Age-aware Logic and Memory Co-Placement for RRAM-FPGAs

Yuan Xue[†], Chengmo Yang[†] and Jingtong Hu[‡]

[†]University of Delaware, [‡]Oklahoma State University
{xueyuan, chengmo}@udel.edu jthu@okstate.edu *

ABSTRACT

Resistive RAM (RRAM) is a promising non-volatile memory (NVM) device which can replace traditional SRAM as on-chip storage for logic and data in FPGAs. While RRAM outperforms SRAM by offering high scalability, low leakage power, and near-zero power-on delay, RRAM-FPGAs have limited programming cycles, and different writes frequencies of memory and logic blocks make the challenge more severe. To overcome this endurance challenge, we propose an age-aware placement framework for RRAM-FPGAs with uniform reconfigurable logic/memory units. The framework, consisting of a dynamic reconfiguration region allocation algorithm and a logic/memory co-placement algorithm, balances write distributions across the entire FPGA according to logic and memory write frequency differences. The proposed algorithms have been integrated into the VTR synthesis flow. Experiments show that the framework achieves 94.9% write reduction, thus effectively extending RRAM-FPGA programming cycles.

1. INTRODUCTION

Reconfigurable architectures such as field programmable gate arrays (FPGAs) are the core hardware solutions for applications that require customization, acceleration, and adaptive workload switching. However, as most of current FPGAs implement on-chip reconfigurable logic and memory components with SRAM, they face the challenges of scalability, power, and volatility. For example, Xilinx's latest 16nm high-end FPGAs only offer less than 100Mb on-chip block memory [1] due to SRAM scalability limitation. SRAM also has high leakage power [2] which challenges both power-consumption and thermal diffusivity. Furthermore, SRAM is volatile and hence requires time-consuming processes to reload the entire design under intermittent power supplies.

Meanwhile, intensive attempts have been made to implement Non-volatile Memory (NVM) based FPGAs. Compared with SRAM-based FPGAs, NVM-based FPGAs are expected to offer more configurable resources, consume less power, and be more resilient to power supply interruptions. Previous works have demonstrated the possibility of implementing FPGA building blocks with various NVMs. In particular, RRAM-FPGA is a promising reconfigurable platform. RRAM is one of the

commercialized NVMs and it outperforms other popular NVMs in cell density and write speed. For example, an RRAM cell has much smaller area and simpler structure than STT-RAM, while its write speed is comparable with STT-RAM [3].

The endurance of RRAM cells is expected to range from 10^6 to 10^{12} [3]. Such a limited endurance challenges the use of RRAM-FPGAs for applications that require intensive reconfigurations to implement different tasks. The issue is even more serious considering data updates that are much more frequent than logic reconfiguration. When implementing a design, logic blocks are only written once during the configuration stage, while memory blocks are written frequently at runtime. Clearly, the massive writes caused by data updates are the limiting factor of RRAM-FPGA endurance. The worst case occurs when one block is repetitively used as memory blocks across multiple designs and accumulates tremendous write counts. As a concrete example, when FPGAs are used to implement stream applications, toggling frequency in memory cells could reach 1MHz [4]. With an endurance of 10^{12} , an RRAM memory block can only last $10^{12}/10^6=10^6$ seconds, which is approximately 12 days.

To avoid the undesired case of wearing a few blocks out much quicker than the rest, one idea is to rotate the positions of memory blocks across different designs. This type of *wear leveling* is feasible as current FPGA architecture allows configurable logic blocks (CLBs) to be used as either logic or memory [1], thus different CLBs can be used as memory blocks across different designs. If the FPGA offers 5.5×10^6 configurable blocks [1] and a design requires 5.28×10^4 on-chip memory blocks [1], wear leveling could potentially improve the lifetime of RRAM-FPGA by 104 times, that is, from 12 days to 3.4 years.

This paper aims to perform wear leveling in RRAM-FPGAs at the *placement* stage. Specifically, an age-aware placement framework is proposed for RRAM-FPGAs with uniform reconfigurable units that can be used as either logic or memory. The framework is composed of two parts: a *dynamic reconfiguration region allocation algorithm* to balance block across the entire FPGA chip, and a *logic/memory co-placement algorithm* to balance block aging within the selected placement region. Both algorithms are integrated into the VTR synthesis flow [5] for experimental evaluation.

