Mapping Optimization Techniques for Coarse-Grained Reconfigurable Architectures

Takuya Kojima

A thesis for the degree of Ph.D. in Engineering
Under the supervision of **Prof. Hideharu Amano**

Graduate School of Science and Technology
Keio University

February 2021

Acknowledgement

Before anything, I am deeply grateful to my supervisor, Professor Hideharu Amano. His persistent support and guidance gave a scientific perception and thinking to me.

Besides, I would like to thank Professor Hiroaki Nishi, Associate Professor Hiroki Matsutani, Associate Professor Masaaki Kondo, Associate Professor Takahiro Yakoh. Without their careful reviews and invaluable advice, I could not have finished writing this thesis.

Special thanks also to all of the collaborators on my research and my labmates. Abobe all, Dr. Akram Ben Ahmed, Dr. Nguyen Anh Vu Doan, and Dr. Hayate Okuhara encouraged me to go to the doctoral course at Keio University. Besides, I could enhance the qualities of my studies, including research in this thesis, thanks to their constructive advice and interesting discussion with them.

Last but not least, I would like to express my deepest gratitude to my family, who have continuously helped and supported me.

Takuya Kojima
Yokohama, Japan
February 2021

Abstract

Coarse-grained reconfigurable architectures (CGRAs) provide high energy efficiency with word-level programmability rather than bit-level ones such as Field Programmable Gate Arrays (FPGAs). Therefore, the CGRAs are expected to be used for embedded systems, IoT (Internet of Things) devices, and edge-computing. In essence, a CGRA is an array of numerous processing elements (PEs). In order to exploit this abundant computation resource, a compiler for CGRAs has to fulfill more tasks compared that for general-purpose processors. Thus, many studies have proposed optimization methods, especially for application mapping, because the performance and energy efficiency strongly depend on optimization at compile time. However, most work focuses only on performance improvement or resource minimization, although such optimization objectives are not always appropriate when considering various use cases.

In this thesis, an application mapping framework using multi-objective optimization based on a genetic algorithm (GenMap) is proposed. The proposed framework does not depend on any specific architecture and can consider various optimization objectives. Thus, users can easily customize fitness functions for optimization. Firstly, this thesis defines the mapping problem to be addressed by the proposed method. Then, a novel formulation of this problem for the genetic algorithm is presented. However, if the whole of the problem is solved by only the genetic algorithm, it needs prohibitive optimization time to obtain reasonable solutions. Therefore, several heuristics are also proposed and integrated into the framework. Furthermore, this thesis provides aggressive power optimization methods based on a dynamic power model and integer linear program (ILP) formulation for leakage minimization.

Three fabricated CGRA chips are evaluated using the proposed framework. Experimental results show that 15.7% of the wire length is reduced while keeping PE utilization compared to conventional methods. In addition, according to real chip experiments, 12.1-46.8% of energy consumption is reduced, and up to 2x speed-up is archived for several architectures compared to the other two approaches.

Contents

List of Figures

List of Tables

1

Introduction

The performance of microprocessors has been enhanced as the feature size of CMOS transistors is shrunk continuously (see Fig. 1.1). The number of transistors in integrated circuits had doubled every 18 months for about five decades. This trend is known as Moore‘s Law. Besides, Dennard‘s scaling rule [3] had led the designers to increase the performance and the functionality in a single LSI chip while keeping the power consumption. This rule is based on the fact that the shrinking of the feature size of transistors results in reduced supply voltage and higher operational frequency. However, it ignores the presence of the leakage current of the transistors. Then, the scaling was ended around 2005 when the power consumption due to the leakage current cannot be ignored, and the scaling-down of the voltage reaches the limit to ensure a correct operation. Thus, the industry moved into the multi-core era, where the number of cores in a single LSI is increased instead of improving the single-core performance and the operational frequency. Although the scaling of transistor size is kept up even now, it is becoming slowdown compared to Moore‘s Law, and the ending of the scaling cannot be avoided. Thus, innovations not depending on the transistor scaling are urgent.

1.1 Demands for novel architectures

Recently, many architects are moving toward domain-specific architectures (DSAs) to tackle the ending of the scaling law. The DSAs do not provide general-purpose computing but rather a highly efficient one for a limited com-

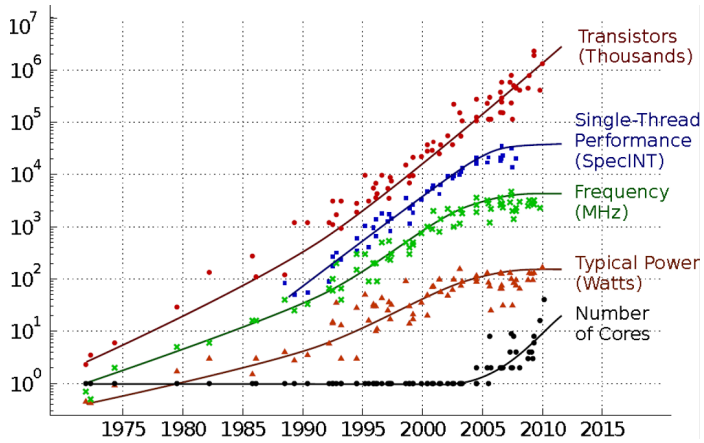


Figure 1.1: Historical trends of processor improvement(source: [1])

putation part such as matrix multiplication. For instance, accelerators for Deep Neural Networks (DNNs) such as Google TPU [4], Intel Spring Crest [5], and Gyrfalcon lightspeeur[6] came out in the market. Each DSA should ideally be implemented as an application-specific integrated circuit (ASIC) chip. However, such killer applications paying off the massive non-recurring engineering (NRE) cost are rare. Besides, the cost, including the designing expense and creating a photomask, becomes increasingly expensive for the recent advanced technologies. In this respect, reconfigurable computing is an essential hardware platform for the DSAs with a smaller NRE cost than the ASIC [7].

Field programmable gate arrays (FPGAs) are the most commonly used reconfigurable device. The major FPGAs are composed of a number of logic blocks consisting of lookup tables (LUTs) and flip-flops [8]. Any logic equations can be realized by using the LUTs. Therefore, users can implement any combinational and sequential circuits on the reconfigurable fabrics as far as they do not require more than the available resources. In addition, digital signal processing (DSP) units and memory blocks are installed even in low-end models for a more efficient computing platform.

Although they can accommodate various kinds of application domains such as database [9] and neural network [10] thanks to their bit-wise reconfigurability, such flexible programmability brings considerable power- and area-overhead. This is because each component is connected with rich interconnections to realize the fine-grained reconfigurability, and the large configuration data has to be stored in memory like on-chip SRAM. Furthermore, the compilation flow is time-consuming, like taking a few days since it contains some complicated steps such as logic synthesis and place-and-route, similar to the

VLSI design process. It causes inefficient application development, resulting in high designing costs.

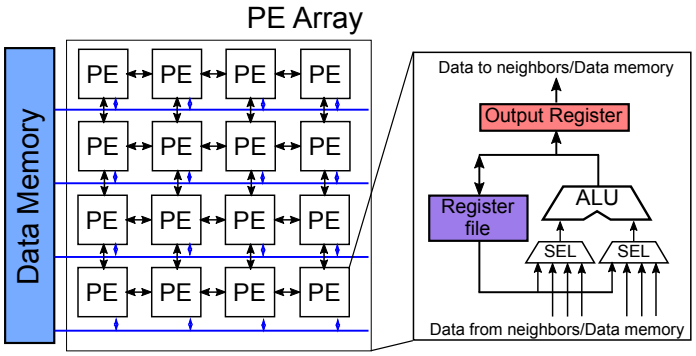


Figure 1.2: PE array of general CGRAs

Coarse-grained reconfigurable architectures (CGRAs) is an alternative solution for the DSAs. CGRAs are optimized in performance and power consumption by supporting data-flow level reconfigurability. In general, they have an array of reconfigurable processing elements (PEs) connected to each other, as shown in Fig. 1.2. The operation for each PE and the interconnection are changed depending on a target application kernel. The PE array is implemented as a part of a well-optimized LSI chip so that CGRAs archive near-ASIC[1] energy efficiency with the programmability. Besides, the coarser reconfigurability contributes to the reduction of configuration data. The reconfiguration time of CGRAs is, therefore, about three orders of magnitude faster than FPGAs [11]. The coarser reconfigurability also relieves heavy compilation tasks of the EDA tool. Thus, CGRA-based designs are often employed as an overlay of FPGAs [12–18].

1.2 Challenges for efficient compilation of CGRAs

The target application kernel can be represented as a data-flow-graph (DFG). The PE array can also be described as a directed graph. The DFG can then be mapped into the PE array graph. The problem of finding a graph-to-graph mapping is known to be NP-complete [19]. Nonetheless, the quality of mapping, including throughput, latency, and energy consumption, strongly depends on the optimization techniques. Therefore, several heuristics have been proposed to optimize the mapping effectively [20, 21]. However, most of them attempt either to improve the speed-up of computation or to save the

[1] application-specific integrated circuit

utilized resources. Therefore, it is impossible to treat a wide variety of use cases by using these uni-objective optimization methods. For instance, there exist unique implementations of CGRAs combined with dynamic voltage and frequency scaling (DVFS) [22,23], body biasing [24–26], emerging non-volatile memory technologies [27–29], and approximate computing [30]. To exploit the advantages of these implementations, a novel optimization technique not dedicated to a specific objective is needed.

1.3 Scope of this thesis and contributions

In order to address the above issue, we propose GenMap, a framework providing a flexible optimization for the CGRA mapping. It searches and optimizes mappings by using a multi-objective genetic algorithm called NSGA-II (Non-dominated Sorting Genetic Algorithm-II) [2]. Thanks to NSGA-II, users can easily customize the optimization objectives, i.e., introducing new ones or remove unnecessary ones depending on their use case. We also present an integer linear program (ILP) formulation to solve a body bias optimization problem to minimize the leakage current. Besides, we develop a dynamic power model considering the glitch propagation effect. For optimizing dynamic power consumption, such a novel model is needed to estimate the power consumption quickly. Then, both the ILP formulation and the dynamic power model are integrated into GenMap. GenMap is agnostic to the specific structure of PE arrays, such as the size of the array, interconnection topology, and fabrication technology. Therefore, it is useful not only for compile-time optimization but also for architecture exploration. However, this thesis focuses only on compile-time optimization.

The contributions of this thesis are as follows:

1. *A generalized ILP formulation for body biasing:* For CGRAs enabling the body biasing, we generalize the optimization problem. It determines the body bias voltages to minimize the leakage power as far as the timing constants are satisfied. As a result of preliminary evaluations, we demonstrate, on average, 17.75% reduced energy consumption compared to the cases without body biasing. (Chapter 4)
2. *Fast estimation of power consumption:* Post-layout simulation is too time-consuming to estimate the power consumption for exploring an optimal mapping. We introduce a simple power model, which is accurate enough to judge which mapping is better. Experiments show that the relative mean error of the model is 10.67-12.91% for different CMOS technologies using parameters fitted based on real chip measurements. Besides, it is more than 10000 times faster than the post-layout simulation. (Chapter 5)

3. *A practical optimization for different architectures:* The target architecture of GenMap can be customized so that we demonstrate our approach can be applied to several different architectures. Compared to the SPKM-based one presented in [31], GenMap achieves 19.8% shorter wire length and needs 20.8% fewer PE columns. In comparison with one of the state-of-the-art mapping tool, CGRA-ME [21], GenMap reduces 15.7% of wire length while keeping the same PE size. In addition, two large DFGs of benchmarks can be mapped by the GenMap, whereas the other two methods fail. (Chapter 6)
4. *An enhancement of routing and other resource mappings:* GenMap includes a routing method based on the A* algorithm with a semi-greedy approach. However, unlike the previous works, we pay attention to path-sharing and routing order to save routing resources. Furthermore, mapping of constant registers and binding of IO ports are formulated as a simple ILP. In this way, the mappability of GenMap is enhanced compared to our previous work. (Chapter 6)
5. *An effective initialization method to generate the first population:* In general, the optimization quality of a genetic algorithm depends on the diversity of the initial population. For that reason, completely randomized initialization is preferred while it can cause slow convergence. In this work, we propose to use a graph-drawing algorithm for the initial set of mappings to improve both the solution quality and convergence speed. (Chapter 6)
6. *Power reduction proven by real chip experiments:* We conduct preliminary experiments to decide some parameters in the power model with the fabricated chips. According to the results, the model can estimate the power consumption with an acceptable deviation from the measured values. Thanks to this model and body biasing, the power optimization achieves 23.4∼46.8% and 12.1∼41.3% of reduction for the energy consumption, 1.54x and 2x of speed-up compared to CGRA-ME and SPKM, respectively. (Chapter 6)

1.4 Structure of this thesis

The rest of this thesis is organized, as shown in Fig. 1.3. In Chapter 2, basic knowledge related to the CGRAs, and the power consumption of CMOS VLSI are explained. Chapter 3 describes the motivation of this work, introducing some related work for mapping methods. Then, the ILP formulation of body bias optimization is presented, and the impact of the body bias control is emphasized in Chapter 4. Chapter 5 shows the effect of glitch propagation in CGRAs, which should be taken into account and presents the estimation

model. In Chapter 6, the proposed mapping algorithm based on the genetic algorithm is described, and a comparison with the other methods is given. Lastly, Chapter 7 summarizes this thesis.

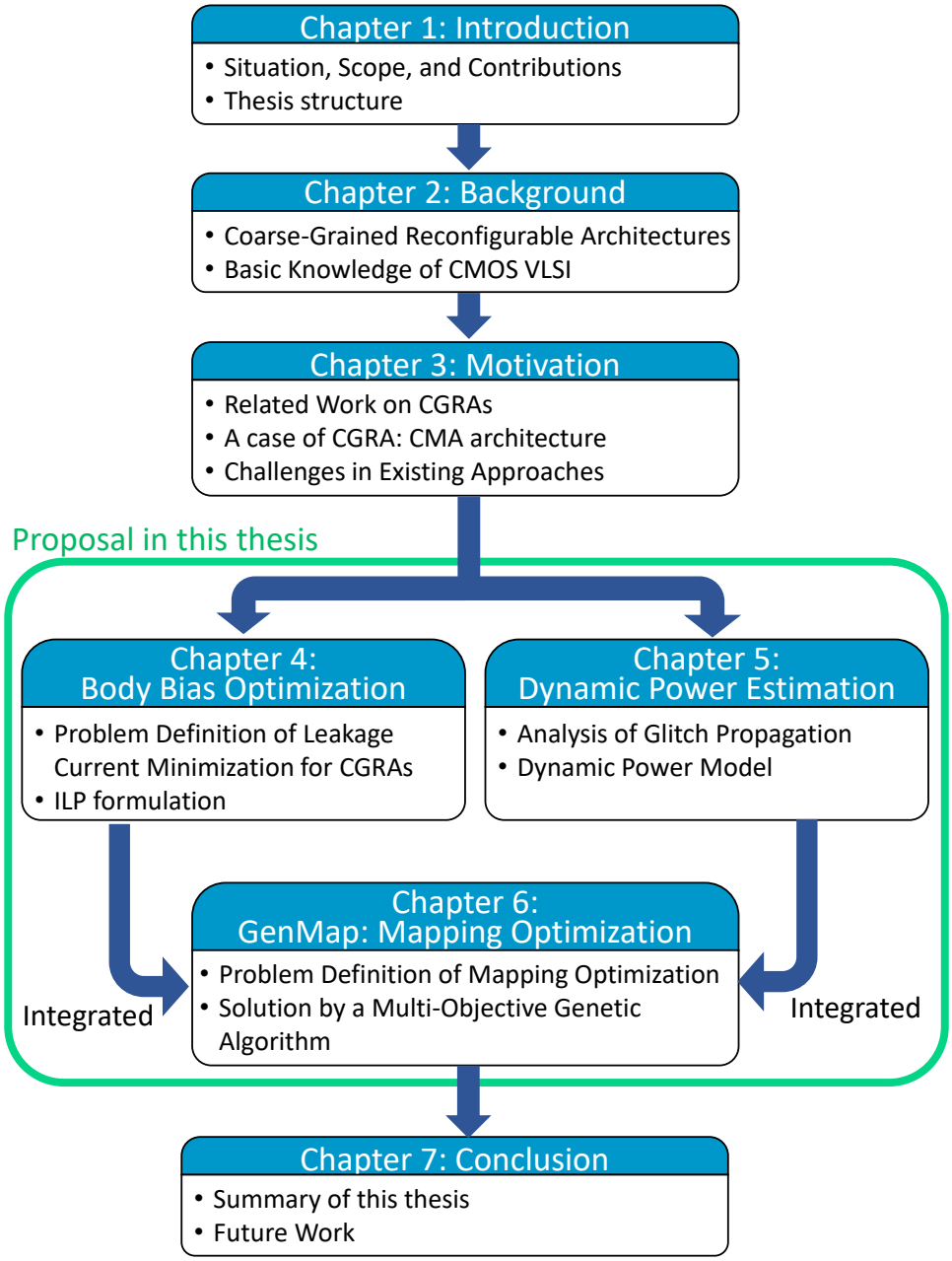


Figure 1.3: Thesis structure

2

Background

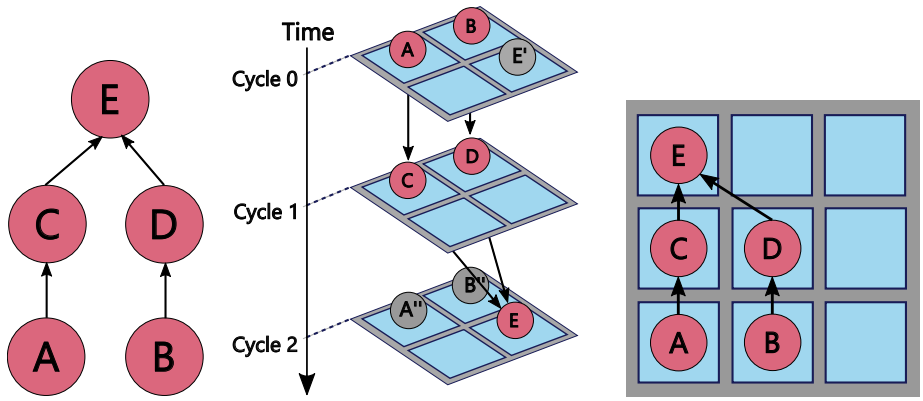
First, this chapter presents the basic knowledge related to this work. Section 2.1 explains an overview of CGRAs, on which this thesis focuses. Especially, the mapping of CGRAs is described in detail. Then, Section 2.2 shows how the power consumption of CMOS VLSI is modeled. The body bias effect is explained in Section 2.3, and the current CMOS technologies are introduced briefly in Section 2.4.

2.1 CGRA: Coarse-Grained Reconfigurable Architecture

2.1.1 A taxonomy of CGRAs

CGRAs are generally utilized as accelerators to execute computational intensive loops instead of general-purpose processors, achieving high energy efficiency. As described in Section 1.2, both the target application and the PE array are represented as directed graphs. The DFG for the application kernel contains nodes, which indicate operations (e.g., addition, multiplication, bit-wise OR, etc.), and the edges, which mean data dependencies between the operational nodes. The compiler has to place the operational nodes on specific PEs and route connections between dependent PEs. This process is formulated as a problem to find an epimorphic subgraph of the PE array graph onto the application DFG [19].

Reconfiguration strategies for CGRAs can be classified into 1) temporal



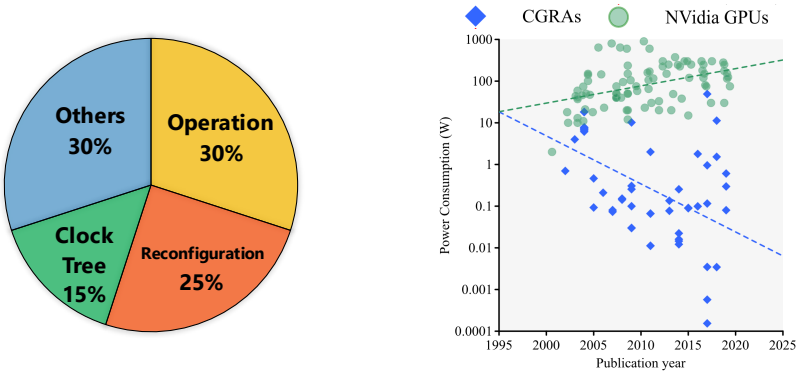
(a) Example of DFG (b) Temporal Mapping (c) Spatial Mapping

Figure 2.1: Reconfiguration of CGRAs

mapping and 2) spatial mapping. In the case of temporal mapping, the PE array graph is extended for several cycles, called time-extended CGRA (TEC). Then, the DFG is mapped into the TEC, as illustrated in Fig. 2.1(b). This style of CGRAs needs to change the configuration of the PE array cycle-by-cycle. A family of CGRAs, such as DRRA [34], MorphoSys [35], ADRES [36], and FloRA [37], follows this mapping policy. In general, multiple DFGs are executed in a software pipeline manner to utilize the PE array efficiently. For the software pipeline, the throughput is determined by the initial interval (II), which is an interval cycle between two iterations. In Fig. 2.1(b), $E^{'}$ represents a node in the previous iteration, and $A^{''}$ and $B^{''}$ are those in the next iteration so that II is equal to 2. Therefore, temporal mapping algorithms usually attempts to minimize the II.

However, this reconfiguration style consumes a large amount of dynamic power. Fig. 2.2(a) shows power breakdown for a temporal mapping CGRA reported in [32]. Around a quarter of the power is consumed for the dynamic reconfiguration. The CGRAs have grown for a few decades, reducing the power consumption seen in Fig. 2.2(b). In contrast to the CGRAs, GPUs have increased the power consumption to boost the clock frequency, that is, computing performance. Although the clock frequency of CGRAs has also been increased, it is no so much as the GPUs. The CGRAs, therefore, focuses on energy efficiency rather than computing performance.

Nevertheless, the improvement of energy efficiency is mostly assisted by the scaling of transistors. Now that we are reaching the limit of transistor scaling, an architectural breakthrough to archive higher energy efficiency is



(a) Power breakdown for a temporal mapping CGRA [32]

(b) Trend of power consumption (source: [33])

needed to keep the growing trend of CGRAs. In this respect, another type of CGRAs based on spatial mapping is promising. It performs a task-by-task reconfiguration to cut down the dynamic power overhead. In contrast to the temporal mapping, the spatial mapping does not switch the configuration while handling a task in an application, as Fig. 2.1(c) describes.

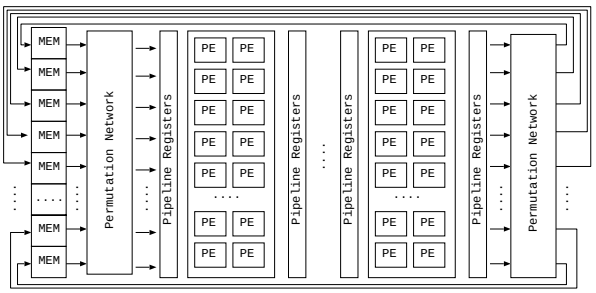


Figure 2.2: Overview of SF-CGRAs

Straight Forward CGRAs (SF-CGRAs), which belong to the spatial mapping CGRAs, contains a simplified PE array for high energy efficiency. SF-CGRAs are designed to make the dataflow on their PE array straightforward, as shown in Fig. 2.2. During a single task of an application, the PE array configuration is static so that computation is driven by sending input data to the PE array. Registers in the PE array behave as pipeline registers. Permutation networks (e.g., crossbar) between the data memory and the input/output of the PE array provide flexible data transfer. For instance, Piperench [38], Kilocore [39], XPP [40], S5 engine [41], REMUS [42], RSPA [43], DT-CGRA [44],

and RHP-CGRA [45] are classified into SF-CGRAs.

Unlike the temporal mapping CGRAs, when the target DFG contains more operational nodes than the size of PEs, the compiler for SF-CGRAs has to divide the application kernel into sub-tasks, each of which fits the PE array. It causes extra reconfiguration overhead in the execution time due to context switching. Thus, fast reconfiguration techniques like [46] are employed to mitigate the overhead. Besides, optimizing task partitioning is another research topic for SF-CGRAs, and some heuristics are proposed [47,48]. General graph partitioning algorithms for directed acyclic graphs (DAGs) like [49,50] can also be utilized for this purpose. This thesis focuses on spatial mapping towards more energy-efficient computing, combined with aggressive power optimization.

2.1.2 Advantage of CGRAs over FPGAs

As introduced in Section 1.1, the CGRAs sacrifice such a fine granularity of reconfiguration as FPGAs. Instead, the CGRAs has the definite advantage of the smaller reconfiguration overhead in terms of area, power consumption, and reconfiguration time, compared to the FPGAs. Moreover, the coarser reconfigurability contributes to a more simplified compilation.

Fig. 2.3 explains the difference in the general compilation flow between the FPGAs and the CGRAs. The target application for the FPGAs is designed with either a hardware description language (HDL) such as Verilog HDL or a high-level language such as C/C++ and OpenCL. In the latter case, the source codes specify the behavior and algorithm of the circuit, and then, they are compiled into HDL codes (a.k.a. high-level synthesis). The HDL code is translated into an optimized gate-level netlist (i.e., logic synthesis), similarly to VLSI design flow. A combinational circuit on most FPGAs is realized with a number of k-input LUTs [8]. Therefore, the logic gates of netlist are grouped for each group to fit in the k-input. This process is called technology mapping and is usually based on a cut-enumeration algorithm [51]. Likewise, the grouped netlist (i.e., LUT-level netlist) is clustered because recent FPGAs are composed of logic blocks (LBs) containing the multiple LUTs. Then, each node of the clustered netlist (i.e., LB-level netlist) is placed onto a corresponding LB, and dependent LBs are routed through the interconnection network of the FPGAs. It is not rare that this compilation flow takes several days for a large-scale circuit. Besides, if the generated implementation violates any design rules or timing-constraints, the user has to modify the design. Hence, this complicated compilation flow causes inefficient application development. The efficiency is crucial, especially for an application domain where the requirement is updated constantly.

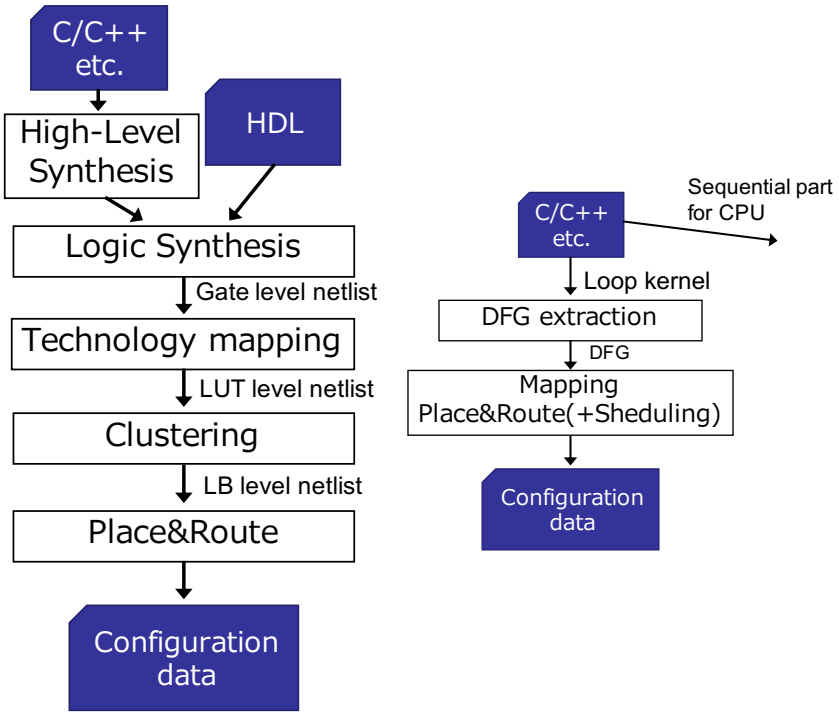


Figure 2.3: Comparison of compilation flow between FPGAs (left) and CGRAs (right)

On the contrary, the CGRAs need fewer compilation steps than FPGAs because of the coarser granularity of reconfiguration. Like the high-level synthesis of FPGAs, the application is written in C-based description language [52], C language with pragma directives [53], and OpenCL [54]. The target kernel to be executed on the PE array is extracted as the DFG like in [55]. As mentioned above, the CGRAs are usually used as an accelerator so that the other part of the program, including control of the CGRA, is handled by a host processor. The extracted DFG is mapped into the PE array. For the temporal mapping CGRAs, the compiler has to determine the time-slot for each operational node (i.e., scheduling) as well as placement and routing. Various kinds of mapping algorithms are proposed. They are introduced in Chapter 3 as the related work of this thesis.

CGRA-based designs are often implemented on the FPGAs [12–18], and the target applications are compiled for the CGRAs. This approach is called an overlay of the FPGAs. Even for a domain-specific use such as deep learning, an FPGA vendor provides sophisticated hardware designs like CGRAs with a software stack including compiler and libraries [56]. Therefore, the CGRAs

are increasingly promising.

2.2 Power consumption of CMOS VLSI

For understanding energy-efficient computing, we have to figure out how power is consumed in the CMOS VLSI and know the fundamental property of the power dissipation. Generally, the power consumption can be classified into two types: 1) dynamic power (P_{dyn}) and 2) static power (P_{st}). Therefore, the total power consumption is denoted by the following equation.

$$P_{\text{total}} = P_{\text{dyn}} + P_{\text{st}} \tag{2.1}$$

2.2.1 Dynamic power

The dynamic power is attributed to the switching of signals. When a transition of a gate output from low to high (rising) or from high to low (falling), the parasitic capacitance of the transistor is charged or discharged, and then, the power is dissipated. The amount of power (called switching power) is calculated as follows.

$$P_{\text{switching}} = \alpha C f V_{DD}^2 \tag{2.2}$$

where α is switching activity, C is the capacitance, f is clock frequency, and V_{DD} is the power supply voltage.

