Leveraging Emerging Nonvolatile Memory in High-Level Synthesis with Loop Transformations

Shuangchen Li * Ang Li† Yuan Zhe† Yongpan Liu† Peng Li‡
Guangyu Sun§ Yu Wang† Huazhong Yang† Yuan Xie*
University of California, Santa Barbara* Tsinghua University†
University of California, Los Angeles‡ Perking University§
{shuangchenli, yuanxie}@ece.ucsb.edu* ypliu@tsinghua.edu.cn†

***Abstract*—To mitigate the "Power Wall" challenges for both mobile devices and data centers, accelerator-rich architecture with normally-off mode has been intensively studied recently. Power/energy optimization in high-level synthesis for accelerator design is critical for such accelerator-rich architecture. The emerging nonvolatile memory (NVM), offers many benefits such as ultra-low leakage power, high density, and instant power-on/off, and therefore is a promising alternative for the hardware accelerator design to achieve further power reduction. However, such NVM suffers from large write energy and latency, which brings new challenges for the buffer allocation in the custom accelerator design. This paper presents the first framework that optimizes NVM allocation in high-level synthesis for custom accelerator design, considering loop transformations. It solves the loop transformation, buffer allocation, and buffer type selection to minimize the memory power consumption, while under area, bandwidth, and performance constraints. This paper formulates the optimization problem, and solves it with a problem-specific designed stimulated annealing solution. Experiments demonstrate 32% extra power reduction compared with the previous method without optimizing loop transformations.**

I. Introduction

Power/energy optimization is one of the critical design goals [1], [2]. Power gating is one of the most effective techniques to reduce the power consumption of VLSI chips. The logic part's power can be completely cut off when it is idle. However, the memory part cannot, so as to keep the data retention. Furthermore, not only do mobile chips apply power gating for low power purpose, but recent studies revealed that the duty cycle of high-performance chips also decrease significantly in order to meet the thermal budget. For example, prior work projected that the amount of *Dark Silicon* may reach up to 50-80% at 8 nm technology nodes [3]. In such normally-off systems, the power consumption of memory will dominate the total chip power consumption. Consequently, designing energy-efficient memory system is critical.

The emerging nonvolatile memory (NVM) technology brings a revolution for the normally-off systems. It offers ultra-low leakage power, high density, instant sleep/wakeup, and data persistence property. In addition, Spin-Transfer Torque RAM (STT-RAM) is an emerging CMOS-compatible non-volatile memory technology based on Magnetic Tunnel Junctions (MTJ) as a memory bit [4]. Consequently, recent work studied how to use such NVM in microprocessors to achieve low power design in registers [5], cache [6] or scratch-pad memory [7], and main memory [8], with chip prototyping on fabricated nonvolatile processors [9], [10]. However, none of them leverage NVM for the accelerator design in accelerator-rich architectures, which becomes even more important and widely adopted in order to pursue energy-efficient computation [11], [12].

Adopting such NVM technology in accelerator designs is non-trivial, with challenges from both NVM devices and accelerator architectures. Since NVMs have large write energy [13], asymmetric read/write operations [14], and limited endurance [15], special designs are needed. Furthermore, the accelerators have more irregular variance, different from the rather regular microprocessor architecture. For example, unlike cache or starch-pad memory, buffers in accelerators are distributed and work in parallel. The high-level synthesis (HLS) [16] is a promising solution to handle such complexities.

Plenty of work had explored memory optimization in HLS. Zuo *et al.* [17] explored throughput-aware memory partition via loop transformations. Cong *et al.* investigated the buffer hierarchy and allocation with loop transformations [18], [19], [20] in HLS. The objective was to minimize buffer size while meeting the bandwidth constraint. However, none of them deploy NVM for power optimization. Most recently, Li *et.al* allocated hybrid NVM buffers to minimize power [21], but loop transformation was not considered, which seriously limited the optimization space.

Given the C program to be synthesized and the area, bandwidth, and performance constraints, this paper minimizes the power consumption by solving the optimal loop transformation, buffer allocation, and buffer type (NVM/SRAM) selection. *To our best knowledge, this is the first HLS framework leveraging NVM with loop transformations [22]. The specific contributions include,*

- Proposing a model with Grouped Dynamic Data Reuse Graph (GD-DRG) and Hybrid Allocation Vector (HAV), in order to capture the data reuse feature under different loop transformations and different buffer types (for NVM especially),
- Formulating the hybrid memory allocation as integer nonlinear programming, and designing a scalable simulated annealing algorithm with self-cooling scheme and prior knowledge-based neighbor selection,
- Validating the proposed framework with seven real benchmarks, which shows 48% extra power saving on average by considering loop transformations, and
- Finding that traditional minimal buffer sizing does not necessarily guarantee a minimal power design if NVMs are considered, and the proposed hybrid memory system is especially good for wide bandwidth applications.

