Liquid Silicon-Monona: A Reconfigurable Memory-Oriented Computing Fabric with Scalable Multi-Context Support

Yue Zha
Department of Electrical and Computer Engineering
University of Wisconsin - Madison
yzha3@wisc.edu

Jing Li
Department of Electrical and Computer Engineering
University of Wisconsin - Madison
jli587@wisc.edu

Abstract

With the recent trend of promoting Field-Programmable Gate Arrays (FPGAs) to first-class citizens in accelerating compute-intensive applications in networking, cloud services and artificial intelligence, FPGAs face two major challenges in sustaining competitive advantages in performance and energy efficiency for diverse cloud workloads: 1) limited configuration capability for supporting light-weight computations/on-chip data storage to accelerate emerging search-/data-intensive applications. 2) lack of architectural support to hide reconfiguration overhead for assisting virtualization in a cloud computing environment.

In this paper, we propose a reconfigurable memory-oriented computing fabric, namely *Liquid Silicon-Monona* (*L-Si*), enabled by emerging nonvolatile memory technology i.e. RRAM, to address these two challenges. Specifically, *L-Si* addresses the first challenge by virtue of a new architecture comprising a 2D array of physically identical but functionally-configurable building blocks. It, for the first time, extends the configuration capabilities of existing FPGAs from computation to the whole spectrum ranging from computation to data storage. It allows users to better customize hardware by flexibly partitioning hardware resources between computation and memory, greatly benefiting emerging search- and data-intensive applications. To address the second challenge, *L-Si* provides scalable multi-context architectural support to minimize reconfiguration overhead for assisting virtualization. In addition, we provide compiler support to facilitate the programming of applications written in high-level programming languages (e.g. OpenCL) and frameworks (e.g. TensorFlow, MapReduce) while fully exploiting the unique architectural capability of *L-Si*. Our evaluation results show *L-Si* achieves 99.6% area reduction, 1.43× throughput improvement and 94.0% power reduction on search-intensive benchmarks, as compared with the FPGA baseline. For neural network benchmarks, on average, *L-Si* achieves 52.3× speedup, 113.9× energy reduction and 81% area reduction over the FPGA baseline. In addition, the multi-context architecture of *L-Si* reduces the context switching time to $\sim 10ns$, compared with an off-the-shelf FPGA ($\sim 100ms$), greatly facilitating virtualization.

ACM Reference format:
Yue Zha and Jing Li. 2018. Liquid Silicon-Monona: A Reconfigurable Memory-Oriented Computing Fabric with Scalable Multi-Context Support. In *Proceedings of 2018 Architectural Support for Programming Languages and Operating Systems, Williamsburg, VA, USA, March 24–28, 2018 (ASPLOS '18)*, 15 pages.
https://doi.org/10.1145/3173162.3173167

ASPLOS '18, March 24–28, 2018, Williamsburg, VA, USA

ACM ISBN 978-1-4503-4911-6/18/03…$15.00
https://doi.org/10.1145/3173162.3173167

1 Introduction

Field-Programmable Gate Arrays (FPGAs) are attractive in many computational domains due to their ability to achieve high throughput and predictable latency while providing programmability, low power and time-to-value [102][38][70][67][30]. While traditionally being used in embedded systems, FPGAs have recently begun to make their way into data centers and the cloud. For instance, Amazon and Alibaba (partnered with Intel) have launched FPGAs for Acceleration-as-a-Service on the cloud i.e., AWS EC2 F1 instances [3] and the Aliyun cloud [43]. Microsoft [10] and Baidu [68] have also deployed FPGAs in their data centers for web search ranking and machine learning applications. While FPGAs provide the lucrative benefits of fine-grained parallelism and high flexibility to accelerate compute-intensive applications, they face *two* major challenges that hinder further performance improvements for diverse cloud workloads.

The *first* challenge is the lack of efficient architectural support for accelerating emerging search-/data-intensive workloads that have low compute-to-memory access ratios [103]. The configuration capability of state-of-the-art FPGAs is highly skewed towards the computation side. Thus, compute-intensive applications with high compute-to-memory access ratios could greatly benefit from it. However, current FPGAs do not provide sufficient configuration support to cover the other side of the spectrum on the light-weight computation and memory sides. For instance, FPGAs do not have a dedicated configuration mode for implementing a ternary content addressable memory (TCAM) - a light-weight compute hardware specialized in performing massively parallel search inside data [69]. Most existing workarounds rely on brute force

to emulate the equivalent TCAM function at the cost of excessive hardware resources including a large amount of on-chip memory and configurable logic. Moreover, the configuration support for on-chip memory is limited, in both capacity (limited to ~ 100's MB [97][98]) and location (embedded as hard IP at fixed location by vendors and unchangeable by users after manufacturing). These limitations make it difficult to exploit data locality [27] when implementing data-intensive applications.

