Stress-Aware Loops Mapping on CGRAs with Dynamic Multi-Map Reconfiguration

Jiangyuan Gu, Shouyi Yin, Leibo liu, and Shaojun Wei

Abstract—With VLSI process technology scaling into nano-scale, the increasingly serious aging issues (e.g., NBTI and HCI aging effects) have brought a significant threat to system reliability. Coarse-grained reconfigurable architectures (CGRAs) exhibit the feature to reconfigure and execute different mapping schemes (Maps) dynamically, compensating for each other to mitigate aging issues effectively. In this paper, a two-stage stress-aware loops mapping algorithm is first proposed for the CGRA-mapped designs by jointing the intra-kernel and inter-kernel stress optimizations. With pipelining techniques, the intra-kernel stress optimization employs the stress-aware force-directed and effective MCC (Maximal Compatibility Class) methods to optimize operations' placement and mapping distribution on processing elements (PEs), which helps to avoid overmany operations to be mapped on the same PEs and reduce the accumulated stresses. By leveraging the dynamic reconfiguration feature, the inter-kernel stress optimization develops a multi-map scheduling method to reconfigure a set of ordered maps on CGRA dynamically, which diversifies the PEs' usage and compensates for the stresses on different PEs among them. Experimental results show that our approach can reduce the maximum stress by 82.0% for NBTI and 70.4% for HCI, and improve the aging efficiency by 6.01X and MTTF by 3.16X averagely, while keeping the optimized performance.

Index Terms—CGRAs, aging mitigation, loop mapping, NBTI and HCI, stress optimization, intra-kernel, inter-kernel

1 Introduction

Nowadays, the mobile devices and intelligent equipments have been required to meet tighter constraints on cost, performance and power. As reported in ITRS2015 [1], coarse-grained reconfigurable architectures (CGRAs) have been regarded as one of the most emerging computing architectures, receiving a huge attention due to their high performance and energy-efficiency [2], [3], [4], [5]. As shown in Fig. 1, a typical CGRA mainly consists of a homogenous and 2-D meshed array of processing elements (PEs), host controller, context and data memories. Each PE has an Arithmetic Logic Unit (ALU), register file and internal configuration unit, which can be configured with different logic and arithmetic operations via the configuration controller [2], [3], [5], [6]. The new operation configuration information can be pre-fetched into configuration buffers to prepare for PEs during previous operations execution, which can hide and eliminate the explicit configuration time during execution.

With process technology scaling down into nano-scale continuously, Very Large Scale Integration (VLSI) designs, as well as CGRAs, have been being facing significant reliability issues, due to the increasingly serious aging effects, such as Negative Biased Temperature Instability (NBTI), Hot Carrier Injection (HCI), Electromigration (EM) and Time-dependent Dielectric Breakdown (TDDB) [7], [8]. They usually increase threshold voltage and delay, and then decrease system Mean Time To Failure (MTTF) and reliability. Among them, NBTI and HCI are two most significant ones dominating the aging influence on devices and circuits [8], [9], [10]. In [11], it indicates the NBTI-induced increase in the switching delay of a PMOS transistor is near up to 20% in 10 years. In [10], the average degradation of the critical path delay increases by 10.1% for NBTI and by 3.3% for HCI.

As depicted in Fig. 2, the NBTI aging effect exhibits a unique characteristic that *Stress* and *Recovery* phases alternate during dynamic execution, but the HCI aging effect incurs only one *Stress* phase. The detailed aging mechanisms of them are given out in Section 2. Since the degradations accumulated for both NBTI and HCI significantly depend on the execution time of devices and circuits, the longer running causes the larger stresses (or degradation) accumulated on them. For the aging issues on CGRAs, more operations are executed on PEs, which also means longer stress times and causes larger stresses on them. Our target is to reduce the maximum stresses accumulated on PEs and balance the stress distribution on CGRA.

Since many applications spend most of the execution time on loops [12], [13], it is also necessary for CGRAs to pay more attention to them [14], [15], [16]. With pipelining techniques [13], loops are transformed into *loop kernels* by overlapping several loop bodies to execute on PEs repeatedly [15], [17]. Different loop kernels have different operation distributions on PEs, causing different stress distributions on them. Moreover, since executing loops repeatedly causes larger stresses accumulated on some PEs more easily, loops usually pose a significant influence on both performance and aging issues

• *The authors are with the Institute of Microelectronics, Tsinghua University, Beijing 100084, China. E-mail: yinsy@tsinghua.edu.cn.*

Manuscript received 18 June 2017; revised 6 Feb. 2018; accepted 7 Mar. 2018. Date of publication 19 Mar. 2018; date of current version 8 Aug. 2018.
(Corresponding author: Shouyi Yin.)
Recommended for acceptance by M. Huebner.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2018.2816955

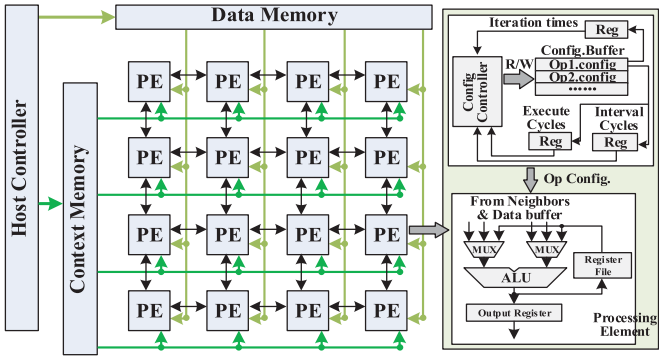


Fig. 1. A typically homogenous and 2-D meshed 4×4 CGRA.

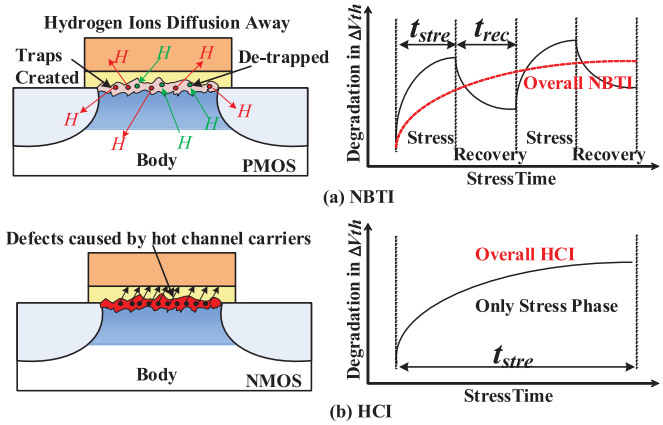


Fig. 2. Mechanisms of NBTI and HCI aging effects.

of CGRA-mapped designs. However, the existing works of loops mapping on CGRAs rarely consider aging issues, but are mostly focused on performance optimization [15], [17], [18], [19].

Therefore, we propose a stress-aware loops mapping approach to choose the optimized loop maps with stress optimization efforts in the CGRA-mapped designs, which distinguishes itself in such following contributions:

i) With the two-stage smart hill-climbing search idea [20], [21], a heuristic intra-kernel and inter-kernel jointed stress optimization framework is developed and extracted successfully for the CGRA-mapped designs at system level. The intra-kernel stress optimization serves as the global search stage to identify an optimized starting loop map. Then the inter-kernel stress optimization plays as the local search stage to start from the selected loop map before and search for its neighborhood for multiple better loop maps to reconfigure on CGRA dynamically.

ii) For the intra-kernel stress optimization, a stress-aware force-directed scheduling method (sFDS) [16], [22] is introduced to schedule operations at different timeslots with the pipelining technique [13], [15], [16], which helps to avoid operations, especially of the same types, to map on the same PEs. Also, a modified Maximal Compatibility Classes (MCC) method [15], [23], [24] is designed to map operations and disperse the stresses on more PEs, which finally reduces the maximum stresses accumulated within one loop kernel.

iii) For the inter-kernel stress optimization, a multi-map scheduling method is proposed by leveraging the dynamic reconfiguration feature of CGRA, which tries to select a set of ordered loop maps from four different geometric transformations to reconfigure and execute them on CGRA dynamically. This effectively diversifies the PE usages and compensates for the stresses accumulated on different PEs among those loop kernels, which can further decrease the maximum stresses accumulated on PEs and balance the stress distribution on CGRA.

The rest of the paper is organized as follows. In Section 2, we give the background and related works of this paper. Then, a motivation example of our work is given in Section 3. Section 4 formulates the problem of our jointly stress-aware loops mapping on CGRA. Section 5 gives the efficient solution of this loop mapping and optimization solution in details. In Section 6, the experimental results demonstrate the effectiveness of our proposed two-stage stress-aware loop mapping approach. Section 7 concludes this paper.

2 Background and Related Works

2.1 Basic Aging Models

1) *NBTI Aging Effect.* The general physical mechanism of NBTI, as shown in Fig. 2a, is explained quantitatively with the reaction-diffusion model [25], [26]. In the stress phase (e.g., turn on), hydrogen ions that pacifies the dangling bonds in the interface region diffuse away, leaving behind traps (or defects). In the recovery phase (e.g., turn off), some traps are de-trapped. Just those traps increase the threshold voltage and prolong the delay of devices and circuits.

In the stress phase, n_1 can be 1/4 or 1/6 depending on the fabrication process. k is *Boltzmann* constant. E_a is a model parameter. A_{NBTI} is the fitting model parameter. t_{stre} is the time span under stress phase. d_0 is the *pre-aging* delay. Then, the aging stress for NBTI under the stress phase (Δd_{stre}) can be described as follows [25], [26], [27]

$$\Delta d_{stre} = A_{NBTI} \cdot e^{\left(\frac{-E_a}{kT}\right)} \cdot t_{stre}^{n_1} \cdot d_0. \tag{1}$$

In the recovery phase, t_{rec} is the time span under recovery phase. η is a model parameter ($\eta = 0.35$). R_N is called *recovery coefficient*, dependent on t_{rec} and t_{stre}. Then, the total aging stress for NBTI after the recovery phase, denoted as Δd_{NBTI}, can be described below [25], [26], [27]

$$\begin{aligned} \Delta d_{NBTI} &= \Delta d_{stre} \cdot (1 - R_N) \\ R_N &= \sqrt{\eta \cdot t_{rec}/(t_{stre} + t_{rec})}. \end{aligned} \tag{2}$$

2) *HCI Aging Effect.* As shown in Fig. 2b, the physical mechanism of HCI is explained by hot channel carriers created by the accelerated electrons in the electric field of conducting channel [8], [26], [27]. When reaching a certain higher velocity and colliding with the gate oxide interface, some electrons generated from electron-holes might get trapped in the Si/SiO2 interface near the drain, causing threshold voltage increased and delay prolonged.

Here, n_2 can be 0.5, A_{HCI} is the fitting model parameter. E_b is a technology dependent factor. α is the activity factor and f is the frequency. $\alpha \cdot f$ is known as the activity rate,

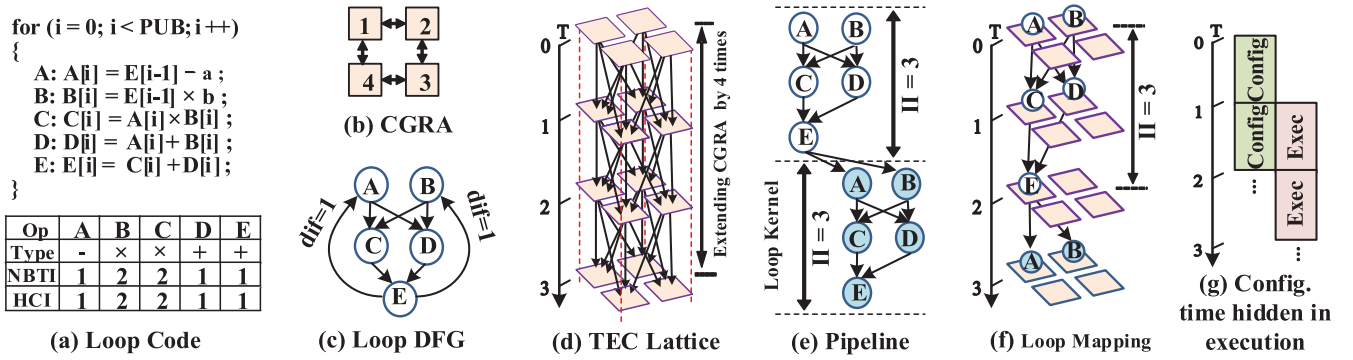


Fig. 3. Example of Loops mapping on CGRA, which is similar to a graph-to-graph matching problem and equivalent to find a maximal compatible mapping part between the DFG of loop kernel and TEC lattice.

denoted as AR. Then, the aging stress for HCI, denoted as Δd_{HCI}, can also be described as follows [26], [27]

$$\Delta d_{HCI} = A_{HCI} \cdot \alpha \cdot f \cdot e^{(\frac{-E_b}{kT})} \cdot t_{stre}^{n_2} \cdot d_0. \quad (3)$$

Obviously, aside from the impact of model parameters,[1] the significant factors of the stress time (t_{stre}) and temperature (T) of both NBTI and HCI aging effects are related to the operations execution on PEs, which indicates that we can optimize the operations scheduling and mapping on them to mitigate the aging issues on CGRA at system level.