II. Overview

This section introduces the HLS framework, and then demonstrates the importance of power/energy optimization for memory designs. The challenges of memory design using such NVM (such as the latency/energy overhead associated with the write operations) are highlighted, followed by a motivating example.

A. HLS Framework

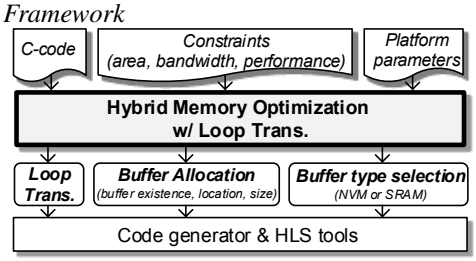


Fig. 1: The HLS Framework Flow

The HLS framework optimizes the hybrid on-chip buffer allocation with loop transformations. The inputs are a synthesizable C program, design constraints (on-chip memory area, off-chip memory bandwidth, and performance), and platform parameters (technology node and NVM characteristics). The design knobs consist of loop transformations, buffer allocations (whether and where to insert the buffer, and the buffer size), and buffer type (NVM or SRAM) selections. The power optimized results are translated into the modified C-code (Fig. 5(c) is an example), based on which a commercial HLS tool is able to generate the desired accelerator.

B. Impact of Memory's Power Consumption

Fig. 2 shows the power consumption breakdown for nine typical accelerator design. On average, memories take more than 50% of the total power consumption, so utilizing NVM's leakage power advantages is promising. Moreover, accelerators usually work in normally-off mode for either power or thermal consideration. When the accelerator is idle, the logic part is able to be power-gated, but memory part is not, due to the requirement of data retention. Therefore, in Fig. 3, as the duty cycle decreases, the memory power tends to dominate. By utilizing NVM, which supports both power-gating and data retention, accelerators are able to benefit more from normally-off mode. Therefore, utilizing NVM is significant for total accelerators power reduction.

C. Impact of NVM

Table I compares NVM with SRAM [23]. It shows that the NVM has smaller leakage (Lkg) and area (A), but larger write energy (E^w) and latency (L^w). Fig. 4 (the right part) shows power breakdown of both a NVM and a SRAM. In the SRAM, the leakage dominates (49%). However, write energy dominates (66%) in the NVM. Therefore, proper optimizations are needed to utilize the advantages of both NVM/SRAM and avoid their weaknesses.

	E^r (pJ)	E^w (pJ)	L^r (ns)	L^w (ns)	Lkg (μW)	A (μm^2)
SRAM	6.479	3.823	6.744	6.742	21.03	116697
STT-RAM	7.92	27.799	7.424	11.233	4.24	36027

TABLE I: NVM v.s. SRAM (32-nm, 64KB)

According to the above analysis, both buffer sizes and access patterns lead to various design choices on NVM/SRAM. Fig. 4 (the left part) shows the energy consumption trend with different buffer size (x-axis) and access patterns (legends). NVMs and SRAMs have very different energy consumption curves. SRAM's energy (solid lines) barely changes with memory access patterns but is sensitive to the buffer size. However, the NVM (dotted lines) is just the opposite. Accordingly, the boundary of NVM/SRAM selection (black crosses) changes from 64KB to 256KB when the memory access changes from 1W6R (one write with six read) to 6W1R. Therefore, loop transformations, which change both buffer sizes and data access patterns, add new dimensions to memory optimization in HLS. An example below shows the loop transformation's impact.

D. Motivating Example

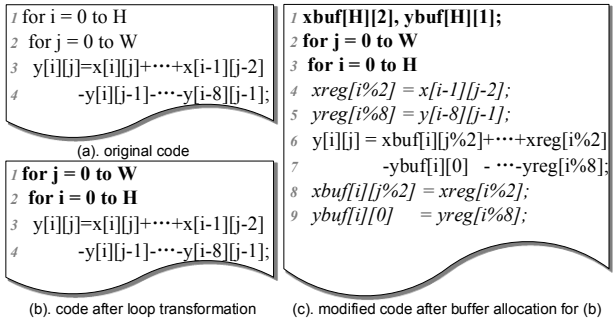


Fig. 5: Motivation Example C-code

The motivation example is a 2D recursive digital filter, which is used for spectral factorization [24]. Fig. 5 shows the original code (simplified for a clearer look), the code after a loop transformation (permutation), and the modified code with buffer allocations. Fig. 6 shows the generated HLS hardware and Table II shows the results of various optimizations.

As we observe, the on-chip buffer allocation is able to save the data access power significantly. Row 2 demonstrates that almost 37% power is saved if memory optimization is applied. Moreover, the hybrid memory provides extra power savings in Row 4. It reduces extra 5% power compared with the traditional SRAM-only solution (Row 2) and even with NVM-only solution (Row 3).