In addition to the switching power, short circuit current between supply and ground is another factor of the dynamic power consumption. It is also caused when the transistor switches.

Eq. (2.2) suggests that lowering the supply voltage V_{DD} contributes to the reduction of dynamic power consumption drastically. However, it degrades the speed of signal transition. DVFS leverages the trade-off between power consumption and performance while considering the status of workloads.

2.2.2 Static power

The static power is consumed regardless of the circuit switching. For process technologies larger than the 90-nm node, the static power consumption can be neglected since the dynamic power is the dominant component of the total power consumption. However, the static power consumption had been increased exponentially beyond the 65-nm process. It consequently became an obstacle to Dennard‘s scaling rule [57].

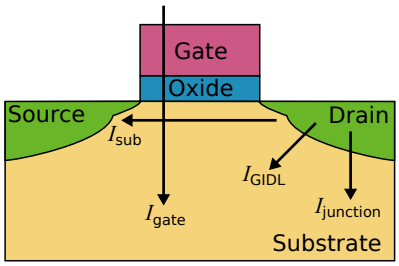


Figure 2.4: Mechanisms of leakage current

The static power is mainly caused by four kinds of leakage current: 1) subthreshold leakage (I_{sub}), 2) gate leakage (I_{gate}), 3) junction leakage (I_{junction}), and 4) gate-induced-drain-leakage GIDL (I_{GIDL}), as illustrated in Fig. 2.4. Then, static power is described as follows:

$$P_{\text{st}} = (I_{\text{sub}} + I_{\text{gate}} + I_{\text{junction}} + I_{\text{GIDL}})V_{DD} \tag{2.3}$$

Subthreshold leakage

The subthreshold leakage is the major contributor to the total static power consumption. This current is passed between the source and drain of the CMOS transistor. Its characteristic is explained by the following equation:

$$I_{\text{sub}} = I_{\text{off}} 10^{\frac{V_{gs}+\eta(V_{ds}-V_{DD})-k\gamma V_{sb}}{S}} (1 - e^{\frac{-V_{ds}}{v_T}}) \tag{2.4}$$

where I_{off} is the subthreshold current at $V_{gs} = 0$ and $V_{ds} = V_{DD}$, V_{gs}, V_{ds}, V_{sb} are respectively gate-source, drain-source, and source-body (substrate) voltages, η and k_γ are respectively coefficients related to the DIBL (drain-induced barrier lowering) and body bias effect, and S is the subthreshold slope [58]. As the threshold voltage decreases, the subthreshold current increases exponentially. In Dennard's scaling rule, the threshold voltage was reduced linearly so that the subthreshold current brought about a serious problem for the short channel transistors. However, it can be reduced by the body bias effect, which controls the threshold voltage by changing the voltage between source and body (V_{sb}). It is described in Section 2.3.

Gate leakage

In addition to the subthreshold leakage, the short channel transistors face the challenge of growing gate leakage current. This current is associated with

electron tunneling through the gate oxide. It is modeled as the following equation:

$$I_{\text{gate}} = WA\left(\frac{V_{DD}}{t_{\text{ox}}}\right)e^{-B\frac{t_{\text{ox}}}{V_{DD}}} \tag{2.5}$$

where W is the gate width, A and B are parameters depending on the process technology, and t_{ox} is the thickness of the gate oxide. As the equation indicates, its effect strongly depends on the thickness of the oxide. As scaling the process technologies, the thickness is also reduced. However, the thickness below 20 $\mathring{A}$ results in a significant increase in the gate leakage current. It can be suppressed by introducing high-k dielectrics for the insulator instead of SiO_2 gate oxide [59] while the manufacturing complexity is increased.

Junction leakage and GIDL

Even though the p-n junctions between the substrate and the source/drain are reverse-biased in normal operation, it still causes a small leakage current. Especially if both the drain and source are heavily doped, band-to-band tunneling (BTBT) accounts for a large amount of the junction leakage [60].

GIDL is attributed to the high field effect in a gate-drain overlap region due to the biased gate. As the thickness of the oxide decreases or V_{DD} becomes high, the electric field is enhanced, and then, GIDL is also increased [60].

2.3 Body bias control

As explained in the previous section, the subthreshold leakage depends on the threshold voltage (V_t). In order to control the threshold voltage, body bias control can be available. The effect of body bias is explained by the following equation:

$$V_t = V_{t0} + \gamma(\sqrt{\phi_s + V_{sb}} - \sqrt{\phi_s}) \tag{2.6}$$

where V_{t0} is the threshold voltage at $V_{sb} = 0$, γ is a coefficient of body bias, and ϕ_s is the surface potential at threshold [58].

Changing threshold voltage has an impact on the gate dealy as well as the subthreshold leakage. α-Power law [61] estimates the gate delay τ as follows.

$$\tau = k\frac{CV_{DD}}{(V_{DD} - V_t)^{\alpha}} \tag{2.7}$$

As the source-body voltage V_{sb} becomes higher than zero, the threshold voltage is increased, resulting in the reduction of subthreshold leakage, whereas the

gate delay is increased, that is, degrades the operational frequency of the circuit. This status is called the reverse body bias. On the contrary, in the case of $V_{sb} < 0$, the transistor works at a higher operational frequency at the expense of the subthreshold leakage. This situation is referred to as the forward bias. Likewise, the state without such a control of the body bias (i.e., $V_{sb} = 0$) is called the zero bias and provides a regular operation. In this way, the possibility of a trade-off between the performance and the leakage current can be exploited after the chip fabrication.

However, the control range for the bulk CMOS is limited because a strong reverse bias instead causes an increase in the junction leakage. Silicon on insulator (SOI) overcomes this drawback, as explained later.

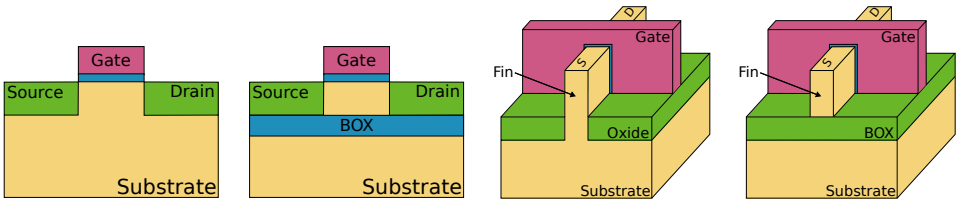
Some studies present multiple body bias domains in a single LSI chip [62, 63]. In other words, different domains can respectively operate with different threshold voltages. Thereby, the reverse bias is available for domains not including time-critical modules as far as the timing constraints are satisfied. Conversely, the forward bias to boost the operational frequency can be applied only to time-critical modules, minimizing the cost of leakage increase. Some FPGAs [64–66] and CGRAs [26, 67] change the body bias voltage for each domain depending on the configuration to save the power consumption.

In the case of multi-VDD design to use low voltage (VDDL) and high voltage (VDDH), level shifters are needed between the voltage domains. In contrast, body bias domain partitioning does not require them since the high level of signals is common for each body bias domain. Thus, power- and area-overheads regarding the domain partitioning is small.

2.4 Recent CMOS technology

The typical bulk CMOS has several challenges in the transistor scaling. As introduced previously, an issue regarding the leakage power consumption is a factor of them. Besides, the short channel transistor needs a highly doped channel in the traditional scaling, whereas it causes a variation of electrical properties such as the threshold voltage. It loses the possibility of performance scaling since lowering V_{DD} is restricted by the worst case of the threshold voltage. Therefore, a couple of innovations taking the place of the bulk CMOS has occurred.

Fig.2.5 shows the novel transistor structures. The first (Fig. 2.5(b)) is called a fully depleted silicon on insulator (FD-SOI). The transistor layer of FD-SOI is the same as the bulk CMOS. However, it is isolated from the substrate by a thin buried oxide (BOX) layer. Thanks to the BOX layer, the junction leakage due to BTBT can be eliminated. As a result, better control



(a) Bulk CMOS (b) Planar FD-SOI (c) Bulk FinFET (d) SOI-FinFET

Figure 2.5: Transistor structure for each technology

of the body bias effect is provided compared to the bulk CMOS. The BOX layer can reduce the parasitic capacitance between the source and drain. Moreover, the transistor layer is thin enough to fully deplete the channel under the gate. Thus, the short channel effect, which the bulk CMOS suffers from can be mitigated. ST Microelectronics 28-nm [68], GlobalFoundries 22-nm [69], and Renesas SOTB 65-nm[70] process technologies are categorized into the FD-SOI.

The second (Fig. 2.5(c)) is a multi-gate transistor, so-called FinFET. This type of transistor has a thin vertical fin, and it forms the channel. The gate is wrapped around the channel, that is, multiple gates are formed around the channel, as shown in this figure. It contributes to a fully depleted channel and offers better electrostatic control of the channel compared to the bulk CMOS. Consequently, faster switching and smaller leakage current are archived.

Intel's 22-nm FinFET called Tri-Gate transistor is the first one for a commercial CPU [71]. Then, most of the advanced process technologies below 22-nm, including the latest node, TSMC 5-nm [72] and Samsung 5-nm [73], belong to a family of FinFET. Besides, a hybrid type integrating the FinFET structure on SOI like Fig. 2.5(d) as in [74] is proposed. Although the body bias control can work for the bulk FinFET and SOI-FinFET, the possibility of the trade-off is limited compared to the FD-SOI [75].

2.4.1 SOTB: a case of FD-SOI

A part of the studies in this thesis is based on some chips implemented with the SOTB process, which is an FD-SOI technology. As illustrated in Fig. 2.6, the transistor of SOTB forms a triple-well structure. Besides V_{DD} and V_{SS}, it has two additional terminals, VBN and VBP, for body bias voltages of NMOS transistor and PMOS transistor, respectively. VBN for NMOS transistors is given to p-well. That is, if $VBN = 0$, the transistor works with a normal threshold level since V_{sb} is equal to 0. If a reverse bias ($VBN < 0$) is given, the threshold voltage increases, and thus, the leakage current is reduced while

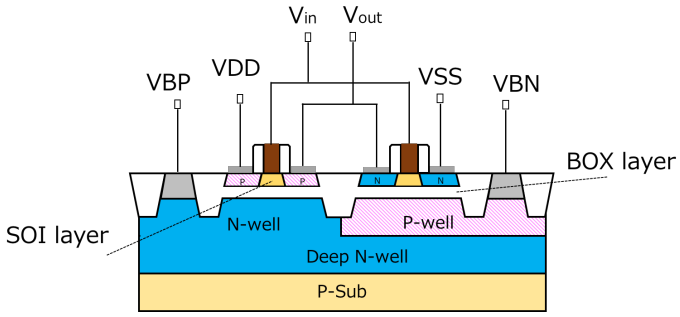


Figure 2.6: Cross-sectional view of the SOTB MOSFET

the delay time is extended, as explained in Section 2.3. On the contrary, a forward bias ($VBN > 0$) decreases the threshold voltage, which boosts the operational frequency with an increase in the leakage current. In the case of PMOS transistors, VBP is given to the n-well. Here, the transistors are formed on a thin BOX layer, as shown in Fig. 2.6. Therefore, in this case, $VBP = V_{DD}$ means the zero bias. When $VBP > V_{DD}$, this corresponds to the reverse bias, while $VBP > V_{DD}$ is for the forward bias.

Here, the bias voltage is equally given to both the NMOS and PMOS so that $VBP + VBN = V_{DD}$ is satisfied. Hereinafter, we, therefore, express the level of body bias solely with the value of VBN. Table 2.1 summarizes the body bias effect on the SOTB process.

Table 2.1: Effect of body bias control

	VBN	VBP	Performance	Leakage
Reverse bias	< 0	$> V_{DD}$	Degraded	Decreased
Zero bias	$= 0$	$= V_{DD}$	Normal	Normal
Forward bias	> 0	$< V_{DD}$	Enhanced	Increased

3

Motivation

This chapter discusses previous studies related to the optimization method for CGRAs. Then, we explain the details of a CGRA chosen as a case study in this thesis. Finally, we conclude few studies can provide general-purpose optimization.

3.1 Related work on optimization techniques for CGRAs

There are several possibilities of optimization in CGRAs, and various kinds of techniques have been proposed to date. They can be classified into three categories depending on the phase when the optimization is applied: 1) design-time, 2) runtime, and 3) compile-time.

3.1.1 Design-time optimization

Design-time optimization usually aims to improve power consumption and area of the CGRA fabric at an HDL-level and a circuit-level. For instance, RADISH [76] provides an automated design of more efficient PEs, which consist of fully customized ALUs. It employs a genetic algorithm to reduce power consumption and area as far as the programmability is kept enough for a target application domain. In [24], the size of body bias domains on a CGRA has been analyzed, focusing on leakage power reduction and area overhead. It also uses a genetic algorithm in order to decide an optimal bias voltage for each

domain. Besides, many works provide CGRA templates to explore the design space and to find the most energy-efficient architecture [77–80]. Compared to the other two optimization phase, design-time optimization is more tolerant regarding simulation duration because it is not needed to be carried out once the design is fixed. Nonetheless, it cannot take care of the target application adaptively. Although comprehensive design space exploration is indispensable for DSAs, too specialized architectures might degrade the flexibility.

3.1.2 Runtime optimization

Runtime optimization is generally a part of just-in-time (JIT) compilation. It has the advantage of leveraging the runtime status of the CGRA. For multi-threaded CGRAs like [81], resources on the CGRA such as PE array and data memory are shared among the multiple applications (threads). In other words, available resources for an application are different depending on the time of execution. Thus, an application binary compiled in advance does not always be executable due to resource conflict. [82] and [83] transform the pre-build application mapping into another one dynamically to improve resource efficiency. Although runtime optimization is the most adaptive, it has to be completed in the order of milliseconds or microseconds. Therefore, the optimization time is the top priority, and limited optimization is consequently provided.

3.1.3 Compile-time optimization

Unlike the multi-threaded CGRAs, most of CGRAs assume the PE array is occupied by a single application. Even in the case of handling multiple applications, context switching is employed. Therefore, many studies, including this thesis, focus on compile-time optimization, that is, offline mapping optimization. This is because the efficiency of the CGRAs strongly depends on the quality of the mapping.

As explained in Section 2.1, the temporal mapping includes scheduling as well as placement and routing. However, the graph representation, TEC in Fig. 2.1(b), integrates the scheduling problem into the placement and routing. Then, most methods are based on Iterative modulo scheduling (IMS) [84], which is a software pipelining technique to schedule the instructions with as a small initial interval as possible. These algorithms start with minimum II (MII), which is a lower bound of II due to resource constraint and data dependencies between iterations. If it fails in the scheduling, then the target II is incremented. It is repeated until a valid scheduling is obtained. Thereby, they attempt to minimize the II.

For the temporal mapping CGRAs, a classical approach using simulated annealing (SA) has been proposed with DRESC [85]. However, the simulated annealing causes a long compilation time. EMS [86] addresses the long compilation time by efficient routing strategy and also improves the performance. Since temporal mapping usually suffers from routing between PEs which depend on each other, many sophisticated routing methods have been proposed. EPIMap [19] formulates the problem generally and enhances the routing ability considering re-computation. REGIMap [87] extends EPIMap [19] to exploit register files distributed over the PE array as routing resources. MEMMap [88] utilizes memory units as well as the register files. Then, RAMP [20] adaptively finds routing by taking every routing resources into account. TAEM [89] is similar to RAMP but improves the compilation time significantly.

However, Zhao *et al.* [90] argue that there still exists an inefficient process of mapping attributed to the integration of scheduling into the placement and routing. Then, they propose a reorganized algorithm to reduce the compilation time and also to improve the II.

The above methods, except for DRESC, are deterministic approaches so that they tend to get stuck in local optima. CRIMSON [91] is a randomized approach of IMS to tackle the issue. Despite a simplified algorithm to be completed within a millisecond, it can obtain mappings with almost the same II as RAMP.

In addition to the above methods, various heuristics are employed like a force-directed approach [92], quantum-inspired evolutionary algorithm [93], a formulation based on graph minor theory [94]. HyCUBE [95] is a CGRA which has an enhanced interconnection network, and a novel routing algorithm to leverage the rich network has also been presented.

However, they all do not attempt to reduce power consumption. Although the researches presented in [22, 96] consider dual-VDD to reduce unnecessary power consumption, they give priority to improve the performance (II).

Instead of performance improvement, optimization for spatial mapping usually aims at saving utilized resources since it is supposed to improve the possibility of obtaining a valid mapping and shorten the computation latency. SPKM [31] is a graph drawing based approach to save the number of PE rows. It suggests applying power gating to unused rows. It solves two types of ILP, 1) column-wise scattering and 2) row-wise scattering, iteratively until a valid mapping is found while increasing the number of used rows. DFGNet [97] and RLMap [98] are based on deep learning. The former employs CNN (Convolutional Neural Network) only to reduce the compilation time, whereas the latter aims to save rectangular area covering the utilized PEs by using Deep Q-Network (DQN). PolyMap [99] employs a genetic algorithm to search for good loop tiling and optimizes the mapping in the same way as SPKM.

Table 3.1: Summary of mapping algorithms

Methods	Reconf.	Objective	Category	Power reduction
DRESC[85]	Temporal	min. II	SA	×
EMS[19] EPIMap[87] REGIMap[87] MEMMap[88] HyCUBE[95] RAMP[20] TAEM[89] Zhao *et al.*[90] Chen *et al.*[94] Fell *et al.*[92]	Temporal	min. II	Deterministic	×
CRIMSON[91]	Temporal	min. II	Randomized IMS	×
Gu *et al.*[22] Xu *et al.*[96]	Temporal	min. II	Deterministic	Dual-VDD[1]
Lee *et al.*[93]	Temporal	min. latency	QEA	×
CGRA-ME [102–104]	Temporal or Spatial	min. routing resources	SA or ILP	∆[2]
Canesche *et al.*[105]	Temporal or Spatial	min. buffer	Graph-based traversal	×
SPKM[31]	Spatial	min. PE rows	ILP	∆[3]
Zhou *et al.*[101][4]	Spatial	min. wire len.	ACO	×
DFGNet[97]	Spatial	success in mapping	CNN	×
RLMap[98]	Spatial	min. used PE	DQN	×

[1] Higher priority to improvement of performance (II) than power consumption
[2] The authors imply the power reduction depending on the used parameters.
[3] The authors assume the power supply to unused PEs is gated
[4] Placement algorithm is not included

In general, the routing problem of CGRAs is defined as Steiner tree problem similar to VLSI design [100]. The Steiner tree problem is also known as the NP-complete problem so that Zhou *et al.*[101] have proposed a routing algorithm for the spatial mapping CGRAs based on a metaheuristic, ant colony optimization (ACO). However, it contributes to up to 6% of wire length reduction compared to a greedy approach.

CGRA-ME [102] is a framework that can model a wide range of CGRAs and able to evaluate them. Besides, it includes application mappers using

simulated annealing [102] and with ILP formulation [21]. They can treat both temporal and spatial mapping for minimizing the utilized resources. Nowatzki *et al.*[106] provides a similar ILP formulation with a more general architecture description than CGRA-ME. Although such ILP formulations guarantee an optimal solution, the optimization time increases exponentially for large scale problems. Thus, for large application DFGs and large PE arrays, it is difficult to solve the ILP within an acceptable compilation time. Instead of the optimal solution, Canesche *et al.* [105] have proposed a sophisticated graph traversal method to find a mapping efficiently. It also supports both temporal and spatial mappings. It makes hundreds of instances, each of which traverses the graph in a different order. Then, it picks the best solution from these instances. Therefore, in spite of its greedy heuristic, it can archive the same mapping quality as CGRA-ME within a millisecond.

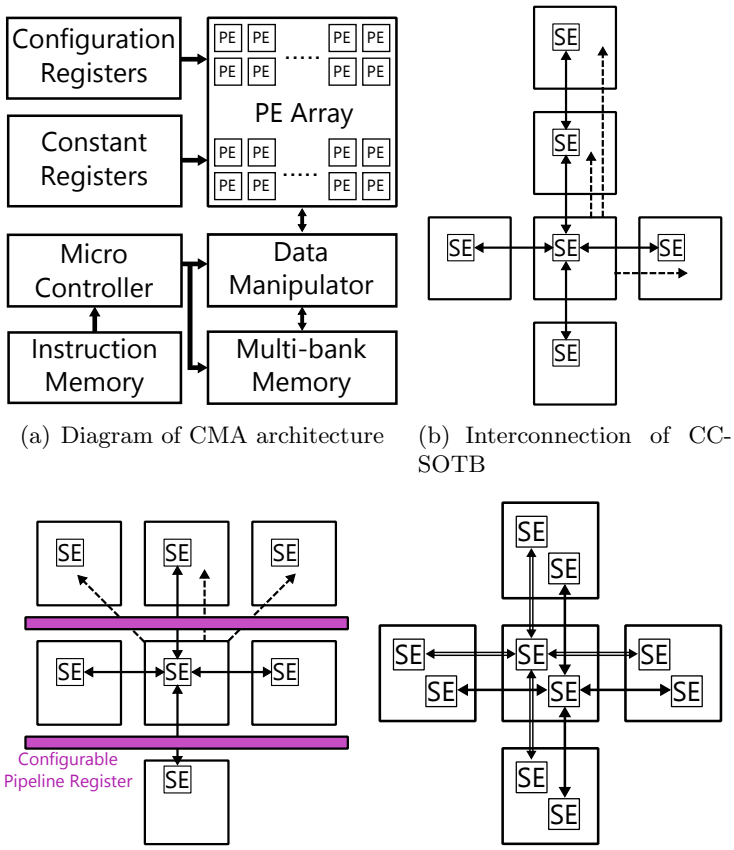
Table 3.1 summarize the mapping algorithms. In the case of the spatial mapping CGRAs, the reduction of resources like SPKM, RLMap, and CGRA-ME's mappers leads to energy saving implicitly. There still exist possibilities to reduce power consumption further by applying power reduction techniques such as body biassing. Nevertheless, all of them cannot consider such aggressive power optimization.

3.2 Cool mega array: a case of CGRA

3.2.1 Architecture overview

Cool mega array (CMA) is one of SF-CGRAs and the target architecture of this thesis. Like other CGRAs, CMA has a PE array, as illustrated in Fig. 3.1(a), and it works following the spatial mapping fashion. However, unlike others, each PE does not have a register file to hold intermediate results. In summary, the PE array is composed of a combinational circuit to cut down the dynamic power consumption drastically since a clock signal for the PE array is not needed. According to [67], it achieves a quite high energy efficiency: more than 700 MOPS (Million Operations Per Second) per milliwatt. Such a PE array brings about a long critical path delay, and thereby system clock frequency would be degraded. In order to prevent it, the PE array of CMA is designed to be a multi-cycle execution unit. The number of cycles is programmable, depending on the target application.

Fig. 3.1(a) shows the other modules of the CMA. The micro-controller is a 16-bit tiny RISC micro-controller, which manages the data transfer between the PE array and the multi-bank data memory according to the instruction codes. A data manipulator is placed between the PE array and the data memory. It is composed of several multiplexers to support a flexible data



(a) Diagram of CMA architecture (b) Interconnection of CC-SOTB

(c) Interconnection of CC-SOTB2 (d) Interconnection of NVCMA

Figure 3.1: CMA architecture and PE interconnections

transfer. When multiple data from the data memory are sent with interleaving access to the input of the data manipulator, it transfers each data to a specified output port according to a transfer table. In general, the spatial mapping is less flexible than the temporal mapping, yet CMA can execute various types of computation thanks to the flexible control by the micro-controller and the data manipulator.

When the micro-controller sends input data from the data memory to the PE array, computation on the PE array starts automatically. After a few cycles, the micro-controller writes the computation result back to the data memory. In this thesis, we refer to the former operation as "fetch" and to the latter one as "gather." Fig. 3.2 explains how the micro-controller controls the execution on the PE array. In this example, the execution cycle is set to

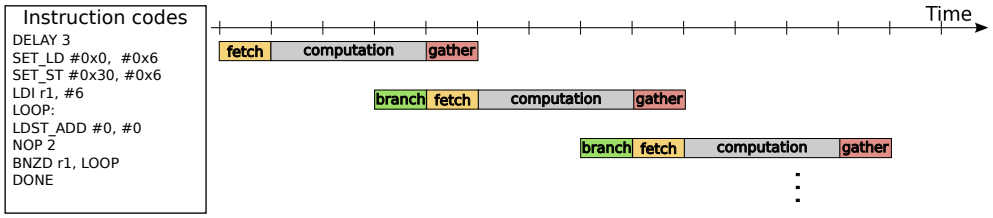


Figure 3.2: 3-cycle execution on non-pipelined PE array

three by the "DELAY" instruction. "SET_LD" and "SET_ST" instructions respectively set configurations such as memory address for the fetch and gather operations. The fetch and gather operations fuse together into an instruction "LDST_ADD." The fetch operation is executed as soon as the "LDST_ADD" is issued. In contrast, execution of the gather operation is delayed like Fig. 3.2. Besides, the micro-controller supports several instructions commonly used in general-purpose RISC processors, such as branch instructions and arithmetic instructions.

3.2.2 Existing implementations

Table 3.2: Architectural and implementation features

Chip	CC-SOTB[24]	CC-SOTB2[107]	NVCMA[28]
Array size	12×8	12×8	8×8
Channel of SE	1ch	1ch	2ch
Direct link	Included	Included	Not included
Pipeline	None	Variable pipeline	None
Body bias control	Controlled (3 domains)	Controlled (5 domains)	Not controlled
Fabrication	Renesas SOTB 65-nm	Renesas SOTB 65-nm	Bulk CMOS Smaller than 65-nm

Several types of CMA architectures have been proposed and fabricated using different process technologies listed in Table 3.2. In this work, we consider three chips: 1) CC-SOTB [24], 2) CC-SOTB2 [107] and 3) NVCMA [28]. Each of them has different features in the PE array. Table 3.2 describes the differences, and the interconnection topology for each architecture is illustrated in Fig. 3.1. PEs for each architecture includes at least one switching element (SE) for an island-style interconnection network. The PEs of NVCMA have two SEs so that two channels of interconnections are available, while those

of CC-SOTB and CC-SOTB2 provide a single channel. Instead, they have another type of interconnection called a direct link. Neighboring PEs are connected by the direct link like, as shown with dashed lines in Fig. 3.1(b) and Fig. 3.1(c).

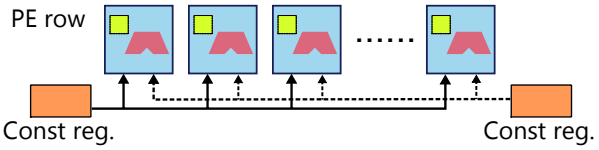
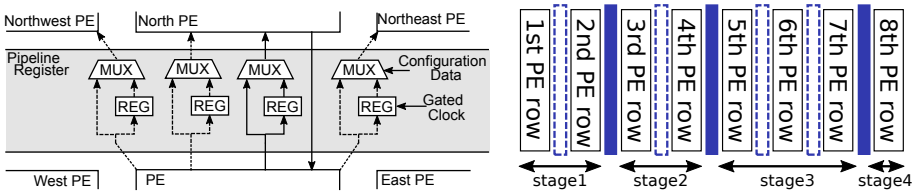


Figure 3.3: Connectivity of constant registers

Constant registers supply constant values used for the computation to the PE array. For all the architectures, sixteen of them are available. However, connectivity between the constant registers and the PEs is restricted. As shown in Fig. 3.3, two constant values are provided for each PE row. If a PE row requires more than two constant values, it needs to borrow unused constant registers from other rows through the interconnection network.



(a) Configurable pipeline registers (b) Example of a pipeline structure

Figure 3.4: Implementation of pipelined PE array on CC-SOTB2

CC-SOTB2 has a limited number of configurable pipeline registers placed between every PE row. The pipeline registers yield a variable pipeline structure according to their configuration, as illustrated in Fig. 3.4(a). When a pipeline register is activated, it works as a usual flip-flop. On the contrary, a deactivated register just passes the input data to the upper PE row without latching. Such a register bypassing is commonly used in CGRAs[14, 98, 108, 109]. In this case, the clock signal for the unused flip-flops is gated to reduce unnecessary dynamic power dissipation. Fig. 3.4(b) shows an example of a pipeline structure configuration. In this example, three pipeline registers are activated, and thus, a 4-stage pipeline is formed. Note that each pipeline stage can also be assumed as a multi-cycle unit.

Fig 3.5 describes the execution control for the 4-stage pipelined PE array. In this example, each stage takes two cycles so that the computation results

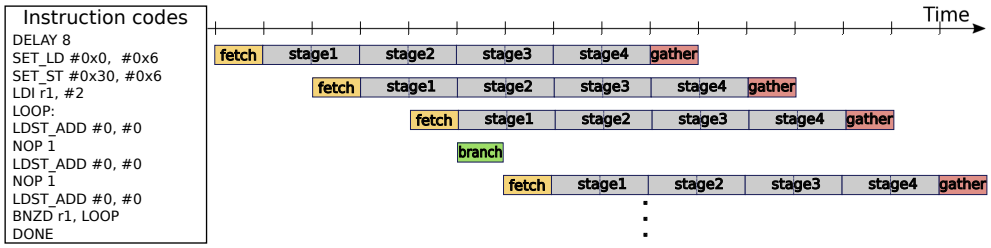


Figure 3.5: 8-cycle execution on 4-stage pipelined PE array

are yielded after eight cycles. Compared to non-pipelined PE arrays like CC-SOTB and NVCMA, it contributes to higher throughput and shorter critical path length on the PE array. However, too many activated pipeline registers bring about a large amount of dynamic power consumption. Therefore, the pipeline structure should be optimized, considering the target application, operational frequency, and performance requirements.