2.2 Basic Loops Mapping on CGRAs

As depicted in Fig. 3, mapping loops to CGRAs is equivalent to mapping loop kernels to the time-extended CGRA (TEC) lattice with feasible *initiation interval* (II) [15], [16]. II is the number of cycles between the start of two consecutive iterations of loop kernels. MII is the minimal II of loop kernels and $MII = Max(ResMII, RecMII)$ [13], [28], where $ResMII$ is the resource-constrained MII and $RecMII$ is the recurrence-constrained MII. The smaller II means the next loop kernel can start earlier and brings better performance. As shown in Fig. 3e, the *loop kernel* with $II = 3$ is a repeated unit of operations and formed by overlapping several loop iterations, represented by a data flow graph (DFG) $D = (V_d, E_d)$, where V_d and E_d are respectively its set of nodes and edges. The TEC is formed by extending CGRA along the time-axis by II times [15], denoted by $T_{II} = (V_c, E_c)$, where PE_j^t is the jth PE at timeslot t. As shown in Fig. 3d, $PE1$ at $T=1$ is represented by PE_1^1. As shown in Fig. 3f, mapping loops on CGRA is similar to a graph-to-graph matching problem and equivalent to find a maximal compatible mapping part between DFG of loop kernel and TEC lattice, which is a known NP-complete problem [15], [17]. Here, a modified MCC method is employed to find the valid loop mapping schemes, called *loop maps* and denoted as Map, which is described in Section 5 in detail. Moreover, since loops are executed repeatedly, only some operation configurations are required for each PE. They are usually pre-fetched and prepared for PEs during previous operations or loops execution [3], [6], causing the explicit configuration times hidden into loops execution on PEs.

1. Aside from the simulation overhead at transistor/device level and its complexity, the detailed transistor/device-level manufacturing information and parameters of chips are usually not available for the CGRA/FGPA-mapped users or designers at system level [27].

2.3 Related Works

With technology process scaling down, NBTI and HCI aging effects dominate the nominal degradation and reduce the circuits lifetime [9], [31]. So they should be considered when mapping on CGRAs. Especially for loops, they usually dominate the total execution times and induce aging issues more easily due to their repeated execution style.

Recently, many studies about loops mapping on CGRAs have been proposed to achieve higher performance [15], [17], [18], [19], where pipelining techniques [13] are usually introduced to reduce IIs by overlapping successive loop kernels. In [18], an edge-centric modulo scheduling method (EMS) is proposed to achieve smaller II with considering routing problem. In [15], EPIMap formalizes loops mapping on CGRA as a graph epimorphism problem and attempts to find a maximal common subgraph (MCS) between DFG and TEC [32]. In [16], the force-directed loops mapping approach is proposed to solve the scheduling and mapping problem on CGRA, which achieves an optimal resource utilization with the total execution time reduced by 39% on average. However, they are mainly aimed to achieve higher performance but rarely to consider aging issues at all.

Unfortunately, few existing works are related to the aging problems on CGRAs. And the most-related works worthy of reference are mainly focused on Field Programming Gate Arrays (FPGAs). Based on system-level abstractions for both BTI and HCI, an easier and effective stress estimation model is proposed in [27] to predict the aging-induced degradation for guiding the mapping procedures of FPGA-based designs. In [33], three feasible strategies are based on exploiting spare logic and interconnect resources for degradation mitigation. In [7], a stress-aware module placement method distributes the stress evenly on the reconfigurable resources, which provides a 30% decrease in degeneration with considerable improvement in lifetime. By leveraging the reconfiguration feature to distribute stresses among multiple placement and mapping schemes, a wear-levelling technique in [34] reconfigures patterns of resources periodically and makes the stresses on all components more even, which brings a significant improvement in system reliability. In [29], a novel module diversification method is proposed to create different configurations for reconfigurable modules, which balances and mitigates the aging process in FPGA-based runtime reconfigurable systems. In [30], a cross-layer aging-aware placement method introduces the intra- and inter-region optimization strategy among different regions of FPGA-based

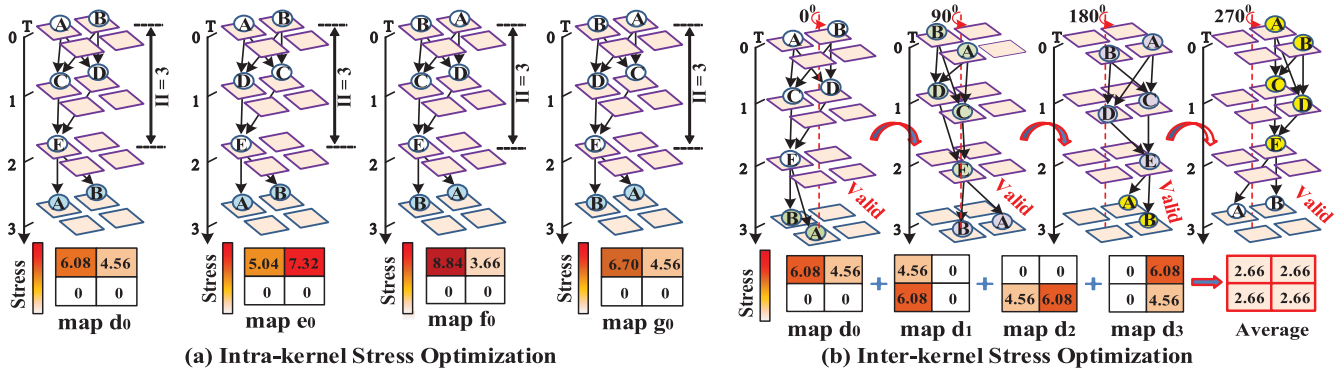


Fig. 4. (a) The intra-kernel stress optimization optimizes the loop operations mapping on TEC lattice within a loop kernel; (b) The inter-kernel stress optimization reconfigures multiple loop maps to execute on CGRAs dynamically, which diversifies the PE usages and compensates for the stresses on different PEs among them [29], [30].

reconfigurable architectures, which reduces the maximum stresses by 64% and 35% and then brings 177% and 14% MTTF improvement for HCI and BTI with a negligible performance cost.

3 Motivation

As discussed above, it is significant for both low aging and high performance mapping schemes to joint stress optimization into loop scheduling and mapping procedures. When more operations are executed on PEs, the stress times will be enlarged to cause larger stresses on them. So we can reduce the stresses by optimizing operations mapping on CGRA.

As shown in Fig. 3, a loop with 5 operations, {A, B, C, D, E}, is mapped to a 2×2 meshed CGRA. Two loop-carried dependencies are from E to A and from E to B. Operation A is subtraction, D and E are additions, and B and C are multiplications. According to the NBTI and HCI aging models in Section 2, if operations are all under the same conditions, the aging stresses induced by both of them are proportional to the pre-aging delays ($\triangle d_{NBTI} \propto d_0$ and $\triangle d_{HCI} \propto d_0$). Under 65-nm technology, the synthesis results of Design Compiler indicate that the pre-aging delay of 16-bit multiplier and adder are respectively 2.27 and 0.98 ns, which are nearly double relation between them. So we assume the normalized stresses (Δd_{stre}) for both d_{NBTI} and d_{HCI} are 1, 2, 2, 1 and 1 for operations A, B, C, D and E. The total stress accumulated on each PE is the sum of $\triangle d_{NBTI}$ and $\triangle d_{HCI}$, which is depicted by different colors and becomes larger from white to dark red.

1)*No Stress Optimization*. As shown in Fig. 4a, the loop map f_0 is obtained without stress optimizations. The execution order of operations on $PE1$ is (B, C, E, ...). According to Equation (2), operation A at T = 0 has 1 cycle for stress phase and 2 cycle for recovery phase, so its recovery coefficient R_N is simply $\sqrt{0.35 \cdot 2/(1+2)} = 0.48$. Similarly, R_Ns for operations B, C and E are 0, 0.34 and 0.48, respectively. The total stress accumulated on $PE1$ for one loop kernel is $(2 \times (1-0) + 2)$+$(2 \times (1-0.34) + 2)$+$(1 \times (1-0.48) + 1) = 8.84$. In map f_0, since it has only 3 operations of (B, C, E) and 2 operations (B, C) are of the same type on $PE1$, it causes the maximum stress of 8.84 on $PE1$, but 3.66, 0 and 0 on $PE2$, $PE3$ and $PE4$, which are very unbalanced.

2)*Intra-Kernel Stress Optimization*. As discussed before, we can reduce the stresses by avoiding operations, especially of the same types, to be mapped on the same PEs. Four different loop maps {d_0, e_0, f_0, g_0}, are depicted in Fig. 4a, which all have the optimal $II = 3$. Similar to map f_0, R_Ns of operations (A, C, E, ...) on $PE1$ in map d_0 are (0.48, 0.48, 0.48, ...). So the total stress of $PE1$ is $(1 \times (1-0.48)+1)$+$(2 \times (1-0.48)+2)$+$(1 \times (1-0.48)+1) = 6.08$. Then, the maximum stresses for those four loop maps are {8.84, 6.08, 6.32, 6.60}, respectively. Although there are 3 operations (A, C, E) executed on $PE1$ in map d_0, they are all of different types, each of which has longer recovery time. Thus, the loop map d_0 causes much smaller maximum stress than map f_0, which is reduced from 8.84 to 6.08.

3)*Inter-Kernel Stress Optimization*. Since these operations of repeated loops are always executed on those mapped PEs, it still brings much uneven stresses on PEs more easily. For example, if loop map d_0 repeats execution 4 times, the maximum accumulated stress is $4 \times 8.84 = 34.56$ for simplicity. To further reduce the stresses on CGRA, multiple different loop maps are reconfigured dynamically. They can diversify PE usages and compensate for the stresses accumulated on different PEs among them, which makes the stress distribution on PEs more balanced. In Fig. 4b, 4 rotationally symmetric loop maps, {d_0, d_1, d_2, d_3}, are reconfigured and executed dynamically, which can cause the average stress variance near to 0 on CGRA. Also, any two successive loop maps of them obey the necessary data dependencies to route the intermediate data. The results of operations E at T = 2 in map d_0 is directly consumed by operation A and B at T = 3 in map d_1. Consequently, the maximum stress for those four loop maps is $6.08 + 4.56 = 10.64$, and further reduced from 6.08 to 2.66 averagely.

In summary, by optimizing the operations of loop kernels mapping on TEC in the intra-kernel stress optimization and reconfiguring multiple loop maps on CGRA dynamically in the inter-kernel stress optimization, we can effectively reduce the maximum stresses accumulated on PEs and balance the stress distribution on CGRA.

4 Problem Formulation

4.1 Loops Mapping on CGRAs

Given a loop DFG $D = (V_d, E_d)$ and CGRA $C = (V_c, E_c)$, TEC $T_{II} = (V_t, E_t)$ represents CGRA extended by II times along time-axe. $PE_j^t \in V_t$, represents the jth PE at timeslot t in TEC. II^* is the optimal II achieved for the best loop mapping scheme, called loop Map^*. We employ *Compatibility* concept to formulate the loops mapping from DFG to TEC

[17], [32]. If an operation OP in DFG can be mapped to a PE PE_j^t in TEC, the mapping pair (OP, PE_j^t) is compatible, called *compatibility mapping pair*. For example in Fig. 3f, since there is an edge between operations A and C and a path from PE_1^0 to PE_2^1, (A, PE_1^0) and (C, PE_2^1) can co-exist in this loop map and are compatible with each other.

Definition 1 (Valid Mapping). *A loop mapping function $f : V_d \rightarrow V_t$, is a valid loop map Map on TEC. 1): iff $\forall u, v \in V_d$, $(u, v) \in E_d$, then $\exists f(u) = PE_{j1}^{t1}, f(v) = PE_{j2}^{t2} \in V_t$, and there must be a path from $f(u)$ to the node $f(v)$. 2): $\exists II = II^*$, guaranteeing the best performance.*

Since performance optimization is still the first priority in many applications and optimization problems [15], [17], [19], II is also optimized first in our solution.

4.2 Aging Stress Model on CGRAs

In this paper, the *Stress* is defined as the sum of the degradation of delays induced from all the functional operations executed on each PE. As discussed in Sections 2 and 3, it mainly depends on the numbers, types and scheduled timeslots of operations that are executed on them [26], [27]. Also, we define the *kernel stress distributions*, shortened as KSD, is the average stress distribution accumulated on PE array of CGRA for only one loop kernel.