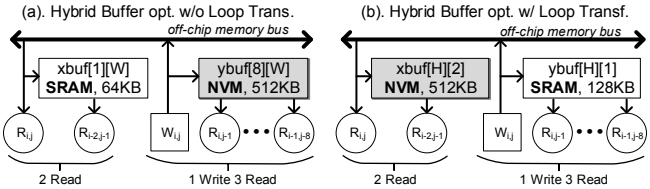


Fig. 6: Motivation Example Hardware

Opt. strategy		Memory allocation	Power (μW)
w/o. optimization		no buffer allocated	1670
w/o Loop Trans.	SRAM-only	Fig. 6(a) but SRAM-only	1053
	NVM-only	Fig. 6(a) but NVM-only	1015
	hybrid	Fig. 6(a) hybrid	968
w/ Loop Trans.	*hybrid*	*Fig. 6(b), this work*	*896*

TABLE II: Motivation Example Result

In addition, more improvements are achieved if loop transformations are considered in Row 5, because it tunes both buffer size and access patterns. As Fig. 4 pointed out, they have a non-trivial impact on the buffer type selection. Fig. 6(b)

Fig. 2: Acc. power breakdown

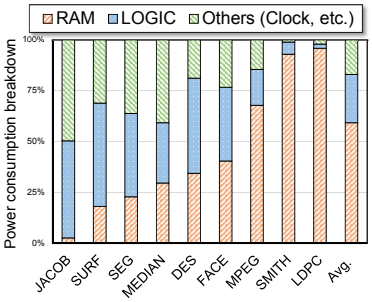


Fig. 3: Acc. with normally-off

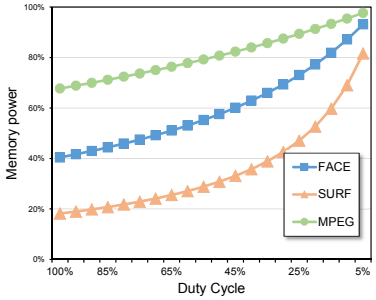
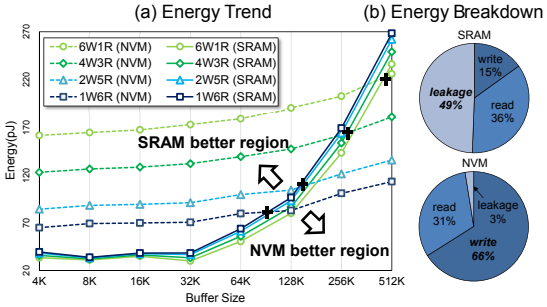


Fig. 4: Memory Energy Trend



shows the optimal buffer allocation for the program after loop transformation (C-code in Fig. 5(b)). It saves power by enlarging the buffer for array x (who has fewer accesses) but shrinking the buffer for array y (who has more accesses).

III. Modeling

In this section, we introduce the concepts of polyhedral model, data reuse and hybrid memory architecture, which correspond to the design choices needs optimization in Fig. 1.

A. Polyhedral Model

The *polyhedral model* [25] is a flexible and expressive representation for loop nests with statically predictable control flow. To save space, detailed modeling is shown in [26]. $\mathcal{D}_S$ denotes the iteration domain of statement S. Given two statements R and S and a data dependence $R \rightarrow S$, a dependence polyhedron $\mathcal{D}_{R,S}$ contains all pairs of dependent instances $\langle \vec{x}_R, \vec{x}_S \rangle$. To construct a program optimization, we build a collection of schedules $\vec{\Theta}$=$\{\Theta^{S_1}, \ldots, \Theta^{S_n}\}$, that is a list of the statement scheduling function for each statement in the program, so that for all dependent instances the producer instance is scheduled before the consumer.

B. Data Reuse

To bridge the gap between the extraordinary computing power provided by hardware accelerators and the limited memory bandwidth, on-chip scratchpad memories are used to buffer intermediate data. Augmenting the traditional data reuse graph [19], [27] with group dependence information, a novel data reuse representation, the Grouped Dynamic-Data Reuse Graph (GD-DRG) is proposed in this paper to reduce the complexity combining loop transformation with data reuse. Fig. 7(a) shows the GD-DRG example for the C code in Fig. 5(a).

An array access in the original code is represented by a vertex v_k^g in graph $G(v, e)$, where g is the group index and k is the node index within the group. The order of the vertices shows the data dependency between array accesses and is able to be changed dynamically with loop transformation. Considering the semantic, the first vertex in any group v_0^g must correspond to either an array write operation or a new array. Each vertex contains two attributes: W_k^g denotes write/read, and AC_k^g denotes the access count[1].

Edges, denoted as e_k^g, only exist between v_k^g and v_{k+1}^g, i.e., neighbor vertices within the same group. An edge represents a possible buffer allocation in hardware. Each edge has a weight BS_k^g representing the buffer size[1].AC_k^g, BS_k^g are calculated from $\vec{\Theta}$ for each loop transformations [19], [28], [29].