CC-SOTB and CC-SOTB2 have been fabricated with Renesas SOTB introduced in Section 2.4. Besides, circuits of both fabricated chips are divided into several body bias domains. As described in Table 3.2, CC-SOTB and CC-SOTB2 have three and five domains, respectively. The PE array of CC-SOTB2 is divided into four domains, as follows: 1-5th, 6th, 7th, and 8th PE rows. For CC-SOTB, it is merged into one single domain.

Since the data transfer with the micro-controller and the computation in the PE array are executed in an overlapped manner, as shown in Fig. 3.2, their performance should be balanced. In order to keep the balance, other parts of CC-SOTB2, including the micro-controller, belongs to a different domain from the PE array. Besides, the configuration registers of CC-SOTB is separated from the domain of the micro-controller. However, this thesis focuses only on controlling the body bias voltages of the PE array.

As for NVCMA, it has been fabricated with another technology. However, the details of the technology are confidential due to the industry secrecy of our collaborator. Although the NVCMA contains non-volatile memories, considering them is out of the scope of this work.

3.2.3 Combination of body bias control and variable pipeline

For the variable pipelined PE array, the delay time for each stage should be balanced since the throughput is determined by the longest stage delay. Therefore, even if the other stages complete the computation in a shorter delay time, it does not increase the throughput. Although the variable pipeline can roughly adjust the balance of the delay time, there still remains a variation of

delay time between each stage. The body bias control offers a possibility to eliminate such a variation, that is, to reduce the leakage power consumption by leveraging the trade-off, as explained in Section 2.3. When a timing constraint from users is not severe, it also brings more reduction of the leakage power. This is because the reverse bias can be applied even to the slowest stage. In contrast, the forward bias can improve the delay time of the bottleneck stage to satisfy a strict time constraint when high performance is required.

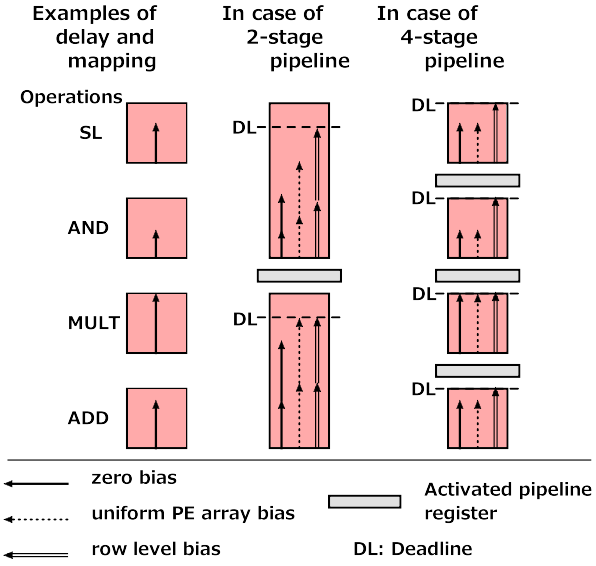


Figure 3.6: Row-level body bias control with pipeline registers (2 and 4 stages)

Nevertheless, a uniform bias on the whole PE array, such as CC-SOTB, produces a limited effect. Fig. 3.6 explains the limitation of the uniform control. It shows two examples of pipeline structure for the same application mapping. The length for each arrow indicates the delay time of the corresponding PE. In the case of the 2-stage pipeline, the uniform bias can reduce the leakage power for both stages, compared to the zero bias (i.e., without body bias control). However, it misses the possibility of more reduction for the second stage since a more strong reverse bias causes a timing violation in the first stage. To make matters worse, the uniform control cannot adjust the balance of delay time due to the second stage.

To alleviate the limitation, we consider a row-level body bias domain for the PE array, as also shown in Fig. 3.6, to finely balance the delay time of each pipeline stage. This can allow more flexible choices on the bias voltages compared to a uniform bias. With a row-level design, each row is implemented with its own body bias domain and receives its own bias voltage. Since all

the pipeline registers are outside of the PE array domain and implemented in the same domain as the micro-controller, they can work at the same clock frequency. The delay scaling problem in the clock tree does not have to be taken into account since all the flip-flops and the whole clock tree are implemented in a single body bias domain. From the layout point of view, the overhead of separating the body bias domains is negligible because the same macro corresponding to a single row is used regardless of whether body bias domains are separated or not because of a common layout policy. For the use of macro cells, isolation cells, and well separation are needed in any case. Therefore, the eventual overhead to consider would come from a generator which can deliver multiple body bias voltages. Although each PE row needs its own body bias generator, we can employ significantly low overhead on-chip generators like [110, 111].

Using a row-level body bias control, we can apply a reverse body bias to every stage whose delay is shorter than the largest one until they become (nearly) equal. Conversely, a forward body bias can be supplied only to stages whose delay is longer than the shortest one. Even if the delay of each stage is not exactly the same, the row-level control leads to a more balanced pipeline compared to the uniform case.

Though CC-SOTB2 is based on the same concept as the row-level body bias, the domains of the actually fabricated chip are not row-level due to a restriction on the number of I/O pins. However, in order to study the benefit of the row-level control, Chapter 4 considers generalized body bias domains, including the row-level and the uniform control. Then, the real chip evaluation in Chapter 6 is based on the actually implemented domains.

3.3 Challenges in the previous optimization approaches

As introduced previously, most mapping algorithms are based on a specific optimization objective, such as computation performance. Nonetheless, considering the fact that some CGRAs, including CMA, have unique characteristics, these mapping algorithms lose a generality of optimization. As a result, they can not exploit such a unique characteristic. Besides, users do not always require maximum performance. Thus, the optimization objectives should be customizable by the users. Moreover, only a few methods [21, 95] treat the dedicated switching elements inside PEs, and the others ignore it.

In Chapter 4, this thesis first proposes an algorithm to adaptively decide an optimal body bias voltage for each domain according to the performance required from the users. In addition, it emphasizes the impact of the row-level body bias control with evaluation results. Then, for more aggressive

power optimization, a dynamic power estimation technique is needed. Thus, Chapter 5 presents a dynamic power model and demonstrates its accuracy for the three different chips of CMA as described above. Finally, in Chapter 6, this thesis proposes GenMap, an optimization framework not depending on a specific optimization objective. GenMap provides a multi-objective optimization, that is, optimizing different objectives simultaneously by using a multi-objective genetic algorithm called NSGA-II. As a case study, this thesis shows the above power optimization techniques can be deployed in GenMap thanks to the generality of optimization objectives, and the energy reduction is archived compared to the previous approaches.

4

Body bias optimization

In this chapter, we propose a body bias optimization method, considering the variable pipeline structure like CC-SOTB2 to save power consumption. Assuming that the pipeline structure and the body bias voltages are controlled simultaneously, there are several possibilities of trade-offs, as shown in Table 4.1.

Table 4.1: Trade-off between performance and power associated with body bias control and variable pipeline

	Number of pipelined stage	
	large	small
Performance	high	low
Dynamic power of register and clock tree	increases	decreases
Dynamic power of the glitches	decreases	increases
	Body bias voltage	
	forward bias	reverse bias
Performance	low	high
Static power	decreases	increases

For instance, we can observe that the power consumption induced by

glitches decreases as the number of activated pipeline registers increases. Glitches are unneeded signal transitions caused by the different delay times between inputs of the PEs. Without pipeline registers, they are propagated to the PEs in the upper rows and will therefore imply an increase in consumption. Using pipeline registers allows to limit this propagation, and thus the induced power consumption, but at the cost of an overhead related to the registers and the associated clock tree. This propagation effect is detailedly discussed in Chapter 5. This example shows that more advanced analyses are required to assess the trade-off possibilities between performance and power consumption, which both depend differently on the pipeline registers configuration and the body bias control.

4.1 Problem definition

On the basis of the aforementioned trade-off information, we can define the problem as the following bi-objective optimization problem: given an application, how to optimize the power consumption and the performance of the PE array with choices on simultaneously the body bias voltages and the pipeline structure.

The equations required to model this problem can be formulated as follows:

$$\begin{aligned} P_{\text{st}} &= \sum_{i=0}^{N_{\text{domain}}-1} P_{\text{leak},i}(VBN_i) \\ &\quad + P_{\text{leak,reg}} + P_{\text{leak,clk}} && (4.1) \\ \boldsymbol{preg} &= \{preg_0, preg_1, ..., preg_{N_{\text{reg}}-1}\} && (4.2) \\ preg_k &= \begin{cases} 1 & \text{if the } k\text{-th} \\ & \quad \text{pipeline register is activated} \\ 0 & \text{otherwise} \end{cases} && (4.3) \\ P_{\text{dyn}} &= f_{\text{req}} \times \Bigg(E_{\text{comb}}(\boldsymbol{preg}) \\ &\qquad + \sum_{k=0}^{N_{\text{reg}}-1} (E_{\text{reg}} + E_{\text{clk}}) preg_k \Bigg) && (4.4) \\ D_l &= \sum_{\substack{v \in l\text{-th} \\ \text{datapath}}} D_v(VBN) && (4.5) \end{aligned}$$

where:

- VBN_i is the body bias voltage supplied to i-th domain,
- $P_{\text{leak},i}(VBN)$, $P_{\text{leak,reg}}$, and $P_{\text{leak,clk}}$ are the leak power of respectively PEs in the i-th domain on VBN, a pipeline register, and the clock tree,
- $preg_k$ represents the configuration of the k-th pipeline register,
- $\boldsymbol{preg}$ is a vector whose elements are $preg_k$ and expresses the pipeline structure of the PE array, assuming the PE array has totally N_{reg} pipeline registers,
- P_{dyn} and P_{st} are respectively the dynamic and static power of the PE array (considering body bias control and pipeline structure),
- $E_{\text{comb}}(\boldsymbol{preg})$, E_{reg}, and E_{clk} are the dynamic energy consumption of respectively the combinational circuits (PEs), a pipeline register, and clock tree, and
- D_l and $D_v(VBN)$ are the delay time of respectively the l-th datapath and the node v (ALU or SE) supplied with VBN; D_l is therefore calculated as the sum of the delays caused by the nodes located in the l-th datapath.

Then, the optimization problem is to minimize the sum of P_{dyn} and P_{st}. Note that E_{comb} depends on the pipeline structure (i.e., $\boldsymbol{preg}$) because of the glitch propagation effect. Besides, the static power is regarded as the subthreshold leakage since the body bias basically affects only the subthreshold current. This thesis assumes implementations with FD-SOI technologies so that other types of leakage current are negligible. In the case of the other CMOS technologies, the amount of the other leakage currents is constant. Thus, even if these values are not negligible size, this problem formulation is valid.

Although this chapter focuses on the PE array with multiple body bias domains and variable pipeline, this formulation can be applied to the non-pipelined PE array (e.g., CC-SOTB) as a case of $N_{\text{reg}} = 0, E_{\text{reg}} = 0, E_{\text{clk}} = 0$. Likewise, it is also valid for the uniform body bias domain (i.e., $N_{\text{domain}} = 1$).

4.2 Preliminary analysis

The parameters in the above model, such as P_{leak} or $P_{\text{comb}}(\boldsymbol{preg})$, are obtained by several simulations. The used environments are shown in Table 4.2. The design used in the simulations is based on a chip layout of CC-SOTB2. It is a 6mm×3mm chip designed with the same environments as shown in the table.

Table 4.2: Simulation environments for preliminary evaluation

Design	Verilog HDL
Process Library name	Renesas 65-nm LPT-8
Synthesis	Synopsys Design Compiler 2016.03-SP4
Place and route	Synopsys IC Compiler 2016.03-SP4
Temperature	25 °C
Delay and leak power simulation	Synopsys HSIM 2012.06-SP2 V_{DD}: 0.55, 0.65, 0.75, 0.85 V
Dynamic power simulation	Synopsys Prime Time 2012.06-SP2 VDD: 0.55 V
Behavioral simulation	Cadence NC-Verilog 10.20-s131

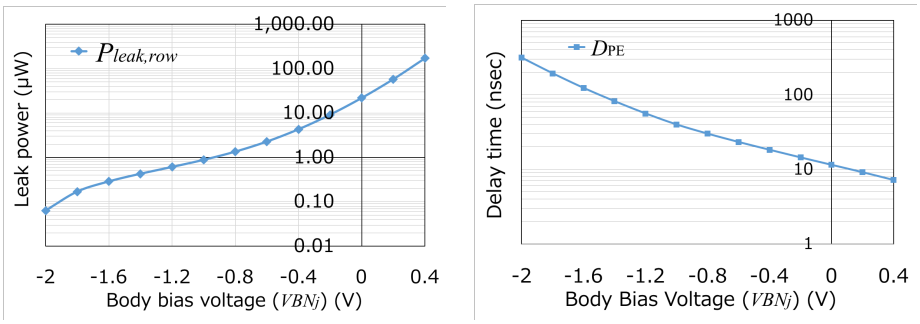
Considering the row-level body bias control, $P_{\text{leak},i}$ corresponds to the leakage power of a PE row, and the number of domains N_{domain} is equal to that of rows, that is, eight for CC-SOTB2. All PE rows are considered to be implemented with the same macro. Therefore, each $P_{\text{leak},i} (0 \leq i \leq 7)$ is regarded as an identical value. Hereinafter, $P_{\text{leak,row}}$ is used for all PE row instead of $P_{\text{leak},i}$.

$P_{\text{leak,row}}$ and D_v are simulated for each value of V_{DD} (0.55, 0.65, 0.75, and 0.85 V) with HSIM by changing the body bias voltages (VBN) every 0.2 V from -2.0 V to 0.4 V. Here, we treat VBN as a discrete value. This is because typical body bias generators like [111] are based on a digital-analog-converter so that available voltages are discrete. Hence, we assume N_{bb} types of body bias voltages (in the above case, $N_{\text{bb}} = 13$). $P_{\text{leak,row}}$ is calculated as an average of two input patterns, all inputs being set either to low level or high level. The simulated $P_{\text{leak,row}}$ for each VBN with $V_{DD} =$ 0.55 V is shown in Fig 4.1(a). For an ALU, D_v depends on an assigned operation. For example, the delay time of integer multiplication is generally longer than the bit-wise logic operations such as bit-wise AND. It also depends on its input values. Hence, the input values which can provide the critical path for each operation are determined using the reports from IC compiler. The D_v for each VBN with ADD operation and $V_{DD} =$ 0.55 V is shown in Fig 4.1(b). Both results clearly demonstrate the trade-off between the performance and

Table 4.3: Simulated applications

Application	Description
gray	24 bit (RGB) gray scale
sepia	8 bit sepia filter
af	24 bit (RGB) alpha blender
sf	24 bit (RGB) sepia filter
dct	8-point DCT

static power described in Table 4.1. Similar tendencies of the delay time for the other operations are observed (the full results are shown in Appendix A).



(a) Leak power per PE row (V_{DD}=0.55 V) (b) Delay time of PE (ADD operation, V_{DD}=0.55 V)

Figure 4.1: Examples of simulation results

E_{comb}, E_{reg}, and E_{clk} depend on the running application. Five applications are simulated, as shown in Table 4.3. For each of them, the dynamic power (at the combinational circuit, registers, and clock tree) are simulated at a certain frequency and a V_{DD} of 0.55 V using PrimeTime, and E_{comb}, E_{reg}, and E_{clk} are obtained dividing each dynamic power by the frequency. The values for other V_{DD} voltages used in our simulation (0.65, 0.75, or 0.85 V) are scaled from those of 0.55 V based on Eq. (2.2).

An analysis of the solution space shows that its size is $2^7 \times 13^8$. Indeed, CC-SOTB2 can configure $2^{N_{\text{reg}}} = 2^7 = 128$ patterns of pipeline structure since it is possible to choose to use it or not for each of the seven registers. For the row-level body bias, given that each of the eight rows in the PE array can select among thirteen possible voltages, there are $N_{\text{bb}}^{N_{\text{domain}}} = 13^8$ possibilities of body bias assignment. As a test, for one pipeline structure, it takes 3 hours to elicit and simulate all these possibilities on a 1.6GHz dual-core Intel Core i5 with 8GB of DDR3 RAM.

Given the size of the solution space and the complex formulation of some equations (e.g., P_{dyn}), techniques such as metaheuristics could be applied since they have been used successfully for similar cases, providing interesting solutions in an acceptable amount of time. However, a close examination of the problem shows that it is possible to formulate this problem as an 0-1 ILP (0-1 Integer Linear Problem), which guarantees optimality, unlike metaheuristics. Indeed, when the pipeline structure is fixed, that is, ***preg*** is fixed, P_{dyn} is constant. Therefore, with the remaining equations being linear, it is possible to formulate this problem as only 128 ILPs (one for each pipeline structure). Moreover, its bi-objective nature can be simplified by considering the performance as a timing constraint. Since the design focus of the CMA is low power, the problem can be re-formulated as follows: given an application and a fixed pipeline structure, how to optimize the power consumption of the PE array while reaching required performance with choices on the body bias voltages. This methodology is then repeated for each pipeline structure, as summarized in Fig. 4.2. The leak power is minimized by an ILP, while the dynamic power is optimized by the ILP iterations.

4.3 ILP model

The ILP can then be formulated as follows:

$$isVBN_{ij} = \begin{cases} 1 & \text{if the } i\text{-th domain} \\ & \quad \text{is set with } j\text{-th } VBN \\ 0 & \text{otherwise} \end{cases} \tag{4.6}$$

$$\min P_{\text{st}} = \sum_{i=0}^{N_{\text{domain}}-1} \sum_{j=0}^{N_{\text{bb}}-1} P_{leak,i,j}\ isVBN_{ij} \tag{4.7}$$

subject to

$$\sum_{j=0}^{N_{\text{bb}}-1} isVBN_{ij} = 1 \quad \forall j = \{0, 1, \ldots, N_{\text{domain}} - 1\} \tag{4.8}$$

$$D_l = \sum_{\substack{v \in l\text{-th} \\ \text{datapath}}} \sum_{j=0}^{N_{\text{bb}}-1} D_{v,j}\ isVBN_{ij} \tag{4.9}$$

$$D_l \leq D_{req}, \quad \forall \text{ datapath } l \tag{4.10}$$

$$isVBN_{ij} = \{0, 1\}, \quad \forall i = \{0, 1, \ldots, N_{\text{domain}} - 1\}, \quad \forall j = \{0, 1, \ldots, N_{\text{bb}} - 1\} \tag{4.11}$$

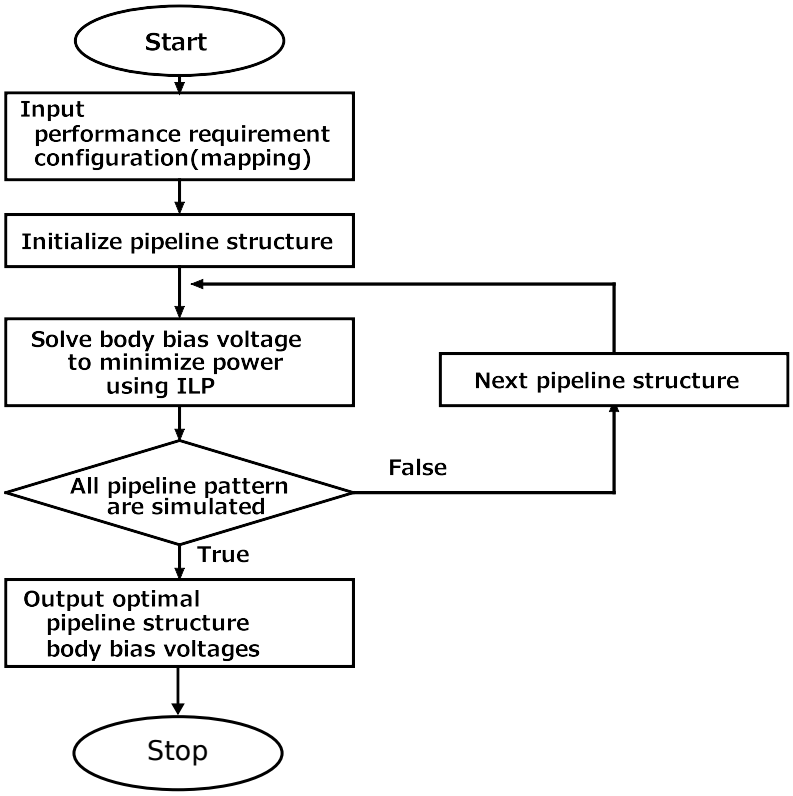


Figure 4.2: Algorithm flow-chart to find an optimal body bias assignment and pipeline structure

where a set of $isVBN_{ij}$ are decision variables, the constraint (4.8) ensures that the same body bias voltage is supplied to the PEs belonging to the same domain, D_{req} is the inverse of the required frequency f so that the constraint (4.10) guarantees that any timing violation does not occur. It is worth noting that $P_{leak,reg}$, and $P_{leak,clk}$ are constant (not controlled by body bias) and therefore do not have to be included in the objective function.

4.4 Evaluation

4.4.1 Optimization results

To analyze the possibilities of the proposed method, we perform the leakage power optimization assuming several different performance requirements and for each application described in Section 4.2. Two examples of results are presented in Table 4.4 and Fig. 4.3, where the performance is described as

the number of executed operations per second, the simulated application is *gray*, and V_{DD} is set at 0.55 V. These results clearly demonstrate that the optimal pipeline structure and body bias voltages are different depending on the requirement, and the proposed method can solve them exactly. Each ILP can be solved within 0.1 seconds. Thus, compared to the Brute-force search as in the preliminary analysis in Section 4.2, the proposed method reduces the optimization time by six orders of magnitude.

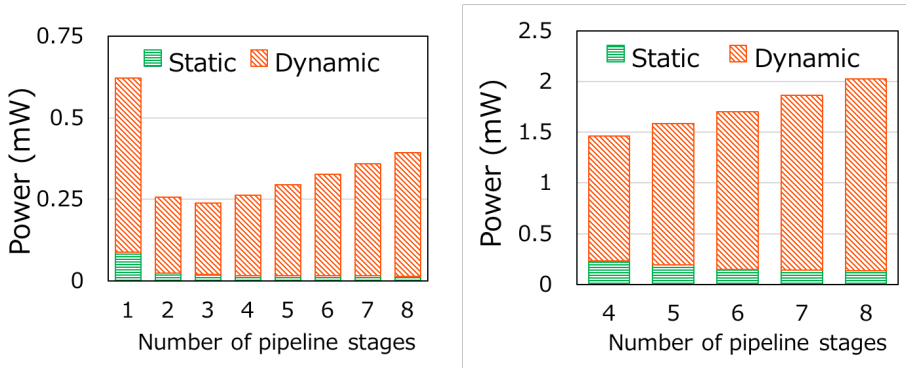
Table 4.4: Examples of optimization results (*gray*)

Requirement of 7.8 $\times 10^8$ ops/s

Pipeline register i	0	1	2	3	4	5	6	
$preg_i$	0	0	1	1	0	0	0	
Row number	0	1	2	3	4	5	6	7
VBN	-0.8	-0.8	-1.0	-0.6	-1.2	-1.2	-1.4	-1.4

Requirement of 3.9 $\times 10^9$ ops/s

Pipeline register i	0	1	2	3	4	5	6	
$preg_i$	0	0	1	1	1	0	0	
Row number	0	1	2	3	4	5	6	7
VBN	0.0	0.0	-0.2	0.2	0.2	0.0	0.0	-0.2



(a) Req. of 7.8×10^8 ops/s (b) Req. of 3.9×10^9 ops/s

Figure 4.3: Minimized power for each pipeline stages

In the case of 7.8 $\times 10^8$ operations/sec, the result of $preg_i$ indicates that the number of an optimal pipeline is three. As shown in Fig. 4.3(a), when low performance such as 7.8 $\times 10^8$ operations/sec is requested, static power is extremely low due to a strong reverse bias such as -1.0 V, and the dynamic power accounts for most of the consumption. The single-stage structure implies a large dynamic power because of the glitch propagation due to the large

combinational circuit.

On the contrary, when high performance such as 3.9 $\times 10^9$ operations/sec is requested (Fig. 4.3(b)), static power is not as small as in the low-performance case. Indeed, in order to achieve the requirement, the third and fourth PE rows are given a forward bias such as 0.2 V, which causes an increase in static power. In Fig. 4.3(b), the power when the number of pipeline stages is 1, 2, and 3 is not shown since such a low stage pipeline cannot satisfy the performance requirement even if a strong forward bias is applied.

4.4.2 Performance and energy reduction

Next, we analyze the impact of the row-level body bias control. To evaluate the energy reduction achieved by the row-level control, we simulate other policies of body bias control as a comparison basis:

- control for the whole PE array (uniform)
- without body bias control (zero bias)

Fig. 4.4 shows that using the body bias control allows reaching higher achievable performance for *gray application* thanks to applying the forward bias. For instance, without body bias control (zero bias), the performance cannot exceed 3.12 $\times$ 10^9 operations/sec, whereas both the uniform control and the proposed method allow higher performance values. The results of the other application can be found in Appendix B.

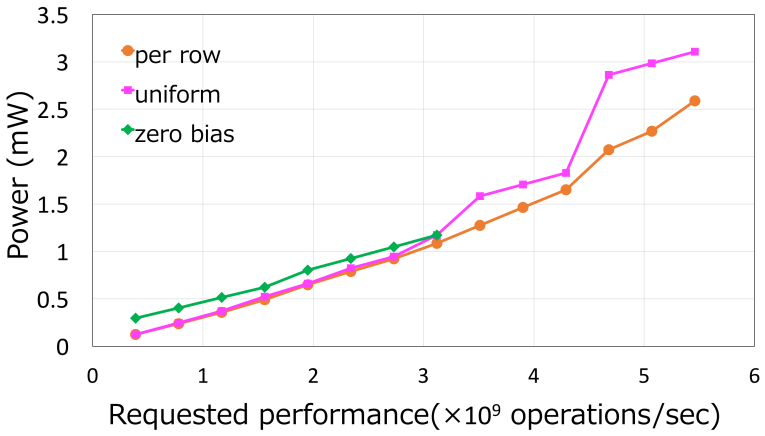


Figure 4.4: Comparisons between each method ($V_{DD} = 0.55$ V)

Furthermore, unlike the uniform control, the row-level control can keep a steady increase in the power even at high performance. This can be explained

by the need to apply a forward bias to the whole PE array in order to meet the requirement in the uniform case, which results in a drastic power increase. On the contrary, with the row-level control, the forward bias has to be applied only to the row which causes a bottleneck in the critical path. In Table 4.5, optimized values of VBN with both controls at the highest performance point are shown. By using values of leakage power, which are shown in Fig 4.1(a), the leakage power of the row-level control is calculated to be 0.6270 mW, while in the uniform case, the leakage power is 1.373 mW. However, the dynamic power of the proposed method and uniform control are 1.951 mW and 1.724 mW, respectively. Therefore, the power reduction due to the row-level control reaches about 500 μW.

Table 4.5: Optimized VBN_i in the case of 5.46 $\times 10^9$ operations/sec (*gray*)

Uniform control

i	0	1	2	3	4	5	6	7
VBN_i (V)	0.4							

Row-level control

i	0	1	2	3	4	5	6	7
VBN_i (V)	0.2	0.4	0.0	0.4	0.4	-0.2	0.0	-0.8

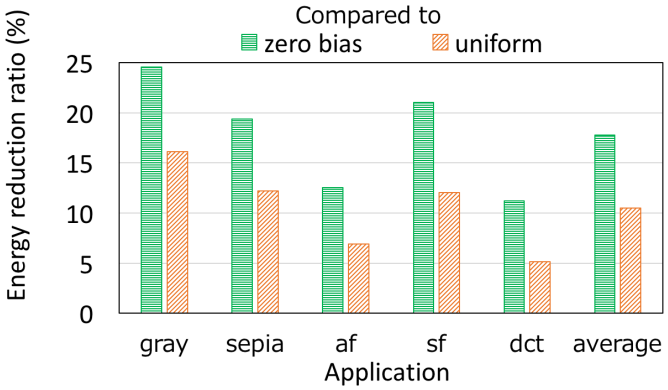


Figure 4.5: Energy reduction ratio by the row-level control for each application ($VDD = 0.55$ V)

To compare the energy between different methods, the average energy of all performances is calculated for each application and for each method. Fig. 4.5 illustrates the reduction ratio of the energy between the row-level control and the other two policies. With the row-level control, it is possible to achieve an energy consumption of up to 24.5% and 16.1% lower than respectively the zero bias and the uniform cases (the best reduction with *gray* application).

On average, the consumption is 17.75% and 10.49% lower than respectively the zero bias and the uniform cases.

Figure 4.6: Static power reduction ratio by the row-level control (VDD = 0.55 V, *gray*)

To discuss the effectiveness of the row-level body bias control, we also focus on the static power. Fig. 4.6 shows the reduction ratio of the static power where the *gray* application is simulated. The row-level control results in a reduction of 91.9% and 65.8% when compared with respectively the zero bias and uniform cases. Fig. 4.6 also reveals that the static power is not always lower than the uniform case. For example, when 1.56×10^9 operations/sec is requested, the static power in the case of row-level control is higher by 90.9%. Nevertheless, the total power with the proposed method is always lower. At such performance, as an increase in the static power occurs, a change in the optimal pipeline structure is also observed. If the pipeline structure or the body bias control are considered independently, it is then impossible to adjust the balance between the static and the dynamic powers and consequently misses such an optimal solution.