To the best of our knowledge, this is the first work that considers logic and memory co-placement in NVM-based FPGAs. In the rest of this paper, Section 2 reviews background of RRAM-FPGAs and related work. Section 3 describes the proposed co-placement framework, including the algorithms and implementations. Section 4 presents the experimental results, while Section 5 concludes the paper.

2. BACKGROUND

RRAM-FPGA: One representative FPGA architecture that includes configurable logic blocks (CLB), switch boxes (SB),

*This work is supported by NSF CCF-1527464 and 1527506.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

DAC '17, June 18-22, 2017, Austin, TX, USA

© 2017 ACM. ISBN 978-1-4503-4927-7/17/06...\$15.00

DOI: <http://dx.doi.org/10.1145/3061639.3062198>

Figure 1: RRAM-FPGA architecture

and connection blocks (CB), is shown in Figure 1. CLBs are the main functional blocks, SBs are global routing resources, and CBs are responsible for locally connecting CLBs and SBs. Taking Xilinx FPGA as a reference, each CLB contains eight 6-input Lookup tables (LUT6), and each LUT6 consists of two 5-input Lookup tables (LUT5) and two flip-flops. The SBs, CBs, and LUTs can all be implemented with NVM cells. Various types of NV-FPGA architectures have been proposed recently [6, 7, 8]. Among these designs, RRAM-FPGA is one of the most promising platforms. RRAM can be easily integrated with CMOS technique and its fabrication is backend-of-the-line (BEOL) compatible. It is one of the commercialized NVMs. A typical RRAM cell consists of one transistor and one resistor (1T1R) [9]. Its structure is shown in Figure 1(e). It works by changing the resistance across a dielectric solid-state material (a thin film metal oxide such as TiO_2). By applying bias voltage, the resistance can be adjusted to High Resistance State (HRS) and Low Resistance State (LRS) to represent logic '0' and '1', respectively.

Researchers have proposed a number of RRAM-FPGA implementations. In [8], the authors propose RRAM-based CLB with reduced programming complexity and high-performance SB. In [10], 3D stacking RRAMs are utilized to implement different blocks: 1R cells are used to implement LUTs, while 2R cells are used to implement SBs and CBs. In [3], RRAM-based programmable interconnect design is proposed. These works demonstrate the feasibility and advantages of deploying RRAMs in FPGAs.

Logic/memory hybrid structure: In traditional FPGAs, CLBs are used for logic implementation and are configured only once for each design, while on-chip *block RAM* is used as memory and updated frequently at runtime. To offer better design flexibility, Xilinx has developed a type of CLB that can be configured as either logic or memory, known as *distributed RAM*. Compared with block RAM, distributed RAM is more efficient in terms of resource utilization, performance, and power, and additionally has shorter clock-to-output delay and fewer placement restrictions [1]. However, due to on-chip resource limitation, currently only a small portion (around 25%) of CLBs is capable of achieving this flexibility. As RRAM-FPGAs are capable of achieving much higher density, it is feasible to develop a hybrid architecture wherein every CLB can be configured as either logic or memory. This structure not only improves design flexibility and circuit timing, but also prolongs the lifetime of RRAM-FPGAs as it allows different CLBs to be used as memory blocks across different designs.

Related work on age-aware placement: A few works have considered the aging issue in SRAM-FPGAs. In [11], multiple placement solutions are generated for the same design in order to satisfy different aging requirements. In [12], an offline placement algorithm for a fixed set of accelerators is proposed. It partitions on-chip SRAM resources into multiple regions and generates multiple placement solutions for each accelerator. In [13], a table is used to store qualified candidate regions with different aging information. A stress-aware placement is proposed in [4] to balance transistor toggling and improve reliability of SRAM-FPGAs. It pre-defines multiple candidate regions and selects regions for a given design based on region stress information. These works consider SRAM stress and use aging models different from NVMs. Their placements are only for logic and the region partitions are fixed. In contrast, the proposed work targets RRAM-FPGAs and performs logic and memory co-placement to balance writes to different CLBs. It dynamically allocates reconfigurable regions with no pre-determined region boundary.