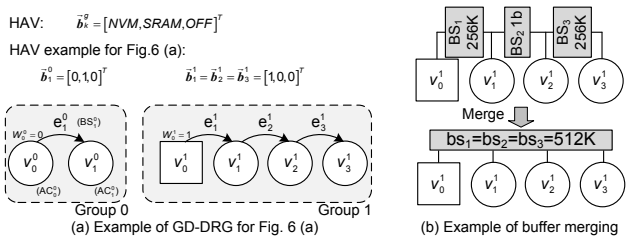


Fig. 7: The Example of GD-DRG and HAV

C. Hybrid Memory

To tackle the challenge of multiple memory types, the Hybrid Allocation Vector (HAV) is proposed. $\vec{\mathbf{b}}_k^g$ is a HAV for the possible buffer corresponding to e_k^g in the GD-DRG. The vector $\vec{\mathbf{b}}_k^g = [b_{\text{NVM}}, b_{\text{SRAM}}, b_{\text{OFF}}]^\intercal$ represents the buffer is NVM or SRAM or not exist but using off-chip access. $||\vec{\mathbf{b}}_k^g|| = 1$ always holds. Fig. 7(a) shows the example HAV to represent buffer allocation in Fig. 6(a). When adjacent buffers e_k^g and e_{k+1}^g have the same HAV, the buffers need to be merged. Fig. 7(b) shows an example. bs_k^g denotes the merged buffer's size, which is calculated as follows,

$$bs_k^g = \sum_{i \in U_g} BS_i^g, \qquad U_g = [l, u] \subset \mathbb{Z}^+, \tag{1}$$

$$\text{where } \vec{\mathbf{b}}_{l-1}^{g\intercal}\vec{\mathbf{b}}_l^g = \vec{\mathbf{b}}_u^{g\intercal}\vec{\mathbf{b}}_{u+1}^g = 0, \text{ and } \forall i, j \in U_g, \vec{\mathbf{b}}_i^{g\intercal}\vec{\mathbf{b}}_j^g = 1.$$

For parameters, the write energy of bs-size NVM, SRAM, and off-chip memory are denoted as $[E_N^W(bs), E_S^W(bs), E_O^W]$. Similar, the read energy, latency, area, and leakage power of bs-size NVM, SRAM, and the off-chip memory are denoted in Table III.

Notation	Description
$\{E_N^W\}/\{E_N^R\}$, $\{Lkg_N\}$ $\{A_N\}$, $\{L_N^W\}/\{L_N^R\}$	sets for variant-size NVM's w/r energy (pJ), leakage (μW), area (μm^2), and latency (ns)
$\{E_S^W\}/\{E_S^R\}$, $\{Lkg_S\}$ $\{A_S\}$, $\{L_S^W\}/\{L_S^R\}$	sets for variant-size SRAM's w/r energy (pJ), leakage (μW), area (μm^2), and latency (ns)
$\{E_O^W\}/\{E_O^R\}$, $\{L_O^W\}/\{L_O^R\}$	off-chip memory 's w/r energy and latency

TABLE III: Memory Parameter Definition

IV. Problem Formulation

The NVM memory allocation with loop transformation's problem is formulated: Solve the valid loop transformation $\vec{\Theta}^*$, the buffer allocation and type HAV $\vec{\mathbf{b}}_k^g$, in order to

minimize memory power, while meets the area, bandwidth, and performance constraints. The formulation is listed, as follows,

$$\min .\{p_{dyn} + p_{lkg}\} \tag{2}$$
$$\text{s.t. } \textit{area constraint:} \quad a < A_{\max} \tag{2a}$$
$$\textit{bandwidth constraint:} \quad bw < BW_{\max} \tag{2b}$$
$$\textit{performance constraint:} \quad clk > CLK_{\min} \tag{2c}$$
$$\textit{dependency constraint:} \quad \forall \langle \vec{i}_n^{S_k}, \vec{i}_m^{S_l} \rangle \in \mathcal{D}_{S_k,S_l}, \; \Theta^{*S_k}\left(\vec{i}_n^{S_k}\right) \prec \Theta^{*S_l}\left(\vec{i}_m^{S_l}\right). \tag{2d}$$

Below, details about power p_{dyn} and p_{lkg}, area a, bandwidth bw, peak clock frequency clk, and data dependency are presented.

Power Calculations Equ. (3) calculates the dynamic power consumption p_{dyn} by summing up every buffer's power in the GD-DRG. AC_k^g denotes the access count, and dyn_k^g is the energy per access. The energy is then divided by execution time $T(bw, clk)$, which is a function of bandwidth and clock rate.

$$p_{dyn} = \sum_{\forall g}\sum_{\forall k} AC_k^g \cdot dyn_k^g / T(bw, clk) \tag{3}$$
$$dyn_0^g = [W_0^g, \widetilde{W}_0^g][E_O^W, E_O^R]^\intercal + [E_N^W(bs_1^g), E_S^W(bs_1^g), 0]\vec{\mathbf{b}}_1^g \tag{3a}$$
$$dyn_k^g = [E_N^R(bs_k^g), E_S^R(bs_k^g), E_O^R]\vec{\mathbf{b}}_k^g + [0,0,1]\vec{\mathbf{b}}_k^g[E_N^W(bs_{k+1}^g), E_S^W(bs_{k+1}^g), 0]\vec{\mathbf{b}}_{k+1}^g \tag{3b}$$

For the first array access dyn_0^g in a group of GD-DRE, it has to write/read off-chip memory (first part of Equ. (3a)), and then if its following read access shares buffer with it, it has to write this buffer. For other following read access dyn_k^g, the NVM/SRAM/off-chip access energy is counted according to the HAV $\vec{\mathbf{b}}_k^g$. In addition, if this access is off-chip access and it shares buffer with its following accesses, it has to write the buffer.