4.4.3 Comparison of V_{DD} control

When the focus is on high performance, there are two ways to achieve it: using a forward bias or increasing V_{DD}. Fig. 4.7 explains that the row-level control

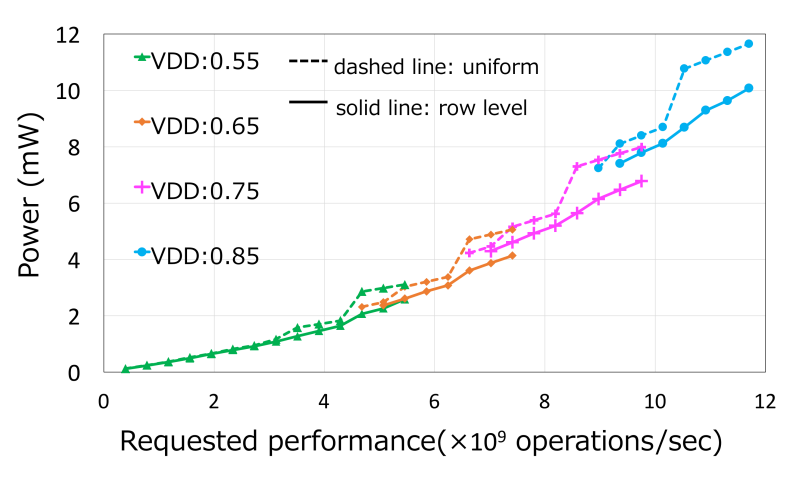


Figure 4.7: Optimization result considering V_{DD} control (*gray*)

with the forward bias is better than using a higer V_{DD}. However, in the case of uniform control, a higher V_{DD} is sometimes more suitable than using the forward bias. In the uniform policy, we can observe ranges of frequencies where a higher V_{DD} implies a lower total power. Even if a higher V_{DD} will increase both static and dynamic powers, it will also improve the switching speed so reverse a bias can be used to decrease the total power. Furthermore, in the range such as $[4.29 \times 10^9, 5.46 \times 10^9]$, lower V_{DD} has to use forward bias to satisfy the performance requirement, which causes a large increase in the leakage power. For instance, between 4.29×10^9 operations/sec and 4.68×10^9 operations/sec, it is more interesting to supply V_{DD} with 0.65 V rather than 0.55 V. On the contrary, a similar phenomenon is not observed regarding the row-level control, which suggests using a V_{DD} as low as possible depending on the requirement. Thus, the row-level control is effective, even with V_{DD} control.

4.5 Summary

In this chapter, we define the leakage power optimization problem for CGRAs. At first glance, it is difficult to find an optimal solution because there exist too complicated trade-offs associated with body bias control and variable pipeline structure. However, detailed examination reveals this problem can be re-formulated as an ILP, which guarantees an optimal solution. Although ILP takes a long time to be solved for large-scale problems because of its NP-hardness [112], our formulation can be solved in an acceptable time. Then,

the optimization results for the row-level body bias control demonstrate up to 24.5% and 16.1% of energy reduction, compared to respectively the case without body bias control and the uniform control. It clearly suggests that CGRA with multiple body bias domains is a promising implementation, and the proposed optimization method is essential to leverage these features.

5

Dynamic power estimation technique

In this chapter, we first explain the influence of glitch propagation for CGRAs, introducing a quantitative analysis. Then, this chapter presents a model to estimate dynamic power consumption quickly and accurately. Lastly, the accuracy of the proposed model is evaluated based on real chip measurements.

5.1 Glitch propagation on PE array

A glitch is an undesired switching caused by two signals deriving with different delays, which consumes a certain power by switching without contribution to the computation. They can be generated by every gate, but especially XOR gates often become a source. Fig. 5.1 shows an example of glitch generation at an XOR gate. In an ideal case, "OUT" does not transit from "0". However, because of the delay of "IN_B," short pulses are generated on "OUT." These unnecessary switchings are called glitches. It is propagated until it reaches registers, so larger combinational circuits, which produce more switching, leads to a larger number of glitches.

Likewise, the different delay times between inputs of the PEs results in such glitches. In the case of the pipelined PE array, such as CC-SOTB2, if a pipeline register is bypassed, they are propagated to the concatenated PEs and then increase the power consumption. In contrast, although activating pipeline registers restrict the propagation, it requires the power for clock distribution

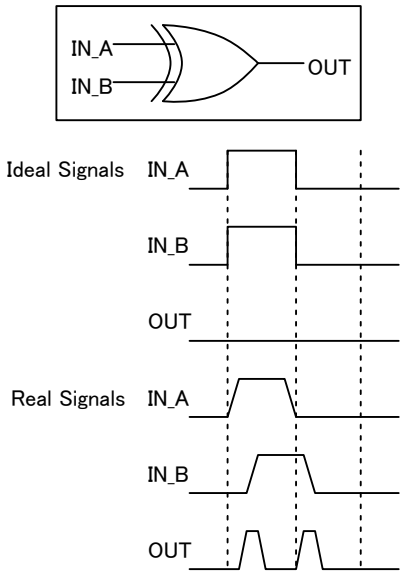


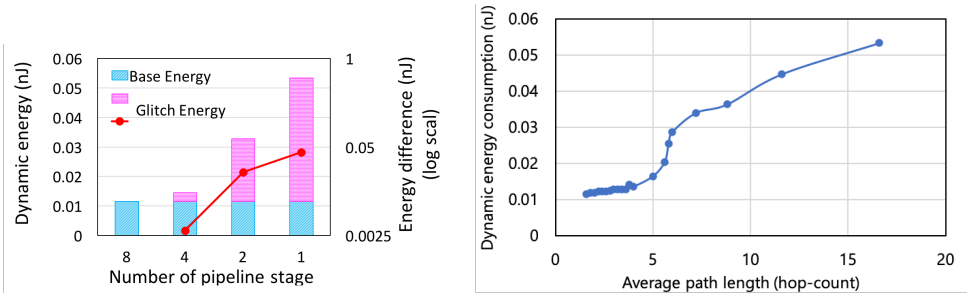
Figure 5.1: An example of glitch generation

and storing data into registers. Therefore, we have to find an optimal pipeline structure, as discussed in Chapter 4. The effect of glitches is more serious for non-pipelined PE arrays, such as CC-SOTB and NVCMA, compared to the pipelined ones since they have no pipeline register. Thus, an application mapping which mitigates the glitch propagation is needed.

The leakage optimization in Chapter 4 assumes the application mapping is fixed, and the dynamic power consumption is obtained by a post-layout simulation to estimate the glitch effect. Nevertheless, such a time-consuming simulation prevents mapping algorithms from exploring the solution space efficiently. That is why a fast and accurate power model is required for mapping optimization. Likewise, FPGAs sometimes produce such a large combinational circuit and suffer from the glitch effect. Hence, some glitch-aware power models for FPGAs have been presented in [113–115]. However, they cannot be applied to CGRAs because of the different granularity of reconfigurable elements between CGRAs and FPGAs.

5.2 Preliminary analysis of glitch propagation

To model the dynamic power consumption considering the glitch effect, several simulations are carried out to obtain the trend of the glitch in the same way in Chapter 4. Fig. 5.2 shows the simulated dynamic power consumption of *gray* application by using the chip layout of CC-SOTB2 under the same



(a) Glitch power for each pipeline structure (b) Relationship between path length and glitch effect

Figure 5.2: Simulated energy consumption of the combinational circuit

condition as Table 4.2 in Chapter 4. Note that they contain only the energy of combinational circuits, and one for clock tree and registers are excluded.

Fig. 5.2(a) explains how the activated pipeline registers avoid glitch propagation. With the 8-stage pipeline structure, each PE is divided with the pipeline registers, and thus, the propagation of glitches is minimized. This result is shown with the blue part of the bar, and the pink part is the difference of the energy when some PE rows are unified into a single pipeline stage. That is, it represents the energy consumption due to propagating glitches. In the non-pipelined case (i.e., single-stage), more than 80% of the dynamic power is caused by the glitches.

Fig. 5.2(a) also shows the difference in energy in the log-scale. In this example, it increases over linearly to the increasing number of unified stages, yet the increase is not exactly exponential. For further examination, more samples of pipeline structure are obtained. They are shown in Fig. 5.2(b). The x-axis represents the average data path length on the PE array. Here, we enumerate all paths from any register outputs to any register inputs for each sample of pipeline structure. Then, the average path length is calculated. As the activated pipeline registers are increased, the average path length usually becomes shorter. The longer paths are formed on the PE array, the more switching due to glitch propagation is caused. In the range of the path length shorter than six, the glitches increase exponentially. However, it becomes linear growth beyond the range. It is expected a part of propagated glitches are decayed through PEs and interconnection networks.

5.3 Dynamic power model

For mapping and pipeline structure optimization, a model which can estimate the dynamic power consumption even for inexperienced application mappings is needed. Besides, it should be simple to calculate the power quickly while keeping the estimation accuracy. Then, we simplify Eq. (2.2) in Chapter 2 as follows.

$$P_{\mathrm{dyn}} = E_{\mathrm{sw}} S_{\mathrm{total}} f + P_{\mathrm{dyn,reg}} \times N_{\mathrm{acitve\_preg}} \tag{5.1}$$

where E_{sw} is the average energy consumption per switching of PEs, and S_{total} is the switching count of the whole PE array with glitches. E_{sw} is regarded as a constant value and depends on the process technology and logic synthesis of the PE, $P_{\mathrm{dyn,reg}}$ is the dynamic power of both the pipeline registers and their clock trees, and $N_{\mathrm{acitve\_preg}}$ is the number of activated pipeline registers. As explained in Section 3.2, clock-gating is applied to the deactivated registers. Thus, they are assumed not to consume dynamic power consumption. Hence, only S_{total} depends on the application mapping. S_{total} is, therefore, modeled by the following equations:

$$S_{\mathrm{total}} = \sum_{v \in V_{\mathrm{alu}} \cup V_{\mathrm{se}}} S(v) \tag{5.2}$$

$$S(v) = \begin{cases} \text{If } v \text{ is an ALU:} & \\ \quad S_{alu}(op) + \beta\gamma^{length} \max\limits_{u \in \mathrm{Preds}(v)} S(u) & (5.3) \\ \text{If } v \text{ is a SE:} & \\ \quad \zeta S(\mathrm{Preds}(v)) & (5.4) \end{cases}$$

where $S(v)$ is the switching count of node v in a set of ALUs (V_{alu}) or a set of SEs (V_{se}), $S_{\mathrm{alu}}(op)$, and S_{se} are respectively that of ALU configured with an operation *op* and switching element (SE). β and γ are propagation factors, *length* is the hop count from the nearest active pipeline register, and $\mathrm{Preds}(v)$ represents the predecessor nodes of v. If $u \in \mathrm{Preds}(v)$ indicates an activated pipeline register, $S(u)$ is regarded as 0 since the pipeline register prevents the glitch propagation. Considering the analyzed trend of glitch propagation, a degree of the glitch propagation can be estimated based on a non-linear fitting function like $\beta\gamma^{length}$. The model calculates the switching count node-by-node. In this way, it can be applied to the different PE designs, which have a different number of SEs, and different interconnection topologies. Compared to the ALUs, the circuit of the SE is more simple, and its latency is, therefore,

much lower. That is why this model considers the SE to generate no more glitches, that is, to transit a part of switching brought from its input signal. ζ denotes a coefficient of the switching transfer of SEs. Please note that each SE has only a single predecessor node.

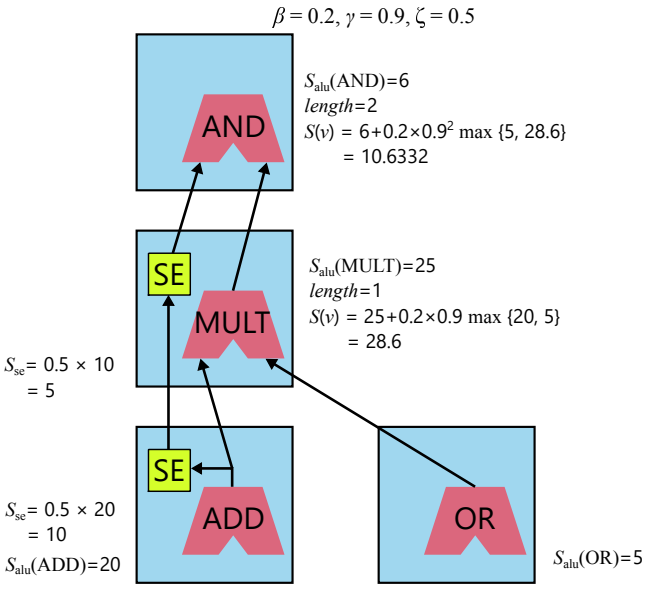


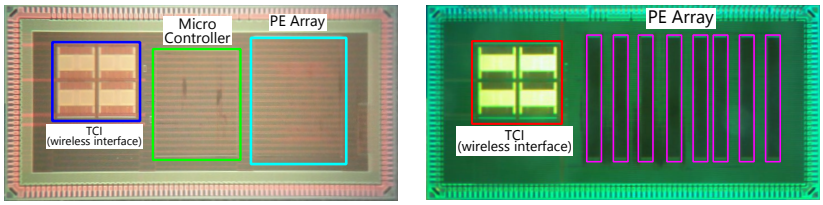
Figure 5.3: An example of the glitch propagation model

Fig. 5.3 shows an example of calculating S_{total} with the model. A blue square in the figure represents a PE containing an ALU and a SE like CCSOTB2. Let the propagation factor β be 0.2, and γ be 0.9, respectively. Then, all pipeline registers are deactivated so that they are omitted in this figure. Since ALUs in the bottom PEs have no input propagated glitches, $S_v = S_{\mathrm{alu}}(op)$. On the contrary, the successor ALU in the second row has two data inputs from two different ALUs, which generate glitches. Here, the biggest switching count is adopted by the max function, and thus the switching count S_v for the ALU is calculated, as shown in the figure. The last ALU is treated in the same way. On the other hand, SEs do not yield more glitches in this model, as described above. In this example, an SE transits 50% of inputted switching. Then, the total switching count S_{total} becomes $(20 + 10) + 5 + (28.6 + 5) + 10.6332 = 79.2332$.

Here, we assume that E_{sw}, β, γ, and ζ are fixed parameters independent from the application programs and obtained with the simulation results or real chip measurements. Besides, given the target application and its mapping, the other parameters are fixed as well. Thus, the total switching count can be estimated when an application program and its mapping are fixed.

5.4 Evaluation

In this section, we carry out several experiments for the three architectures, CC-SOTB, CC-SOTB2, and NVCMA, to evaluate the accuracy of the proposed model. Then, we demonstrate the model can be applied to any PE array architectures regardless of different implementation technologies.



(a) CC-SOTB (b) CC-SOTB2

Figure 5.4: Chip photograph

5.4.1 Obtaining model parameters

All three architectures were designed with Verilog HDL and synthesized by Synopsys Design Compiler using process libraries of the technologies listed in Table 3.2 in Chapter 3. They were then placed and routed with Synopsys IC compiler. Fig. 5.4(a) and Fig. 5.4(b) are the photographs of CC-SOTB and CC-SOTB2 chips, respectively. The size of both chips is 3mm $\times$ 6mm. Although both of them include an inter-chip wireless communication interface[116], it will not be discussed as this falls out of the scope of this work. A photograph of the NVCMA chip is omitted due to confidentiality reasons.

The switching count of the ALU (S_{alu}) is obtained by gate-level simulation with Cadence NC-Verilog. For each ALU operation, the simulation is carried out 3000 times while changing input values randomly. Then, the average switching counts at the output of ALU are recorded. The results are shown in Table 5.1. In comparison with shift operations (SL, SR, and SRA), logical operations (AND, OR, and NOT) bring more switching counts. The switching counts of arithmetic operations (ADD, SUB, and MULT) are the largest and about three times as much as those of shift operations. Both CC-SOTB and CC-SOTB2 are implemented with the same process technology while they are respectively based on different libraries. Thus, even for the same operation, the switching counts of both architectures are different. The netlist of NVCMA is confidential so that we cannot conduct the gate-level simulation. Therefore, the values of CC-SOTB2 is substituted for that of NVCMA.

Table 5.1: Average switching counts of each operation in a PE

Opcode	S_{alu}	
	CC-SOTB	CC-SOTB2
ADD	15.80	17.17
SUB	24.47	20.02
MULT	24.04	31.46
SL	8.762	6.791
SR	6.398	4.973
SRA	8.236	7.318
AND	5.171	5.217
OR	17.11	16.92
NOT	11.12	11.13
XOR	20.71	21.00
CAT	10.99	11.14
SEL	12.75	11.99
GT	18.68	18.66
LT	18.37	18.31
EQL	28.97	28.61

5.4.2 Accuracy of the proposed model

Table 5.2: The results of model fitting

Architecture	# of samples	E_{sw} (pJ/sw)	β	γ	ζ	ME (%)
CC-SOTB	72	0.0263	0.9507	1.0104	0.9372	12.91
CC-SOTB2	1536	0.0836	0.3394	1.0999	0.06879	12.85
NVCMA	111	0.0472	0.5819	1.0515	0.8508	10.67

Lastly, we measure the dynamic power consumption for various kinds of DFGs and mappings to determine the parameters E_{sw}, β, γ, and ζ in this dynamic power model. The supply voltages for CC-SOTB, CC-SOTB2, and NVCMA are respectively 0.55 V, 0.55 V, and 1.20 V. We employ the Levenberg-Marquardt least-squares method to fit these parameters. Table 5.2 describes the results of fitting and mean relative error (ME) for each architecture. CC-SOTB2 needs more samples than the other two architectures because the dynamic power consumption depends on the pipeline structure as well as the mapping. In the case of CC-SOTB2, at most $2^7 = 128$ patterns of pipeline structures are available. Regardless of the process technologies, the model can estimate the dynamic power consumption, as shown in the results of ME. The purpose of the model is to compare solutions from a qualitative point of

view rather than to estimate the power consumption exactly. Therefore, these errors are acceptable.

5.4.3 Comparision with a post-layout simulation

When a post-layout simulation with Synopsys PrimeTime is employed, it takes about six minutes to evaluate the dynamic power consumption for a specific application mapping and a pipeline structure on Intel Xeon E5-2667. In contrast, this model requires less than one millisecond. Thanks to the fast estimation, we can widely explore the solution space of mapping optimization.

5.5 Summary

The effect of glitch on LSI chips should be taken into account since it sometimes causes a prohibitive increase in dynamic power consumption. However, it generally takes much time to estimate the number of glitches with a post-layout simulation. Therefore, a fast estimation technique is essential for power optimization. This chapter proposes a dynamic power model specialized for CGRAs considering the glitch propagation effect. Despite the simplified model, it can calculate the dynamic power consumption with around 10% of error, compared to the actual measurement results. Besides, it reduces the calculation time by five orders of magnitude. Such a fast estimation model contributes to more efficient mapping optimization.

6

GenMap: mapping optimization with genetic algorithm

In this chapter, we first define the application mapping problem and describe the used notations. Then, it shows the solution space of the mapping optimization is extremely large, especially when leveraging the body bias optimization and variable pipeline structure. In order to explore such a huge solution space efficiently, this chapter presents GenMap, which is an optimization framework based on a multi-objective genetic algorithm, NSGA-II [2]. Thanks to the genetic algorithmic approach, there is no limitation regarding the objective functions to evaluate fitness for each solution. Therefore, we integrate the body bias optimization technique in Chapter 4 and the dynamic power model in Chapter 5 into GenMap for a case study. Besides, the benefit of the aggressive optimization by GenMap is demonstrated by comprehensive experiments with the fabricated chips.

6.1 Problem Definition

As introduced in Section 2.1, a data flow graph (DFG), denoted $G(V_d, E_d)$, is given as a target application kernel. The DFG is a directed graph, which consists of nodes indicating operations and edges representing data dependencies between operational nodes. The nodes include constant values and memory

access as well. Therefore, the set of nodes V_d is divided into a few subsets as follows: $V_d = V_{op} \cup V_{const} \cup V_{input} \cup V_{output}$. V_{op}, V_{const}, V_{input}, and V_{output} are the set of computation nodes, constant values, loaded values from the memory, and values to be stored in the memory, respectively. Likewise, the set of edges defined as follows: $E_d = E_{op} \cup E_{const} \cup E_{input} \cup E_{output}$. E_{op}, E_{const}, E_{input}, and E_{output} are the set of edges associated with V_{op}, V_{const}, V_{input}, and V_{output}, respectively. An edge in E_{op} means a simple data dependency between two computational operations, whereas an edge in E_{const} is connected from a constant value to a computational operation. E_{input} and E_{output} are similar to E_{const}.

The PE array is also modeled as a directed graph, denoted $H(V_r, E_r)$, where V_r is denoted as a set of hardware units, and E_r is the connectivity for each unit. In this work, the PE array includes ALUs, SEs, constant registers, input ports, and output ports. Thus, $V_r = V_{alu} \cup V_{se} \cup V_{const\_reg} \cup V_{iport} \cup V_{oport}$.

The problem that we address in this work is to find a valid mapping M: $G \rightarrow H' \subset H$. The valid mapping satisfies the following conditions.

1. $M(V_{op}) \subset V_{alu}$
2. $M(V_{const}) \subset V_{const\_reg}$
3. $M(V_{input}) \subset V_{iport}$
4. $M(V_{output}) \subset V_{oport}$
5. $\forall e \in E_d$: $M(e)$ makes a path between $M(u)$ and $M(v)$
6. $\forall (u, v), (w, x) \in E_d$:
   $u \neq w \Rightarrow M(u, v) \cap M(w, x) = \emptyset$

The conditions 1 to 4 are associated with the correct placement for each DFG node. The last two ones guarantee the routing is accomplished considering path-sharing. Assuming two edges driving the same data, they can share some routing resources.

In general, if a PE array can accommodate a target DFG, there exists more than one valid mapping. Therefore, we have to optimize the mapping as well as just finding a valid one. As mentioned in Section 3.3, the quality of the mapping should be evaluated from various viewpoints. Section 6.3 will explain the criteria of evaluation used in this thesis for the case study.

The search space of this optimization problem is huge, approximately $O(|V_r|^{|V_d|})$. Furthermore, when the body bias control and variable pipeline are considered, it is expanded up to $O(|V_r|^{|V_d|} \times 2^{N_p} \times N_{bb}^{N_{domain}})$ where N_p is the number of pipeline registers, N_{bb} is the size of the set of body bias

voltages, and N_{domain} is the number of body bias domains in the PE array. Therefore, there are few hopes to find an exact solution, so we choose to employ a genetic algorithmic approach to tackle the huge search space. In our case, CC-SOTB2 brings the largest search space due to the variable pipeline and the four-divided body bias domains.

6.2 Proposed framework: GenMap

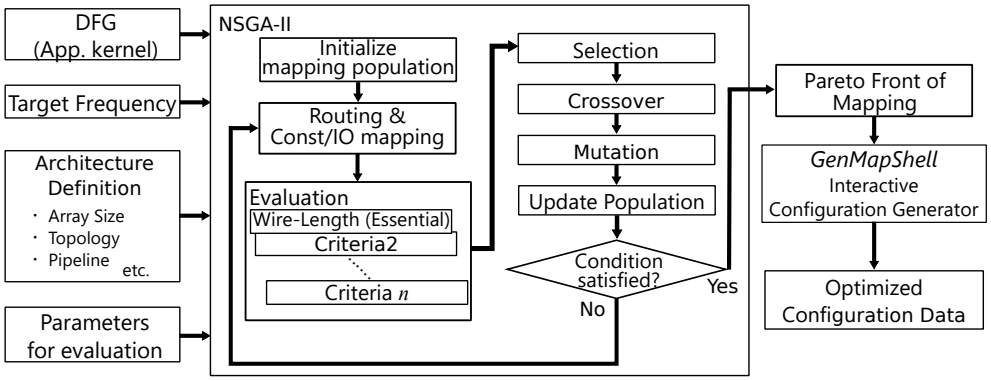


Figure 6.1: Optimization flow of GenMap

In this section, we propose an application mapping framework called GenMap. GenMap introduces a flexible optimization for both architects and programmers. As shown in Fig. 6.1, the overall flow of optimization is based on NSGA-II (Non-dominated Sorting Genetic Algorithm-II) [2], a multi-objective genetic algorithm. In addition to the target application kernel, a target frequency, architecture definition, and parameters such as power consumption and timing information are given to GenMap. The mapping of the application is optimized as far as any timing violation does not occur at the specified frequency. The optimization stops when it reaches the last generation or does not yield any improvement for a fixed number of generations. These conditions are parameterized in GenMap. The architecture definition contains the size of the PE array, interconnection topology, and whether the pipeline structure and body bias voltages can be controlled. The parameters are necessary to calculate the fitness for each criterion. In addition, they depend on the chip implementation of the target architecture. They are obtained from either post-layout simulations or real chip experiments.

6.2.1 Multi-Objective Optimization with NSGA-II

Similar to a general genetic algorithm, NSGA-II iteratively improves a population of solutions as follows:

Step 1: Create an initial population

Step 2: Evaluate fitness values for each individual

Step 3: Select parents

- i: Perform non-dominated sorting and assign Pareto rank to each individual
- ii: Sort individuals for each rank
- iii: Select individuals based on Binary Tournament Selection [117]

Step 4: Crossover the selected pairs of parents to produce children

Step 5: Mutate the selected individuals

Step 6: Evaluate the new individuals as in Step 2

Step 7: Select individuals for the next generation

Step 8: Go to Step 3 if the termination condition is not met.

At each iteration (generation), some individuals are selected as parents. An individual with better fitness is more likely to be selected. Then, children for the next generation are produced based on the selected parents. In the case of the crossover operation, two children are born, combining a pair of parents, while in the case of the mutation operation, the gene of a parent is partially modified.

NSGA-II is an extended algorithm for multi-objective optimization. Particularly, the selection strategy is improved to be able to consider multiple objectives simultaneously based on the Pareto rank and the crowding distance. Multi-objective optimization is generally based on the concept of the dominance relationship between solutions. This relationship is defined as follows:

Definition 6.1 (Dominance). A solution x dominates a solution y if:

- The optimization problem: $\max \boldsymbol{F}(x) = \{f_1(x), f_2(x), ..., f_n(x)\}$
- $f_k(x) > f_k(y), \forall k \in \{1, 2, ..., n\}$

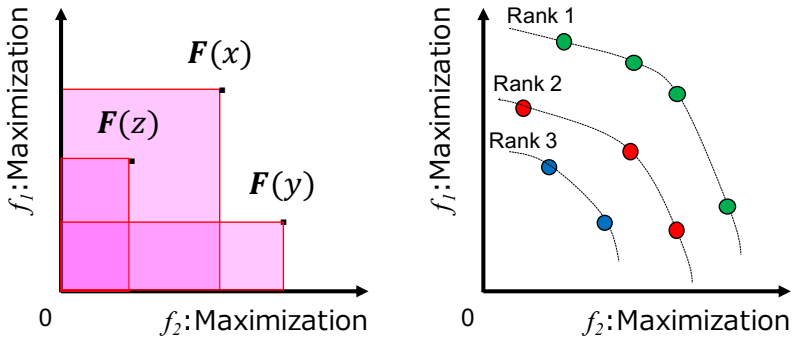
where n objective functions to be maximized are considered.

Fig. 6.2(a) shows examples of the dominance relationship. In this case, the solution x dominates the solution z. In contrast, the solution y is not dominated by x since $f_2(y) > f_2(x)$.

The Pareto rank is calculated based on the dominance between the solutions as follows:

Definition 6.2 (Pareto rank). First, the solutions non-dominated by any other ones are treated as rank 1, and they are eliminated from the population. Then, rank 2 is assigned to the solutions non-dominated in the remaining population. Likewise, rank $i > 2$ is assigned to solutions by repeating the above procedure.

Fig. 6.2(b) illustrates an example of the rank assignment.



(a) Dominance relationship (b) Pareto rank

Figure 6.2: Examples of dominance relationship and Pareto rank for a bi-objective optimization problem

The selection process for the next generation is described in Fig. 6.3. The first criterion is the Pareto rank, and the second one is the crowding distance. First, solutions are selected in the order of their rank. If the size of solutions of rank i (in this example, rank 3) exceeds the remaining capacity of the next population, the solutions are sorted based on the crowding distance. Then, the solutions with larger crowding distances are chosen as the remaining individuals. The other solutions are dismissed.

The crowding distance is used to preserve the diversity of the solutions. It indicates how close an individual to the nearest neighbors. Therefore, a solution with a large crowding distance means a unique solution and should be survived for the next generation. Fig. 6.4 explains how to calculate the crowding distance for a bi-objective optimization problem. The crowding distance of the i-th solution is calculated as the average side length of the cuboid

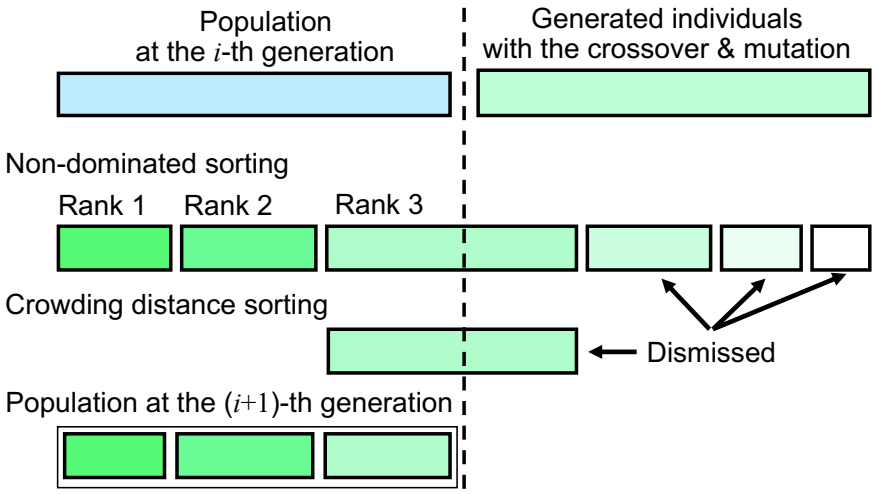


Figure 6.3: Selection flow of the NSGA-II [2]

formed by the nearest solutions. Algorithm 1 describes the calculation more detailedly.