Recently several NVM-based FPGA placement and routing works have been developed. In [14] and [15], placement algorithms are proposed for NVM-FPGAs to reduce the total number of write counts to CLBs during reconfiguration. A routing algorithm is proposed in [16] to minimize the reconfiguration writes to SBs. These works aim at reducing reconfiguration time and cost, through minimizing the *overall writes* on the CLBs and SBs. They only consider the writes on the logic portion. In contrast, the proposed work considers the skewed write distributions due to both logic and memory writes, aiming at minimizing the *maximum writes* across all CLBs.

3. PROPOSED PLACEMENT FRAMEWORK

3.1 Overview of Co-Placement Framework

As stated before, the high block density of RRAMs makes it feasible to develop a hybrid FPGA architecture wherein each CLB can be configured as either logic or distributed RAM. To manage block age information, each CLB has a write counter and the *age management unit* (AMU) is used to monitor and update these counters. The counter size is 40 bits, capable of recording 10^{12} writes. For high-end FPGAs with 5.5×10^6 CLBs [1], the storage overhead of write counters is around 270MB. These counters can be held in FPGA off-chip storage and regarded as part of the configuration memory. The only difference is that configuration memory is updated by users, while counter memory can only be updated by the AMU. To hide counter update latency, the AMU updates counters not when writing a CLB, but upon completing the current running task and in parallel with task switching. The AMU estimates write counts based on running time and toggling rate of the task. Counter read is performed when synthesizing another design or task task and its latency is also short. Specifically, reading 270MB counter memory takes about 0.8 seconds through a 340MB/s SelectMAP configuration interface [1].

Figure 2 shows the overview of the proposed age-aware co-placement framework and its interaction with the conventional FPGA synthesis flow. Given a design at register-transfer level (RTL), first an offline analysis is performed to identify the total number of logic and memory configurable blocks in the design. Based on design size and current block age information, the *dynamic reconfiguration region allocation* step (detailed in Section 3.2) determines both the region shape and its exact on-chip location. An *age threshold* is used to partition blocks into two types. The dark color blocks are *old blocks* with large write counts, while the light color blocks are *young blocks* with

Figure 2: Age-aware co-placement framework

small write counts. The selected region fulfills two constraints: the region size is large enough to hold the entire design, and there exists a sub-region with only young blocks which is large enough to hold the memory blocks. In Figure 2, a 3×2 yellow region located at the top-left part of the FPGA is the selected region. After determining the region, the *logic and memory co-placement* (detailed in Section 3.3) executes to optimize design timing and reduce routing complexity. Note that only young blocks in the region can be used as distributed RAM.

Once the design is implemented on the FPGA, at runtime the AMU monitors the task and estimates CLB write counts based on task running time and toggling rate. Figure 2 shows that after running the design, two more blocks become old. This updated age distribution will be the new starting point of next round design implementation.

3.2 Dynamic Reconfiguration Region Allocation

In traditional FPGA placement, the placement region is a square (or near-square) region starting from a fixed FPGA corner [5]. The proposed region allocation scheme offers much higher flexibility in terms of region location and region shape, by combining *memory cluster searching* and *placement region selection*. Formally speaking, a placement region on FPGA chip can be characterized as $Region = \{X_{low}, Y_{low}, R, C\}$. X_{low} and Y_{low} respectively refer to the horizontal and vertical starting coordinates, while R and C refer to the row counts and the column counts in the region. Given a design with L logic blocks and M memory blocks, the proposed region allocation scheme designates a rectangular region (square or near square) satisfying two basic criteria: $R \times C \geq L + M$ and $N_{mc} \geq M$, where N_{mc} denoting the size of a memory cluster in the $R \times C$ region. *Memory cluster searching* identifies a memory cluster satisfying the second criterion, while *placement region selection* selects a placement region containing the identified memory cluster and satisfying the first criterion.

3.2.1 Memory Cluster Searching

Memory cluster is defined as a rectangular region, in which all CLBs are young. Although any young CLB on the FPGA can be used as a memory block, the proposed framework only uses memory cluster because splitting memories into small distributed blocks tend to impose high pressure on routing, while a memory cluster is more efficient in terms of storage management and data access. Furthermore, since the memory cluster structure is similar to the block RAM, it can be easily applied

to both designs with distributed RAMs and designs with block RAMs.