The leakage power p_{lkg} is the summary of each buffer's leakage according to the HAV, as follows,

$$p_{lkg} = \sum_{\forall g}\sum_{\forall i} [Lkg_N(bs_i^g), Lkg_S(bs_i^g), 0]\vec{\mathbf{b}}_i^g. \tag{4}$$

Constraints The total on-chip memory a can be calculated using Equ. (5), similar with the calculation of leakage power p_{lkg}.

$$a = \sum_{\forall g}\sum_{\forall i} [A_N(bs_i^g), A_S(bs_i^g), 0]\vec{\mathbf{b}}_i^g. \tag{5}$$

For the bandwidth constraint, the total access count for off-chip memory is calculated according to the HAV for each node in the GD-DRG, as follows,

$$bw = \sum_{\forall g}\sum_{\forall k} [0,0,1]\vec{\mathbf{b}}_k^g \cdot AC_k^g. \tag{6}$$

For the performance constraint, it sets the lower boundary for the clock rate. The buffer's longest latency is calculated as follows,

$$clk = \max_{\forall g,k}\{[L_N(bs_k^g), L_S(bs_k^g), 0]\vec{b}_k^g\}^{-1} \tag{7}$$

For the dependency constraint, it ensures the loop transformation $\vec{\mathbf{\Theta}}^*$ is valid. $\mathcal{D}_{S_k,S_l}$ is the dependence polyhedron. Therefore, whatever the code changes, it has to execute $\vec{i}_n^{S_k}$ before $\vec{i}_m^{S_l}$, which is captured by Equ. (2d).

Algorithm 1: Simulated Annealing algorithm

```
1 initializations;
2 while temperature t > t_threshold do
3     cooling the temperature, t' = cooling(t);
4     while not exceed max. iteration time do
5         get a neighbor state, s' = neighbor(s);
6         if E(s')<E(s) or P(E(s'),E(s),t)>p_threshold then
              s = s', t = t' and break ;
7     end
8 end
```

V. A Simulated Annealing Based Solution

In this section, a heuristic solution is proposed based on Simulated Annealing (SA).

A. Mapping Problem to SA Algorithm

A general SA algorithm denotes the current state as s, the temperature as t, the energy as $E(s)$, and the probability transiting from s to a neighbor state s' as P. The algorithm is briefly described in Algorithm 1. At every iteration, it transfers to a more stable state ($E(s') < E(s)$). Nevertheless, it could also transfer to a less stable state at a transition probability. As the annealing temperature cools, the transition probability (exponential with temperature t) also decreases. Table IV shows how to map the particular NVM memory allocation into the SA algorithm. Besides the mapping table, there are two essential factors to be considered: the cooling scheme (Line 3) and neighbor state selection (Line 5).

SA algo.	NVM allocation with loop transformations
initialization	the original Loop Trans. with no buffer allocated
state s	Loop Trans. result $\vec{\mathbf{\Theta}}^*$, allocation result HAVs $\vec{\mathbf{b}}_k^g$
energy $E(s)$	the memory power p by Equ. (2) at state s
temperature t	calculated by Equ. (8)

TABLE IV: Mapping the Problem to the SA Algorithm

B. Cooling Scheme and Neighbor Selection

A specific "automatic" cooling scheme `cooling(t)` is proposed for the NVM allocation in HLS. Cooling scheme should be neither too fast nor too slow, which leads to either sub-optimal or unnecessarily long running time. The proposed method solves the NVM allocation by taking the inherent parameters as temperature. It cools down automatically, which turns out to be an effective cooling scheme regardless of various inputs, as follows,

$$t(s) = \frac{A_{\max} - a}{A_{\max}} + \alpha\frac{bw - BW_{\max}}{BW_{\max}} + \beta\frac{\#\{\vec{\mathbf{b}}_k^g = [0,0,1]^\intercal\}}{\#\{\vec{\mathbf{b}}_k^g\}}. \tag{8}$$

In the equation, the first two parts are the constraint margin of area and bandwidth. The third part is the number of off-chip buffers ($\#\{\vec{\mathbf{b}}_k^g = [0,0,1]^\intercal\}$) divided by the total possible buffer number. It denotes the percentage of buffer not yet allocated. α and β are user-defined weights. A design choice "near" the optimal result is most likely with a small area and bandwidth margin and a high percentage of buffer allocated, which leads to a lower temperature. Therefore, the temperature automatically cools, while the problem converges to a solution. Experimental results in Section 6 demonstrate this cooling scheme is good at achieving solutions with both high quality and efficiency.