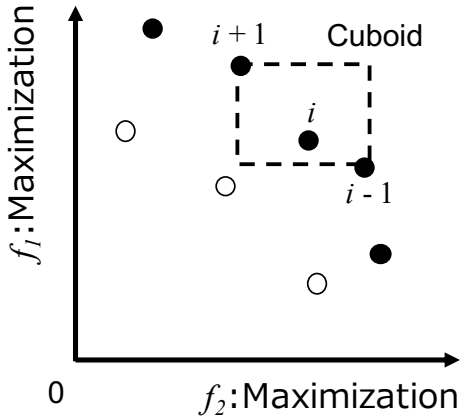


Figure 6.4: Examples of crowding distance calculation [2]

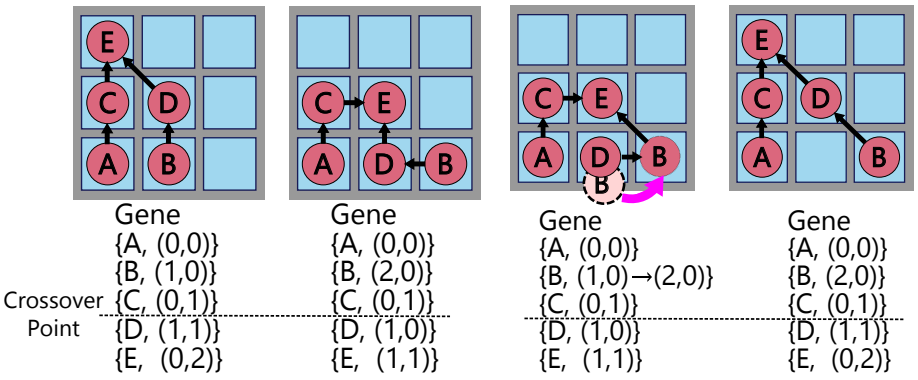
The crossover and mutation occur stochastically, and the probabilities are important parameters in the genetic algorithms. In this work, the crossover and mutation probabilities are respectively 0.7 and 0.3 since they are commonly-used values [118]. In our preliminary analysis (reported in Appendix C), such a condition shows the most stable optimization for GenMap.

Algorithm 1 Pseudo code of crowding distance calculation

Input: A set of individuals to be sorted I
Output: Crowding distance for each individual CD

```
1: l ← |I|                                              /* set # of individuals */
2: for i = 1 ... |I| do
3:    CD[I[i]] ← 0
4: end for
5: for each objective function f_m ∈ F do
6:    sort(I, f_m)          /* sort individuals in the order of their fitness value of f_m */
7:    CD[I[1]] = CD[I[l]] ← ∞                           /* boundary solutions */
8:    f_m^max ← I[l].m
9:    f_m^min ← I[1].m
10:   for i = 2 ... |I| − 1 do
11:      CD[I[i]]+ = (I[i + 1].m − I[i − 1].m)/(f_m^max − f_m^min)
12:   end for
13: end for
```

6.2.2 Gene coding and crossover



(a) Example of parents (b) Born children

Figure 6.5: Gene coding and example of crossover

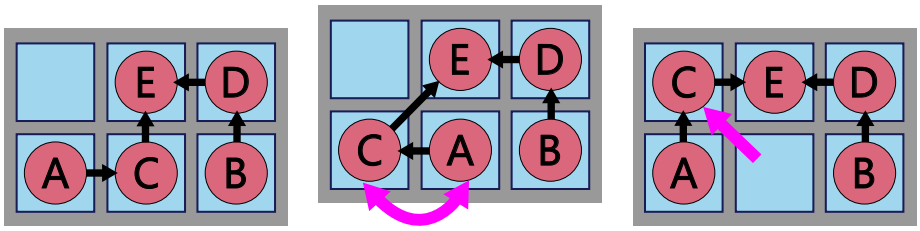
To solve the optimization problem with NSGA-II, a mapping has to be represented as a gene. For GenMap, only the mapping of operation nodes ($M(V_{op})$) and the pipeline structure (if necessary) are coded as the gene. If the gene includes other information such as routing, the crossover and mutation would be too complicated. Thus, every time a new individual is generated, the mapping of the constant registers ($M(V_{const})$) and binding IO ports

($M(V_{input})$, $M(V_{output})$) are determined separately, and then, routing is carried out. Each of them will be explained later in this section. Besides, the assignment of the body bias voltage for each domain is carried out as a part of the power evaluation explained in the next section.

The mapping of operation nodes is represented as a list of PE coordinates, as shown in Fig. 6.5(a). After two parents are selected, a single crossover point is randomly chosen. Then, parts beyond the point are swapped to create the genes of children, as shown in Fig. 6.5. However, such a simple crossover often causes duplicated nodes, such as in the left side of Fig. 6.5(b). In this case, a repair mechanism is performed by randomly moving these nodes to neighboring PEs until the duplication is eliminated.

Regarding the pipeline structure, a bit-vector expression is employed. Assuming the example of Fig. 3.4(b), its genetic description is $\{0, 1, 0, 1, 0, 0, 1\}$ where 1 represents an activated register. The crossover is simpler than that for the operation mapping because duplication does not occur.

6.2.3 Mutation



(a) Selected parent gene (b) mutation (swap) (c) mutation (move)

Figure 6.6: Example of a mutation

We defined two types of mutation: 1) swap and 2) move. When the swap mutation is applied, any two nodes to be swapped are randomly chosen. On the other hand, in the case of the move mutation, a randomly selected node is moved to a free PE. The bit-vector of the pipeline structure is also modified by flipping a randomly selected bit.

6.2.4 Population Initialization

Even though a genetic algorithm attempts to avoid local optima by using mutation, it might take a long time to converge, especially when starting with a fully randomized initial population. Additionally, the diversity of the initial population is important to avoid the algorithm being stuck in a local optimum.

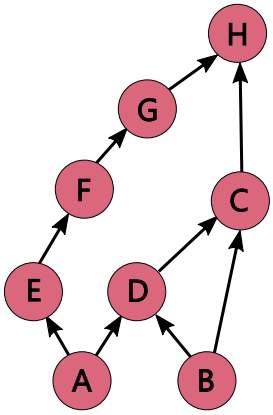
Therefore, the initial population should maintain enough both quality and diversity to ensure fast and good convergence.

Algorithm 2 Initialization of the population

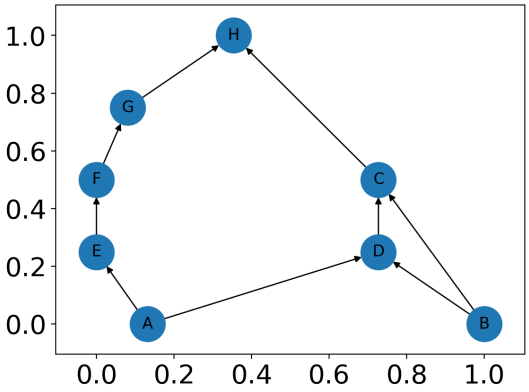
Input: Application DFG, $G_{op}(V_{op}, E_{op})$
Output: Mapping for each node, $M(V_{op})$
1: Calculate $pos(V_{op})$ by $graphviz\_layout(G_{op})$
2: Normalize $pos(V_{op})$
3: Randomly choose the size of mapped rectangle ($W \times H$)
4: **for each** node of V_{op} **do**
5: Set coordinate $(x, y) = pos[V_{op}]$
6: Expand $pos[V_{op}] = (x \times H, y \times H)$
7: **end for**
8: Randomly flip $pos(V_{op})$ horizontally
9: **for each** node of V_{op} **do**
10: Round $pos[V_{op}]$ to neighboring grid point randomly
11: **end for**
12: **for each** grid point **do**
13: **if** more than one node is placed **then**
14: **while** Only one node is placed **do**
15: Randomly selected node is moved to neighboring free grid point
16: **end while**
17: **end if**
18: **end for**

For the initialization, GenMap employs the *dot's* algorithm from the open-source graph visualization software, Graphviz [119]. In general, graph layout algorithms aim at reducing the number of crossing edges and keeping each edge length as equal as possible. For the mapping problem, such an algorithm can decrease the congestion of wiring among PEs. A few methods are based on the graph drawing algorithm [31, 92]. Besides, *dot's* algorithm maintains the hierarchical structure of a directed graph, which is especially suitable for SF-CGRAs.

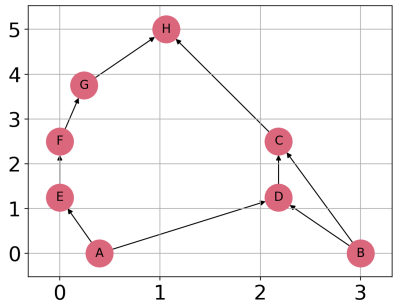
The procedure of making a mapping for the initial population is described in Algorithm 2. In the beginning, the position for each node is laid out by the *dot's* algorithm. Then, they are normalized, as shown in Fig. 6.7(b). The normalized positions are expanded to a PE rectangle big enough to contain each node, the size being randomly chosen. For instance, for a size of 4 x 6 PEs, the position for each node is changed, as shown in Fig. 6.7(c). In addition, the layout is stochastically flipped horizontally. For another example, Fig. 6.7(e) indicates a modified layout for a size of 2x7 PEs with such flipping. Each



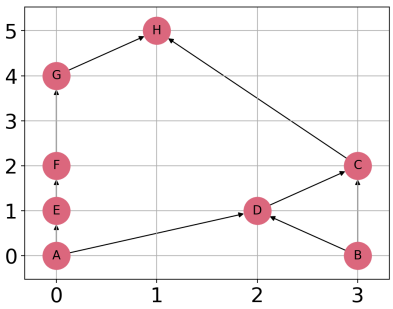
(a) Example of application DFG



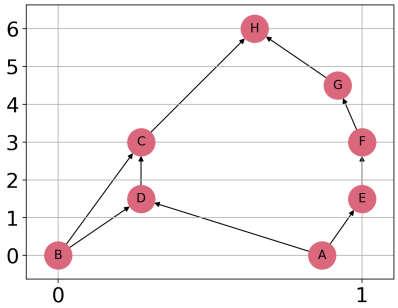
(b) Normalized layout by *dot's* algorithm



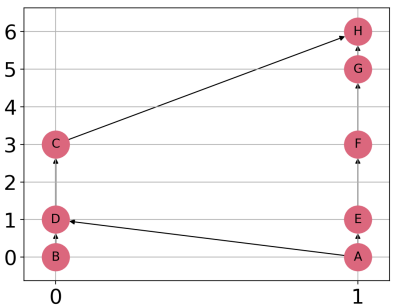
(c) Expanded to 4x6 PEs



(d) Rounded to 4x6 PEs



(e) Expanded to 2x7 PEs with horizontal flipping



(f) Rounded to 2x7 PEs

Figure 6.7: Examples of mapping initialization

grid point corresponds to one PE. Each node is moved to the adjacent grid point randomly, that is, four options: 1) upper left, 2) upper right, 3) lower

left, and 4) lower right. Due to such a randomized rounding, PE rectangles of the same size can provide different mappings. If there exists any duplicated node at the same grid point, they are moved again, in the same manner as duplications are removed in the crossover operation of genes (Fig. 6.5(b)). The initial population is made by repeating this procedure.

6.2.5 Routing Method

Waste of the resources by unoptimized routing can cause a lower probability of valid mapping so that effective heuristics are essential. However, for our approach, routing between dependent PEs has to be performed each time a new mapping is generated. Therefore, time-consuming heuristics cannot be applied. That is why our routing algorithm is based on a greedy search using the A$*$ algorithm [120] iteratively. Although such a greedy approach cannot guarantee the global optimum, using an Ant Colony Optimization would reduce the total wire length by only a few percent compared to the greedy heuristics [101].

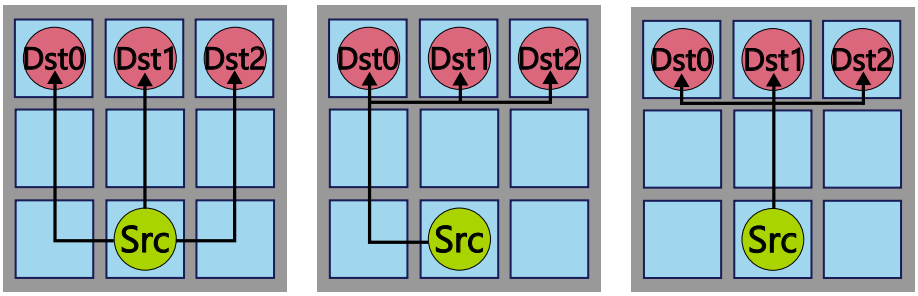
Algorithm 3 Overall flow of routing

Input: Application DFG, $G(V_d, E_d)$
Model of PE array, $H(V_r, E_r)$
Mapping for each node, $M(V_{op})$

1: computation_routing($G_{op}(V_{op}, E_{op}), H, M(V_{op})$)
2: $M(V_{const}) \leftarrow$ const_reg_mapping()
3: const_routing($G_{const}(V_{const}, E_{const}), H, M(V_{const})$)
4: $M(V_{input}) \leftarrow$ input_port_mapping()
5: input_routing($G_{input}(V_{input}, E_{input}), H, M(V_{input})$)
6: $M(V_{output}) \leftarrow$ output_port_mapping()
7: output_routing($G_{output}(V_{output}, E_{output}), H, M(V_{output})$)

As shown in Algorithm 3, our routing method is composed of four steps: 1) computation routing, 2) constant routing, 3) input routing, and 4) output routing. They are associated with the routing of $E_{op}, E_{const}, E_{input}$, and E_{output}, respectively. Each gene has the only mapping of (V_{op}) so that mapping of other resources such as constant registers is carried out in this phase. Details of these mapping are explained later. Our preliminary analysis indicates this routing order brings good results in many cases.

Algorithm 4 describes the *computation_routing* procedure in Algorithm 3. Other routings, including *const_routing*, are almost the same. Thus, their pseudo codes are omitted. In order to save the routing resources, path-sharing has to be considered, even with the greedy approach. This method induces



(a) No path-sharing (b) Bad path-sharing (c) Better path-sharing

Figure 6.8: Impact of path-sharing

path-sharing by changing routing cost adaptively. Without path-sharing, the routing resources can be wasted, as illustrated in Fig. 6.8(a). Even though such a data path can be shared, the three outgoing edges are routed along separated routes.

Initially, the routing cost of all edges in the PE array (E_r) is set to 1. Routed dependencies in E_{op} are classified by their source nodes. Edges from the same source node are grouped. Then, the smaller group is routed with the higher priority, such as in [86]. If a node has only an outgoing edge and is routed with a lower priority, it causes a roundabout routing due to the lack of resources. However, in the case of a high fan-out node, such a roundabout way is utilized by sharing it with other successor nodes.

While edges from the same source node are routed, the cost of routing resources already utilized for these edges is temporarily set to 0. In this way, path-sharing is encouraged. Each edge in the same group is routed depending on the Manhattan-distance in order to avoid bad path-sharing, as shown in Fig. 6.8(b). In the case of Fig. 6.8(b), $Src \rightarrow Dst0$ is routed ahead of $Src \rightarrow Dst1$ so that other paths are influenced by the first routing way. On the other hand, by sorting the edges based on the Manhattan-distance, a good path-sharing such as in Fig. 6.8(c) is provided. $N_{penalty}$ is a high penalty cost of routing defined to already used resources to avoid overuse. Therefore, if the A$*$ algorithm cannot find any path, or the length of a found path is larger than the penalty cost, then the routing process is considered as failed.

6.2.6 Constants and IO mapping

The mapping of the constant registers has to be treated differently from the mapping of operation nodes. As illustrated in Fig. 6.9, when several nodes (Ⓐ and Ⓒ) require the same constant value ("1"), a single mapping (Fig. 6.9(a)),

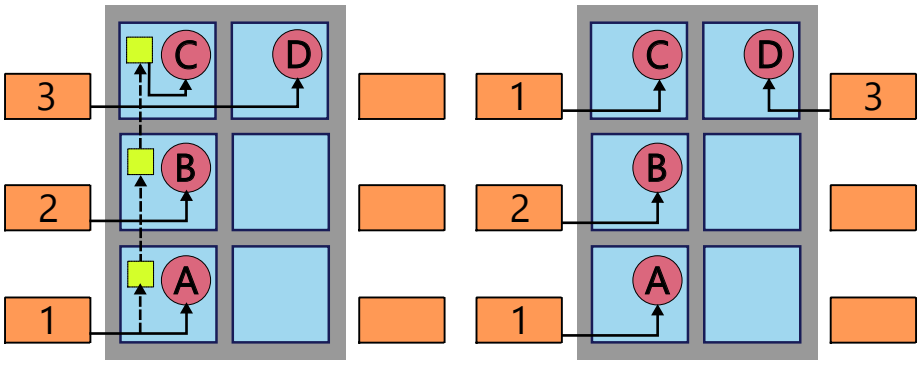
Algorithm 4 Routing method in GenMap

Input: Application DFG, $G_{op}(V_{op}, E_{op})$
Model of PE array, $H(V_r, E_r)$
Mapping for each node, $M(V_{op})$

1: $E_{list} \leftarrow \emptyset$
2: set_routing_cost(E_r, 1)
3: **for each** $v \in V_{op} \subset V_d$ such that outdeg$(v) > 0$ **do**
4: $E_{from,v} = (v, w) | w \in$ Successors(v)
5: Append $E_{from,v}$ to E_{list}
6: **end for**
7: Sort E_{list} by ascending size of $|E_{from,v}|$
8: **for each** $E_{from,v}$ of E_{list} **do**
9: Sort $E_{from,v}$ by ascending Manhattan-distance between $M(v)$ and $M(w), (v, w) \in E_{from,v}$
10: $R_{used} \leftarrow \emptyset$
11: **for each** $e \in E_{from,v}$ **do**
12: Set $(v, w) = e$
13: $p =$ find_astar_path$(H, M(v), M(w))$
14: **if** Path not found or $p > L_{penalty}$ **then**
15: **return** Fail in routing
16: **else**
17: set_routing_cost(p, 0)
18: Append p to R_{used}
19: **end if**
20: **end for**
21: set_routing_cost(R_{used}, $L_{penalty}$)
22: **end for**

which assigns each constant value only to one constant register, would waste routing resources. On the other hand, if that constant value is allowed to be mapped multiple times onto different constant registers, such as in Fig. 6.9(b), it could result in a more appropriate constant mapping.

Assigning different constant registers for each operation node remains nevertheless a difficult task due to the limited number of constant registers, as explained in Section 3.2. Therefore, the multi-mapping is determined while considering the resource limitation. We formulate this problem with a simple integer linear program as follows:



(a) In the case of single mapping (b) In the case of multi-mapping

Figure 6.9: Different mapping strategy for constant registers

$$isMap_{e,creg} = \begin{cases} 1 & \text{if the constant register } creg \\ & \text{is utilized for edge } e \\ 0 & \text{otherwise} \end{cases} \tag{6.1}$$

$$\min \sum_{\substack{e \in E_{const} \\ creg \in V_{const\_reg}}} isMap_{e,creg} \times dist_{e,creg} \tag{6.2}$$

subject to

$$\forall e \in E_{const} : \sum_{creg \in V_{const\_reg}} isMap_{e,creg} = 1 \tag{6.3}$$

For any pair: $e_1 = (u_1, v_1), e_2 = (u_2, v_2) \in E^2_{const}$

$$\begin{cases} \text{If } u_1 \text{ is equal to } u_2, \forall creg \in V_{const\_reg} : \\ \quad isMap_{e_1,creg} + isMap_{e_2,creg} \leq 2 & (6.4) \\ \text{else}, \forall creg \in V_{const\_reg} : \\ \quad isMap_{e_1,creg} + isMap_{e_2,creg} \leq 1 & (6.5) \end{cases}$$

where $isMap_{e,creg}$ is a decision variable indicating whether a constant register $creg$ is assigned for the routing of an edge e. $dist_{e,creg}$ is the distance between $creg$ and the successor nodes of e, and is calculated with the A$*$ algorithm. Constraint (6.3) ensures that each constant value is mapped onto a constant

register. Constraint (6.4) allows a constant register to be shared with two operation nodes if possible. Finally, constraint (6.5) guarantees that different constant values are not mapped onto the same constant register.

Similarly to the constant mapping, input port mapping is decided with the same formulation, whereas output ports are bound by choosing the nearest port in a greedy manner.

6.3 Model and Objectives

After the routing is performed, each mapping is evaluated with respect to several objectives. The NSGA-II supports both minimization and maximization. This thesis considers four evaluation functions: 1) wire length, 2) mapping width, 3) power consumption, and 4) time slack for our case study. This section will describe each of them. Another work of ours [121] gives more examples of the objectives for readers interested in fault tolerance for a non-volatile CGRA.

6.3.1 Wire Length

Although the objectives can be customized in GenMap, the wire length is an important objective that cannot be omitted since finding a valid mapping depends on it. After all the routing steps are completed, unused resources are eliminated. Then, the total length of the resource graph is calculated as the wire length. That is, the obtained value of wire length is not the physical wire length on the LSI chip but rather an estimation. The objective function is formulated as follows, considering failure in routing at any step:

$$\min L_{wire} = L_{success} + N_{penalty} \times |E_{unrouted}| \tag{6.6}$$

where $L_{success}$ is wire length routed so far, $N_{penalty}$ is the same value as in Algorithm 4, and $E_{unrouted}$ is the set of unrouted edges.

Thereby, the optimization reduces the number of unrouted edges at an early stage. After it finds a valid mapping, that is, the second term in Equation 6.6 is equal to 0, it seeks mappings with a shorter wire length by reducing $L_{success}$.

6.3.2 Mapping Width

Mapping width is also crucial from the viewpoint of throughput. In many cases, application kernels offloaded to CGRAs have data-level parallelism so

that the mapping can be duplicated horizontally by loop unrolling. For instance, assuming a mapping width of 3 and 12 columns of PE array, the unrolling factor for the mapping is $\lfloor 12/3 \rfloor = 4$. The mapping width can be obtained by analyzing which utilized PE is the rightmost PE like Equation 6.7, where "x_coord(r)" returns zero-based x-coordinate of the PE containing r. The objective is to minimize the width in order to increase the unrolling factor, that is, the throughput.

$$W_{map} = \max\{\text{x\_coord}(r) \mid \forall \text{ used } r \in V_{alu} \cup V_{se}\} \tag{6.7}$$

6.3.3 Power Consumption

Power consumption is one of the most considerable concerns in this thesis. Here, we focus on the power consumption of the PE array since that of other modules is mostly the same regardless of the mapping. As Eq. (2.1) describes, total power consumption is the sum of the dynamic power P_{dyn} and the static power P_{st}. At the evaluation phase of GenMap, mapping for each individual is already fixed. Besides, the pipeline structure is included in the gene code (if the PE array is pipelined). Therefore, the best body bias voltage for each domain can be obtained by using the ILP formulation in Chapter 4. In this way, the optimal static power consumption for each individual is evaluated. The dynamic power consumption is also evaluated with the dynamic power model in Chapter 5. Then, the sum of both evaluated power is employed as the fitness value regarding power consumption.

6.3.4 Time Slack

Unlike the other objectives, the time slack is to be maximized. At first glance, this seems meaningless since the timing constraint only has to be met. Nevertheless, in the case of timing violation due to long wire length, the time slack takes a negative value, which is a behavior often observed at the early stages of the optimization with a severe timing constraint (the detailed analysis is described in Appendix E). When any mapping satisfying the timing constraint is still not found, the closer to zero the time slack of mapping is, the more likely it is to be selected for the next generation. Consequently, a valid mapping can be obtained efficiently and quickly, thanks to this objective. It is formulated as follows:

$$\max T_{slack} = D_{req} - \max_{l \in \text{datapaths}} D_l \tag{6.8}$$

where D_{req} and D_l are defined as in the equation. (4.10).

6.4 Evaluation

In this section, we carry out several experiments for the three architectures to evaluate the benefit of optimization in GenMap. Some of them are based on real chip measurements with the same fabricated chips as in chapter 5. Then, we demonstrate that our approach is effective and practical regardless of the PE array architectures and their implementations.

6.4.1 Evaluation Setup

Table 6.1: Experimental conditions & means

	CC-SOTB	CC-SOTB2	NVCMA
V_{DD}	0.55 V		1.2 V
VBN	-0.8∼+0.4 V by 0.2 V steps		N/A
Delay time	Synopsys HSIM		Scaled
Switching count	Cadence NC-Verilog		N/A
Power consumption	Real chip measurement		

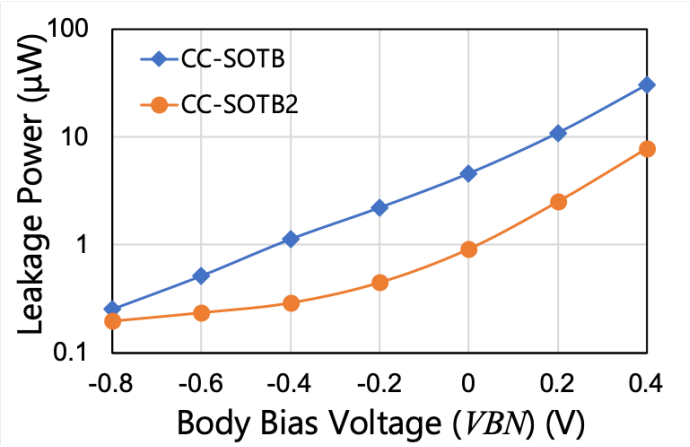


Figure 6.10: Measured leakage power per PE

Before performing the optimization with GenMap, we have to first determine some of the parameters which have appeared in Section 6.3. The delay time of an ALU for each operation and that of an SE (D_v) are simulated using Synopsys HSIM under the conditions listed in Table 6.1, similar to Chapter 4. The delay of NVCMA is estimated by scaling that of CC-SOTB2 since the simulation for NVCMA is not available for the same reason as Section 5.4. For

Table 6.2: Selected application kernels

Kernel	Description	# of nodes	# of edges
af	24-bit alpha blender	24	49
gray	24-bit gray scale	13	27
sepia	8-bit sepia filter	12	27
sf	24-bit sepia filter	20	41
dct	4-point discrete cosine transform	18	40
fft	Radix-4 fast Fourier transform	38	84
aes	Advanced encryption standard	45	94

the dynamic power model, we use the same parameters as the fitting results in Section 5.4.

For CC-SOTB and CC-SOTB2, body bias control is applied so that we measure the leakage power of both chips for each body bias voltage. Fig. 6.10 shows the leakage power per PE for each chip. Although they are implemented with the same Renesas SOTB technology, the used libraries and synthetic conditions of both implementations are different from each other. Therefore, the leakage power is also different.

Benchmark applications

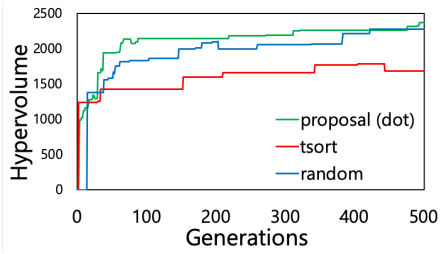
We choose seven application kernels of various sizes from image, signal, and cryptographic processing, as summarized in Table 6.2.

Comparative approaches

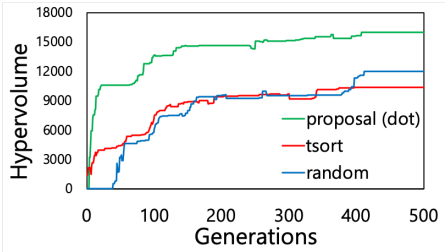
In order to define a comparison basis for our proposed method, mappings for each kernel and for each architecture are also obtained by using two other methods: 1) SPKM [31] and 2) the ILP version of CGRA-ME mapper [21]. Though CGRA-ME supports another version of mapper using simulated annealing [102], in all cases of our experiments, the ILP one is much faster and better quality. CGRA-ME can express a wide range of CGRAs so that the three target architectures are also supported. On the contrary, SPKM cannot perform routing considering SEs within the PEs. Therefore, only the mapping for each node is decided by the two types of ILP: 1) column-wise scattering and 2) row-wise scattering. Then, routing is carried out by using our method.

6.4.2 Quality of Optimization

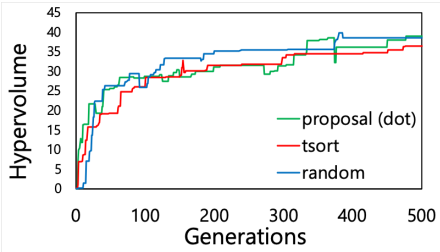
First, we analyze the convergence of the optimization. The *af* application is chosen for the evaluation, which contains a medium-sized DFG in the bench-



(a) Result of CC-SOTB



(b) Result of CC-SOTB2



(c) Result of NVCMA

Figure 6.11: Hypervolume indicator for each generation in the case of *af* application

marks. The hypervolume indicator [122] is used for assessing the quality of the optimization. It can evaluate both the convergence and the diversity of the population by calculating the volume of objective space that is dominated. In order to demonstrate the effectiveness of the proposed initialization method, the other two methods are also evaluated. The first one is a fully randomized initialization (random), and in the second one, utilized PEs are selected randomly, while the PEs are assigned in topological order of the kernel DFG (tsort). The analysis results are shown in Fig. 6.11. The hypervolumes until the five hundredth generation are reported for each architecture and for each method. In the case of CC-SOTB2 (Fig. 6.11(b)), which has the largest search space, the proposed method can quickly improve the quality and the diversity of solutions when compared to the other two methods. Although the *random* method would still improve the population, the convergence speed is slower. The *tsort* version mostly stops the evolution after two hundred generations. The proposed method does work effectively for CC-SOTB (Fig. 6.11(a)). However, even the *random* method can reach the same mapping quality as the proposed one since the search space of CC-SOTB is not so large as CC-SOTB2. On the contrary, there is little difference between these three methods for NVCMA (Fig. 6.11(c)) since it has the smallest search space. The proposed

method is, therefore, effective for a wide range of PE array architectures. Besides, from these results, the size of generations seems to be enough at around two hundred for all cases. Considering more complicated kernels such as *aes*, we set the maximum generation size to three hundred in the subsequent evaluations.