The goal of *memory cluster searching* is to find a rectangular young block region whose size N_{mc} is no less than the requested memory size M . Mathematically, this searching problem is very similar to the classical problem of “submatrix with all 1s” [17]. Each CLB can be represented by a 0 or 1 in a binary matrix, with 0 denoting “old” block and 1 denoting “young” block. The problem of finding the largest submatrix with all 1s, which corresponds to the largest young block region, can be solved optimally with dynamic programming (DP) in the complexity of $O(n)$ [17], where n is the total number of CLBs in the FPGA.

Given the 0/1 matrix representation of the RRAM-FPGA, the proposed work employs DP to search for, not the largest all-1 submatrix, but an all-1 submatrix whose size is no less than M . One possibility is to search for the smallest all-1 submatrix whose size is no less than M , which is called the “best fit”. Another possibility is to use the “first fit”—the first all-1 submatrix found in the search process whose size is no less than M . The first fit largely reduces the average searching time since in most cases there is no need to traverse the entire matrix. Table 1 reports the normalized running time of “first fit” and “best fit” as well as their impact on the maximum block writes in the FPGA. It can be seen that on average, DP finds the first fit when traversing only 26% of the entire matrix, while the maximum block writes of the two strategies are almost the same. Another benefit of “first fit” is to get a memory cluster relatively close to FPGA boundaries, since the search process always starts from one corner of the FPGA. This will help reduce I/O timing. Given these benefits, the proposed framework adopts the first fit strategy.

Another constraint in the memory cluster searching is the *aspect ratio* of the candidate region, defined as the larger value of width over height or height over width, as shown below:

$$AR = \max\left\{\frac{width}{height}, \frac{height}{width}\right\} \leq \left\lfloor \frac{L + M}{N_{mc}} \right\rfloor \quad (1)$$

In Equation (1), $L + M$ is the total number of blocks in the design and N_{mc} is the number of blocks in the memory cluster. Constraining the aspect ratio is to avoid narrow and long memory clusters, as those shapes tend to increase the delay for accessing memory blocks, leading to inferior design timing.

If DP terminates but the size of the largest all-1 submatrix is smaller than M , it fails to find a feasible solution. This means that the RRAM-FPGA does not have enough young blocks, and it is necessary to adjust the *age threshold* that differentiates young and old blocks.

A concrete example of the search process is shown in Figure 3(a). The young blocks (in light color) and old blocks (in dark color) are determined based on the current *age threshold*. The memory cluster searching always starts from one corner of the FPGA, and the starting point rotates clockwise upon another search, as denoted by the red dashed arrows. This rotation aims to avoid the case of placing designs repetitively at the same corner. Formally speaking, the starting point of the i^{th} design will be the $(i\%4)^{th}$ corner. The design in Figure 3(a) needs 5 logic blocks and 3 memory blocks. A region

Table 1: Searching strategy comparison

	First fit	Best fit
Running Time	0.262	1
Maximum Block Writes	1.061	1

Figure 3: Dynamic reconfiguration region allocation

with 4 young blocks is selected as the memory cluster, highlighted with the yellow frame. The selected 2×2 region is the first fit. Note that the 4×1 region at the bottom left corner is not selected since its aspect ratio is 4, which is larger than $(L + M)/N_{mc} = (5 + 3)/4 = 2$.

3.2.2 Age Threshold Adjustment

In the proposed framework, a block is considered young if and only if its write count is smaller than the age threshold. Initially, the age threshold is set to the average block age. Throughout FPGA lifetime, the *age threshold* is adjusted under two conditions. First, before implementing a given design, the framework checks the current number of young blocks in the FPGA, denoted as N_{young} , and updates the age threshold if the ratio of young blocks N_{young}/N_{block} is less than $M/(L+M)$, the ratio of young blocks required by the design. Second, if *memory cluster searching* is not able to find a qualified cluster, it is necessary to adjust the age threshold. Every time the age threshold adjusts as following:

$$Threshold_{new} = \frac{Threshold_{old} + \frac{\sum_{i=1}^{N_{block}} Age_i}{N_{block}}}{2} \quad (2)$$

In Equation (2), the current threshold and the current average on-chip block age is averaged to get the new threshold. This means that the threshold increases linearly, and linear increment prevents quick aging of blocks.