Neighbor state selection `neighbor(s)` needs specific designs to solve the NVM allocation problem. A neighborhood is defined as changing the loop transformation or one HAV ($\vec{\mathbf{b}}_k^g$). The selection should be random. However, in order to

speed up the convergence, it takes both history and prior knowledge by following three rules: (i) It applies the tabu search and explored states are banned. (ii) Adding small buffers has a higher priority, because small buffers usually provide considerable power reduction with small overheads. They are most likely to be included in the allocation. (iii) Removing or changing buffers within a connected and merged buffer has a lower priority, because it breaks the merged buffer. Separated buffers are usually less power efficient than a merged buffer.

VI. Experimental Results

This section starts with the experiment environment set up. The experiment data are presented to show both the effectiveness and superiority of this work, the constraints' impacts, and the comprehensive analysis of the algorithm.

A. Experimental Setup

The parameters of both NVM (STT-RAM) and SRAM come from NVSim [23], with 32nm technique node. Off-chip memory's parameter comes from a low power DRAM [30]. The experiment uses seven benchmarks. Detailed setup for experiment is shown in [26]. In order to focus on the contribution of the proposed optimization and to make fair comparisons among different applications, the experiments below focus on the memory part power consumption. The experiments set duty cycle as 100% during the evaluation for fairness, and the power saving is more significant in a real scenario with reasonable duty cycle.

B. Effectiveness of Proposed Method

Table V shows the effectiveness of the proposed method. The table compares the power consumptions with and without optimization (no buffer allocated). It concludes that by applying the proposed NVM allocation optimization, it achieves up to 91.1% power reduction and an average reduction of 64.5%. Buffer allocation optimization saves a large amount of power of off-chip data accesses.

	Jacobi	Seidel	Fdtd	Gemver	Sobel	Rdf	CNN	Avg.
w/o. opt.	4175	1676	975	1460	2520	1670	6997	–
this work	1810	493	514	478	707	897	620	–
power sav.	56.6%	70.6%	47.3%	67.3%	71.9%	46.3%	91.1%	**64.5%**

TABLE V: Buffer Allocation's Power Saving (μW)

Fig. 8 shows the superiority of the proposed method. First, we compare this work with the method without optimizing loop transformation, marked as *w/o. Loop Trans*. [21]. It shows that the proposed method is 62.5% better in *Gemver* benchmark, and is 32.4% better on average. The loop transformation provides a powerful design knob. It is able to shrink buffer sizes and tune access patterns, both of which potentially saves more power. Therefore, it is necessary to consider loop transformation during optimization.

In addition, Fig. 8 also compares this work the method target at minimize total buffer size with Loop Trans. and hybrid NVM, marked by *min.buffer size* [19]. The proposed method is up to 29.6% better in *Fdtd* benchmark, and 16% better on average.

The power breakdown in Fig. 9 provide further insight of the superiority of this work. The *min.buffer size* is bad, because

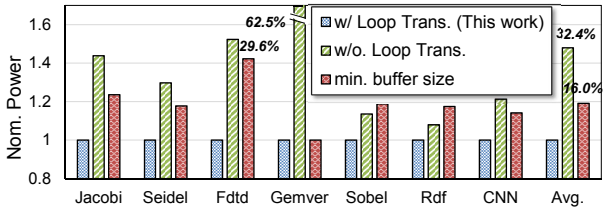


Fig. 8: Experiment Result for Seven Benchmarks

it may end up with a huge buffer with several tiny buffers, whose total buffer size is small, but the leakage is worse. The write and read are not balanced to achieve power efficiency, either. The *w/o. Loop Trans.* is better on balancing the power consumptions but limited by design space, which is the not best, either.

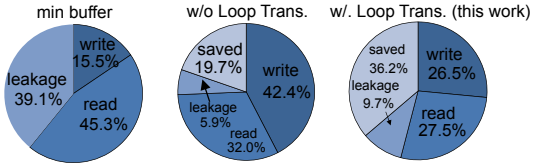


Fig. 9: On-chip Power Breakdown for Rdf[2]

C. Impacts of Different Constraints

Fig. 10 shows the impact of area constraint on the NVM allocation results. There are two observations. First, it shows the tradeoff between area and power. The proposed method is represented by the lower curve. As the area constraint shrinks along the x-axis, the on-chip buffers split and turn smaller. Moreover, it tends to prefer the NVM buffer, due to its advantage of small area. Consequently, the power consumption gets larger. Second, it shows the proposed method's advantage over *w/o. Loop Trans.* (shown by the upper curve). It is even worse when the area constraint is tight. This is because missing a design knobs, the design space is limited, which leads to more power waste. Even worse, it fails to get any solution when area constraint reaches 7000.