6.4.3 Mapping Ability

Next, we compare the quality of the obtained mappings between GenMap and two other methods. Fig. 6.12 and Table 6.3 present a comparison of the optimized wire length. GenMap produces several optimized solutions so that we can choose from two types of solutions: 1) $\text{GenMap}_{\text{wire}}$ and 2) $\text{GenMap}_{\text{width}}$. The first one yields the shortest wire length, and the second is mapped with the smallest width. If the mapping process does not finish within 24 hours, it is regarded as a timeout. **TO** and **F** in Table 6.3 mean the timeout and failure to map the kernel, respectively. Likewise, a comparison of the optimized map width is shown in Fig. 6.13 and Table 6.4.

Table 6.3: Comparison of wire length for each method

	CC-SOTB						
	af	*gray*	*sepia*	*sf*	*dct*	*fft*	*aes*
$\text{GenMap}_{\text{wire}}$	41	24	28	35	44	167	132
$\text{GenMap}_{\text{width}}$	53	24	30	35	67	167	132
CGRA-ME	45	43	35	46	66	**TO**	**TO**
SPKM	57	34	32	**F**	50	**TO**	**F**
	CC-SOTB2						
	af	*gray*	*sepia*	*sf*	*dct*	*fft*	*aes*
$\text{GenMap}_{\text{wire}}$	38	27	32	34	44	160	167
$\text{GenMap}_{\text{width}}$	56	27	32	34	65	174	172
CGRA-ME	69	40	38	43	64	**TO**	**TO**
SPKM	62	29	41	48	57	**TO**	**F**
	NVCMA						
	af	*gray*	*sepia*	*sf*	*dct*	*fft*	*aes*
$\text{GenMap}_{\text{wire}}$	71	38	41	56	61	212	232
$\text{GenMap}_{\text{width}}$	83	39	42	56	64	212	235
CGRA-ME	103	53	45	61	88	**TO**	**TO**
SPKM	79	40	44	64	85	**TO**	**F**

Table 6.4: Comparison of mapping with for each method

	CC-SOTB						
	af	*gray*	*sepia*	*sf*	*dct*	*fft*	*aes*
$\text{GenMap}_{\text{wire}}$	5	2	4	3	6	8	10
$\text{GenMap}_{\text{width}}$	3	2	3	3	4	8	10
CGRA-ME	3	2	3	3	4	**TO**	**TO**
SPKM	4	3	4	**F**	5	**TO**	**F**
	CC-SOTB2						
	af	*gray*	*sepia*	*sf*	*dct*	*fft*	*aes*
$\text{GenMap}_{\text{wire}}$	6	2	3	3	6	9	12
$\text{GenMap}_{\text{width}}$	3	2	3	3	4	8	11
CGRA-ME	3	2	3	3	4	**TO**	**TO**
SPKM	3	3	3	4	5	**TO**	**F**
	NVCMA						
	af	*gray*	*sepia*	*sf*	*dct*	*fft*	*aes*
$\text{GenMap}_{\text{wire}}$	4	6	4	3	7	8	8
$\text{GenMap}_{\text{width}}$	3	2	3	3	4	8	7
CGRA-ME	3	2	3	3	4	**TO**	**TO**
SPKM	3	3	4	4	5	**TO**	**F**

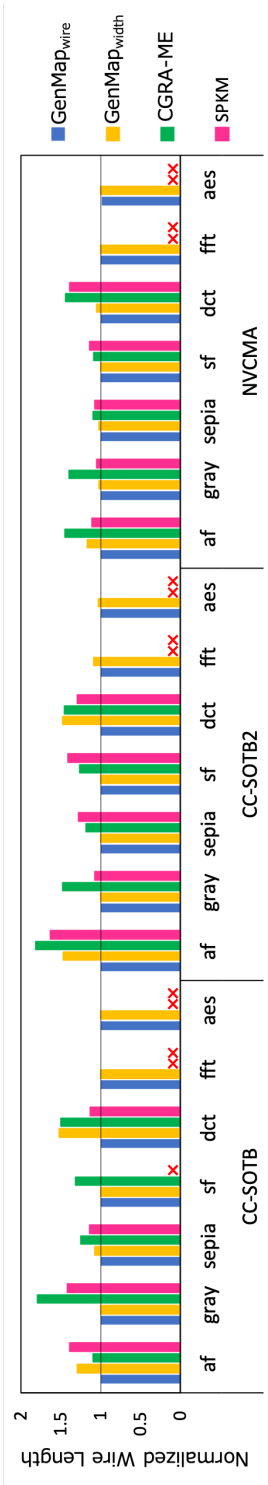


Figure 6.12: Optimized wire length for each architecture and for each method

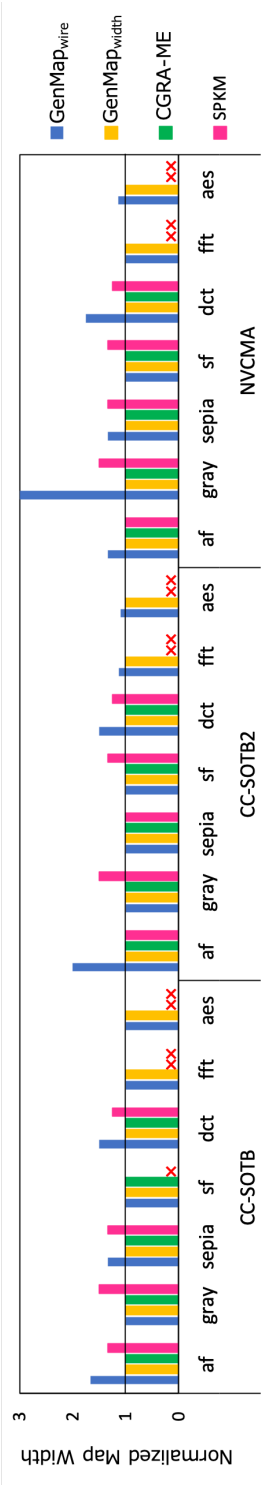


Figure 6.13: Optimized mapping width for each architecture and for each method

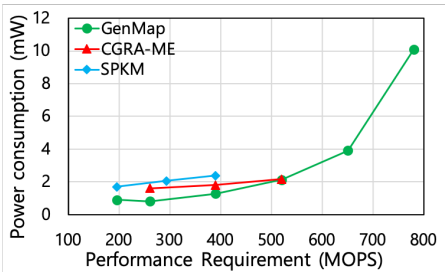
For all three architectures, GenMap can map all seven kernels within a few hours. CGRA-ME and SPKM can map small DFGs quickly (e.g., within 10 seconds for *gray*). Nevertheless, they cannot find a valid mapping of both *fft* and *aes* for all architectures within 24 hours due to less scalability of the ILP-based methods. In addition, the SPKM fails to map the *sf* kernel for CC-SOTB and *aes* for all architectures. In contrast, GenMap is robust against an increase in the complexity and the size of DFGs by using a genetic algorithm. Our approach, therefore, demonstrates a higher mapping ability than the other two methods.

CGRA-ME is good at resource saving thanks to its sophisticated ILP formulation so that it can map the five kernels with the smallest width for all cases. In the case of GenMap, if we give priority to the mapping width (choosing $\text{GenMap}_{\text{width}}$), the same mapping width is also achieved, as shown in Fig. 6.13. Moreover, $\text{GenMap}_{\text{width}}$ shows an average reduction of 15.7% wire length compared to the CGRA-ME since NSGA-II optimizes all objectives simultaneously. Although SPKM succeeded in mapping the same kernels as CGRA-ME except for the CC-SOTB architecture, it shows around 20% larger width than CGRA-ME and $\text{GenMap}_{\text{width}}$. Focusing on $\text{GenMap}_{\text{wire}}$, it achieves the shortest wire length for all cases. Compared to CGRA-ME and SPKM, on average, 26.8% and 19.8% of wire lengths are saved, respectively.

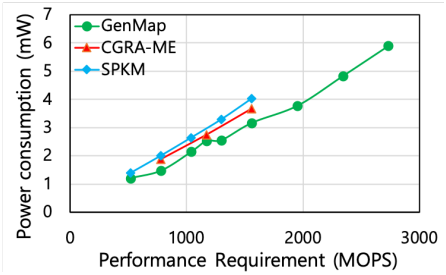
6.4.4 Energy Consumption and Speed Up

Lastly, we discuss the power reduction achieved by GenMap. Power optimization of GenMap depends on the target frequency, as explained in Section 6.3. In order to evaluate this effect, the benchmark applications are optimized at various kinds of target frequencies. Please note that the frequency here means the time interval of data input to the PE array and is not the operating frequency for the system, including the micro-controller. Among the obtained mappings, we choose the best case for estimated power consumption and then measure the actual power consumption. Unfortunately, the fabricated chip of CC-SOTB contains a defect in the PE array, and the upper two PE rows do not work properly. Thus, the number of PE row is limited to be six for CC-SOTB in this power evaluation. CGRA-ME and SPKM cannot determine the pipeline structure for CC-SOTB2 by themselves. Hence, two types of composition — 1) non-pipelined and 2) fully-pipelined — are employed for their evaluations. The first does not use any pipeline registers, while the second activates all of them. Then, one with the smallest power consumption is picked up.

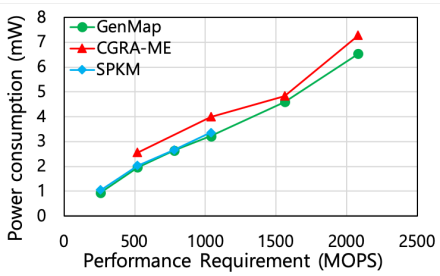
Fig. 6.14 shows the power consumption for each architecture when the *gray* kernel is mapped. The x-axis represents the performance defined in MOPS



(a) Result of CC-SOTB



(b) Result of CC-SOTB2



(c) Result of NVCMA

Figure 6.14: Comparison of power consumption for each architecture in the case of *gray* application

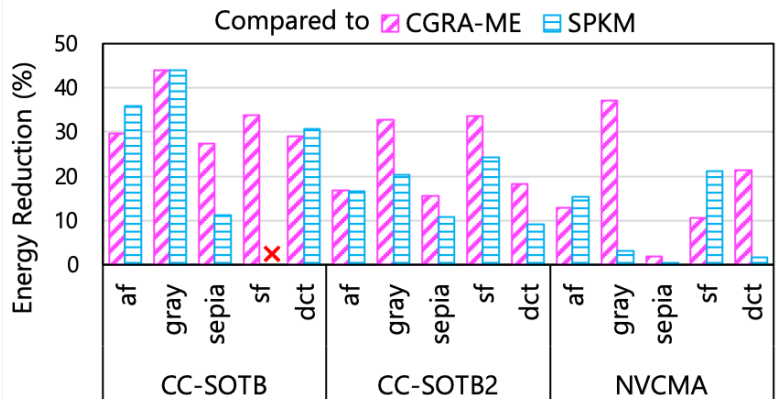


Figure 6.15: Average energy reduction by GenMap

(Million Operations Per Second), instead of the target frequency. This is because each method could generate a mapping with different width and thereby achieve different throughput due to loop unrolling count. Both CGRA-ME and SPKM do not consider the target frequency. Therefore, in both cases, the power consumption increases linearly according to the increase in performance.

On the other hand, GenMap does cut down unnecessary power consumption, especially when low performance is required since its optimization focuses not only on the mapping but also the body bias effect (CC-SOTB and CC-SOTB2) and the pipeline structure (CC-SOTB2). Among the other benchmarks, similar trends are observed (the full results are shown in Appendix D).

Average energy reduction for each application across various performance levels is summarized in Fig. 6.15. In the case of CC-SOTB, 46.8% and 41.3% of energy consumption are reduced on average, compared to CGRA-ME and SPKM, respectively. If the body bias is not controlled adaptively, the leakage current accounts for around 40∼60% of power consumption in the CC-SOTB. Thus, the mapping with body bias optimization contributes to considerable energy reduction. For CC-SOTB2, GenMap archives 23.4% and 16.0% smaller energy consumption than CGRA-ME and SPKM, respectively, thanks to optimizing the pipeline structure and the body bias simultaneously. Although unlike the other two architectures, NVCMA has no opportunity for optimization other than the mapping, GenMap demonstrates 25.1% and 12.1% of energy saving compared to CGRA-ME and SPKM, respectively.

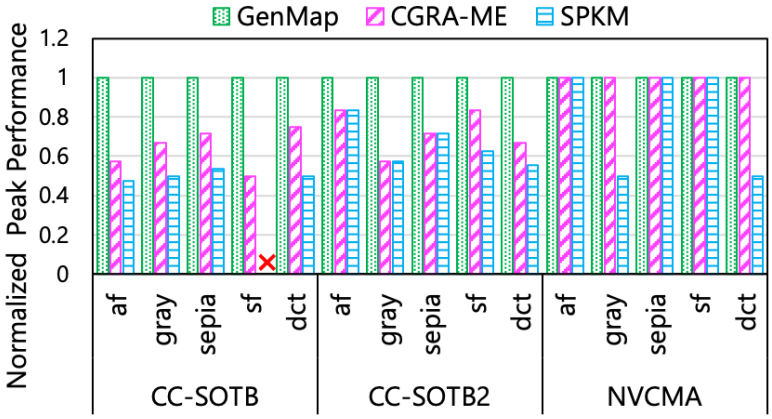


Figure 6.16: Peak Performance for each method

Furthermore, GenMap can satisfy more severe timing constraints than the other two methods. For instance, it results from a forward bias at the expense of increased leakage power. Thereby, GenMap presents 2.0x and 1.55x average speedups for CC-SOTB over CGRA-ME and SPKM, respectively. The pipeline optimization of GenMap also helps CC-SOTB2 to work at higher performance, and consequently, the peak performance is improved by 1.37x and 1.50x than CGRA-ME and SPKM, respectively. On the other hand, GenMap and CGRA-ME archive the same performance for NVCMA since they provide the same mapping width, i.e., loop unrolling count. However, SPKM maps

gray and *dct* kernels into larger widths of PEs than the other methods. This is why SPKM shows lower peak performance for NVCMA.

6.5 Summary

This chapter has introduced GenMap, which is a new mapping optimization method for CGRAs. GenMap is based on a multi-objective genetic algorithm called NSGA-II so that it can optimize the mapping from various kinds of viewpoints, including power consumption. The flexibility in the objective functions enables GenMap to provide any kind of optimization associated with the mapping. As the results of experiments conducted with three fabricated chips, GenMap contributes to a 15.7% reduction of wire length compared to conventional methods. Moreover, GenMap shows energy-saving mappings as well as higher peak performance.

7

Conclusion and future work

7.1 Conclusion

CGRAs are promising hardware accelerators with a reconfigurable PE array. In particular, SF-CGRAs, such as CMA, bring high energy efficiency by employing spatial mapping strategies. However, only a few mapping methods for SF-CGRAs attempt to reduce the power consumption aggressively since the transistor scaling by itself has made a sufficient improvement of the energy efficiency for a few decades. Towards the Post-Moore Era, the mapping optimization combined with some low power techniques, such as body biasing, is needed.

This thesis first proposes two important techniques in power optimization for CGRAs: 1) body bias optimization method based on Integer Linear Program to minimize the leakage current and 2) dynamic power estimation method. Then, GenMap, a general optimization framework for CGRAs based on a multi-objective genetic algorithm, is proposed. For a case study, the above two techniques are integrated into GenMap as an objective function of optimization. The experimental results show the well-formulated search space exploration of GenMap achieves shorter total wire length and better utilization of PE array compared to the existing two approaches. Besides, up to 46.8% of energy saving is offered by the integrated techniques for aggressive power optimization. In addition, the body bias control contributes to up to 2× speed up of performance thanks to the forward biassing.

7.2 Future work

The evaluation results about GenMap are obtained with the real chip experiments to demonstrate the effectiveness for practical use. However, the tested chips are fabricated with older generation technology than the latest one. Thus, the question remains whether the proposed techniques are also useful for the latest technology nodes such as FinFETs. Although the fabrication cost is prohibitively expensive, we can use some free process design kits such as FreePDK15 [123] and ASAP 7nm [124]. For further investigation on the benefit from the proposed method, we should implement the CGRAs with such advanced processes and evaluate our method for these implementations.

Besides, the proposed method has been applied only to a family of CMA so that enough study on other CGRAs is not undertaken yet. Nonetheless, it already turned out that GenMap can be used for some spatial mapping CGRAs like [45]. We, therefore, have to extend our evaluation to other CGRAs.

As argued in Section 2.1, task partitioning in advance of the mapping is sometimes needed due to the limited size of the PE array. Considering that the target application is represented as a directed graph, the problem of finding an optimal task partitioning is defined as a graph partitioning problem. However, finding a good quality partitioning is also an NP-complete problem [50]. The optimization of both mapping and partitioning are tightly dependent. Therefore, we had no choice but to sacrifice an opportunity to optimize either mapping or partitioning. For instance, partitioning of the kernel is solved with a sophisticated heuristic, whereas the mapping is handled by a greedy algorithm as in [48]. For future work, a novel method without loss of optimality in both mapping and partitioning for CGRA is required.

Bibliography

[1] C. Moore, "Data processing in exascale-class computer systems," in *The Salishan Conference on High Speed Computing*. sn, 2011. [Online]. Available: https://www.lanl.gov/conferences/salishan/salishan2011/3moore.pdf

[2] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE transactions on evolutionary computation*, vol. 6, no. 2, pp. 182–197, 2002.

[3] R. H. Dennard, F. H. Gaensslen, H.-N. Yu, V. L. Rideout, E. Bassous, and A. R. LeBlanc, "Design of ion-implanted MOSFET's with very small physical dimensions," *IEEE Journal of Solid-State Circuits*, vol. 9, no. 5, pp. 256–268, 1974.

[4] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, 2017, pp. 1–12.

[5] A. Yang, "Deep Learning Training at Scale Spring Crest Deep Learning Accelerator (Intel® Nervana™ NNP-T)," in *2019 IEEE Hot Chips 31 Symposium (HCS)*. IEEE, 2019, pp. 1–20.

[6] S. Ward-Foxton. Gyrfalcon unveils fourth ai accelerator chip — ee times. [Online]. Available: https://www.eetimes.com/gyrfalcon-unveils-fourth-ai-accelerator-chip/#

[7] J. L. Hennessy and D. A. Patterson, *Computer Architecture, Sixth Edition: A Quantitative Approach*, 6th ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2017.

[8] H. Amano, *Principles and Structures of FPGAs*. Springer, 2018.

[9] P. Papaphilippou and W. Luk, "Accelerating Database Systems Using FPGAs: A Survey," in *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*, 2018, pp. 125–1255.

[10] K. Guo, S. Zeng, J. Yu, Y. Wang, and H. Yang, "[DL] A survey of FPGA-based neural network inference accelerators," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 12, no. 1, pp. 1–26, 2019.

[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: Taxonomy, Challenges, and Applications," *ACM Computing Surveys (CSUR)*, vol. 52, no. 6, pp. 1–39, 2019.

[12] A. K. Jain, D. L. Maskell, and S. A. Fahmy, "Are coarse-grained overlays ready for general purpose application acceleration on fpgas?" in *2016 IEEE 14th Intl Conf on Dependable, Autonomic and Secure Computing, 14th Intl Conf on Pervasive Intelligence and Computing, 2nd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech)*. IEEE, 2016, pp. 586–593.

[13] X. Li, A. K. Jain, D. L. Maskell, and S. A. Fahmy, "A time-multiplexed FPGA overlay with linear interconnect," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018, pp. 1075–1080.

[14] A. Werner, F. Fricke, K. Shahin, F. Werner, and M. Hübner, "Automatic Toolflow for VCGRA Generation to Enable CGRA Evaluation for Arithmetic Algorithms," in *International Symposium on Applied Reconfigurable Computing*. Springer, 2019, pp. 277–291.

[15] I. Taras and J. H. Anderson, "Impact of FPGA Architecture on Area and Performance of CGRA Overlays," in *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2019, pp. 87–95.

[16] J. Mandebi Mbongue, D. Tchuinkou Kwadjo, and C. Bobda, "Flexitask: A flexible fpga overlay for efficient multitasking," in *Proceedings of the 2018 on Great Lakes Symposium on VLSI*, 2018, pp. 483–486.

[17] C. Liu, H.-C. Ng, and H. K.-H. So, "QuickDough: A rapid FPGA loop accelerator design framework using soft CGRA overlay," in *2015 International Conference on Field Programmable Technology (FPT)*. IEEE, 2015, pp. 56–63.

[18] L. B. D. Silva, R. Ferreira, M. Canesche, M. M. Menezes, M. D. Vieira, J. Penha, P. Jamieson, and J. A. M. Nacif, "READY: A Fine-Grained Multithreading Overlay Framework for Modern CPU-FPGA Dataflow Applications," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 18, no. 5s, pp. 1–20, 2019.

[19] M. Hamzeh, A. Shrivastava, and S. Vrudhula, "EPIMap: Using epimorphism to map applications on CGRAs," in *Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE*. IEEE, 2012, pp. 1280–1287.

[20] S. Dave, M. Balasubramanian, and A. Shrivastava, "RAMP: resource-aware mapping for CGRAs," in *Proceedings of the 55th Annual Design Automation Conference*. ACM, 2018, p. 127.

[21] M. J. Walker and J. H. Anderson, "Generic connectivity-based CGRA mapping via integer linear programming," in *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2019, pp. 65–73.

[22] J. Gu, S. Yin, L. Liu, and S. Wei, "Energy-aware loops mapping on multi-vdd CGRAs without performance degradation," in *Design Automation Conference (ASP-DAC), 2017 22nd Asia and South Pacific*. IEEE, 2017, pp. 312–317.

[23] S. M. Jafri, M. A. Tajammul, A. Hemani, K. Paul, J. Plosila, and H. Tenhunen, "Energy-aware-task-parallelism for efficient dynamic voltage, and frequency scaling, in cgras," in *2013 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*. IEEE, 2013, pp. 104–112.

[24] Y. Matsushita, H. Okuhara, K. Masuyama, Y. Fujita, R. Kawano, and H. Amano, "Body bias grain size exploration for a coarse grained reconfigurable accelerator," in *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, Aug 2016, pp. 1–4.

[25] N. A. V. Doan, Y. Matsushita, N. Ando, H. Okuhara, and H. Amano, "Multi-objective optimization for application mapping and body bias control on a CGRA," in *2017 IEEE 11th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSOC)*. IEEE, 2017, pp. 143–150.

[26] J. M. Kühn, A. B. Ahmed, H. Okuhara, H. Amano, O. Bringmann, and W. Rosenstiel, "MuCCRA4-BB: A fine-grained body biasing capable

DRP," in *2016 IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS XIX)*. IEEE, 2016, pp. 1–3.

[27] Z. Chen, H. Zhou, and J. Gu, "R-Accelerator: An RRAM-Based CGRA Accelerator With Logic Contraction," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 11, pp. 2655–2667, 2019.

[28] T. Ikezoe, H. Amano, J. Akaike, K. Usami, M. Kudo, K. Hiraga, Y. Shuto, and K. Yagami, "A Coarse Grained-Reconfigurable Accelerator with energy efficient MTJ-based Non-volatile Flip-flops," in *2018 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*. IEEE, 2018, pp. 1–6.

[29] K. Usami, S. Akiba, H. Amano, T. Ikezoe, K. Hiraga, K. Suzuki, and Y. Kanda, "Non-Volatile Coarse Grained Reconfigurable Array Enabling Two-step Store Control for Energy Minimization," in *2020 IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS)*. IEEE, 2020, pp. 1–3.

[30] O. Akbari, M. Kamal, A. Afzali-Kusha, M. Pedram, and M. Shafique, "PX-CGRA: Polymorphic approximate coarse-grained reconfigurable architecture," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018, pp. 413–418.

[31] J. W. Yoon, A. Shrivastava, S. Park, M. Ahn, R. Jeyapaul, and Y. Paek, "SPKM: A novel graph drawing based algorithm for application mapping onto coarse-grained reconfigurable architectures," in *Proceedings of the 2008 Asia and South Pacific Design Automation Conference*. IEEE Computer Society Press, 2008, pp. 776–782.

[32] N. Ozaki, Y. Yasuda, M. Izawa, Y. Saito, D. Ikebuchi, H. Amano, H. Nakamura, K. Usami, M. Namiki, and M. Kondo, "Cool Mega-Arrays: Ultralow-Power Reconfigurable Accelerator Chips," *IEEE Micro*, vol. 31, no. 6, pp. 6–18, Nov 2011.

[33] A. Podobas, K. Sano, and S. Matsuoka, "A Survey on Coarse-Grained Reconfigurable Architectures From a Performance Perspective," *IEEE Access*, vol. 8, pp. 146 719–146 743, 2020.

[34] M. A. Shami and A. Hemani, "Partially reconfigurable interconnection network for dynamically reprogrammable resource array," in *ASIC, 2009. ASICON'09. IEEE 8th International Conference on*. IEEE, 2009, pp. 122–125.

[35] H. Singh, M.-H. Lee, G. Lu, F. J. Kurdahi, N. Bagherzadeh, and E. M. Chaves Filho, "MorphoSys: an integrated reconfigurable system for data-parallel and computation-intensive applications," *IEEE transactions on computers*, no. 5, pp. 465–481, 2000.

[36] B. Mei, F.-J. Veredas, and B. Masschelein, "Mapping an H. 264/AVC decoder onto the ADRES reconfigurable architecture," in *Field Programmable Logic and Applications, 2005. International Conference on*. IEEE, 2005, pp. 622–625.

[37] D. Lee, M. Jo, K. Han, and K. Choi, "FloRA: Coarse-grained reconfigurable architecture with floating-point operation capability," in *Field-Programmable Technology, 2009. FPT 2009. International Conference on*. IEEE, 2009, pp. 376–379.

[38] H. Schmit, D. Whelihan, A. Tsai, M. Moe, B. Levine, and R. R. Taylor, "PipeRench: A virtualized programmable datapath in 0.18 micron technology," in *Custom Integrated Circuits Conference, 2002. Proceedings of the IEEE 2002*. IEEE, 2002, pp. 63–66.

[39] B. Levine, "Kilocore: Scalable, High Performance and Power Efficient Coarse Grained Reconfigurable Fabrics," in Proc. of International Symposium on Advanced Reconfigurable Systems, 2005, pp. 129–158.

[40] M. Petrov, T. Murgan, F. May, M. Vorbach, P. Zipf, and M. Glesner, "The XPP architecture and its co-simulation within the simulink environment," in *International Conference on Field Programmable Logic and Applications*. Springer, 2004, pp. 761–770.

[41] J. M. Arnold, "S5: The Architecture and Development Flow of a Software Configurable Processor," in *Proc. of the 4th IEEE Int'l Conf. on Field Programmable Technology (ICFPT2005)*, December 2005, pp. 120–128.

[42] L. Liu, D. Wang, M. Zhu, Y. Wang, S. Yin, P. Cao, J. Yang, and S. Wei, "An energy-efficient coarse-grained reconfigurable processing unit for multiple-standard video decoding," *IEEE Transactions on Multimedia*, vol. 17, no. 10, pp. 1706–1720, 2015.

[43] Y. Kim, M. Kiemb, C. Park, J. Jung, and K. Choi, "Resource sharing and pipelining in coarse-grained reconfigurable architecture for domain-specific optimization," in *Design, Automation and Test in Europe*. IEEE, 2005, pp. 12–17.

[44] X. Fan, H. Li, W. Cao, and L. Wang, "DT-CGRA: Dual-track coarse-grained reconfigurable architecture for stream applications," in *Field Programmable Logic and Applications (FPL), 2016 26th International Conference on*. IEEE, 2016, pp. 1–9.

[45] A. Podobas, K. Sano, and S. Matsuoka, "A template-based framework for exploring coarse-grained reconfigurable architectures," in *2020 IEEE 31st International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, 2020, pp. 1–8.

[46] T. Kojima and H. Amano, "A Configuration Data Multicasting Method for Coarse-Grained Reconfigurable Architectures," in *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2018, pp. 239–2393.

[47] I. B. Mahapatra, U. Agarwal, and S. Nandy, "DFG partitioning algorithms for coarse grained reconfigurable array assisted RTL simulation accelerators," in *2018 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT)*. IEEE, 2018, pp. 1–6.

[48] W. Sheng, W. He, J. Jiang, and Z. Mao, "Pareto optimal temporal partition methodology for reconfigurable architectures based on multi-objective genetic algorithm," in *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum*. IEEE, 2012, pp. 425–430.

[49] J. Herrmann, J. Kho, B. Uçar, K. Kaya, and Ü. V. Çatalyürek, "Acyclic partitioning of large directed acyclic graphs," in *2017 17th IEEE/ACM international symposium on cluster, cloud and grid computing (CCGRID)*. IEEE, 2017, pp. 371–380.

[50] O. Moreira, M. Popp, and C. Schulz, "Evolutionary multi-level acyclic graph partitioning," *Journal of Heuristics*, pp. 1–29, 2020.

[51] J. Cong, C. Wu, and Y. Ding, "Cut ranking and pruning: Enabling a general and efficient FPGA mapping solution," in *Proceedings of the 1999 ACM/SIGDA seventh international symposium on Field programmable gate arrays*, 1999, pp. 29–35.