3.2.3 Placement Region Selection

Once the memory cluster is determined, the entire design placement region can be decided as well. Obviously, this region is a rectangle containing the identified memory cluster. Furthermore, taking design performance and routability into consideration, square and near-square regions are preferable. Finally, considering I/O access delay, the placement region should be as close to FPGA boundaries as possible.

The proposed framework adopts a two-step process to select the placement region: first an initial region is selected, then if necessary, the region is reshaped to improve its I/O timing.

Given a design containing L logic blocks and M memory blocks and a pre-selected memory cluster, the initial region is the smallest square that can hold the design and cover the memory cluster. The number of rows R and columns C are both set to $\lceil \sqrt{L + M} \rceil$. As an example, the design in Figure 3 has 5 logic blocks and 3 memory blocks, thus the region in 3(b) has $R=C=3$ initially. Since the placement region has to

fully cover the memory cluster, To reduce I/O timing as much as possible, the initial region position is chosen to approach FPGA boundaries to the maximum extent. In the best case, the initial region locates at an FPGA corner.

If the initial placement region does not locate at an FPGA corner, region reshaping may bring it closer to the boundary and hence help improve I/O timing. Reshaping increases (decreases) R and decreases (increases) C by certain amount. To ensure the region shape to be near-square, the following two constraints are imposed:

$$R + C = 2\lceil \sqrt{L + M} \rceil \quad (3)$$

$$R \times C \geq L + M \quad (4)$$

Because $R + C$ remains constant and the initial shape is a square, every reshape will reduce the size of the selected region. Given Inequality (4), the reshaping process terminates when the region is not large enough to hold the design any more, This ensures that the final shape of the selected region is always a near-square rectangle, thus preserving design timing and routability to the maximum extent.

The goal of reshaping is to get as many as possible IO pins closer to FPGA boundaries, which mathematically is equivalent to minimizing the *margin area*, presented as the two blue rectangles S_1 and S_2 in Figure 3(b). As the *reshaping box* represented as $A \times B$ is constant, the *margin area* can be easily computed using the following equation:

$$Area_{margin} = S_1 + S_2 = (A - x) \times y + x \times (B - y) \\ = 2x^2 + (B - A - 4\lceil \sqrt{L + M} \rceil)x + 2A\lceil \sqrt{L + M} \rceil \quad (5)$$

Here, x and y respectively represent the possible values of C and R . y is eliminated from the final expression since from Equation (3), it is known that $y = 2\lceil \sqrt{L + M} \rceil - x$. A, B, L and M are all constants. Therefore, the final placement region can be determined by finding the x in the range given by Equation (4) that minimizes Equation (5).

3.3 Age-aware Co-Placement

Once the placement region is determined, the logic and memory blocks are placed in the select region through the proposed co-placement algorithm. It is based on VTR timing driven placement [5] which first randomly generates a placement solution and then iteratively selects two blocks to swap and accepts the swap decision if it brings timing benefits.

The VTR timing driven placement is only for logic blocks. To account for both logic and memory blocks, the proposed co-placement has three major changes on top of it. First, the co-placement is age-aware and needs to differentiate the old blocks and young blocks. Second, memory blocks are constrained to locate only in the memory cluster, while logic blocks can locate either inside or outside the cluster since the cluster may not be fully used by memory blocks. Third, validity of the swapping decision is also constrained by block type. Logic-to-logic swapping and memory-to-memory swapping are allowed, but logic-to-memory swapping is allowed only if both blocks are located in the memory cluster.