Fig. 11 shows the bandwidth constraint's impact. The lower curve shows the proposed method's result. The upper curve shows the result from *min.buffer size* method. It shows that the *min.buffer size* performs even better with high performance, wide bandwidth memory systems.

Perf. Contr.	Loop Trans.	Buffers	Power (uW)
20ns	permutation	512k NVM, 128k SRAM	896.5
15ns	permutation	512k SRAM, 128k SRAM	996.8
10ns	permutation	128k SRAM	984.1
6.8ns	original	64k SRAM	1506.7

TABLE VI: Performance Constraint's Impact on RDF

Table VI shows performance (clock period) constraint's impact. It shows the tradeoff between performance and power. When the performance constraint shrinks from 20ns to 15ns, SRAM is used to replace NVM, in order to avoid 512KB NVM's large write latency. When the constraint further shrinks, buffers with large latency are abandoned. As a result, it sacrifices power reduction to meet the performance constraint. When the constraint reaches 6.8ns, the loop transformation result also changes, in order to provide smaller and hence faster buffer, which still saves power.

D. SA Algorithm Discussion

The proposed SA algorithm is both accurate and efficient. Table VII shows the running time of the proposed algorithm,

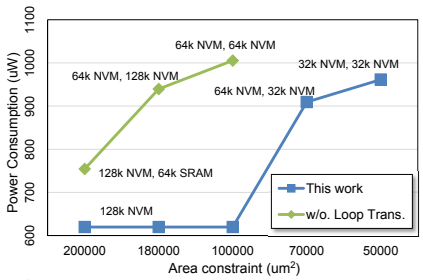


Fig. 10: Area Constr.'s Impact (CNN)

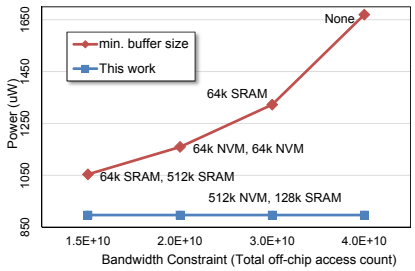


Fig. 11: Bandwidth Const.'s Impact (RDF)

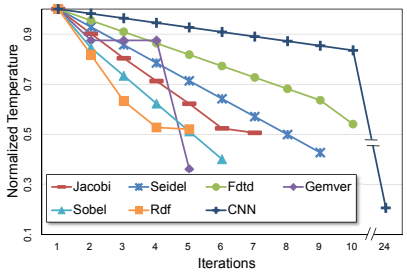


Fig. 12: Cooling in the SA algorithm

	Jacobi	Seidel	Fdtd	Gemver	Sobel	Rdf	CNN
running time (μs)	198	383	506	102	104	67	420
speedup to NILP	$1x10^3$	$8x10^4$	∞	$3x10^3$	$2x10^2$	$1x10^3$	∞
speedup to Brute Force	$4x10^6$	$1x10^6$	$1x10^6$	$4x10^4$	$3x10^6$	$2x10^5$	∞

TABLE VII: Running Time (all methods with same results)

compared with the Brute Force method and the non-linear programming (NILP) method [31]. The proposed algorithm returns the same result as the other two, which are optimal. In addition, both Brute Force and NILP are not scalable, which leads to failure in the larger benchmarks. On the other hand, the SA algorithm leads to a 3-6 orders of magnitude speedup compared with the other two methods.

Fig. 12 shows the temperature cooling procedure under different benchmarks. With specific designed cooling scheme, all the benchmarks converge within 10 iterations (except for CNN with 24 iterations).

The correctness and efficiency owe to the special cooling scheme. In order to provide the insights on how to select cooling parameters in SA algorithm, we evaluate the different cooling parameters to compare their convergence speed and the quality of the solution. Both a linear and an exponential cooling function are used as comparisons. Table VIII shows the running time and errors. *linear 0.1* means the linear cooling sets its step as 0.1 and *exp 0.9* means the factor of the exponential cooling scheme is 0.9.

cooling scheme	Faster but with Error			Slower	
	exp 0.9	linear 0.3	linear 0.25	linear 0.1	exp 0.95
running time	-63.04%	-62.49%	-53.36%	2.31%	28.60%
error	27.14%	113.08%	48.69%	0.00%	0.00%

TABLE VIII: Analysis of Cooling Scheme on CNN

VII. Conclusion

Accelerator-rich architecture working in normally-off mode has been studied for low power design. In such accelerator-rich architecture, the power consumption of memory is a major contributor of the total power consumption. The emerging NVM offers a promising solution, with design challenges associated with the write overheads. This paper proposes to leverage NVM in accelerator design with a proposed HLS framework for hybrid memory allocation with loop transformations. The experimental result shows 32% extra power saving compared to prior work.

References

[1] D. S. Holmes *et al.*, "Energy-efficient superconducting computingłpower budgets and requirements," *TAS*, vol. 23, no. 3, p. 1701610, 2013.