[52] V. Tunbunheng and H. Amano, "Black-Diamond: a Retargetable Compiler Using Graph with Configuration Bits for Dynamically Reconfigurable Architectures," in Proc. of The 14th SASIMI, 2007, pp. 412–419.

[53] S. Dave and A. Shrivastava, "CCF: A CGRA Compilation Framework." [Online]. Available: https://github.com/MPSLab-ASU/ccf

[54] H.-S. Kim, M. Ahn, J. A. Stratton, and W. H. Wen-mei, "Design evaluation of opencl compiler framework for coarse-grained reconfigurable arrays," in *2012 International Conference on Field-Programmable Technology*. IEEE, 2012, pp. 313–320.

[55] M. Mukherjee, A. Fell, and A. Guha, "DFGenTool: A dataflow graph generation tool for coarse grain reconfigurable architectures," in *2017 30th International Conference on VLSI Design and 2017 16th International Conference on Embedded Systems (VLSID)*. IEEE, 2017, pp. 67–72.

[56] M. S. Abdelfattah, D. Han, A. Bitar, R. DiCecco, S. O'Connell, N. Shanker, J. Chu, I. Prins, J. Fender, A. C. Ling *et al.*, "DLA: Compiler and FPGA overlay for neural network inference acceleration," in *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2018, pp. 411–4117.

[57] M. B. Taylor, "A landscape of the new dark silicon design regime," *IEEE Micro*, vol. 33, no. 5, pp. 8–19, 2013.

[58] N. H. Weste and D. Harris, *CMOS VLSI design: a circuits and systems perspective*. Pearson Education India, 2015.

[59] N. S. Kim, T. Austin, D. Baauw, T. Mudge, K. Flautner, J. S. Hu, M. J. Irwin, M. Kandemir, and V. Narayanan, "Leakage current: Moore's law meets static power," *computer*, vol. 36, no. 12, pp. 68–75, 2003.

[60] K. Roy, S. Mukhopadhyay, and H. Mahmoodi-Meimand, "Leakage current mechanisms and leakage reduction techniques in deep-submicrometer CMOS circuits," *Proceedings of the IEEE*, vol. 91, no. 2, pp. 305–327, 2003.

[61] T. Sakurai and A. R. Newton, "Alpha-power law MOSFET model and its applications to CMOS inverter delay and other formulas," *IEEE Journal of solid-state circuits*, vol. 25, no. 2, pp. 584–594, 1990.

[62] J. M. Kühn, H. Amano, W. Rosenstiel, and O. Bringmann, "Leveraging FDSOI through body bias domain partitioning and bias search," in *Design Automation Conference (DAC), 2016 53nd ACM/EDAC/IEEE*. IEEE, 2016, pp. 1–6.

[63] F. Conti, D. Rossi, A. Pullini, I. Loi, and L. Benini, "Energy-efficient vision on the PULP platform for ultra-low power parallel computing," in *Signal Processing Systems (SiPS), 2014 IEEE Workshop on.* IEEE, 2014, pp. 1–6.

[64] M. Hioki, T. Sekigawa, T. Nakagawa, H. Koike, Y. Matsumoto, T. Kawanami, and T. Tsutsumi, "Fully-functional fpga prototype with fine-grain programmable body biasing," in *Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays.* ACM, 2013, pp. 73–80.

[65] M. Hioki and H. Koike, "Low Overhead Design of Power Reconfigurable FPGA with Fine-Grained Body Biasing on 65-nm SOTB CMOS Technology," *IEICE TRANSACTIONS on Information and Systems*, vol. 99, no. 12, pp. 3082–3089, 2016.

[66] Lewis, David and Ahmed, Elias and Cashman, David and Vanderhoek, Tim and Lane, Chris and Lee, Andy and Pan, Philip, "Architectural enhancements in stratix-iii™and stratix-iv™," in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays.* ACM, 2009, pp. 33–42.

[67] K. Masuyama, Y. Fujita, H. Okuhara, and H. Amano, "A 297mops/0.4mw ultra low power coarse-grained reconfigurable accelerator CMA-SOTB-2," in *2015 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, Dec 2015, pp. 1–6.

[68] FD-SOI Technology Platform - STMicroelectronics. [Online]. Available: https://www.st.com/content/st_com/en/about/innovation---technology/FD-SOI/fd-soi-technology-platform.html

[69] Highly integrated 5G mmWave mobile FEMs& TRXs using 22FDX RF. [Online]. Available: https://www.globalfoundries.com/sites/default/files/product-briefs/highly_integrated_5g_mmwave_mobile_fems_and_trxs_using_22fdx_rf_10jun2020.pdf

[70] Ishigaki, Takashi and Tsuchiya, Ryuta and Morita, Yusuke and Sugii, Nobuyuki and Kimura, Shin'ichiro, "Ultralow-power LSI Technology with Silicon on Thin Buried Oxide (SOTB) CMOSFET," *Solid State Circuits Technologies, Jacobus W. Swart (Ed.), ISBN: 978-953-307-045-2, InTech*, pp. 146–156, 2010.

[71] M. Bohr and K. Mistry, "Intel's revolutionary 22 nm transistor technology," *Intel website*, 2011.

[72] TSMC and OIP Ecosystem Partners Deliver Industry ' s First Complete Design Infrastructure for 5nm Process Technology. [Online]. Available: https://pr.tsmc.com/english/news/1987

[73] Samsung Successfully Completes 5nm EUV Development to Allow Greater Area Scaling and Ultra-low Power Benefits – Samsung Global Newsroom. [Online]. Available: https://news.samsung.com/global/samsung-successfully-completes-5nm-euv-development-to-allow-greater-area-scaling-and-ultra-low-power-benefits

[74] C. Lin, B. Greene, S. Narasimha, J. Cai, A. Bryant, C. Radens, V. Narayanan, B. Linder, H. Ho, A. Aiyar *et al.*, "High performance 14nm soi finfet cmos technology with 0.0174 μm 2 embedded dram and 15 levels of cu metallization," in *2014 IEEE International Electron Devices Meeting*. IEEE, 2014, pp. 3–8.

[75] W.-T. Chang, C.-T. Shih, J.-L. Wu, S.-W. Lin, L.-G. Cin, and W.-K. Yeh, "Back-biasing to performance and reliability evaluation of UTBB FDSOI, bulk FinFETs, and SOI FinFETs," *IEEE Transactions on Nanotechnology*, vol. 17, no. 1, pp. 36–40, 2017.

[76] M. Willsey, V. T. Lee, A. Cheung, R. Bodík, 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, no. 3, pp. 407–418, 2018.

[77] G. Ansaloni, P. Bonzini, and L. Pozzi, "Heterogeneous coarse-grained processing elements: A template architecture for embedded processing acceleration," in *2009 Design, Automation & Test in Europe Conference & Exhibition*. IEEE, 2009, pp. 542–547.

[78] B. Mei, A. Lambrechts, J.-Y. Mignolet, D. Verkest, and R. Lauwereins, "Architecture exploration for a reconfigurable architecture template," *IEEE Design & Test of Computers*, vol. 22, no. 2, pp. 90–101, 2005.

[79] A. Lambrechts, P. Raghavan, M. Jayapala, F. Catthoor, and D. Verkest, "Energy-aware interconnect-exploration of coarse grained reconfigurable processors," in *Proc. of Workshop on Application Specific Processors*, vol. 23, 2005.

[80] D. Suh, K. Kwon, S. Kim, S. Ryu, and J. Kim, "Design space exploration and implementation of a high performance and low area coarse grained

reconfigurable processor," in *2012 International Conference on Field-Programmable Technology*. IEEE, 2012, pp. 67–70.

[81] D. Voitsechov, O. Port, and Y. Etsion, "Inter-thread communication in multithreaded, reconfigurable coarse-grain arrays," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2018, pp. 42–54.

[82] X. Man, L. Liu, J. Zhu, and S. Wei, "A General Pattern-Based Dynamic Compilation Framework for Coarse-Grained Reconfigurable Architectures," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[83] S. M. Jafri, G. Serrano, M. Daneshtalab, N. Abbas, A. Hemani, K. Paul, J. Plosila, and H. Tenhunen, "Transpar: Transformation based dynamic parallelism for low power CGRAs," in *2014 24th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2014, pp. 1–8.

[84] B. R. Rau, "Iterative modulo scheduling: An algorithm for software pipelining loops," in *Proceedings of the 27th annual international symposium on Microarchitecture*, 1994, pp. 63–74.

[85] B. Mei, S. Vernalde, D. Verkest, H. De Man, and R. Lauwereins, "Exploiting loop-level parallelism on coarse-grained reconfigurable architectures using modulo scheduling," *IEEE Proceedings-Computers and Digital Techniques*, vol. 150, no. 5, pp. 255–261, 2003.

[86] H. Park, K. Fan, S. A. Mahlke, T. Oh, H. Kim, and H.-s. Kim, "Edge-centric modulo scheduling for coarse-grained reconfigurable architectures," in *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*. ACM, 2008, pp. 166–176.

[87] M. Hamzeh, A. Shrivastava, and S. Vrudhula, "REGIMap: register-aware application mapping on coarse-grained reconfigurable architectures (CGRAs)," in *Design Automation Conference (DAC), 2013 50th ACM/EDAC/IEEE*. IEEE, 2013, pp. 1–10.

[88] S. Yin, X. Yao, D. Liu, L. Liu, and S. Wei, "Memory-aware loop mapping on coarse-grained reconfigurable architectures," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 5, pp. 1895–1908, 2016.

[89] M. Kou, J. Gu, S. Wei, H. Yao, and S. Yin, "TAEM: fast transfer-aware effective loop mapping for heterogeneous resources on CGRA," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[90] Z. Zhao, W. Sheng, Q. Wang, W. Yin, P. Ye, J. Li, and Z. Mao, "Towards Higher Performance and Robust Compilation for CGRA Modulo Scheduling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 9, pp. 2201–2219, 2020.

[91] M. Balasubramanian and A. Shrivastava, "CRIMSON: Compute-Intensive Loop Acceleration by Randomized Iterative Modulo Scheduling and Optimized Mapping on CGRAs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 11, pp. 3300–3310, 2020.

[92] A. Fell, Z. E. Rákossy, and A. Chattopadhyay, "Force-directed scheduling for data flow graph mapping on coarse-grained reconfigurable architectures," in *ReConFigurable Computing and FPGAs (ReConFig), 2014 International Conference on*. IEEE, 2014, pp. 1–8.

[93] G. Lee, S. Lee, K. Choi, and N. Dutt, "Routing-aware application mapping considering steiner points for coarse-grained reconfigurable architecture," in *International Symposium on Applied Reconfigurable Computing*. Springer, 2010, pp. 231–243.

[94] L. Chen and T. Mitra, "Graph minor approach for application mapping on cgras," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 7, no. 3, p. 21, 2014.

[95] M. Karunaratne, A. K. Mohite, T. Mitra, and L.-S. Peh, "HyCUBE: A CGRA with reconfigurable single-cycle multi-hop interconnect," in *Design Automation Conference (DAC), 2017 54th ACM/EDAC/IEEE*. IEEE, 2017, pp. 1–6.

[96] B. Xu, S. Yin, L. Liu, and S. Wei, "Low-Power Loop Parallelization onto CGRA Utilizing Variable Dual VDD," *IEICE TRANSACTIONS on Information and Systems*, vol. 98, no. 2, pp. 243–251, 2015.

[97] S. Yin, D. Liu, L. Sun, L. Liu, and S. Wei, "DFGNet: Mapping dataflow graph onto CGRA by a deep learning approach," in *Circuits and Systems (ISCAS), 2017 IEEE International Symposium on*. IEEE, 2017, pp. 1–4.

[98] D. Liu, S. Yin, G. Luo, J. Shang, L. Liu, S. Wei, Y. Feng, and S. Zhou, "Data-Flow Graph Mapping Optimization for CGRA with Deep Reinforcement Learning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018.

[99] D. Liu, S. Yin, Y. Peng, L. Liu, and S. Wei, "Optimizing spatial mapping of nested loop for coarse-grained reconfigurable architectures," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 11, pp. 2581–2594, 2014.

[100] A. B. Kahng, J. Lienig, I. L. Markov, and J. Hu, *VLSI physical design: from graph partitioning to timing closure.* Springer Science & Business Media, 2011.

[101] L. Zhou, J. Zhang, and H. Liu, "Ant Colony Algorithm for Steiner Tree Problem in CGRA Mapping," in *Information Science and Control Engineering (ICISCE), 2017 4th International Conference on.* IEEE, 2017, pp. 198–202.

[102] 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 *Application-specific Systems, Architectures and Processors (ASAP), 2017 IEEE 28th International Conference on.* IEEE, 2017, pp. 184–189.

[103] S. A. Chin, K. P. Niu, M. Walker, S. Yin, A. Mertens, J. Lee, and J. H. Anderson, "Architecture exploration of standard-cell and fpga-overlay cgras using the open-source cgra-me framework," in *Proceedings of the 2018 International Symposium on Physical Design.* ACM, 2018, pp. 48–55.

[104] S. A. Chin and J. H. Anderson, "An architecture-agnostic integer linear programming approach to CGRA mapping," in *Proceedings of the 55th Annual Design Automation Conference.* ACM, 2018, p. 128.

[105] M. Canesche, M. Menezes, W. Carvalho, F. Torres, P. Jamieson, J. A. Nacif, and R. Ferreira, "TRAVERSAL: A Fast and Adaptive Graph-based Placement and Routing for CGRAs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2020.

[106] T. Nowatzki, M. Sartin-Tarm, L. De Carli, K. Sankaralingam, C. Estan, and B. Robatmili, "A General Constraint-Centric Scheduling Framework for Spatial Architectures," in *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser.

PLDI '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 495–506.

[107] T. Kojima, N. Ando, Y. Matshushita, H. Okuhara, N. A. V. Doan, and H. Amano, "Real chip evaluation of a low power CGRA with optimized application mapping," in *Proceedings of the 9th International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies*. ACM, 2018, p. 13.

[108] Y. Park, H. Park, and S. Mahlke, "CGRA express: accelerating execution using dynamic operation fusion," in *Proceedings of the 2009 international conference on Compilers, architecture, and synthesis for embedded systems*. ACM, 2009, pp. 271–280.

[109] G. Ansaloni, L. Pozzi, K. Tanimura, and N. Dutt, "Slack-aware scheduling on coarse grained reconfigurable arrays," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011*. IEEE, 2011, pp. 1–4.

[110] Hiroki Nagatomi, et. al., "A 361nA Thermal Run-away Immune VBB Generator using Dynamic Substrate Controlled Charge Pump for Ultra Low Sleep Current Logic on 65nm SOTB," in *Proceedings of the SOI-3D-Subthreshold Microelectronics Technology Unified Conference*, Oct. 2014, pp. 1–2.

[111] M. Blagojević, M. Cochet, B. Keller, P. Flatresse, A. Vladimirescu, and B. Nikolić, "A fast, flexible, positive and negative adaptive body-bias generator in 28nm fdsoi," in *2016 IEEE Symposium on VLSI Circuits (VLSI-Circuits)*, June 2016, pp. 1–2.

[112] B. Korte and J. Vygen. Springer Berlin Heidelberg, 2018.

[113] L. Cheng, D. Chen, and M. D. Wong, "GlitchMap: An FPGA technology mapper for low power considering glitches," in *Design Automation Conference, 2007. DAC'07. 44th ACM/IEEE*. IEEE, 2007, pp. 318–323.

[114] S. Cromar, J. Lee, and D. Chen, "FPGA-targeted high-level binding algorithm for power and area reduction with glitch-estimation," in *Design Automation Conference, 2009. DAC'09. 46th ACM/IEEE*. IEEE, 2009, pp. 838–843.

[115] A. A. Gaffar, J. A. Clarke, and G. A. Constantinides, "Modeling of glitch effects in FPGA based arithmetic circuits," in *Field Programmable Technology, 2006. FPT 2006. IEEE International Conference on*. IEEE, 2006, pp. 349–352.

[116] Y. Take, H. Matsutani, D. Sasaki, M. Koibuchi, T. Kuroda, and H. Amano, "3D NoC with inductive-coupling links for building-block SiPs," *IEEE Transactions on Computers*, vol. 63, no. 3, pp. 748–763, 2012.

[117] C. A. C. Coello, G. B. Lamont, D. A. Van Veldhuizen *et al.*, *Evolutionary algorithms for solving multi-objective problems*. Springer, 2007, vol. 5.

[118] L. Davis, "Adapting operator probabilities in genetic algorithms," in *Proceedings of the third international conference on Genetic algor ithms*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1989, pp. 61–69.

[119] "Graphviz - Graph Visualization Software," https://www.graphviz.org/.

[120] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[121] T. Ikezoe, T. Kojima, and H. Amano, "A Coarse-Grained Reconfigurable Architecture with a Fault Tolerant Non-Volatile Configurable Memory," in *2019 International Conference on Field-Programmable Technology (ICFPT)*. IEEE, 2019, pp. 81–89.

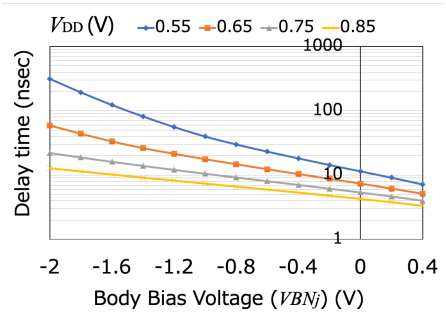
[122] E.-G. Talbi, *Metaheuristics: From Design to Implementation*. Jhon Wiley & Sons, Jun 2009.

[123] K. Bhanushali and W. R. Davis, "FreePDK15: An open-source predictive process design kit for 15nm FinFET technology," in *Proceedings of the 2015 Symposium on International Symposium on Physical Design*, 2015, pp. 165–170.

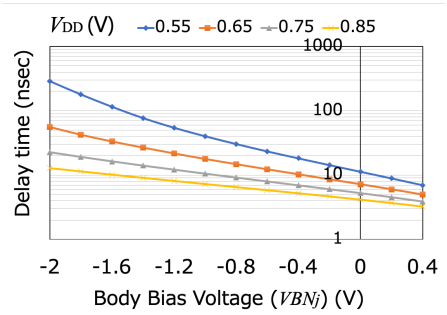
[124] L. T. Clark, V. Vashishtha, L. Shifren, A. Gujja, S. Sinha, B. Cline, C. Ramamurthy, and G. Yeric, "ASAP7: A 7-nm finFET predictive process design kit," *Microelectronics Journal*, vol. 53, pp. 105–115, 2016.

Appendicies

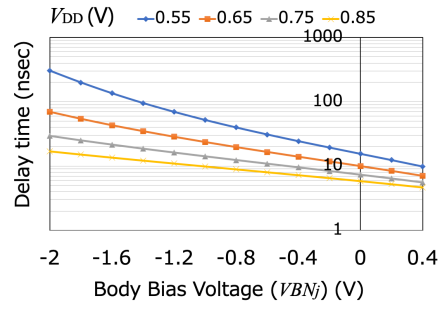
A Full results of the simulated delay time



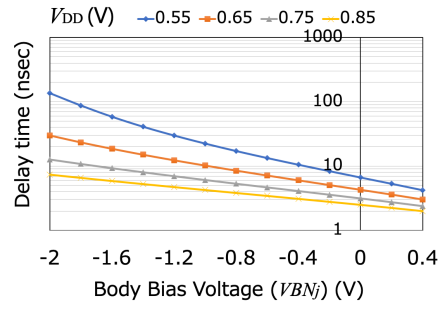
(a) ADD operation



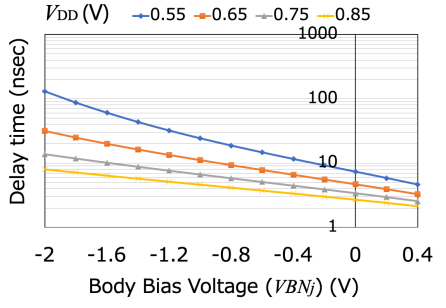
(b) SUB operation



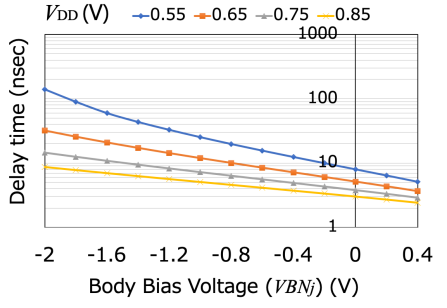
(c) MULT operation



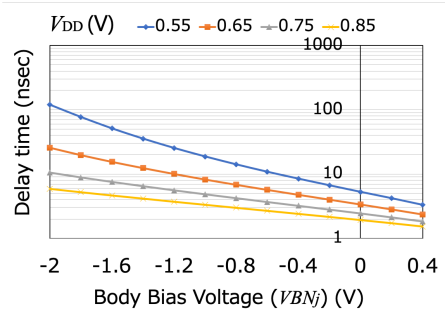
(d) SL operation



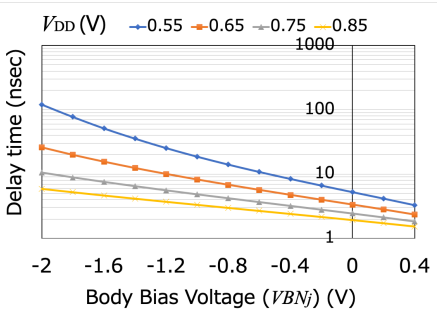
(e) SR operation



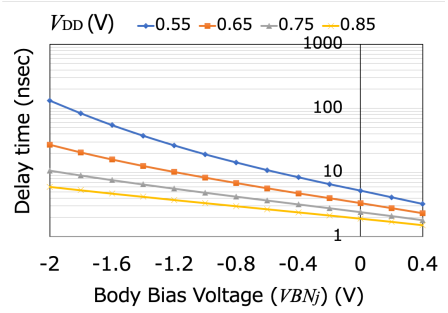
(f) SRA operation



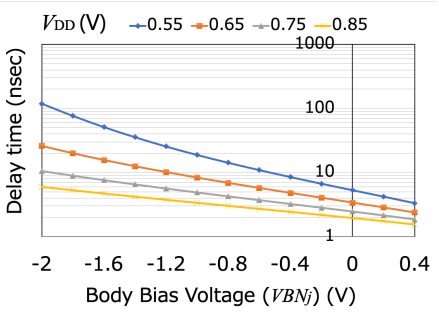
(g) AND operation



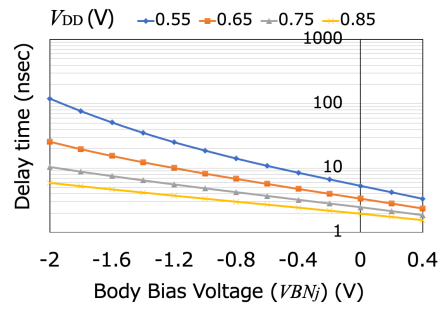
(h) OR operation



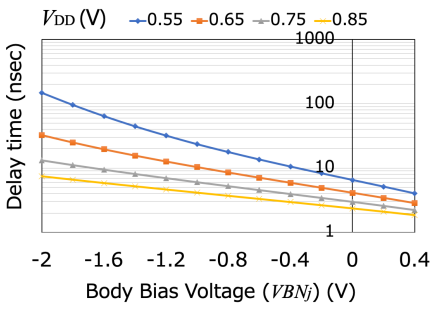
(i) NOT operation



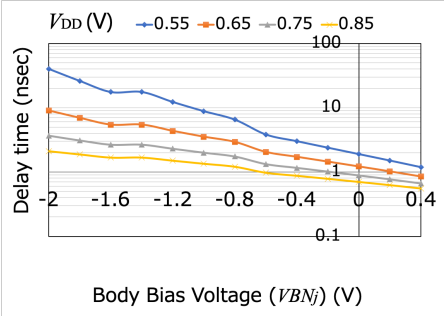
(j) XOR operation



(k) CAT operation



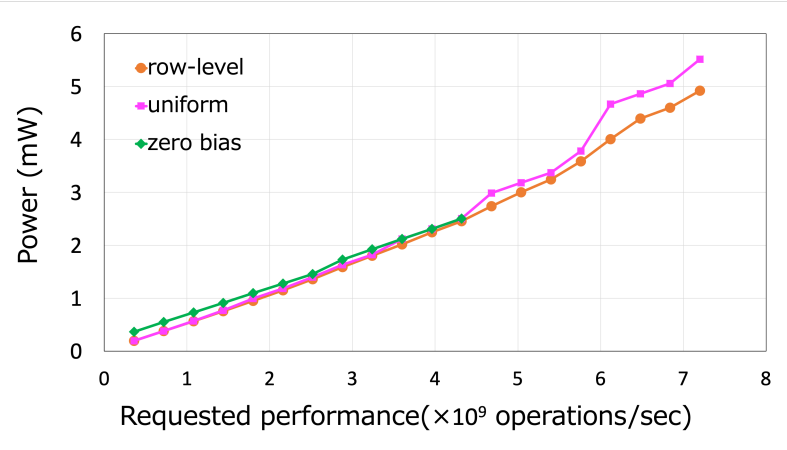
(l) SEL operation



(m) SE

Figure A.2: Delay time of ALU for each operation and SE simulated with Synopsys HSIM (CC-SOTB2)

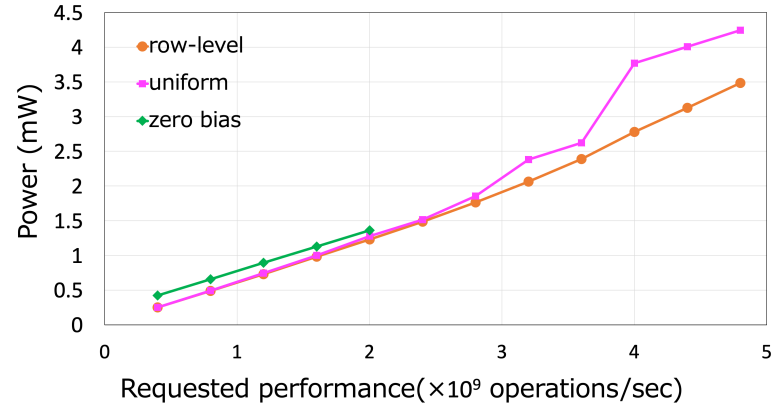
B Full results of body bias optimization



(a) *af* application

(b) *gray* application

(c) *sepia* application



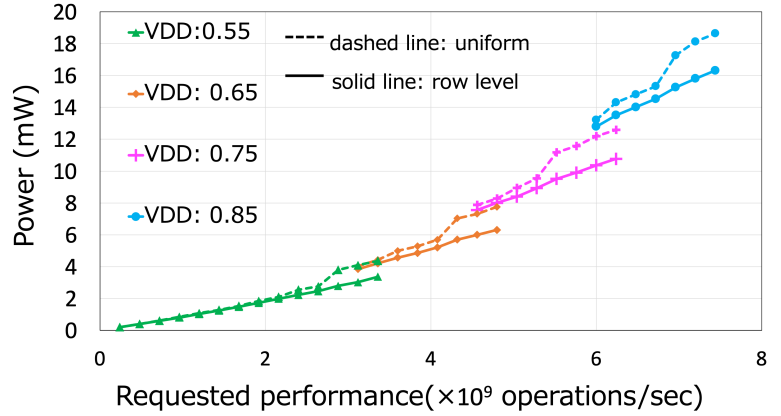
(d) *sf* application

(e) *dct* application

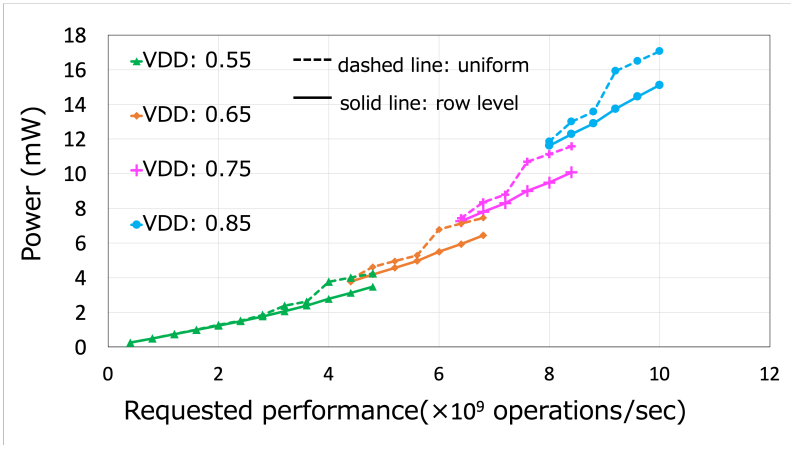
Figure B.1: Comparison between the row-level body bias control and the other policies for each application (CC-SOTB2)

(a) *af* application

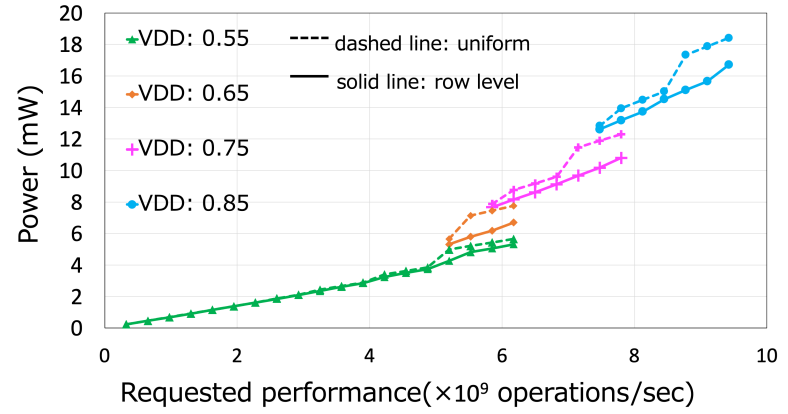
(b) *gray* application



(c) *sepia* application



(d) *sf* application



(e) *dct* application

Figure B.2: Comparison between the row-level body bias control and the uniform control considering V_{DD} control for each application (CC-SOTB2)

C Analysis of the crossover and mutation probabilities

The crossover and mutation probabilities used in Chapter 6 are commonly-used values, and they are also appropriate for our cases according to the following evaluation.