Algorithm 1 shows the proposed age-aware co-placement algorithm in detail. The co-placement starts from a randomly generated initial placement which first places memory blocks in the memory cluster and then places logic blocks on the remaining unused blocks in the placement region. The main body of the algorithm is a two-level nested loop. SR_{limit} is the swapping region which is initialized to the placement area and then updated at the end of the outer *while* loop. In each iteration of the inner *for* loop, two CLBs are randomly selected from

Algorithm 1 Age-aware logic and memory co-placement

Input: $Region = \{X_{low}, Y_{low}, R, C\}$, MC = memory cluster, $Type[R \times C]$ = block type, CLB-level Netlist

Output: Co-placement results

- 1: Place memory blocks in MC , update $Use[R \times C]$;
- 2: Place logic blocks in the unused blocks in $Region$, update $Use[R \times C]$;
- 3: Compute initial timing costs bb_cost and td_cost ;
- 4: Initialize T , initialize swapping region $SR_{limit} \leftarrow region$;
- 5: **while** $Exit() = 0$ **do**
- 6: **for** $Iter \leftarrow 0$ to $Inner_Loop_Limit$ increase by 1 **do**
- 7: Randomly select $Blk1$ and $Blk2$ in SR_{limit} ;
- 8: **if** $(Use[Blk1] \neq Use[Blk2])$ **and**
- $Type[Blk1] \neq Type[Blk2]$ **then**
- 9: Reject;
- 10: **else**
- 11: $P_{new} \leftarrow$ swap $Blk1$ and $Blk2$ in P ;
- 12: $\Delta bb_cost \leftarrow bb_cost(P_{new}) - bb_cost(P)$;
- 13: $\Delta td_cost \leftarrow td_cost(P_{new}) - td_cost(P)$;
- 14: $\Delta Cost \leftarrow \Delta bb_cost + \Delta td_cost$;
- 15: $r \leftarrow$ a random value in $(0, 1)$;
- 16: **if** $\Delta Cost < 0$ or $e < r^{-\frac{\Delta Cost}{T}}$ **then**
- 17: Accept swap; $P \leftarrow P_{new}$;
- 18: Update bb_cost and td_cost ;
- 19: **end if**
- 20: **end if**
- 21: **end for**
- 22: Reduce T , reduce SR_{limit} ;
- 23: **end while**

SR_{limit} at line 7. $Use[i]$ records whether block i is used for memory or logic currently, and $Type[i]$ indicates whether i is inside or outside the memory cluster. Line 8 checks whether it is a logic-to-memory swap and whether the two blocks are not both located in the memory cluster. If both conditions are true, the swap is rejected directly. Otherwise, a cost function that combines wire length bb_cost and timing delay td_cost is used to evaluate each swap decision at lines 12–14. At line 16, a swap is definitely accepted if it is beneficial and probabilistically accepted even if it increases the cost function, based on a simulated annealing process [18] in which T is the temperature of annealing. If the swapping is accepted, the related costs are updated. The *while* loop terminates when the difference between the max and min costs accepted at temperature T is small enough.

4. EXPERIMENTAL EVALUATION

4.1 Experimental Methodology

The proposed co-placement algorithm is incorporated into VTR 7.0 CAD flow [5]. A system simulator is used to perform region allocation and estimate FPGA age distributions. The experimental FPGA architecture is Stratix IV GX, with a 40nm fabrication library and an area-delay model offered by VTR 7.0. Each CLB consists of 8 LUTs, and each LUT has 6 inputs.

The “Proposed” placement is compared to two algorithms. The “Baseline” placement is purely timing-driven, and both the shape and the location of the placement region are fixed without any consideration of age variations or block types. “Optional Region (OR)” placement is similar to a previous placement scheme [11]. It pre-defines the region shape and divides the chip into multiple fixed regions. Every time it selects regions based on average block age without differentiating memory and logic blocks. A set of large Titan benchmark circuits which include both logic and memory blocks [19] are used for evaluation purposes, shown in Table 2.

Table 2: Benchmark Circuits

	Benchmark	Total CLB#	Logic#	Memory#
1	stereo_vision	2886	2751	135
2	neuron	3345	3160	185
3	sparcT1_core	3887	3756	131
4	des90	4808	3950	858
5	SLAM_spheric	6479	6392	87
6	dart	7008	6478	530
7	bitonic_mesh	8503	6846	1657
8	stap_qrd	10200	9600	600
9	cholesky_bdti	16031	15719	312
10	gsm_switch	20545	18835	1710

Figure 4: Maximum write curves

4.2 Block Age Distribution

To obtain block age distributions, a total number of 500 runs are simulated. In each run the 10 benchmarks in Table 2 are configured on the FPGA one by one. When a CLB is used as memory, its write count is assumed to be linearly proportional to both the benchmark running time and the clock rate. Effectiveness of the proposed approach in mitigating unbalanced writes is evaluated from two perspectives: the maximum block write count and the overall block write distribution.