[2] K. Sekar, "Power and thermal challenges in mobile devices," in *MobiCom*. ACM, 2013, pp. 363–368.

[3] H. Esmaeilzadeh *et al.*, "Dark silicon and the end of multicore scaling," in *ISCA*. IEEE, 2011, pp. 365–376.

[4] C. Lin *et al.*, "45nm low power CMOS logic compatible embedded STT MRAM utilizing a reverse-connection 1T/1MTJ cell," in *IEDM*, 2009.

[5] Y. Wang *et al.*, "Pacc: A parallel compare and compress codec for area reduction in nonvolatile processors," *TVLSI*, pp. 1491–1505, 2014.

[6] G. Sun *et al.*, "A hybrid solid-state storage architecture for the performance, energy consumption, and lifetime improvement," in *HPCA*, 2010, pp. 1–12.

[7] P. Wang, "Designing scratchpad memory architecture with emerging stt-ram memory technologies," in *ISCAS*, 2013, pp. 1244–1247.

[8] C. Xu *et al.*, "Understanding the trade-offs in multi-level cell reram memory design," in *DAC*. IEEE, 2013, pp. 1–6.

[9] Y. Chen *et al.*, "Processor caches built using multi-level spin-transfer torque ram cells," in *ISLPED*. IEEE, 2011, pp. 73–78.

[10] Y. Wang, Y. Liu *et al.*, "A 3us wake-up time nonvolatile processor based on ferroelectric flip-flops," in *ESSCIRC*, 2012, pp. 149–152.

[11] J. Cong *et al.*, "Architecture support for accelerator-rich cmps," in *DAC*. ACM, 2012, pp. 843–849.

[12] N. Goulding-Hotta *et al.*, "The greendroid mobile application processor: An architecture for silicon's dark future," *IEEE Micro*, vol. 31, no. 2, pp. 86–95, 2011.

[13] X. Dong, X. Wu *et al.*, "Circuit and microarchitecture evaluation of 3d stacking magnetic ram (mram) as a universal memory replacement," in *DAC*, 2008, pp. 554–559.

[14] G. Sun *et al.*, "A novel architecture of the 3d stacked mram l2 cache for cmps," in *HPCA*, 2009, pp. 239–249.

[15] J. Wang, "i2wap: Improving non-volatile cache lifetime by reducing inter-and intra-set write variations," in *HPCA*, 2013, pp. 234–245.

[16] J. Cong *et al.*, "High-level synthesis for fpgas: From prototyping to deployment," *TCAD*, vol. 30, no. 4, pp. 473–491, 2011.

[17] W. Zuo, *et al.*, "Improving polyhedral code generation for high-level synthesis," in *CODES+ISSS*, 2013, p. 15.

[18] J. Cong *et al.*, "Combined loop transformation and hierarchy allocation for data reuse optimization," in *ICCAD*, 2011, pp. 185–192.

[19] J. Cong, P. Zhang *et al.*, "Optimizing memory hierarchy allocation with loop transformations for high-level synthesis," in *DAC*.

[20] L.-N. Pouchet, P. Zhang *et al.*, "Polyhedral-based data reuse optimization for configurable computing," in *FPGA*, 2013, pp. 29–38.

[21] S. Li *et al.*, "Nonvolatile memory allocation and hierarchy optimization for high-level synthesis," in *ASP-DAC*. IEEE, 2015, pp. 166–171.

[22] ——, "Leveraging nonvolatile memory in high-level synthesis for accelerator-rich architecture," in *NVMW*, 2015.

[23] X. Dong *et al.*, "Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *TCAD*, vol. 31, no. 7, pp. 994–1007, 2012.

[24] M. Ekstrom *et al.*, "Two-dimensional spectral factorization with applications in recursive digital filtering," *TASSP*, vol. 24, pp. 115–128, 1976.

[25] P. Feautrier, "Some efficient solutions to the affine scheduling problem. part ii. multidimensional time," *International journal of parallel programming*, vol. 21, no. 6, pp. 389–420, 1992.

[26] S. Li *et al.*, "Leveraging emerging nonvolatile memory in high-level synthesis with loop transformations," 2015, http://www.ece.ucsb.edu/~shuangchenli/TR/ISLPED_TR.pdf.

[27] I. Issenin *et al.*, "Drdu: A data reuse analysis technique for efficient scratch-pad memory management," *TODAES*, vol. 12, no. 2, p. 15, 2007.

[28] M. Kandemir *et al.*, "Compiler-directed scratch pad memory hierarchy design and management," in *DAC*. ACM, 2002, pp. 628–633.

[29] W. Li, "Compiling for numa parallel machines," Cornell University, Tech. Rep., 1994.

[30] H.-W. Lee *et al.*, "25.3 a 1.35 v 5.0 gb/s/pin gddr5m with 5.4 mw standby power and an error-adaptive duty-cycle corrector," in *ISSCC*. IEEE, 2014, pp. 434–435.

[31] "Lingo 14.0," http://www.lindo.com/.