We compared ten conditions of the probabilities using the hypervolume indicator. Here, the mutation probability is equal to 1− the crossover(cx) one. The *af* application is optimized for CC-SOTB2 within 300 generations for each condition. The CC-SOTB2 brings the largest search space, and the difference of searching capability due to the above conditions has a great impact on the mapping quality and convergence speed considerably. Then, the same experiments are repeated several times in order to confirm the robustness.

Fig. C.1 shows the hypervolume evaluations for the best five conditions. The condition of 0.7 crossover probability (i.e., 0.3 for mutation), which we used in Chapter 6, shows the most efficient search. For other samples, similar tendencies are observed.

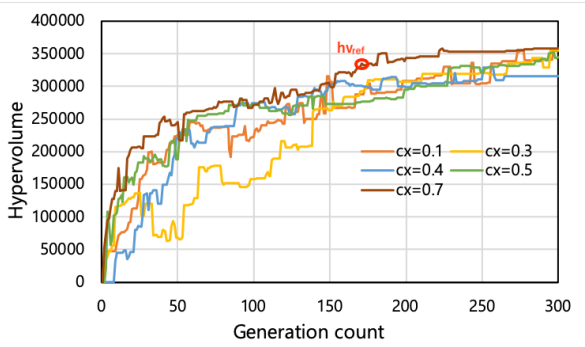


Figure C.1: Comparison of the hypervolume evolution (sample #0)

Next, we analyzed how fast the convergence speed is for each condition. For an explanation, we introduce the following notation:

$hv_{P_{cx}}(n)$: hypervolume at n-th generation with P_{cx} crossover probability (1)

Here, we regard $hv_{ref} = 0.9 \times hv_{0.7}(300)$ as a baseline of comparison, as shown in Fig. C.1. The comparison results are shown in Table. C.1 and Table. C.2. "N/A"s mean those conditions cannot reach the baseline until 300 generations. In Table. C.2, the negative values indicate those conditions reach the baseline earlier than the case of 0.7 crossover probability. Although 0.7 crossover probability does not always show the fastest convergence speed as summarized in

Table. C.2, it shows the most stable searching. For example, 0.8 crossover probability in sample #2 reaches the baseline earlier, while it does not work well for the other samples. Hence, 0.7 crossover probability is appropriate for our cases as well.

Table C.1: Generation count reaching hv_{ref}

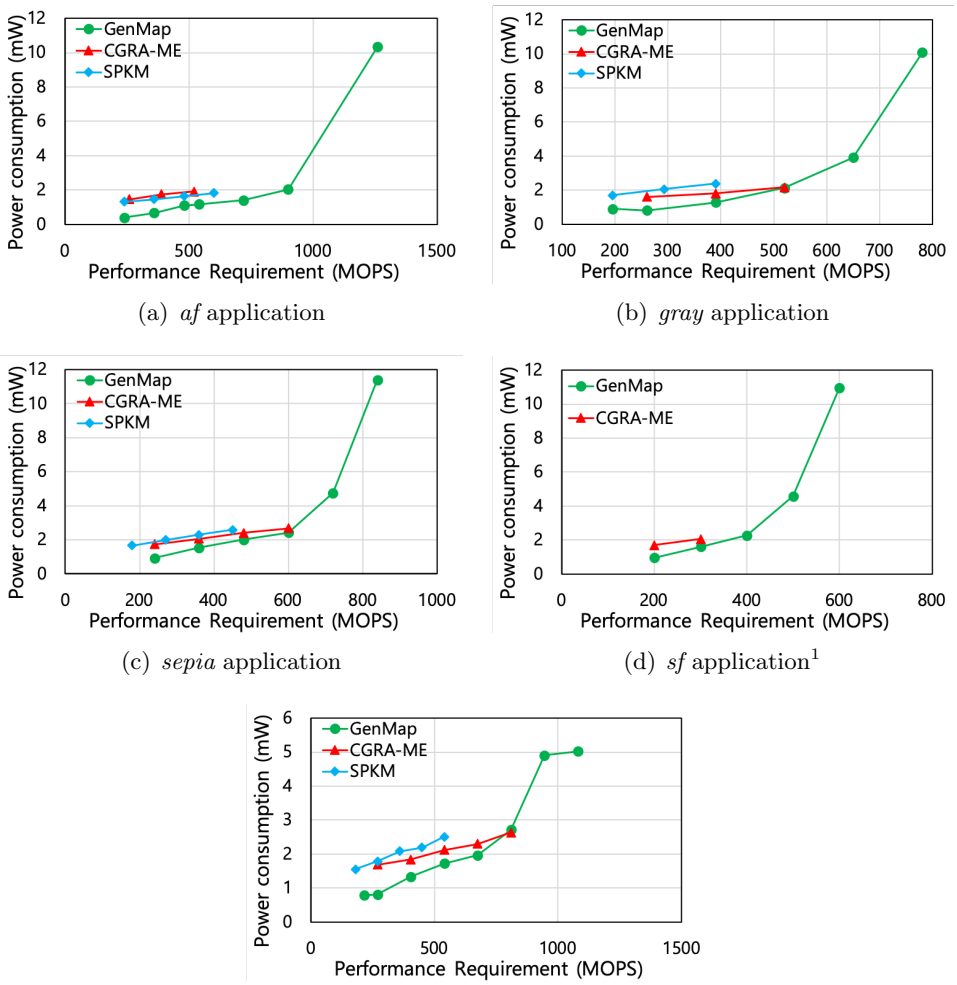
	crossover probability										
sample	0	0.1	0.2	0.3	0.4	0.5	0.6	**0.7**	0.8	0.9	1
#0	N/A	225	N/A	267	251	228	N/A	**167**	N/A	N/A	N/A
#1	N/A	79	173	143	69	132	N/A	**50**	N/A	N/A	106
#2	N/A	102	149	115	42	199	80	**57**	37	37	104
#3	N/A	196	N/A	267	86	100	133	**128**	152	N/A	N/A

Table C.2: Difference in the generation count from the case of 0.7 crossover probability

	crossover probability										
sample	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
#0	N/A	29	123	93	19	82	N/A	0	N/A	N/A	56
#1	N/A	58	N/A	100	84	61	N/A	0	N/A	N/A	N/A
#2	N/A	45	92	58	-15	142	23	0	-20	-20	47
#3	N/A	68	N/A	139	-42	-28	5	0	24	N/A	N/A
Avg.		50		97.5	11.5	64.25		0			

D Measurement results of power optimization with GenMap

CC-SOTB



(a) *af* application

(b) *gray* application

(c) *sepia* application

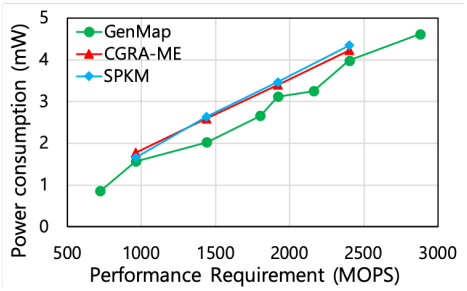
(d) *sf* application[1]

(e) *dct* application

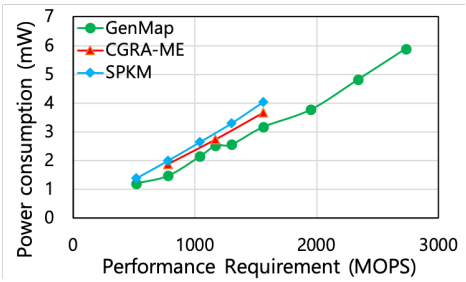
Figure D.1: Comparison of optimization result regarding power consumption for each application (CC-SOTB)

[1]SPKM fails to map *sf* application for CC-SOTB

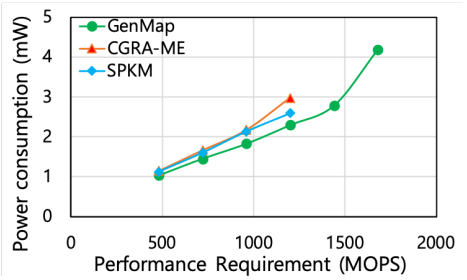
CC-SOTB2



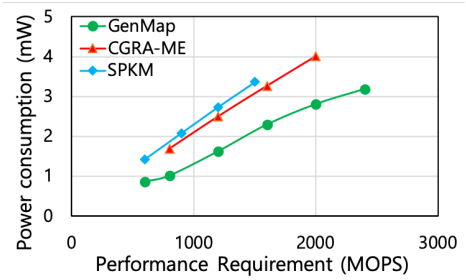
(a) *af* application



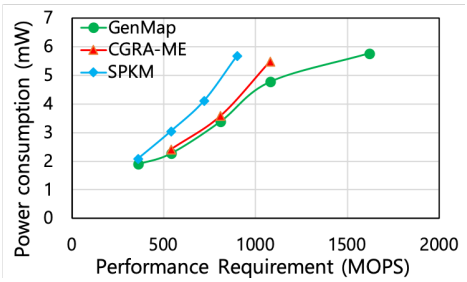
(b) *gray* application



(c) *sepia* application



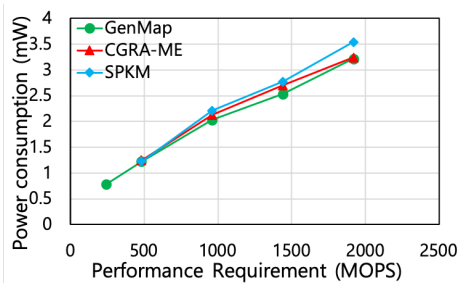
(d) *sf* application



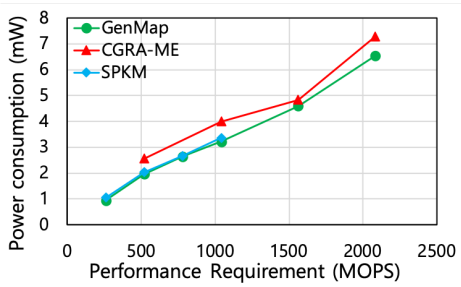
(e) *dct* application

Figure D.2: Comparison of optimization result regarding power consumption for each application (CC-SOTB2)

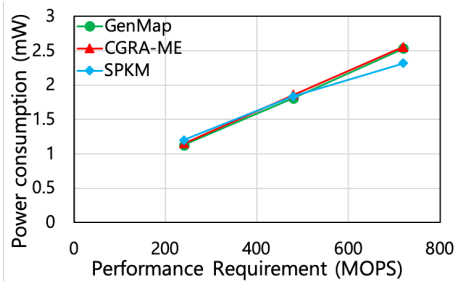
NVCMA



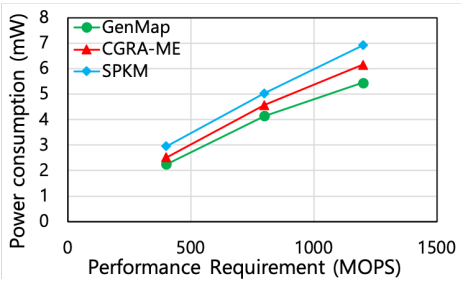
(a) *af* application



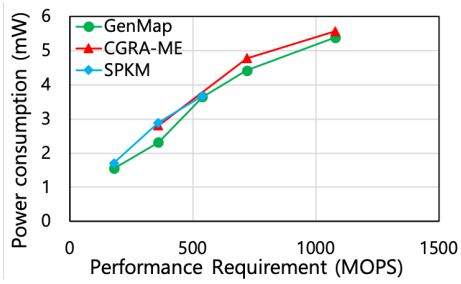
(b) *gray* application



(c) *sepia* application



(d) *sf* application



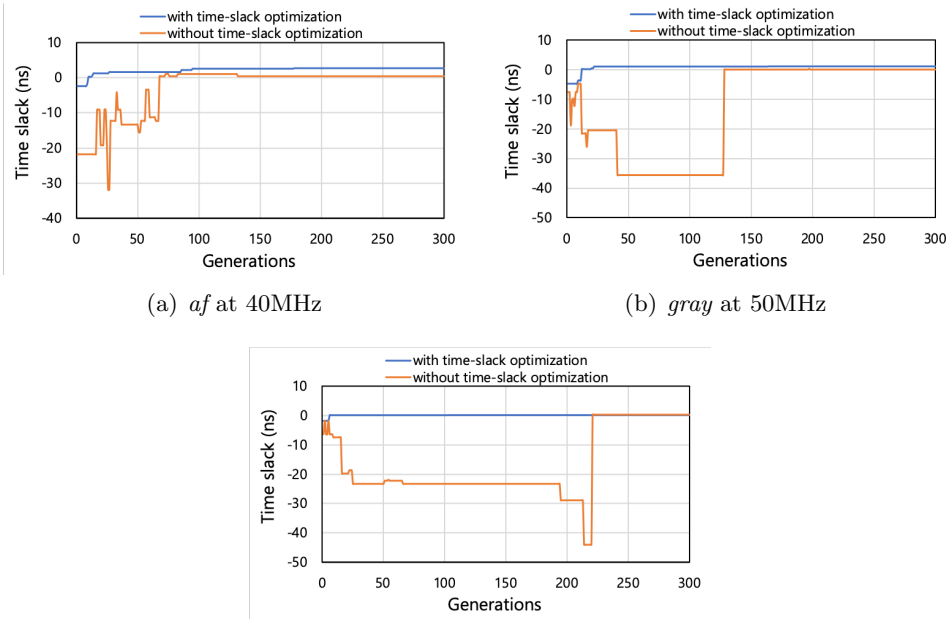
(e) *dct* application

Figure D.3: Comparison of optimization result regarding power consumption for each application (NVCMA)

E Effect of time slack objective

The time slack objective contributes to an efficient search for a valid mapping rather than a guarantee that no timing violation occurs. Here, we assume two mappings M_A and M_B where $T_{slack}(M_A) = -10$ nsec and $T_{slack}(M_B) = -5$ nsec, that is, both violate the timing constraint. Without the objective function, which returns the slack time explicitly, both mappings can be considered to be the same mapping quality. However, it is obvious M_B is better than M_A in terms of slack time. If such a better solution is missed at the selection phase, the genetic algorithm cannot optimize the solutions efficiently. Thus, we should treat the slack time explicitly.

To demonstrate the efficiency by introducing the time slack objective, we carried out additional experiments. Three applications (*af, gray, sepia*) are optimized by GenMap with either considering the time slack or not. For each application, a severe timing constraint is given. Fig. E.1 shows the transition of the best time slack during the optimization. The target architecture is CC-SOTB2. If GenMap does not consider the objective, it makes the mapping worse in terms of the time slack. Then, it takes a long time to obtain a mapping which meets the timing requirement. In the case of *sepia*, a valid mapping is found at the 8th generation with the dedicated objective for the time slack, while it is obtained at the 221st generation without the objective.



(a) *af* at 40MHz

(b) *gray* at 50MHz

(c) *sepia* at 65MHz

Figure E.1: Effect of time slack consideration

Publications

Related Papers

Journal Papers

[1] Takuya Kojima, Nguyen Anh Vu Doan, Hideharu Amano, GenMap: A Genetic Algorithmic Approach for Optimizing Spatial Mapping of Coarse Grained Reconfigurable Architectures, *IEEE Transactions on Very Large Scale Integration Systems*, Vol. 28, no. 11, pp.2383–2396, November 2020.

[2] Takuya Kojima, Naoki Ando, Hayate Okuhara, Ng. Anh Vu Doan, Hideharu Amano, Optimization of Body Biasing for Variable Pipelined Coarse-Grained Reconfigurable Architectures, *IEICE Transactions on Information and Systems*, Vol.E101-D, No.6, pp.1532–1540, Jun 2018.

International Conference Papers

[3] Takuya Kojima, Naoki Ando, Yusuke Matsushita, Hayate Okuhara, Nguyen Anh Vu Doan, Hideharu Amano, Real Chip Evaluation of a Low Power CGRA with Optimized Application Mapping, In *Proc. of the 9th International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies (HEART2018)*, pp,1–6, Toronto, Canada, June, 2018.

[4] Takuya Kojima, Naoki Ando, Hayate Okuhara, Hideharu Amano, Glitch-aware Variable Pipeline Optimization for CGRAs, In *Proc. of the 2017 International Conference on ReConFigurable Computing and FPGAs(ReConFig 2017)*, pp.1–6, Cancun, Mexico, December, 2017.

[5] Takuya Kojima, Naoki Ando, Hayate Okuhara, Ng. Anh Vu Doan, Hideharu Amano, Body Bias Optimization for Variable Pipelined CGRA, In *Proc. of the 27th International Conference on Field-Programmable Logic and Applications (FPL)*, pp.1–4, Ghent, Belgium, September, 2017.

Other Papers

Journal Papers

[6] Takeharu Ikezoe, Takuya Kojima, Hideharu Amano, Recovering faulty Non-volatile Flip Flops for Coarse-Grained Reconfigurable Architectures, *IEICE Transactions on Electronics*, June 2021. (Accepted)

[7] Takuya Kojima, Takeharu Ikezoe, Hideharu Amano, CubeSim: A Cycle Accurate Simulator for Multicore System with 3D SiP, *IEICE Transactions on Information and Systems*, April 2021. (In Japanese)(Accepted)

[8] Takuya Kojima, and Hideharu Amano, A Fine-Grained Multicasting of Configuration data for Coarse-Grained Reconfigurable Architectures, *IEICE Transactions on Information and Systems*, Vol.E102-D, No.7, pp.1247–1256, July 2019.

International Conference Papers

[9] Ayaka Ohwada, Takuya Kojima, Hideharu Amano, MENTAI: A Fully Automated CGRA Application Development Environment that Supports Hardware/Software Co-design, 23rd Workshop on Synthesis And System Integration of Mixed Information technologies (SASIMI2021), March 2021. (Accepted)

[10] Ayaka Ohwada, Takuya Kojima, Hideharu Amano, Compiler Framework for Spatial Mapping CGRA using LLVM, In *Proc. of the 2020 IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS)*, Poster 8, Kokubunji, Japan, April 2020.

[11] Takeharu Ikezoe, Takuya Kojima, Hideharu Amano, A Coarse-Grained Reconfigurable Architecture with a Fault Tolerant Non-volatile Configurable Memory, In *Proc. of the 2019 International Conference on Field-Programmable Technology (FPT)*, pp.81–89, Tianjin, China, December 2019.

[12] Hideto Kayashima ,Takuya Kojima, Hayate Okuhara, Tsunaaki Shidei, Hideharu Amano, Real Chip Performance Evaluation on Through Chip Interface IP for Renesas SOTB 65nm Process, In *Proc. of 7th International Symposium on Computing and Networking Workshops (CANDARW' 19)*, pp.269–274, Nagasaki, Japan, November 2019.

[13] Ryohei Tomura, Takuya Kojima, Hideharu Amano, A Real chip evaluation of a CNN accelerator SNACC, In *Proc. of the 22st Workshop on Synthesis And System Integration of Mixed Information technologies (SASIMI2019)*, pp.62–67, Tainan, Taiwan, October 2019.

[14] Sayaka Terashima, Takuya Kojima, Hayate Okuhara, Kazusa Musha, Hideharu Amano, Ryuichi Sakamoto, Masaaki Kondo, Mitaro Namiki, A Preliminary Evaluation of Buiding Block Computing Systems, In *Proc. of the 2019 IEEE 13th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC-2019)*, pp.312–319, Singapore, October 2019.

[15] Takuya Kojima, Naoki Ando, Yusuke Matsushita, Hideharu Amano, Demonstration of Low Power Stream Processing Using a Variable Pipelined CGRA, In *Proc. of the 29th International Conference on Field Programmable Logic and Applications (FPL)*, pp.411–412, Barcelona, Spain, September 2019.

[16] Takuya Kojima, Hideharu Amano, Refinements in Data Manipulation Method for Coarse Grained Reconfigration Architectures, In *Proc. of the 14th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC 2019)*, pp.113–120, York, United Kingdom, July 2019.

[17] Hideto Kayashima, Takuya Kojima, Hayate Okuhara, Tsunaaki Shidei, Hideharu Amano, Real Chip Performance Evaluation of Inductive Coupling TCI IP, In *Proc. of the 2019 IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS)*, Poster 16, Yokohama, Japan, April 2019.

[18] Takuya Kojima, Hideharu Amano, A Configuration Data Multicasting Method for Coarse-Grained Reconfigurable Architectures, In *Proc. of the 28th International Conference on Field Programmable Logic and Applications (FPL)*, pp.239–242, Dublin, Ireland, August, 2018.

[19] Takeharu Ikezoe, Takuya Kojima, Hideharu Amano, Junya Akaike, Kimiyoshi Usami, Keizo Hiraga, Yusuke Shuto, Kojiro Yagami, A micro-controller for MTJ-based Non-volatile Flip-flops for data verification, *Proc. of the 2018 IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS)*, Poster 10, Yokohama, Japan, April 2018.

[20] Sayaka Terashima, Takuya Kojima, Hayate Okuhara, Yusuke Matsushita, Naoki Ando, Mitaro Namiki, Hideharu Amano, A shared memory chip for twin-tower of chips, In *Proc. of the 21st Workshop on Synthesis And System Integration of Mixed Information technologies (SASIMI2018)*, pp.353–358, Shimane, Japan, March 2018.

[21] Ryuichi Sakamoto, Ryo Takata, Jun Ishii, Masaaki Kondo, Hiroshi Nakamura, Tetsui Ohkubo, Takuya Kojima, Hideharu Amano, Scalable Deep Neural Network Accelerator Cores with Cubic Integration using Through

Chip Interface, In *Proc. of the 2017 International SoC Design Conference (ISOCC 2017)*, pp.155–156, Seoul, Korea, November, 2017.

[22] Ryuichi Sakamoto, Ryo Takata, Jun Ishii, Masaaki Kondo, Hiroshi Nakamura, Tetsui Ohkubo, Takuya Kojima, Hideharu Amano, The Design and Implementation of Scalable Deep Neural Network Accelerator Cores, In *Proc. of the 2017 IEEE 11th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*, pp.13–20, Seoul, Korea, September, 2017.

[23] Takuya Kojima, Naoki Ando, Hayate Okuhara, Ng. Anh Vu Doan, Hideharu Amano, Power Optimization for CGRA with Control of Variable Pipeline and Body Bias Voltage, In *Proc. of the 2017 IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS)*, Poster 6, Yokahama, Japan, April 2017.

Domestic Conference Papers and Technical Reports

[24] 大和田彩夏, 小島拓也, 天野英晴, CGRA アプリケーションの IP ベース設計環境の提案, 信学技報, vol. 120, no. 121, CPSY2020-6, pp.37–42, 2020 年 7 月.

[25] 小島拓也, 大和田彩夏, 天野英晴, 深層学習を用いた CGRA の効率的なアプリケーションマッピング手法, 信学技報, vol. 120, no. 121, CPSY2020-7, pp.43–48, 2020 年 7 月.

[26] 小島拓也, 池添赳治, 天野英晴, 3 次元積層型ヘテロジニアスプロセッサのためのシミュレータ開発とその応用, 信学技報, vol. 119, no. 428, CPSY2019-103, pp.93–98, 2020 年 2 月.

[27] 大和田彩夏, 小島拓也, 天野英晴, LLVM を用いた CGRA 向けソフトウェア開発環境の構築と評価, 信学技報, vol. 119, no. 428, CPSY2019-109, pp.145–150, 2020 年 2 月.

[28] 小島拓也, 天野英晴, CGRA のためのアプリケーションマッピングフレームワーク GenMap の実装と実機評価, 信学技報, vol. 119, no. 282, VLD2019-29, pp.1–6, 2019 年 11 月. (**IEEE CEDA AJJC Design Gaia Best Poster Award**)

[29] 茅島秀人, 小島拓也, 奥原颯, 四手井綱章, 天野英晴, チップ間誘導結合無線通信技術の実機評価, 信学技報, vol. 119, no. 286, CPSY2019-48, pp.59–64, 2019 年 11 月.

[30] 戸村遼平, 小島拓也, 天野英晴, 坂本龍一, 近藤正章, CNN アクセラレータ SNACC の実チップ評価, 信学技報, vol. 119, no. 286, CPSY2019-49, pp.65–70, 2019 年 11 月.

[31] 池添赳治, 小島拓也, 天野英晴, 不揮発性構成メモリを用いた耐故障性粗粒度再構成可能アーキテクチャ, 信学技報, vol. 119, no. 208, RECONF2019-28, pp.39–44, 2019 年 9 月.

[32] 天野英晴, 茅島秀人, 小島拓也, 坂本龍一, 近藤正章, 並木美太郎, ビルディングブロック型積層システムの性能評価, 信学技報, vol. 119, no. 147, CPSY2019-17, pp.1–6, 2019 年 7 月.

[33] 小島拓也, 天野英晴, 粗粒度再構成可能アーキテクチャCMA におけるメモリバンクアクセスの改良, 信学技報, vol. 119, no. 147, CPSY2019-22, pp.85–90, 2019 年 7 月.

[34] 茅島秀人, 小島拓也, 奥原颯, 四手井綱章, 天野英晴, 誘導結合 TCI IP の実チップにおける性能測定, LSI とシステムのワークショップ 2019, ポスターセッション, no.26, 2019 年 5 月.

[35] 天野英晴, 茅島秀人, 四手井綱章, 小島拓也, ルネサス SOTB65nm 用 Through Chip Interface IP の実機評価, 信学技報, vol. 119, no. 25, VLD2019-5, pp.31–36, 2019 年 5 月.

[36] 小島拓也, 天野英晴, 3 次元積層型 CGRA のためのアプリケーション割り当て手法の検討, 信学技報, vol. 118, no. 375, CPSY2018-51, pp.37–42, 2018 年 12 月.

[37] 寺嶋爽花, 小島拓也, 武者千嵯, 奥原颯, 天野英晴, ツインタワー型共有メモリチップを用いた CNN アプリケーションの高速化, 信学技報, vol. 118, no. 375, CPSY2018-76, pp.125–130, 2018 年 12 月.

[38] 茅島秀人, 小島拓也, 奥原颯, 天野英晴, 誘導結合 ThruChip Interface の検証方式の実チップ実装, 信学技報, vol. 118, no. 339, CPSY2018-42, pp.53–58, 2018 年 12 月.

[39] 小島拓也, 安藤尚樹, 松下悠亮, 奥原颯, Nguyen Anh Vu Doan, 天野英晴, 可変パイプラインを持つ低消費電力アクセラレータ CCSOTB2 によるストリーム処理, 信学技報, vol. 118, no. 325, CPSY2018-33, pp.1–5, 2018 年 11 月. (**IEICE CPSY Young Presentation Award**)

[40] 小島拓也, 安藤尚樹, 松下悠亮, 奥原颯, Nguyen Anh Vu Doan, 天野英晴, 多目的遺伝的アルゴリズムを用いた CGRA マッピング最適化手法と実チップ評価, 信学技報, vol. 118, no. 215, RECONF2018-31, pp.67–72, 2018 年 9 月.

[41] 松下悠亮, 小島拓也, 門本淳一郎, 黒田忠広, 天野英晴, マルチコア積層システム Cube-2 の実装と評価, 情報処理学会第 80 回全国大会講演論文集, pp.1:81-–1:82, 2018 年 3 月.

[42] 寺嶋爽花, 小島拓也, 奥原颯, 松下悠亮, 安藤尚輝, 並木美太郎, 天野英晴, ツインタワーのためのメモリチップ, 情報処理学会第 80 回全国大会講演論文集, pp.1:87-–1:88, 2018 年 3 月.

[43] 小島拓也, 安藤尚樹, 天野英晴, 可変構造パイプラインを持つ粗粒度再構成アクセラレータ CCSOTB2, 情報処理学会第 80 回全国大会講演論文集, pp.1:89-–1:90, 2018 年 3 月.

[44] 小島拓也, 安藤尚輝, 奥原颯, 天野英晴, グリッチ削減のためのパイプライン構造最適化, 信学技報, vol. 117, no. 279, RECONF2017-41, pp.25–30, 2017 年 11 月.

[45] 安藤尚輝, 小島拓也, 天野英晴, 可変パイプライン CGRA の実チップ評価, 信学技報, vol. 117, no. 279, RECONF2017-41, pp.19–24, 2017 年 11 月.

[46] 寺嶋爽花, 小島拓也, 奥原颯, 松下悠亮, 安藤尚輝, 並木美太郎, 天野英晴, ツインタワー用共有メモリチップの開発, 信学技報, vol. 117, no. 273, VLD2017-34, pp.43–48, 2017 年 11 月.

[47] 小島拓也, 安藤尚輝, 奥原颯, Ng.Doan Anh Vu, 天野英晴, 整数計画問題を用いたパイプライン型 CGRA のボディバイアス電圧最適化, 信学技報, vol. 117, no. 46, RECONF2017-16, pp.81–86, 2017 年 5 月.

[48] 小島拓也, 安藤尚輝, 松下悠亮, 奥原颯, 天野英晴, パイプライン段数とボディバイアス電圧制御によるパイプライン型 CGRA の電力削減手法の検討, 信学技報, vol. 116, no. 510, CPSY2016-140, pp.51–56, 2017 年 3 月.

[49] 大久保徹以, 小島拓也, 天野英晴, 高田遼, 石井潤, 坂本龍一, 近藤正章, 中村宏, 無線 3 次元積層チップを用いた Deep Learning アクセラレータのコンパイラツールチェーン, 信学技報, vol. 116, no. 510, CPSY2016-155, pp.357–362, 2017 年 3 月.