Figure 4 shows the maximum block write counts at each of the 500 execution rounds. The count of “Baseline” increases much faster than the other two schemes. At the 500th round, the baseline count is more than 7.7 times larger than “OR” and 19.8 times larger than “Proposed”. While “OR” and “Proposed” have relatively similar increasing trends, the proposed scheme increments more slowly. Overall, the proposed scheme reduces the maximum block write count by 94.9% and 61.0% compared to “Baseline” and “OR”, respectively. These data clearly confirm the capability of the proposed scheme in prolonging RRAM-FPGA’s lifetime.

Figure 5 shows the overall write distributions on RRAM-FPGA under the three different placement schemes. The write distributions are shown at the granularity of 600×600 partitions, and each sub-region’s write count is represented by the average writes on CLBs within the region. The “Baseline” write distribution in Figure 5(a) is highly skewed since in the original VTR, region allocation always starts from the top-left corner and placement does not take block age into consideration. The write distribution of the “OR” scheme in Figure 5(b) is more-or-less balanced within each region, but the write distribution across regions is unbalanced. This is due to the limitation that OR only offers fixed region partition and one design can never be placed across multiple regions. Finally, Figure 5(c) shows the write distribution of the proposed scheme, which is much more balanced than the other two schemes. The four corners are most frequently used since they are the starting points of memory cluster searching. The tiny rectangles in different colors are the foot prints of memory clusters.

Figure 5: Block write distributions

Figure 6: Critical path delay (in nano-seconds)

Table 3: Placement running time (in seconds)

	Benchmark	VTR placement	Proposed placement	
			region	placement
1	stereo_vision	460.25	0.01	461.36
2	neuron	445.11	0.02	446.64
3	sparcT1_core	426.21	0.01	426.73
4	des90	876.96	0.07	918.34
5	SLAM_spheric	1097.81	0.01	1098.01
6	dart	1001.74	0.04	1008.45
7	bitonic_mesh	1899.17	0.10	2010.43
8	stap_qrd	2536.61	0.03	2546.52
9	cholesky_bdti	1720.29	0.02	1720.97
10	gsm_switch	4419.67	0.07	4456.10

4.3 Design Timing

To evaluate critical path delay, the proposed co-placement algorithm is compared to the baseline VTR placement which separates logic and memory and is purely timing-driven. As stated before, the implementation using hybrid logic/memory blocks is expected to have shorter critical path delay than traditional FPGAs that separate memory and logic. The critical path delay results are shown in Figure 6. It can be seen that the proposed co-placement consistently reduces the critical path delay for all the benchmarks. The average reduction is 7.25% and the maximum reduction is 10.2%.

4.4 Placement Speed

The running time of VTL and the proposed placement algorithms are collected, shown in Table 3. Compared with VTR placement, the proposed placement additionally has a region allocation step. Yet the running time of region allocation is trivial since its complexity is only $O(n)$ and the “first fit” strategy is applied to further speed up the search process. The running time of the proposed placement is close to that of VTR. Although the proposed placement needs to place more blocks (logic and memory block) than VTR (only logic blocks), the block swapping acceptance condition in the proposed placement is stricter. As a result, the speeds of two placements are close. Overall, the proposed placement imposes only 1.4% overhead in placement speed, which is trivial.

5. CONCLUSIONS

In this paper an age-aware co-placement technique is presented for RRAM-FPGAs with blocks that can be used as either logic or memory. The goal is to reduce the maximum write counts to CLBs and mitigate unbalanced write distributions so as to increase RRAM-FPGA programming cycles. A dynamic placement region allocation scheme and an age-aware co-placement algorithm are proposed to alleviate the block aging problem in RRAM-FPGAs. The proposed algorithms are integrated into the VTR synthesis flow. The experiments show that the proposed work reduces maximum block writes significantly, balances the write distributions efficiently, and delivers better design timing with almost no degradation to placement speed compared with previous placement works. Such increased programming cycles and design performance qualify RRAM-FPGAs for next-generation applications that require high adaptivity and reconfigurability.