Given a loop DFG D and CGRA C, their sizes of vertices are respectively n and m. TEC is denoted as $T_{II} = (V_t, E_t)$. Map is a valid map for the loop kernel, which includes some necessary mapping information, such as scheduled timeslots, number and types of operations mapped on PEs. $\overrightarrow{S_{NBTI}} = (sn_i)_{1\times n}$ is the stress (Δd_{stre}) vector of operations on one PE under stress phase due to NBTI, which can be calculated by the aging model in Section 2. $\overrightarrow{R_N} = (rn_i)_{1\times n}$, here rn_i is the recovery coefficient of operation i, which is mainly related to the scheduling and mapping status of operations. Similarly, $\overrightarrow{S_{HCI}} = (sh_i)_{1\times n}$ is the stresses vector of operations on one PE under stress phase due to HCI. The total stress (Δd) accumulated on one PE is the sum of those from NBTI and HCI caused by all operations executed on it. $\vec{A} = (a_i)_{1\times m}$ is the vector of the number of operations mapped on PEs for a loop kernel, and $\Sigma_{i=1}^{m} a_i = n$. The KSD of a map is denoted as $\overrightarrow{KSD} = (ks_i)_{1\times m}$. ks_i is the total stress Δd_i on the ith PE, which can be obtained as follows:

$$ks_i = \sum_{j=1}^{a_i} (sn_j \cdot rn_j + sh_j). \tag{4}$$

Assume $CSet = (Map_1, \ldots, Map_N)$ is a set of loop map candidates. II_i is the II of Map_i. $\overrightarrow{KSD}_1$, …, $\overrightarrow{KSD}_N$ are respectively their kernel stress distributions. S_1^{max}, …, S_N^{max} are the maximum stress of Map_1, …, Map_N, respectively. σ_1^{stress}, …, σ_N^{stress} are the stress distribution variances of them. The optimized loop maps are the ones with the smallest S^{max} and σ^{stress} on CGRAs, denoted as Map^*. Their average KSD on CGRA is denoted as $\overrightarrow{KSD}_{ave}$. S_{ave}^{max} and σ_{ave}^{stress} are respectively the average S^{max} and σ^{stress} of them. Then, we obtain such equation below

$$\overrightarrow{KSD}_{ave} = \frac{\sum_{i=1}^{N} \overrightarrow{KSD}_i}{N}. \tag{5}$$

For example in Fig. 4a, the total stress accumulated on $PE1$ in the loop map d_0 is caused by the operations A, C and E due to NBTI and HCI, which is about $(1 \times (1 - 0.48) + 1) + (2 \times (1 - 0.48) + 2) + (1 \times (1 - 0.48) + 1) = 6.08$. In Fig. 4b, after four loop maps reconfiguration on CGRA, KSD_{ave} is roughly $\{2.66, 2.66, 2.66, 2.66\}$ for $PE1$, $PE2$, $PE3$ and $PE4$, so the $S_{ave}^{max} = 2.66$ and $\sigma_{ave}^{stress} \approx 0$.

4.3 Whole Problem Definition

Given a loop DFG $D = (V_d, E_d)$ and CGRA $C = (V_c, E_c)$. TEC is denoted as $T_{II} = (V_t, E_t)$. II^* is the optimized II for the loop map Map^*. The *Problem Definition* of our proposed stress-aware loops mapping problem is to find a set of ordered maps $MapSet$ with the optimized II^* to reconfigure on CGRAs dynamically, and tries to reduce the S^{max} and σ^{stress} on CGRAs as possible. Such that:

1) For each valid map in $MapSet$, $\forall u, v \in V_d, (u, v) \in E_d$, $\exists f(u), f(v) \in V_t$, $(f(u), f(v)) \in E_t$.
2) For each valid map in $MapSet$, $\forall u, v \in V_d, (u, v) \in E_d$, $\exists f(u), f(v) \in V_t$, $(f(u), f(v)) \in E_t$.
3) Two successively reconfigured maps in $MapSet$ should satisfy the data dependencies between them.
4) S_{ave}^{max} and σ_{ave}^{stress} are minimized on the premise of the optimized II^*.

5 Solution

In this paper, a heuristic stress-aware loops mapping optimization framework is first proposed for the CGRA-mapped designs at system level, which integrates the two-stage smart hill-climbing searching idea [20], [21] into our intra- and inter-kernel stress optimization procedures. This hill-climbing strategy is an improvement of depth priority search, which can use the feedback information to help to generate better solutions [20], [21]. At the intra-kernel stress optimization, it can be regarded as the global search stage to identify an optimized starting loop map, which adopts an effective sFDS approach [16], [22] and MCC method [15], [23], [24] to reduce S^{max} and σ^{stress} within one loop map. At the inter-kernel stress optimization, it serves as the local search stage to start from the selected starting loop map before and resumes searching for its neighborhood to find multiple better loop maps to reconfigure on CGRA, which employs a multi-map scheduling method to further decrease S^{max} and σ^{stress} by diversifying the PE usages and compensating for the stresses among multiple different loop maps.

5.1 Stress-Aware FDS Scheduling

As we know, different numbers, types and scheduled timeslots of operations in TEC lattice can bring different t_{stre} and t_{rec} on PEs, which usually causes different S^{max} and σ^{stress}. Thus, two rules can guide loop kernels to map on CGRA: (i) Avoid operations, especially of the same arithmetic types, to be mapped on high-stressed PEs that have been mapped many operations. (ii) Try to map operations on more PEs to disperse the stresses accumulated on them.

Therefore, the intra-kernel stress optimization introduces a stress-aware force-directed scheduling method (sFDS) to optimize the scheduled timeslots of operations of loop kernels in the TEC lattice [16], [22]. To guarantee the best performance, pipelining technique [13] is adopted and begins

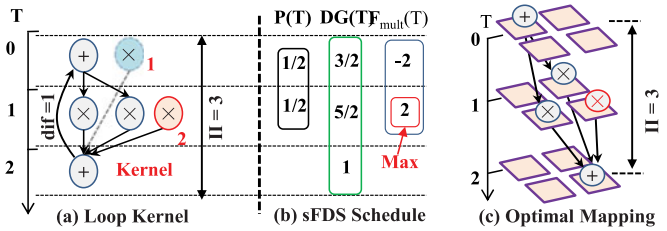


Fig. 5. sFDS tries to schedule more operations (especially of the same types) at the same timeslots in TEC, avoiding to map on the same PEs at different timeslots.

searching from MII (minimal II) iteratively [15], [16]

$$F_k(j) = \sum_{i=1}^{II} DG(i) \cdot x_k(i) \cdot s_k. \tag{6}$$

Given a DFG of one loop kernel and CGRA, $P(k,i)$ is the *probability* of operation k scheduled at timeslot i. $x_k(i)$ is the variation of $P(k,i)$ of operation k when scheduled at timeslot i. The probability distribution graphs (DGs) are generated based on the loop kernel after *pipelining*, $DG(i) = \sum_k P(k,i)$. $\overrightarrow{SK} = (s_k)_{1\times n}$ is the vector of the stresses (Δd_{stre}) of all operations under stress phase due to NBTI and HCI. So $s_k = sn_k + sh_k$ according to Equation (4). $F_k(j)$ is the force of operation k when placed at timeslot j, which can be calculated by Equation (6). It can be used to guide operations to schedule at one certain timeslot in TEC. When $F_k(j)$ is larger, scheduling operation k at timeslot j brings a larger stress reduction by avoiding many operations to be mapped on the same PEs. After a schedule of loop kernel is determined, it is used to search the loop maps in which more PEs are mapped. If failing to find out valid maps, we go back to the latest scheduled operation and reselect another schedule by sFDS to search again.

Different to original FDS [22], sFDS tries to schedule more operations at the same timeslots in TEC, helping to avoid mapping many operations on the same PEs at different timeslots. So those better schedules are first searched to find the optimized loop maps with the lowest S^{max} and σ^{stress}. For an example in Fig. 5, the multiplication of red color can be scheduled at $T=0$ or $T=1$. When it is scheduled at $T=1$, the force is $F_{mult}(1) = \frac{3}{2}\times(0-\frac{1}{2})\times(2+2)+\frac{5}{2}\times(1-\frac{1}{2})\times(2+2) = 2$, which is larger than $F_{mult}(0) = \frac{3}{2}\times(1-\frac{1}{2})\times(2+2)+\frac{5}{2}\times(0-\frac{1}{2})\times(2+2) = -2$ when scheduled at $T=0$. So this multiplication is scheduled at $T=1$, causing three multiplications scheduled at the same timeslot $T=1$. In Fig. 5c, an optimal loop map with the lowest S^{max} and σ^{stress} is just obtained from it, because three multiplications are all mapped to three different PEs and the stresses are dispersed on more PEs.

5.2 Stress-Aware Loop Maps Searching

As discussed before, finding valid loop maps is equivalent to search the maximal compatibility classes (MCCs) in the compatibility matrix (CM) [15], which is constructed by the DFG of loop kernel and TEC lattice. In this paper, a modified MCC searching method [23], [24] is introduced to search these valid loop maps with the lowest S^{max} and σ^{stress}, which optimizes the numbers and types of operations on PEs and tries to map them on more PEs. Since it finds out all the potentially feasible (OP, PE_j^t) mapping pairs at once and then remove the invalid ones that cannot co-exist, rather than trace back to add them into MCC one by one like [15], [32], this MCC searching method can help to guide to find the lower-aging loop maps more effectively and rapidly.

5.2.1 Compatibility Matrix Construction

The first step is to form compatibility mapping pairs (CPs), $cp = (OP, PE_j^t)$, which are achieved from the loop kernel D_{ker} and TEC $T_{II} = (V_t, E_t)$. Due the symmetrical feature of CGRA, some equivalent CPs that are mapped symmetrically for the first operation can be removed. As shown in Figs. 6a and 6b, a scheduled loop kernel is obtained by the sFDS method above, which has 4 operations and is mapped to a 1×2 CGRA. In Fig. 6c, 7 CPs are totally generated since CP (A, PE_2^0) is removed and only $cp_1 = (A, PE_1^0)$ is generated for operation A. Then, CM M is constructed by those CPs, which are the entries of this CM. Its values represent the corresponding compatible relationship between these two CPs cp_i and cp_j, denoted as $R(cp_i, cp_j)$, whose value is 0 or 1. If cp_i and cp_j are compatible, $R(cp_i, cp_j) = 1$ and this cell in CM is assigned value 1. Otherwise, the cell is assigned value 0. As an example of CM depicted in Fig. 6c, all the entries are related to those 7 CPs above.

There are mainly two rules for CM construction [15], [32]. (i) The same PE_j^t must not execute two different operations, and the same OP mustn't be mapped to two different PEs. For example, since operations A and D are mapped to PE_1^0, $R(cp_1, cp_6) = 0$. (ii) If there is an edge between the operations in DFG, there must be a communication path between the PEs in TEC. Otherwise, these two CPs are incompatible. For example, for two CPs cp_1 and cp_2, an edge exists between A and C and a path exists between PE_1^0 and PE_2^1, so they are compatible and $R(cp_1, cp_3) = 1$.

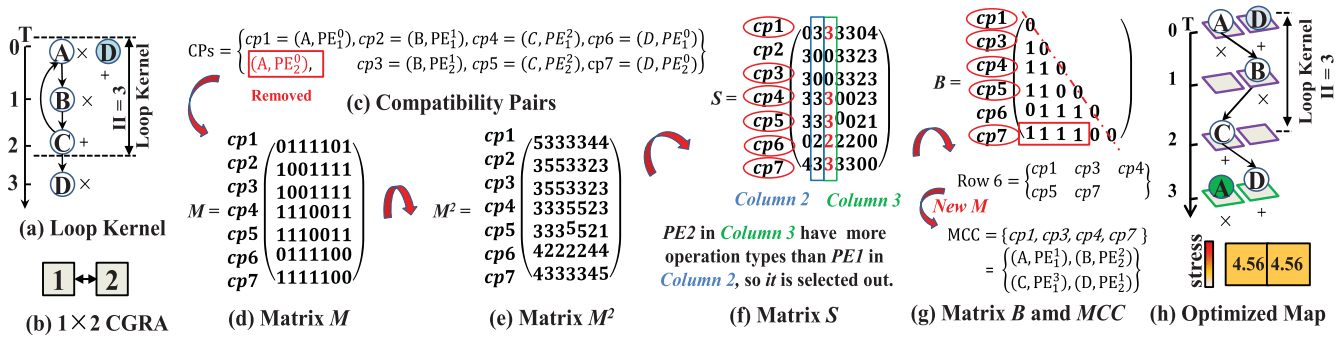


Fig. 6. A MCC searching example: It optimizes the numbers and types of operations on PEs and tries to map to more PEs during loop maps searching procedures. The loop maps with lowest S^{max} and σ^{stress} are found as the optimized ones.