The *second* challenge is the lack of architectural support for assisting virtualization in a multitasking and resource-sharing cloud computing environment, which requires efficient management and minimization of reconfiguration overhead in time and space. Off-the-shelf FPGAs are single-context devices which only store one context of configuration. As computation and configuration are *mutually exclusive* operations in such a single-context architecture, FPGAs cannot commence computation until the loading of the configuration (bitstream) completes, resulting in high reconfiguration overhead [95]. As such, existing cloud services only support the dedicated allocation of specific FPGAs for a single user/application [3] *without* sharing to simplify the resource management and task scheduling. Several recent works [9] [14][50][32][4] proposed to virtualize and share the resources of an FPGA to improve per-device utilization, but require complex resource allocation and task scheduling algorithms to hide the long reconfiguration time. Alternatively, the multi-context FPGA shows promise [26][11][84] in easing virtualization by providing native hardware support for storing multiple configurations to enable a fast context switch, but incurs prohibitive costs of adding a large amount of SRAMs and multiplexers (~ 40% area increase for one context) in practical implementations.

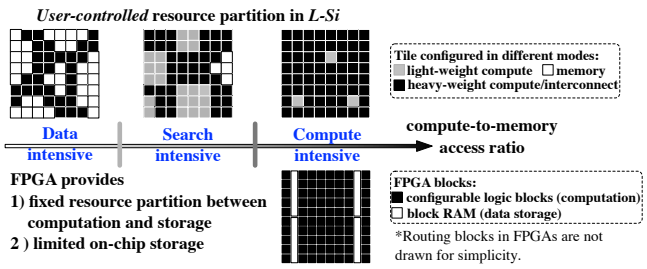

Figure 1. *L-Si* provides a *user-controlled* resource partition to cover the whole spectrum, from data-intensive to compute-intensive. On the contrary, FPGAs only provide an efficient support on compute-intensive applications.

In this work, we propose a reconfigurable memory-oriented computing fabric based on RRAM technology [94], namely *Liquid Silicon-Monona* (also referred to as *L-Si*), to tackle these two challenges. More specifically, *L-Si* addresses the *first* challenge by introducing a new building block and a new architectural organization. As shown in Fig. 1, it consists of a 2D array of physically identical tiles, each of which can be configured into one or a combination of the following four modes: 1) heavy-weight compute, 2) light-weight compute, 3) interconnect, and 4) memory. To our best knowledge, such an architecture, for the *first* time, extends the configuration capabilities of existing FPGAs from computation to the whole spectrum, ranging from full computation to full memory, or any intermediate state (partial memory and partial computation). Moreover, the dedicated light-weight configuration mode in *L-Si* enables simple and native implementations of high performance search and binarized neural network functions [25], making it particularly beneficial for a wide range of search-/data-intensive applications. Thus, such a hardware substrate gives users more flexibility in customizing hardware (not just the compute units in FPGAs) to better match an application's characteristics (Fig. 1) for higher performance and energy efficiency.

For the *second* challenge, *L-Si* provides scalable architectural support i.e. multi-context architecture to assist virtualization by leveraging the monolithic 3D stacking capability of RRAM at low cost, versus the prohibitively costly adoption of SRAMs in multi-context FPGAs. In addition, *L-Si* has other architectural benefits over FPGA in routing, logic implementation and security, etc., making it also advantageous in accelerating traditional FPGA workloads (computationally intensive). More details can be found in Section 3.6.

Due to the architectural differences between *L-Si* and FPGA, we also provide a compilation framework (Section 4.2) to facilitate the programming of applications written in high-level programming languages (e.g. OpenCL) and frameworks (e.g. TensorFlow and MapReduce) for *L-Si*. Our goal is to fully exploit the architectural capability of *L-Si* (Section 3.6) to improve code mapping quality on *L-Si* while reducing programming burden and enabling code portability across platforms (CPU/GPU/DSP/FPGA) for heterogeneous computing.

In particular, we made the following major contributions:

1. We develop a reconfigurable memory-oriented computing fabric, namely *L-Si*. To our best knowledge, it is the *first* work to enable a continuum of user-controlled configuration capabilities across the whole spectrum from computation to data storage. Compared with state-of-the-art FPGAs, it provides more flexibility in customizing hardware to better match an application's characteristics and effectively complement FPGAs in accelerating search-/data-intensive applications. The new building block and architectural organization of *L-Si* effectively addresses the first above-mentioned challenge.