6. REFERENCES

- [1] *UltraScale Architecture and Product Overview*, Xilinx, 2016.
- [2] I. Kuon and J. Rose, “Measuring the gap between FPGAs and ASICs,” *TCAD*, vol. 26, Feb. 2006.
- [3] J. Cong and B. Xiao, “FPGA-RPI: A novel FPGA architecture with RRAM-based programmable interconnects,” *TVLSI*, vol. 22, Apr. 2014.
- [4] 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 *ICCAD*, 2015, pp. 38–45.
- [5] J. Luu, J. Goeders, M. Wainberg, A. Somerville, T. Yu, K. Nasartschuk, M. Nasr, S. Wang, T. Liu, N. Ahmed, K. B. Kent, J. Anderson, J. Rose, and V. Betz, “VTR 7.0: Next Generation Architecture and CAD System for FPGAs,” *TRTS*, vol. 7, Jun. 2014.
- [6] Y. Chen, J. Zhao, and Y. Xie, “3D-NonFAR: Three-dimensional non-volatile FPGA architecture using phase change memory,” in *ISLPED*, 2010, pp. 55–60.
- [7] W. Zhao, E. Belhaire, C. Chappert, and P. Mazoyer, “Spin transfer torque (STT)-MRAM-based runtime reconfiguration FPGA circuit,” *TECS*, vol. 9, Oct. 2009.
- [8] P. Gaillardon, D. Sacchetto, G. Beneventi, M. Ben Jamaa, L. Perniola, F. Clermidy, I. O’Connor, and G. DeMicheli, “Design and architectural assessment of 3-D resistive memory technologies in FPGAs,” *TNANO*, vol. 12, Jan. 2013.
- [9] H.-S. P. Wong, H.-Y. Lee, S. Yu, Y.-S. Chen, Y. Wu, P.-S. Chen, B. Lee, F. T. Chen, and M.-J. Tsai, “Metal-oxide RRAM,” *Proceedings of the IEEE*, vol. 100, May. 2012.
- [10] Y.-C. Chen, W. Wang, H. Li, and W. Zhang, “Non-volatile 3D stacking RRAM-based FPGA,” in *FPL*, 2012, pp. 367–372.
- [11] E. Stott, J. S. J. Wong, and P. Y. K. Cheung, “Degradation analysis and mitigation in FPGAs,” in *FPL*, 2010, pp. 428–433.
- [12] 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 *ITC*, 2013, pp. 1–10.
- [13] J. Angermeier, D. Ziener, M. Glab, and J. Teich, “Stress-aware module placement on reconfigurable devices,” in *FPL*, 2011, pp. 277–281.
- [14] Y. Xue, P. Cronin, C. Yang, and J. Hu, “Fine-tuning CLB placement to speed up reconfigurations in NVM-based FPGAs,” in *FPL*, 2015, pp. 1–8.
- [15] Y. Xue, P. Cronin, C. Yang, and J. Hu, “Non-volatile memories in FPGAs: Exploiting logic similarity to accelerate reconfiguration and increase programming cycles,” in *VLSI-SoC*, Oct. 2015.
- [16] Y. Xue, P. Cronin, C. Yang, and J. Hu, “Routing path reuse maximization for efficient NV-FPGA reconfiguration,” in *ASP-DAC*, 2016, pp. 360–365.
- [17] A. Naamad, D. Lee, and W.-L. Hsu, “On the maximum empty rectangle problem,” *Discrete Applied Mathematics*, vol. 8, Jul. 1984.
- [18] M. Huang, F. Romeo, and A. Sangiovanni-Vincentelli, “An efficient general cooling schedule for simulated annealing,” in *ICCAD*, 1986, pp. 381–384.
- [19] K. E. Murray, S. Whitty, S. Liu, J. Luu, and V. Betz, “Titan: Enabling large and complex benchmarks in academic CAD,” in *FPL*, 2013, pp. 1–8.