5.2.2 MCC Searching in CM

Algorithm 1 presents how to find valid loop maps in CM [23], [24]. Assume that $M = (a_i^j)_{n\times n}$ represents CM. Here n is the total CPs and a_i^j is a $R(cp_i, cp_j)$ between two CPs.

First, we calculate the matrix of M^2, $M^2 = (c_i^j)_{n\times n} = M \times M$, and then construct a new matrix $S = (s_i^j)_{n\times n}$. If $a_i^j \neq 0$, $s_i^j = c_i^j$; else $s_i^j = 0$. Then, we scan S by column with the maximal value k^m. Assume that V^m is the size of the remaining unmapped DFG currently. According to the *Theorem* in Appendix 8, if there are at least $V^m - 1$ elements whose values are greater than $V^m - 2$, we first select the columns (c^k) in which more PEs are mapped on operations, especially ones of more types of operations (line 5). Otherwise, we search those with the second-largest value till $k^m < V^m - 2$. As shown in Fig. 6f, since $PE2$ has more types of operations in $Column3$ of S, it is selected to search first. Next, the corresponding CPs of non-zero values in this selected column are stored into CC, and the related rows and columns are selected from M to construct the simplified lower triangular matrix $B = (b_i^j)$. Then, we scan B by row. When $b_i^j = 0$, these CPs from CC are removed. If the size of the remaining CC is equal to V^m, it is a valid MCC and this row is marked. From those marked rows, we select the ones where more PEs are mapped and where PEs that are mapped on more operations have more types of operations. According to the stress model in Section 4, the KSDs of the obtained loop maps can be calculated. From the selected rows, those with the lowest S^{max} and σ^{stress} are further selected as the optimal Map^* and added into $CSet$ (line 10-15). Otherwise, if the number of none-zero cells is larger than $V^m - 1$, a new matrix M^* is obtained by updating the original matrix M. It is based on the rows of most CPs in B, which has less operations but more types on the mapped PEs. Then, we turn back, as depicted in Fig. 6g, to resume searching the newly obtained matrix M^* iteratively, until all cells related to a certain operation in S are zero (line 16-17). Finally, if no valid MCC is found, it means this scheduled D_{ker} fails to map. We go back to the latest scheduled operation successively and select another one by sFDS to resume search again.

For example in Fig. 6, the related matrixes M, M^2, S and B are presented, respectively. The maximal value $k^m = 4$ doesn't satisfy the theorem above, but $k^m = 3$ in $Column2$ and $Column3$ of matrix S does it. $PE2$ in $Column3$ has more operation types than that of $PE1$ in $Column2$, which usually causes lower accumulated stress. So we resume searching in $Column3$ rather than in $Column2$. Then, the resulting triangular matrix B is obtained, whose entries are tr_1, tr_3, tr_5, tr_6 and tr_7 and marked by red circle in S. There is a value "0" in $Row6$ of matrix B, so cp_6 is removed from CC. Since the number of none-zero cells in $Row6$ is larger than $V^m - 1$ ($V^m = 3$ currently), we generate a new matrix M^* to resume again. In Fig. 6h, the finally obtained $MCC = \{cp_1, cp_3, cp_4, cp_7 = \{(A, PE_1^0), (B, PE_2^1), (C, PE_1^2), (D, PE_2^0)\}$ is just one of the optimized loop maps, which has the smallest stress with $S^{max} = (1 \times (1 - 0.48) + 1) + (2 \times (1 - 0.48) + 2) = 4.56$ and $\sigma^{stress} \approx 0$.

5.3 Stress-Aware Dynamic Multi-Map Reconfiguration

To leverage the reconfiguration feature of CGRA, we develop a multi-map scheduling method in the inter-kernel stress optimization to find a set of ordered maps $MapSet$ to reconfigure dynamically, which can compensate for the stress distributions on PEs among those different loop maps and further reduce S^{max} and σ^{stress} on CGRA.

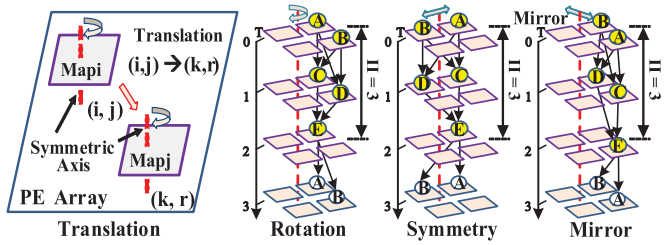


Fig. 7. Map candidates $Cset$ generated by four geometric transformations: *Translation*, *symmetry*, *rotation* and *mirror*.

Algorithm 1. Find MCC in CM with Considering Stress

Require: Loop kernel Schedule D_{ker}, TEC T_{II}, Stress vector SK;
Ensure: $CSet, Success$;
1: $CPairs \leftarrow$ Generate_Compatibility_Pairs(D_{ker}, T_{II});
2: $M \leftarrow$ Construct_CM($CPairs, D_{ker}, T_{II}$);
3: **while** Valid $MapSet$ for reconfiguration is not found **do**
4: $M^2, S \leftarrow$ New_Matrix(CM);
5: $k^m, c^k \leftarrow$ Scan_Column(S); //Scanning S by column
6: // Select the column with less OP numbers but more OP types
7: **if** $k^m < V^m - 2$ **then**
8: **break**;
9: **else**
10: $B, CC \leftarrow$ Simple_Matrix(S, k^m, c^k); //Scanning B by row
11: **if** $|V_{CC}| == V^m$ **then**
12: $S, MCC \leftarrow$ Select_MCC(B, CC, S);
13: $KSD, MCC \leftarrow$ Calculate_KSD(MCC, SK);
14: $Map^*, CSet \leftarrow$ Select_Map($CSet, MCC, KSD$);
15: **else**
16: $M^*, CC \leftarrow$ Update_CM(M, CC, S);
17: continue; // resume search in new CM
18: **end if**
19: **end if**
20: **end while**
21: **if** $CSet == \varnothing$ **then**
22: $Success \leftarrow False$;
23: **else**
24: $Success \leftarrow Ture$;
25: **end if**
26: **return** $Success, CSet, Map^*$;

5.3.1 Set of Map Candidates Generation

To guarantee the reconfigured loop maps to have the same lowest S^{max} and σ^{stress}, four geometric transformations, *translation*, *symmetry*, *rotation* and *mirror*, are designed to generate a set of map candidates $Cset$ based on the homogeneity and symmetry of CGRA. Since each loop map in $Cset$ has the same S^{max} and σ^{stress}, they can further bring both lower S_{ave}^{max} and σ_{ave}^{stress} after compensating for those stress distributions among them. As shown in Fig. 7, several loop maps Map_j can be generated from one map Map_i by translation transformation. With the geometry transformation theory, each translated map Map_j can generate 4 symmetric, 4 rotational and 4 mirror maps. In Fig. 7, symmetry, rotation and mirror maps are depicted, which are all generated from loop map d_0 in Fig. 4. Then, a set of loop map candidates

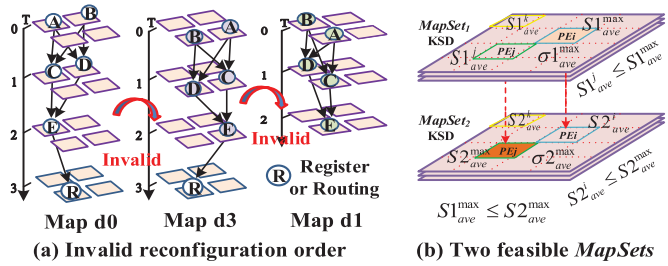


Fig. 8. (a) Maps d_0, d_3 and d_1 don't satisfy necessary data dependencies among them, so this order are *Invalid*; (b) If the average S^{max}_{ave} and σ^{stress}_{ave} of them are reduced, those two feasible *MapSet* candidates can be reconfigured successively.

$CSet_i$ are generated from those geometric transformations for the selected map Map_i.

5.3.2 Single Set of Reconfigurable Maps Construction

For a set of map candidates $CSet_i$ of a certain obtained loop map Map_i in the intra-kernel stress optimization procedure previously, there are two rules to find a single set of reconfigurable maps $MapSet_i$ from $CSet_i$ for Map_i.

i) For two successively reconfigured loop maps in $CSet$, they must satisfy necessary data dependencies. If the PEs mapped by the last operations in the former loop map are interconnected with the PEs mapped by the beginning operations in the latter one, the reconfiguration order of them is *Valid*. Otherwise, it is *Invalid*. For example in Fig. 4b, the results of operation E at T=2 in map d_0 is directly transferred and consumed by A and B at T=3 in map d_1. Since each obtained map in $CSet$ has the optimal II^*, the finally achieved $MapSet$, ordered as {d_0, d_1, d_2, d_3}, still brings the optimized II=II^*=3. For *Invalid* cases in Fig. 8a, loop maps d_0, d_3 and d_1 don't satisfy the data dependencies among them, so extra cycles are needed to prepare for the intermediate data, causing performance degraded.

ii) S^{max}_{ave} and σ^{stress}_{ave} should be minimized. The average KSD, $\overrightarrow{KSD}_{ave}$ of the loop map set $MapSet$, can be calculated by Equation (5) in Section 4. If one loop map maximizes the reduction in S^{max}_{ave} and σ^{stress}_{ave}, we select it as the next reconfigured map and add it into $MapSet$. If no loop map furthers to reduce S^{max}_{ave} and σ^{stress}_{ave}, we stop search. Finally, although several loop map sets $MapSet$s might be obtained, we select the one with lowest S^{max}_{ave} and σ^{stress}_{ave} as the first map set $MapSet$ to reconfigure on CGRA dynamically. For example in Fig. 4b, $MapSet = \{e_0, e_1, e_2, e_3\}$ is just the optimal one with $S^{max}_{ave} = 2.66$ and $\sigma^{stress}_{ave} \approx 0$, so we can select them to reconfigure on CGRA dynamically.

5.3.3 Multi Set of Reconfigurable Maps Searching

To achieve the largest reduction in S^{max}_{ave} and σ^{stress}_{ave} when compensating for those stresses on PEs among different loop maps, we can find more valid and feasible sets of loop maps to reconfigure on CGRA dynamically.

Here, assume that the objective set of reconfigurable loop maps is $MapSet = \{MapSet_1, \ldots, MapSet_M\}$. Each $MapSet_i$ of them is a single set of reconfigurable maps as constructed above for Map_i. To guarantee S^{max}_{ave} and σ^{stress}_{ave} of $MapSet$ to be lower than those of the optimized loop map Map^*, we sort all the loop maps obtained in the intra-kernel stress optimization procedures before by the ascending order of S^{max}_{ave} and σ^{stress}_{ave}, and start from the optimized map Map^* to generate a set of map candidates $CSet$ by those four geometric transformations above. Thus, the first set of reconfigurable maps $MapSet_1$ are generated from Map^*. Then, we ordinally resume searching feasibly reconfigured loop maps from the sorted maps $SMap$

$$\begin{aligned} S2^i_{ave} + S1^{max}_{ave} &\le 2 * S1^{max}_{ave} \\ S2^{max}_{ave} + S1^j_{ave} &\le 2 * S1^{max}_{ave} \\ S2^k_{ave} + S1^k_{ave} &\le 2 * S1^{max}_{ave}. \end{aligned} \tag{7}$$

As shown in Fig. 8b, $MapSet_1$ is a valid set of reconfigured loop maps for Map_1. We generate a set of map candidates $CSet_2$ for Map_2 and find a *Valid* map Map^*_2 from $CSet_2$, which satisfies the necessary data dependencies to reconfigure after $MapSet_1$. Similarly, we use map Map^*_2 rather than Map_2 to construct a set of reconfigurable loop maps $MapSet_2$ from $CSet_2$. Assume that $S1^{max}_{ave}$ and $S2^{max}_{ave}$ are the average maximum stresses accumulated on PEi for $MapSet_1$ and on PEj for $MapSet_2$, respectively. If they satisfy the conditions in Equation (7) above, it means the stresses accumulated on each PE by both $MapSet_1$ and $MapSet_2$, including the PEs with the highest stresses, are all smaller than the largest one when $MapSet_1$ repeats execution two times. When $MapSet_1$ and $MapSet_2$ are executed successively, S^{max}_{ave} and σ^{stress}_{ave} caused by them can be further reduced. So $MapSet_2$ should be added into Map_Set to execute after $MapSet_1$. Finally, we stop search until both S^{max}_{ave} and σ^{stress}_{ave} don't reduce or the sorted maps in $SMap$ are all searched out.

