–387, Nov. 2007.

[7] H-S.Kim, M.Ann, J.A.Sratton, W.Mei, W.Hwu . ULP-SRP: Ultra Low Power Samsung Reconfigurable Processor for Biomedical Applications. In Prof. of ICFPT 2012, pages 329–334, 2012.

[8] M. Motomura. STP Engine, a C-based Programmable HW Core featuring Massively P aralleland Reconfigurable PE Array: its Architecture, Tool, and SystemImplicatio ns. In Prof. of CoolChips XII, 2009.

[9] N. Miura, et al. A-Scalable 3D Heterogeneous Multicore with an Inductive ThruChip Interface. In *IEEE Micro, Vol.33, No.6*, pages 6–15, 2013.

[10] N.Ozaki, et.al. Cool Mega-Arrays: Ultralow-Power Reconfigurable Accelerator Chips. *IEEE Micro, Vol.31*, pages 6–18, 2011.

[11] T.Callahan, J.Hauser, and J.Wawrzynek . The Garp architecture and C compiler. In *IEEE Computer, Vol.33,*, pages 62–69, 2000.

[12] V. Tunbunheng and H. Amano. Black-Diamond: a Retargetable Compiler Using Graph with Configuration Bits for Dynamically Reconfigurable Architectures. In *Proc. of The 14th Workshop on Synthesis And System Integration of Mixed Information technologies (SASIMI)*, pages 412–419, 2007.

[13] V.Tunbunheng, M.Suzuki, H.Amano. RoMultiC: Fast and Simple Configuration Data Multicasting Scheme for Coarse Grain Reconfigurable Devices. In Proc. of IEEE ICFPT, pages 129–136, 2005.

[14] Y. Tuyama, et al. A 45nm 37.3GOPS/W Heterogeneous Multi-Core SoC. In *Proc. of ISSCC Dig. Tech. Papers,*, pages 100–101, 2010.

[15] Z.Ye, A.Mohovos, S.Hauck, and P.Banerjee . CHIMAERA: A high-performance architecture with a tightly-coupled reconfigurable functional unit. In *Int. Symposium on Computer Architecture*, pages 225–235. ACM/IEEE, 2000.