5.4 Whole Stress-Aware Loops Mapping Algorithm

The whole stress-aware loops mapping algorithm is presented in Algorithm 2. First, we calculate the mobility of operations and the length of critical path (L) in DFG. Then MII is determined and used as the beginning II for searching. TEC is created with CGRA under the current II. Then, sFDS method is used to pre-process to eliminate some invalid cases caused by large fan-out or operation amounts [15], [18] and to guide to find better schedules for next maps search [16], [22] (line 5). Next, the intra-kernel stress optimization tries to find the valid loop maps with MCC method discussed before (lines 6-23). If failing to find the valid loop maps, we go back to the latest scheduled operation orderly and select another schedule to map again. When S^{max} and σ^{stress} of the newly found maps don't decrease for several times continuously or the trial times exceed a certain threshold T, the loop map with the smallest S^{max} and σ^{stress} is regarded as the optimized one Map^* (lines 11-15). Then, the inter-kernel stress optimization starts from Map^* to find a set of loop maps $MapSet$ to minimize S^{max}_{ave} and σ^{stress}_{ave} as possible (lines 25-34). Finally, if the scheduled loop kernel D_{kernel} fails to be mapped on CGRA under the current II, we increase the II and resume searching until II exceeds the limit L (lines 37-40). Assume the sizes of loop DFG and CGRA are n and m, respectively. The complexity of our algorithm mainly depends on that of MCC searching procedures, which is about $O(n^2m^2)$ [15], [23]. So the total

complexity of the proposed stress-aware loops mapping algorithm is about $O(n^2m^2LT)$ roughly.

Algorithm 2. Whole Stress-Aware Loops Mapping

Require: Loop DFG D, CGRA C, Stress Vector SK;
Ensure: $MapSet, KSD_{ave}$;
1: $flag \leftarrow False$, $suc \leftarrow False$, $MapSet \leftarrow \varnothing$, $CSet \leftarrow \varnothing$;
2: $D, II_{crit}^{max} \leftarrow$ Determine_Mobility(D);
3: $II \leftarrow$ Determine_MII(D_p);
4: $T_{II} \leftarrow$ Create_TEC(C, II);
5: $D_{kernel} \leftarrow$ sFDS(D, II, SK);
6: **while** Reconfiguration $MapSet$ is not found || $II \leq L$ **do**
7: //**Intra-Kernel Stress Optimization** ('Hill Climbing' Strategy)
8: **while** $flag == False$ **do**
9: $CSet, suc, Map^* \leftarrow$ Find_MCC(D_{kernel}, T_{II}, SK);
10: **if** $suc == True$ **then**
11: **if** Map^* keeps optimal after several times continuously or exceed a certain threshold T **then**
12: $flag \leftarrow True$; **break**;
13: **else**
14: $D_{kernel} \leftarrow$ Re_sFDS(D_{kernel}, II, SK); **continue**;
15: **end if**
16: **else**
17: **if** Failures exceed a certain threshold T^{max} **then**
18: **break**;
19: **else**
20: $D_{kernel} \leftarrow$ Re_sFDS(D_{kernel}, II, SK); **continue**;
21: **end if**
22: **end if**
23: **end while**
24: //**Inter-Kernel Stress Optimization** ('Hill Climbing' Strategy)
25: **while** KSD^{ave} is not optimized && $SMap$ are not searched out **do**
26: $CSet, Map_j^* \leftarrow$ Generate_Candidates($CSet, Map_j$);
27: $CSet \leftarrow$ Check_Dependency($MapSet, CSet$);
28: $MapSet_j \leftarrow$ Find_MapSet($CSet, Map_j^*$);
29: $KSD^{ave} \leftarrow$ Calculate_Stress($MapSet_j, MapSet, SK$);
30: **if** S_{ave}^{max} and σ_{ave}^{stress} is decreased **then**
31: $MapSet \leftarrow$ Add_MapSet($MapSet, MapSet_j$); **continue**;
32: **end if**
33: $MapSet \leftarrow$ Determine_Iteration($MapSet$); // Average method
34: **end while**
35: **if** $suc == True$ **then**
36: **break**;
37: **else**
38: $II \leftarrow II + 1$;
39: $D_{kernel} \leftarrow$ sFDS(D, II, SK); **continue**;
40: **end if**
41: **end while**
42: **if** $MapSet == \varnothing$ **then**
43: **return** $\varnothing$; //Loop DFG D fails to map to target CGRA C;
44: **end if**
45: **return** $MapSet, KSD_{ave}$;

6 Experimental Results

To evaluate the proposed approach, these results are obtained from a cycle-accurate CGRA simulator SimRPU [6]. As shown in Fig. 1, this CGRA has a homogeneous 4×4 PE array with an coupled ARM7 host controller. Each PE has enough local registers and can perform 26 different types of 16/32-bit logic and arithmetic operations, such as xor, not, $shift$, add, sub and the most complex mul operations [5], [6]. With Taiwan Semiconductor Manufacturing Company (TSMC) 65-nm library, the area of one PE is 65,558 um^2 with about 54.63 thousands of gates. A 4-bank shared data memory of 32K-word size is required to be accessed by different PEs. The Direct Memory Access (DMA) controller is in charge of exchanging data between on-chip and off-chip data memory. The working frequency is 200 MHz.

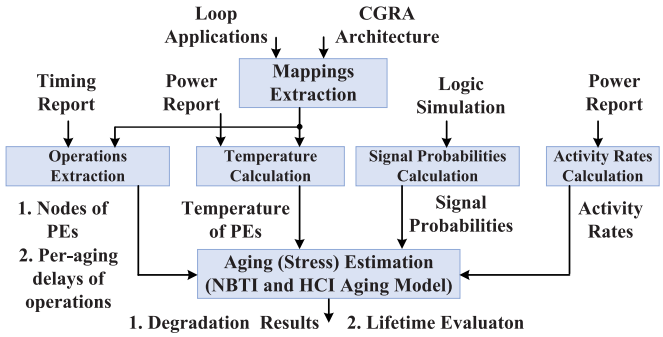


Fig. 9. Overview of evaluating stresses on CGRA.

The NBTI and HCI aging models mentioned in Sections 2 and 4 [26], [27] are used to evaluate the aging effects on delay degradation of PEs when different operations executed on them. The experimental flow is shown in Fig. 9. The mapped loops are fed to Synopsys VCS and Design Compiler to obtain the pre-aging delays, signal probabilities (or duty cycles), activity rates of operations to evaluate the stresses on PEs [26], [27]. The total area and power consumption for a 4×4 PE array are about 1,048,928 um^2 and 244 mW, respectively. As shown in Table 1, the total power consumption of of different loops varies from 31 to 470 mW. They are used to calculate the average temperature of PEs using a widely-used temperature model Hotspot in [35], [36], which causes a temperature range of 319.2∼322.0 K with a variance of 2.8 K. Then, the degradations in delay and lifetime of PEs that executes different operations can be obtained using those NBTI and HCI aging evaluation models, which are used to determine the kernel stress distributions of different loop maps. In all the experiments, the maximum amount of delay degradation (worst case) is used to get both A_{NBTI} and A_{HCI}, which is set to be 10% in 3 years for SP = 1 (always ON → worst case for NBTI) and AR = 200 MHz (maximum frequency → worst case for HCI)[2] [27], [30].

To verify the effectiveness of our approach, as shown in Table 1, 16 loop benchmarks are selected to evaluate from Spec2006, Mibench and PolyBench. Four approaches are employed for comparison with our approach: *Baseline*, *EPIMap*, *SAP*, *SAMD*. The *Baseline* approach introduces neither any stress optimization nor any pipelining technique [13] for performance optimization. The *EPIMap* approach in [15] is only focused on the performance optimization on CGRA using the software pipeline technique, but not employs any stress optimizations. The *SAP* approach in [7] is a stress-aware module placement method to distribute the stress evenly on the reconfigurable PEs within one placement and

2. The worst cases for NBTI and HCI can be taken from users (e.g., in [27]), or alternatively from real measurements (e.g., in [8]).

TABLE 1
Loop Benchmarks Description

Benchmark	Loops	Loops Location (DIR/Filename: Line Number)	Iteration Count	OP_ Num	OP_ Type#	Criti-Path	Loop Carried	Reconfig. Maps#	Temperature (K)	Compilation Times (s)
Spec2006	mshift	gromacs/mshift.c:571-579	1024	20	3	8	No	8	320.7	43.44
	wrf	wrf/collecton.c:141-144	27	5	3	5	Yes	16	319.3	10.27
	calcvir	gromacs/calcvir.c:59-71	256	16	2	4	No	8	322.0	34.62
	Pppm	gromacs/Pppm.c.c:322-326	32	12	4	4	No	8	321.5	16.03
	places	astar/place.cpp:116-123	512	13	4	6	No	8	321.0	34.96
	WayInit	astar/WayInit.cpp:71-80	256	17	7	10	No	8	320.2	38.24
	clincs	gromacs/clincs.c:105-126	32	27	3	6	No	8	320.6	49.13
	vsolve	soplex/vsolve.cc:278-289	100	21	4	10	No	8	320.8	39.93
MiBench	jfdctflt	jpeg/jfdctflt.c:120-166	64	59	3	22	No	8	321.1	92.28
	fft	lame/fft.c:91-119	1024	42	3	13	No	8	320.8	88.12
	jquant2	jpeg/jquant2.cpp:522-529	256	18	4	7	No	8	320.1	35.77
	newmdct	lame/newmdct.c:163-188	1024	84	4	34	No	8	321.2	120.43
PolyBench	fdtd-apml	fdtd-apml/fdtd-apml.c:130-138	1000	21	4	13	No	8	321.1	43.44
	seidel-2d	seidel-2d/seidel-2d.c:64-68	32	9	2	8	Yes	4	319.2	14.11
	syr2k	syr2k/syr2k.c:70-79	16	6	2	4	Yes	16	319.8	11.63
	adi	adi/adi.c:73-77	32	6	3	6	Yes	4	319.4	6.56

mapping scheme. The *SAMD* approach in [29] is a novel stress-aware module diversification approach, which creates multiple different placement and mapping configurations to distribute and reduce the stresses on PEs.

6.1 Reducing Accumulated Stresses on PEs

In our experiment, we repeatedly map the 16 selected loop benchmarks on a 4×4 CGRA by the above five different optimization approaches for comparison.

As discussed before, the total stresses accumulated on PEs significantly depend on the numbers and types of operation executed on them. As shown in Fig. 10, the *Baseline* approach has the largest ratios of operation numbers versus operation types (OP_Num/OP_Type) on PEs in all 16 loop benchmarks. Just because it doesn't employ any pipelining techniques, the operation numbers dominate these ratios and cause more operations mapped on the partial PEs. The *SAP* approach obtains smaller ratios of OP_Num/OP_Type than those by the *EPIMap* approach. Since the *SAP* approach introduces the stress optimization to schedule operations of the same types at the same timeslots, it distributes them to more different PEs and has more operation types, causing smaller OP_Num/OP_Type. The *SAMD* approach also achieves smaller ratios than the *SAP* approach in most kernels, since it can further diversify operations on PEs among different loop maps. Because it can't guarantee to search loop maps from the optimized ones, it sometimes causes worse stress optimization results than the *SAP* approach, such as in *fft* and *seidel-2d*. Our approach can achieve the smallest ratios of OP_Num/OP_Type on PEs in all 16 loops, since it joints the intra- and inter-kernel stress optimizations, which stats from the optimized one to generate multiple reconfigured loop maps. It can scatter and diversify different operations on more PEs to bring smaller OP_Num and large OP_Type, which finally causes those ratios reduced greatly.

Since smaller OP_Num/OP_Type ratios indicate that there are less operations but more operation types on PEs, they usually bring longer recovery times and shorter stress times on PEs, which finally reduces the stresses on them. As shown in Fig. 11, our approach achieves the lowest S_{ave}^{max} in all 16 loops when compared to the *Baseline*, *EPIMap*, *SAP* and *SAMD* approaches, which averagely brings 82.0%, 60.1%, 40.7% and 30.8% reduction in S_{ave}^{max} for NBTI and 79.4%, 49.6%, 32.7% and 30.7% reduction for HCI, respectively. Moreover, it is noted that the HCI aging effect also has commensurate or slightly larger maximum stresses on PEs compared to NBTI in our experiments. So it is significant to consider those two dominant aging effects of NBTI and HCI in our stress-aware loops mapping solutions.

6.2 Balancing Stress Distributions on CGRA

As we know, when more PEs are used, more operations can be mapped to scatter the stresses accumulated on them, helping to achieve more balanced stress distributions.

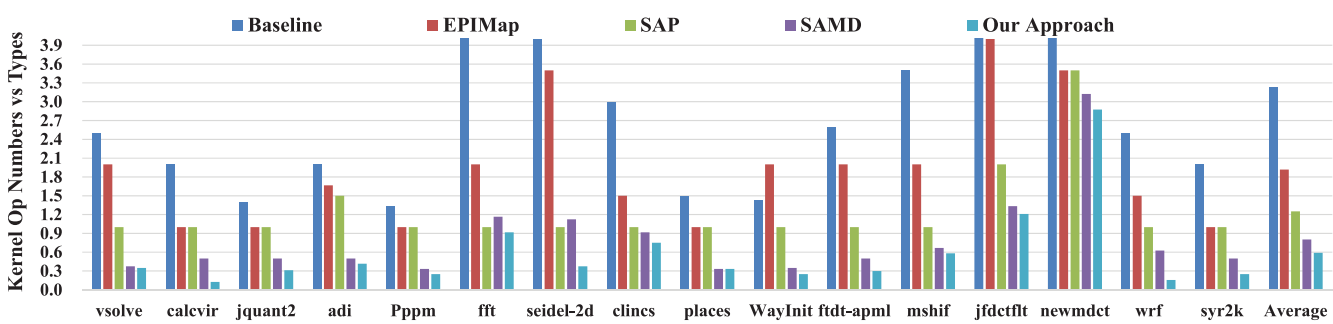


Fig. 10. Our approach achieves the smallest ratios of the operation numbers versus the operation types on PEs.

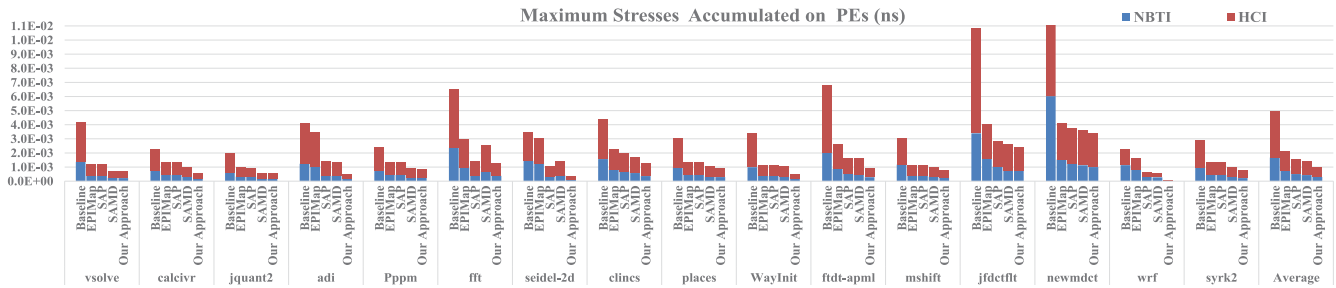


Fig. 11. Our approach achieves the lowest maximum aging stresses on CGRA for one loop kernel averagely.

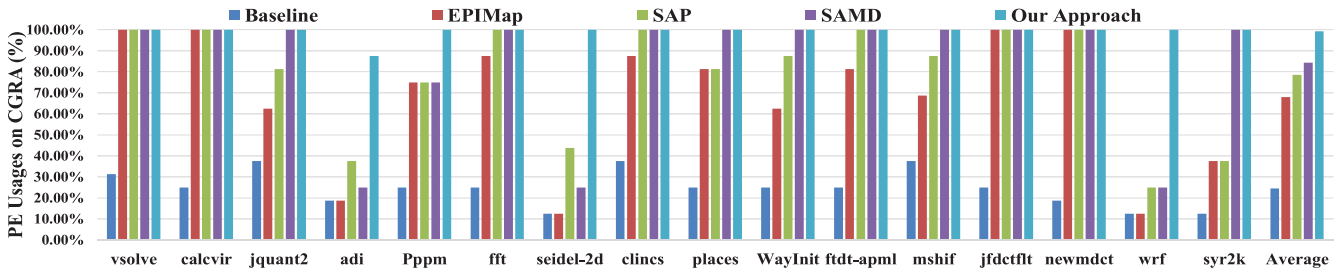


Fig. 12. Our approach achieves the largest PE usages due to pipelining technique and dynamic multi-map reconfiguration.

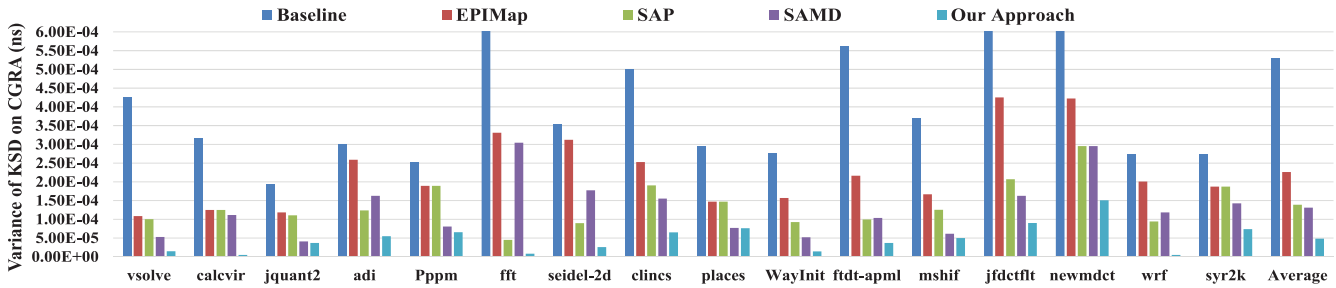


Fig. 13. Our approach achieves the smallest stress variances, which means the most balanced stress distributions on CGRA.

As shown in Fig. 12, the *Baseline* approach has the smallest PE usages in all the 16 loops compared to other four approaches. Since it doesn't employ any pipelining techniques and has a lower execution parallelism, these operations are mostly mapped to partial PEs on CGRA, which causes the PE usages very inadequate. The *SAP* approach obtains larger PE usages than those of the *EPIMap* approach. Because the *SAP* approach not only employs the pipelining technique to improve the parallelism, but also introduces the stress optimization within loop kernels, which helps to map operations on more PEs and scatter the stresses on them. Although the *SAMD* approach generates multiple reconfigured loop maps on PEs arbitrarily, it usually distributes operations onto more PEs and obtains larger PE usages than the *SAP* approach. Since our proposed multi-map scheduling method starts from an optimized one to generate multiple loop maps to reconfigure on CGRA dynamically, our approach can further diversify the PE usages and map operations to more PEs to scatter the stresses accumulated on PEs, which brings the largest PE usages in all 16 loops and a utilization of near 100% in 15 out of 16 loops.

As shown in Fig. 12, our approach achieves the optimized PE usages about 99% averagely, and improves by 303.2%, 45.9%, 26.4% and 17.6% compared to the *Baseline*, *EPIMap*, *SAP* and *SAMD* approaches, whose PE usages are averagely 25%, 68%, 79% and 84%, respectively. Consequently, our approach, as shown in Fig. 13, also obtains the lowest σ_{ave}^{stress} in all 16 loops, and averagely brings 90.9%, 78.6%, 65.2% and 63.1% reduction in σ_{ave}^{stress} compared to the *Baseline*, *EPIMap*, *SAP* and *SAMD* approaches, respectively, which indicates that our approach can bring the most balanced stress distributions on CGRA.

6.3 Keeping Best Performance on Loops Mapping

In the optimization goals, we try to keep the optimized performance (smallest II) when dedicating to minimize S_{ave}^{max} and σ_{ave}^{stress}. As shown Fig. 14, the optimized IIs by achieved the five approaches and the MII of loop benchmarks are presented. It is noted that our approach achieves the smallest MII for 12 out of 16 loop benchmarks. Our achieved IIs are also the optimized ones among all five different approaches in all the 16 loop benchmarks, which are the same as those obtained by all the *EPIMap*, *SAP* and *SAMD* approaches. It is because that our approach also employs the pipelining technique, which attempts to search from MII iteratively. This guarantees to achieve the optimized loop maps with the smallest II in the stress-aware intra-kernel loops optimization procedures. Moreover, two successively reconfigured loop maps in our approach satisfy the data dependency between them, which don't cause extra cycles to route intermediate data in the inter-kernel loops optimization procedure. So our approach doesn't cause the performance degraded when multiple loop maps reconfigured on CGRA dynamically. In short, our approach

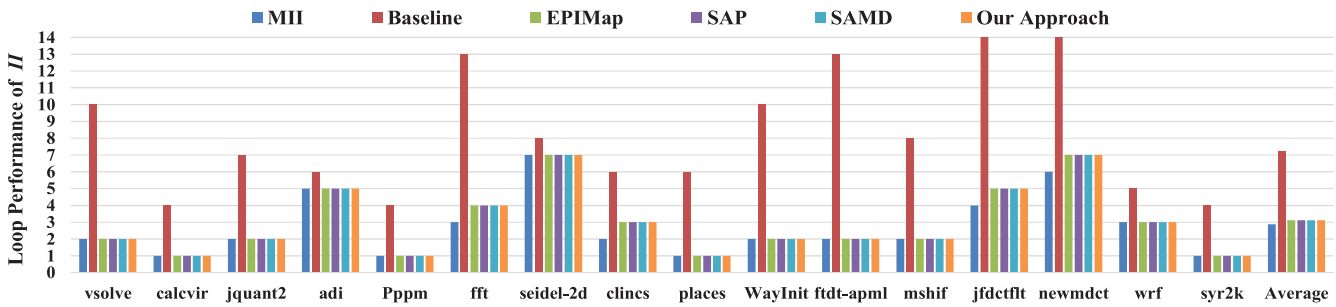


Fig. 14. Our approach achieves the optimized IIs for loops mapping on 4×4 CGRA compared to other four approaches.

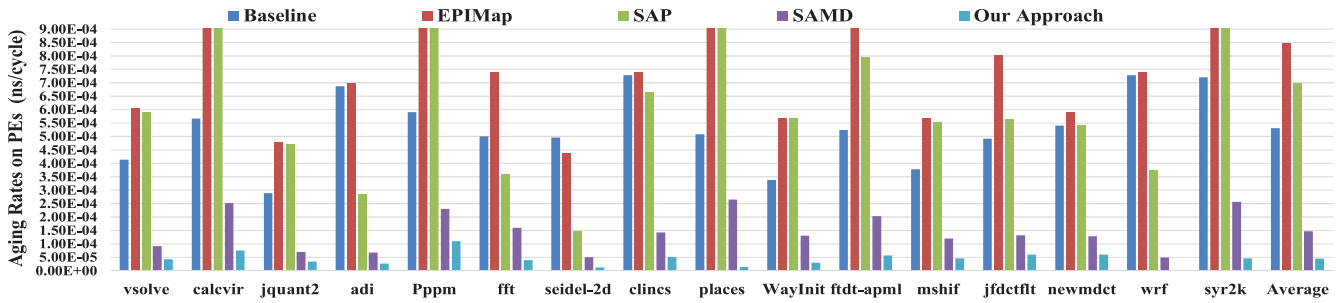


Fig. 15. Our approach achieves the lowest aging rates in all 16 loops on CGRA compared to other four approaches.

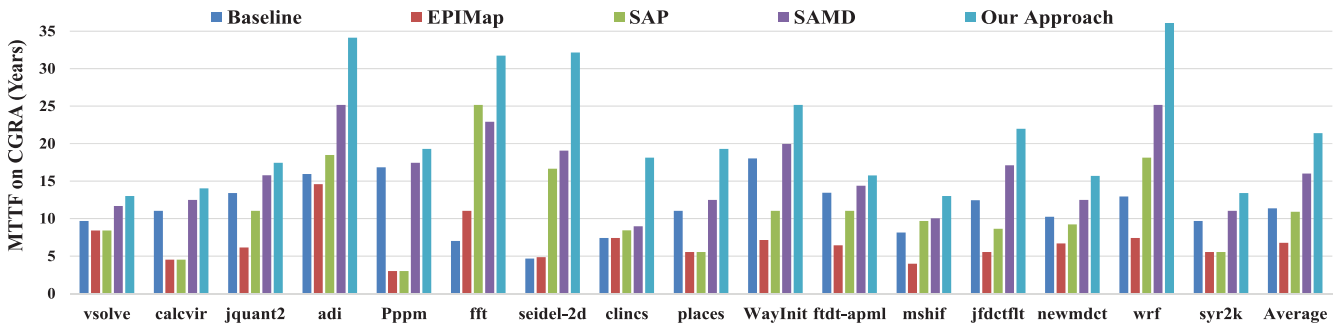


Fig. 16. Our approach achieves the largest MTTFs for all the 16 loops among all the five approaches.

can keep the optimized performance compared to the *EPIMap*, *SAP* and *SAMD* approaches, while effectively reducing and balancing the stresses accumulated on PEs.

6.4 Aging Rate Reduction on PEs

In our experiment, *Aging Rate* can be defined as the average maximum stresses accumulated on PEs per cycle due to both NBTI and HCI aging effects by different operations [27].

As shown in Fig. 15, the *EPIMap* approach has larger aging rates than those of the *Baseline* in all 16 loops. Because pipelining loop kernels can improve the parallelism and makes more PEs to be used. This reduces the number of operations executed on PEs within one loop kernel, which causes the *EPIMap* approach executes one loop kernel faster. So the *EPIMap* approach actually has more operations executed on the same PEs within the same time periods, which finally induces higher stresses and larger aging rates on PEs. The *SAP* approach has larger aging rates than those of the *Baseline* approach in 11 out of 16 loops. It indicates that it is not enough to obtain better operation mapping schemes by employing the stress optimization only in the intra-kernel of loops after pipelining, since it can't guarantee to avoid overmany operations to be mapped on the same PEs. Thus, we should introduce the dynamic multi-map reconfiguration strategy among different loop maps using the inter-kernel stress optimization, which furthers to diversify the PE usages and scatter both the numbers and types of operations to reduce the maximum stresses accumulated on PEs. In the inter-kernel stress optimization, our approach adopts the proposed multi-map scheduling method with these four transformation strategies of *translation*, *symmetry*, *rotation* and *mirror*, which can guarantee to generate more feasible loop maps to reconfigure on PEs dynamically when compared to the *SAMD* approach. As a result, our approach, as shown in Fig. 15, achieves the lowest aging rates in all the 16 loops, which brings about 91.6%, 94.7%, 93.6% and 69.7% reductions averagely when compared to the *Baseline*, *EPIMap*, *SAP* and *SAMD* approaches, respectively.

6.5 MTTF and Aging Efficiency Improvement

In this paper, MTTF is calculated when the delay degradation caused by the NBTI and HCI aging effects exceeds 20% of the pre-delay d_0 for any types of operations on PEs [30].

As shown in Fig. 16, it is noted that the MTTFs achieved by the *Baseline* approach are not only larger than those by the *EPIMap* approach in 14 out of 16 loops but also larger than those by the *SAP* approach in 10 out of 16 loops. Because the smaller IIs obtained by the *EPIMap* approach

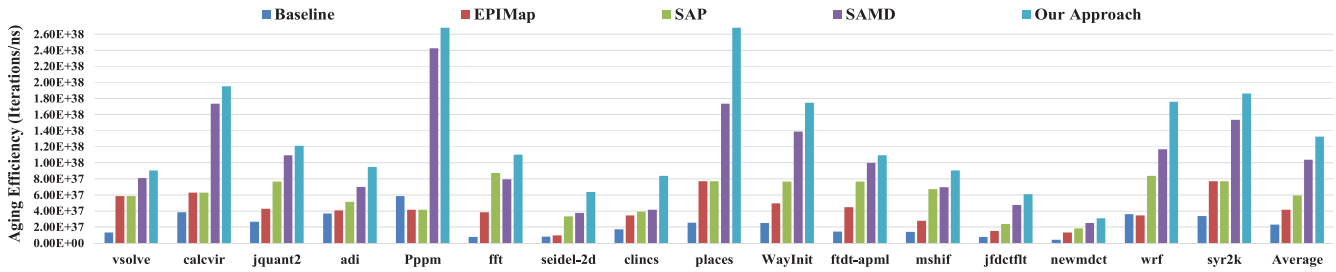


Fig. 17. Our approach achieves the largest aging efficiencies due to the intra- and inter-kernel stress optimizations.

causes overmany operations of the same types mapped on the same PEs within the same time periods. So the total numbers of operations of the same types on some PEs can't be reduced and induce larger stresses on them from a long term, which finally cause smaller MTTFs for the *EPIMap* approach in most loops. As for the *SAP* approach, the results also show that only optimizing stress for a better loop mapping scheme within loop kernels can't fully solve this problem caused by loops pipelining. Due to the strict data dependencies within a loop kernel, it only disperses the operations of the same types to partial PEs. Since the *SAMD* approach and our proposed approach employ the dynamic multiple loop maps to reconfigure on PEs, they can further diversify the operation types and compensate for the stresses accumulated on them, which finally bring larger MTTFs like achieving the lower aging rates in all 16 loops as shown in Fig. 15. Since our approach joints the two-stage intra- and inter-kernel stress optimizations by starting from the optimized one to generate more reconfigured loop maps, it can further scatter the operations to more PEs and reduce the stresses accumulated on them, which helps to bring larger MTTFs than the *SAMD* approach. Thus, our approach, as shown in Fig. 15, achieves the largest MTTFs in all the 16 loops, which finally brings 1.88X, 3.16X, 1.96X and 1.34X improvement averagely compared to the *Baseline*, *EPIMap*, *SAP* and *SAMD* approaches, respectively

$$\text{Aging Efficiency} = \frac{\text{MTTF}}{\text{II} \times \text{Cycle} \times \text{Max\_Degradation}}. \quad (8)$$

Moreover, by analogy with the energy-efficiency concept (Operations/J) [2], [3], [6], *Aging Efficiency* is defined as the average iteration numbers of loop kernels executed on CGRA per degradation unit (Iterations/ns), when the maximum aging stresses exceed 20% of the pre-delay d_0 for any types of operations on PEs. The larger the aging efficiency is, the more loop kernels are executed within the same degradation of delay. The aging efficiency can be calculated in Equation (8) above. As a result, as shown in Fig. 17, our approach can achieve the highest aging efficiencies in all 16 loops and bring 6.01X, 3.18X, 2.23X and 1.28X improvement averagely when compared to the *Baseline*, *EPIMap*, *SAP* and *SAMD* approaches, respectively.

TABLE 2
Normalized Average Results Comparison for Those Obtained Loop Maps Before by Re-Evaluating with [30]

Compared Approaches	S_{ave}^{max}	σ_{ave}^{stress}	Aging Rate	Aging Efficiency	MTTF
Baseline	1.000	1.000	1.000	1.000	1.000
EPIMap	0.465	0.347	1.250	0.360	0.761
SAP	0.364	0.243	1.084	0.618	1.011
SAMD	0.253	0.163	0.705	1.196	1.783
Our approach	0.185	0.068	0.503	2.743	3.458

6.6 Robustness and Optimality Analysis

Our proposed approach is actually a general stresses optimization framework for loops mapping on CGRA, which wouldn't be limited by what stress estimation model is used. The adopted stress estimation model in [27] is effective enough to meet the accuracy need of CGRA-based system designers to decide which mapping solution is better in terms of aging issues. To validate the robustness of our optimization approach against imperfect stress modeling, another transistor-level stress estimation model [30] is adopted to re-evaluate the stresses accumulated on PEs for those loop maps obtained by five mapping approaches before. In the experiments, it is found that those loop maps obtained by our approach still achieve the smallest stresses and most balanced stress distributions on PEs in all 16 loops. From the average results normalized in Table 2, it is also noted that our approach achieves the best and consistent stress optimization results for those obtained loop maps when re-evaluated by [30], indicating that our approach is effective and applicative for different stress estimation models.

Moreover, our proposed stress optimization strategy is a much effective method, which integrates the heuristic two-stage smart hill-climbing idea [20], [21] into our intra- and inter-kernel stress optimization procedures. To validate the optimality, we add a set of comparative experiments with the brute-force search. It is found that our proposed approach achieves similar or equal optimization results in all loop benchmarks. As depicted in Table 3, the average differences of both S_{ave}^{max} and σ_{ave}^{stress} are only 3.71% and 4.96% when compared to the brute-force search, which means that our proposed approach can obtain good and satisfactory stress optimization results in our solutions.

TABLE 3
Average Results Comparison Between our Proposed Approach and the Brute-Force Search Approach

Approaches	S_{ave}^{max} (ns)	σ_{ave}^{stress} (ns)	PE Usage (%)	Aging Rate (ns/cycle)	Aging Effic. (Iter./ns)
Our approach	9.86E-04	4.84E-05	99.2%	4.49E-05	1.33E+38
Brute-force	9.49E-04	4.60E-05	99.2%	4.30E-05	1.38E+38
Difference	3.71% ↓	4.96% ↓	0.0% ↑	4.49% ↓	4.12% ↑

TABLE 4
Results for Four Different-Sized Loops Mapped on Large-Sized 8×8 CGRA

Loop Benchmarks	Approaches	II	$\frac{\text{OP\_Num}}{\text{OP\_Type}}$	S_{ave}^{max} (ns)	PE Usages	Reconfig. Maps$^{\#}$	σ_{ave}^{stress} (ns)	Aging Rates (ns/cycle)	MTTF (Years)	Aging Efficiencies	Compil. Times (s)
wrf	Baseline	5	1.67	3.303E-03	3.13%	1	1.408E-04	6.605E-04	8.64	5.792E+37	15.73
	EPIMap	3	1.50	2.218E-03	3.13%	1	1.047E-04	7.393E-04	7.42	8.294E+37	17.47
	SAP	3	1.00	1.120E-03	6.25%	1	5.862E-05	3.735E-04	18.12	2.025E+38	38.83
	SAMD	3	0.63	9.913E-04	6.25%	4	7.339E-05	8.261E-05	25.17	2.882E+38	19.56
	Our approach	3	0.04	3.282E-05	100.0%	64	1.578E-20	1.710E-08	79.49	8.882E+38	43.14
calcvir	Baseline	4	1.33	2.340E-03	6.25%	1	1.773E-04	5.850E-04	12.49	9.247E+37	45.37
	EPIMap	1	1.00	5.368E-04	25.0%	1	8.722E-05	5.368E-04	3.98	1.334E+38	47.76
	SAP	1	1.00	5.368E-04	25.0%	1	8.722E-05	5.368E-04	3.98	1.334E+38	106.92
	SAMD	1	0.25	4.308E-04	75.0%	4	4.027E-05	1.077E-05	15.69	5.260E+38	51.24
	Our approach	1	0.13	1.686E-04	100.0%	32	6.311E-21	1.054E-05	35.87	1.154E+39	117.92
fft	Baseline	13	6.50	6.278E-03	6.25%	1	4.411E-04	4.829E-04	7.01	1.808E+37	96.73
	EPIMap	2	2.00	1.889E-03	40.6%	1	1.828E-04	9.443E-04	4.53	7.594E+37	101.82
	SAP	2	1.00	1.202E-03	64.1%	1	1.001E-04	6.011E-04	11.04	1.849E+38	226.26
	SAMD	2	0.50	1.171E-03	93.4%	4	7.862E-05	1.464E-04	15.77	2.643E+38	127.38
	Our approach	2	0.33	6.735E-04	100.0%	8	4.627E-05	4.209E-05	26.74	4.219E+38	251.40
newmdct	Baseline	34	11.33	1.837E-02	4.69%	1	1.062E-03	5.401E-04	10.24	1.010E+37	143.64
	EPIMap	2	2.00	1.899E-03	90.6%	1	2.450E-04	9.495E-04	5.12	8.586E+37	151.20
	SAP	2	1.00	1.899E-03	96.8%	1	2.297E-04	9.495E-04	9.68	1.622E+38	335.99
	SAMD	2	1.00	1.807E-03	100.0%	4	1.430E-13	2.259E-04	13.02	2.182E+38	161.41
	Our approach	2	0.81	1.513E-03	100.0%	8	1.145E-04	9.453E-05	16.59	2.781E+38	373.33

6.7 Scalability on Large-Sized CGRA

In order to evaluate the scalability of our stress optimization approach when mapping loops on large-sized CGRAs, four different-sized loop benchmarks *wrf*, *calcvir*, *fft* and *newmdct* are taken as instances to map on a larger 8×8 CGRA. Here, *wrf* is a do-across loop that has a loop-carried dependency. The mapping and stress optimization results of those four loops are summarized in Table 4.

It is noted that our approach is still effective and achieves the smallest IIs, lowest S_{ave}^{max} and σ_{ave}^{stress}, largest PE usages, lowest aging rates, largest MTTFs and highest aging efficiencies for all four loop benchmarks in those five approaches. Our approach achieves 92.1%, 63.5%, 49.8% and 45.6% improvement in S_{ave}^{max} averagely when compared to the *Baseline*, *EPIMap*, *SAP* and *SAMD* approaches, respectively. All the *EPIMap*, *SAP*, *SAMD* and our approaches achieve smaller IIs in three loops except for *wrf*. Because the IIs of *wrf* are mainly dominated by the loop-carried dependency, but the IIs of other three loops significantly depend on the available resources of PEs. Also, since the larger CGRA array makes more operations to be scattered to more PEs, it brings smaller IIs and higher parallelism, causing smaller stresses to be accumulated on them. Thus, when compared to 4×4 CGRA, the S_{ave}^{max} obtained by our approach can be decreased by about 64.9%, 72.0%, 46.8% and 55.2% for those four loops *wrf*, *calcvir*, *fft* and *newmdct*, respectively. Since the smaller IIs can bring higher parallelism and faster execution speed, the aging efficiencies achieved by our approach are also improved by 4.05X, 4.92X, 2.83X and 7.93X for those four loop benchmarks, respectively.

6.8 Compilation Time Evaluation

The compilation time of our approach is also evaluated when running on an Intel Core(TM) i3 CPU PC at 2.6 GHz with 4 GB memory. Tables 1 and 4 show that the compilation time is inevitably increased when the loops become larger and are mapped to larger-sized CGRA. But they are still in the order of magnitude of tens or hundreds of seconds, which are fairly low and acceptable in practice.

7 Conclusion

With massive parallel resources and dynamic reconfiguration feature, a heuristic two-stag stress-aware loops mapping algorithm is proposed for CGRA by integrating the intra-kernel and inter-kernel stress optimization strategies into the loops scheduling and mapping procedures. The intra-kernel stress optimization tries to optimize operations mapping on different PEs within loop kernels, helping to reduce the stresses accumulated on them. The inter-kernel stress optimization tries to generate multiple loop maps to reconfigure dynamically, which furthers to diversify operations mapping and compensate to reduce the stresses accumulated on different PEs. Our experimental results show that our approach not only keeps the smallest IIs of loops but also brings the smallest maximum accumulated stresses and most balanced stress distributions on PEs, indicating that our approach can effectively mitigate the aging issues on CGRA while keeping the optimized performance.

8 Appendix

Theorem 1. *For a simple undirected graph, it is denoted by an adjacent matrix M. If it contains a maximum complete subgraph with m vertices, M^2 has at least $m-1$ elements larger than or equal to $m-2$ in m vertice-related columns (rows) [23], [24].*

Proof. For a complete graph, any two vertices have an edge between them [23]. Since M is an adjacent matrix of a simple undirected graph, the matrix M^k represents the number of paths whose length is k [23], [24]. If a complete graph has n vertices ($n > 2$), there has $n-2$ paths whose

length is 2 between any two vertices. So $M^2 = M \times M$ has $n-1$ elements equal to $n-2$ in any row or column. Thus, if a simple undirected graph has a maximum complete graph of m vertices, there exist at least $m-1$ elements in M^2 larger than or equal to $m-2$ in m vertice-related columns (rows). □

Obviously, the theorem above can help to guide to search the maximum complete subgraphs that is also regarded as maximal compatibility classes (MCCs) [15], [23], [32], which can also be introduced to guide to find the valid loop maps between the DFG of loop kernel and TEC.

ACKNOWLEDGMENTS

This work was supported in part by the China National High Technologies Research Program (No.2015AA016601), the China Major S&T Project (No.2013ZX01033001-001-003) and the NSFC under Grant 61774094.

REFERENCES

[1] (2015). [Online]. Available: https://www.semiconductors.org/main/2015_international_technology_roadmap_for_semiconductors_itrs/

[2] M. Wijtvliet, L. Waeijen, and H. Corporaal, "Coarse grained reconfigurable architectures in the past 25 years: Overview and classification," in *Proc. Int. Conf. Embedded Comput. Syst.: Archit. Model. Simul.*, 2016, pp. 235–244.

[3] F. Bouwens, M. Berekovic, B. De Sutter, and G. Gaydadjiev, "Architecture enhancements for the ADRES coarse-grained reconfigurable array," in *Proc. 3rd Int. Conf. High Perform. Embedded Archit. Compilers*, 2008, pp. 66–81.

[4] H. Amano, Y. Hasegawa, S. Tsutsumi, T. Nakamura, T. Nishimura, V. Tanbunheng, A. Parimala, T. Sano, and M. Kato, "MuCCRA chips: Configurable dynamically-reconfigurable processors," in *Proc. IEEE Asian Solid-State Circuits Conf.*, 2007, pp. 384–387.

[5] M. Zhu, L. Liu, S. Yin, Y. Wang, W. Wang, and S. Wei, "A reconfigurable multi-processor SoC for media applications," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2010, pp. 2011–2014.

[6] L. Liu, D. Wang, S. Yin, Y. Chen, M. Zhu, and S. Wei, "SimRPU: A simulation environment for reconfigurable architecture exploration," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 22, no. 12, pp. 2635–2648, Dec. 2014.

[7] J. Angermeier, D. Ziener, M. Glaß, and J. Teich, "Stress-aware module placement on reconfigurable devices," in *Proc. Int. Conf. Field Programmable Logic Appl.*, 2011, pp. 277–281.

[8] E. A. Stott, J. S. Wong, P. Sedcole, and P. Y. Cheung, "Degradation in FPGAs: Measurement and modelling," in *Proc. 18th Annu. ACM/SIGDA Int. Symp. Field Programmable Gate Arrays*, 2010, pp. 229–238.

[9] K. Kang, S. P. Park, K. Roy, and M. A. Alam, "Estimation of statistical variation in temporal NBTI degradation and its impact on lifetime circuit performance," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, 2007, pp. 730–734.

[10] D. Lorenz, G. Georgakos, and U. Schlichtmann, "Aging analysis of circuit timing considering NBTI and HCI," in *Proc. 15th IEEE Int. On-Line Testing Symp.*, 2009, pp. 3–8.

[11] M. Noda, S. Kajihara, Y. Sato, K. Miyase, X. Wen, and Y. Miura, "On estimation of NBTI-induced delay degradation," in *Proc. 15th IEEE Eur. Test Symp.*, 2010, pp. 107–111.

[12] J. H. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*. Amsterdam, The Netherlands: Elsevier, 2011.

[13] V. H. Allan, R. B. Jones, R. M. Lee, and S. J. Allan, "Software pipelining," *ACM Comput. Surveys*, vol. 27, no. 3, pp. 367–432, 1995.

[14] T. Nowatzki, V. Gangadhar, N. Ardalani, and K. Sankaralingam, "Stream-dataflow acceleration," in *Proc. ACM/IEEE 44th Annu. Int. Symp. Comput. Archit.*, 2017, pp. 416–429.

[15] M. Hamzeh, A. Shrivastava, and S. Vrudhula, "EPIMap: Using epimorphism to map applications on CGRAs," in *Proc. 49th Annu. Des. Autom. Conf.*, 2012, pp. 1280–1287.

[16] A. Fell, Z. E. Rákossy, and A. Chattopadhyay, "Force-directed scheduling for data flow graph mapping on coarse-grained reconfigurable architectures," in *Proc. Int. Conf. ReConFigurable Comput. FPGAs*, 2014, pp. 1–8.

[17] M. Hamzeh and A. Shrivastava, "REGIMap: Register-aware application mapping on coarse-grained reconfigurable architectures (CGRAs)," in *Proc. 50th Annu. Des. Autom. Conf.*, 2013, Art. no. 18.

[18] 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 *Proc. 17th Int. Conf. Parallel Archit. Compilation Techn.*, 2008, pp. 166–176.

[19] B. Mei, S. Vernalde, D. Verkest, H. De Man, and R. Lauwereins, "Exploiting loop-level parallelism on coarse-grained reconfigurable architectures using modulo scheduling," in *Proc. IEE Comput. Digit. Techn.*, 2003, pp. 255–61.

[20] M. Fattah, M. Daneshtalab, P. Liljeberg, and J. Plosila, "Smart hill climbing for agile dynamic mapping in many-core systems," in *Proc. 50th Annu. Des. Autom. Conf.*, 2013, pp. 39:1–39:6.

[21] B. Xi, Z. Liu, M. Raghavachari, C. H. Xia, and L. Zhang, "A smart hill-climbing algorithm for application server configuration," in *Proc. 13th Int. Conf. World Wide Web*, 2004, pp. 287–296.

[22] P. G. Paulin and J. P. Knight, "Force-directed scheduling for the behavioral synthesis of ASICs," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 8, no. 6, pp. 661–679, Jun. 1989.

[23] J.-P. Tremblay and R. Manohar, *Discrete Mathematical Structures with Applications to Computer Science*. New York, NY, USA: McGraw-Hill, 1975.

[24] X. Chen, Q. Lin, Y. Xue, and Y. U. Xian-Zhi, "A new algorithm for maximal compatible classes based on matrix," *J. Logistical*, vol. 26, no. 4, pp. 92–96, 2010.

[25] R. Vattikonda, W. Wang, and Y. Cao, "Modeling and minimization of PMOS NBTI effect for robust nanometer design," in *Proc. 43rd ACM/IEEE Des. Autom. Conf.*, 2006, pp. 1047–1052.

[26] A. Tiwari and J. Torrellas, "Facelift: Hiding and slowing down aging in multicores," in *Proc. 41st IEEE/ACM Int. Symp. Microarchit.*, 2008, pp. 129–140.

[27] A. Amouri and M. Tahoori, "High-level aging estimation for FPGA-mapped designs," in *Proc. 22nd Int. Conf. Field Programmable Logic Appl.*, 2012, pp. 284–291.

[28] B. R. Rau, "Iterative modulo scheduling: An algorithm for software pipelining loops," in *Proc. 27th Annu. Int. Symp. Microarchit.*, 1994, pp. 63–74.

[29] H. Zhang, L. Bauer, M. A. Kochte, E. Schneider, C. Braun, M. E. Imhof, H.-J. Wunderlich, and J. Henkel, "Module diversification: Fault tolerance and aging mitigation for runtime reconfigurable architectures," in *Proc. IEEE Int. Test Conf.*, 2013, pp. 1–10.

[30] H. Zhang, M. A. Kochte, E. Schneider, L. Bauer, H.-J. Wunderlich, and J. Henkel, "STRAP: Stress-aware placement for aging mitigation in runtime reconfigurable architectures," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, 2015, pp. 38–45.

[31] D. K. Schroder and J. A. Babcock, "Negative bias temperature instability: Road to cross in deep submicron silicon semiconductor manufacturing," *J. Appl. Physics*, vol. 94, pp. 1–18, 2003.

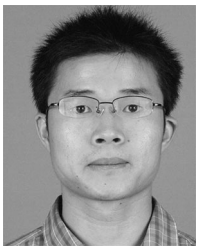
[32] G. Levi, "A note on the derivation of maximal common subgraphs of two directed or undirected graphs," *Calcolo*, vol. 9, 1973, Art. no. 341.

[33] E. Stott, J. S. J. Wong, and P. Y. K. Cheung, "Degradation analysis and mitigation in FPGAs," in *Proc. Int. Conf. Field Programmable Logic Appl.*, 2010, pp. 428–433.

[34] E. Stott and P. Y. Cheung, "Improving FPGA reliability with wear-levelling," in *Proc. Int. Conf. Field Programmable Logic Appl.*, 2011, pp. 323–328.

[35] W. Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron, and M. R. Stan, "HotSpot: A compact thermal modeling methodology for early-stage VLSI design," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 14, no. 5, pp. 501–513, May 2006.

[36] M. Cox, A. K. Singh, A. Kumar, and H. Corporaal, "Thermal-aware mapping of streaming applications on 3D multi-processor systems," in *Proc. 11th Symp. Embedded Syst. Real-time Multimedia*, 2013, pp. 11–20.



Jiangyuan Gu received the BS degree from the School of Microelectronics, Xidian University, Xian, China, in 2014. Currently he is working toward the PhD degree in the Institute of Microelectronics, Tsinghua University, Beijing, China. His research interests include reconfigurable computing, low-powered, and low aging loops mapping for reconfigurable design.

Shouyi Yin received the BS, MS, and PhD degrees in electronic engineering from Tsinghua University, China, in 2000, 2002, and 2005, respectively. He has worked in Imperial College London as a research associate. Currently, he is with the Institute of Microelectronics, Tsinghua University as a associate professor. His research interests include reconfigurable computing, mobile computing, and SoC design.

Leibo Liu received the BS degree in electronic engineering from Tsinghua University, Beijing, China, in 1999 and the PhD degree from the Institute of Microelectronics, Tsinghua University, in 2004. He now serves as an associate professor with the Institute of Microelectronics, Tsinghua University. His research interests include reconfigurable computing, mobile computing, and VLSI DSP.

Shaojun Wei received the PhD degree from the Faculte Polytechnique de Mons, Belguim, in 1991. He became a professor with the Institute of Microelectronics, Tsinghua University, in 1995. He is a senior member of the Chinese Institute of Electronics (CIE). His main research interests include VLSI SoC design, EDA methodology, and communication ASIC design.