2. We present a scalable architectural support to assist virtualization, which addresses the second challenge. Specifically, we add multi-context support to *L-Si* architecture by leveraging the monolithically stacked 3D RRAM arrays to implement multiple planes (contexts) of configuration data, facilitating *dynamic* and *fast* runtime reconfiguration (RTR) at low overhead. We note that the off-the-shelf FPGAs do not have multi-context support due to cost constraints.

3. We provide a compilation framework to facilitate application development for *L-Si*. Specifically, it comprises a flexible front-end that translates applications written in high-level programming languages (e.g. OpenCL) / frameworks (e.g. TensorFlow and MapReduce) into Verilog RTL to ease the programming of *L-Si*, and a custom back-end that further converts Verilog RTL code into bitstreams to fully exploit the architecture capability of *L-Si* for a high mapping quality.

4. We conduct experiments to evaluate *L-Si* on representative benchmarks from real-world applications and provide a comprehensive comparison with: 1) SRAM-based FPGAs (off-the-shelf FPGAs as our baseline); 2) RRAM-based FPGAs; and 3) SRAM-based *L-Si* to provide more insights on the performance and efficiency gain. Our evaluation shows that the RRAM-based *L-Si* outperforms the other three architectures in all benchmark applications, which confirms the net benefits are due to the combined architecture and technology advantages (not simply due to technology) (Section 3.6). Moreover, we show that the light-weight compute mode results in much more performance and efficiency improvement in search-/data-intensive applications than traditional compute-intensive applications. Specifically, *L-Si* achieves 99.6% area reduction, 1.43× throughput improvement and 94.0% power reduction in search-intensive benchmarks, as compared with the FPGA baseline. For data-intensive benchmarks, *L-Si* achieves 52.3× speedup, 113.9× energy reduction and 81% area reduction over the FPGA baseline.

5. Our evaluation also confirms that *L-Si* is a more scalable architecture in providing multi-context support than FPGA. It reduces the context switching time of off-the-shelf FPGAs from $\sim 100ms$ to $\sim 10ns$. As the number of contexts increase, the relative area increase of *L-Si* grows much more slowly than that of the multi-context RRAM-based FPGA, indicating that the improved scalability is not only because of the dense RRAM technology, but also from the architecture itself.

The rest of the paper is organized as follows. Section 2 provides background information. Section 3 describes the architecture of *L-Si*, the operating principles in various configuration modes, the configuration process, the context switching process and the physical design. Section 4 briefly describes system integration and a compilation framework. Section 5 presents evaluation results. Section 6 compares *L-Si* with prior work, followed by Section 7 to conclude the paper.

2 Background

2.1 FPGA Architecture

A typical island-style FPGA architecture is illustrated in Fig. 2, which consists of an array of configurable logic blocks (CLBs), switch blocks (SBs), connection blocks (CBs) and specialized hard IP blocks. The CLBs can be configured to implement arbitrary logic functions while CBs and SBs are used for routing. Hard IP blocks are scarce hardware resources that are used for augmenting the capability of FPGAs in performing specific functions, e.g. block RAM (BRAM) for on-chip storage, but are much less programmable than the others (CLB/CB/SB).

The architecture limitations of state-of-the-art FPGAs include: 1) FPGAs do not provide native hardware support for light-weight computation (i.e., TCAM). 2) On-chip memory (BRAM) is implemented as specialized hard IP blocks with fixed capacity and location and thus cannot be changed by users after manufacturing. Although CLBs can be used as distributed RAMs in some rare cases, not all CLBs support this function due to resource constraints, and the capacity is limited to $\sim 10Mb$ [97][98]. 3) Resources for routing (CBs and SBs) cannot be used to implement other functions e.g., logic. 4) Off-the-shelf FPGAs are single-context devices.

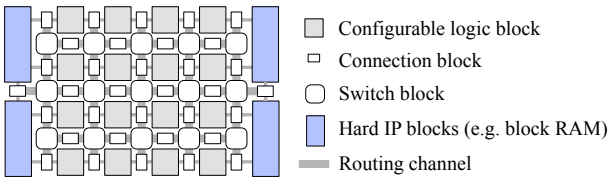

Figure 2. A conceptual diagram of a typical FPGA architecture

2.2 RRAM technology

Resistive random access memory (RRAM) has emerged as one of the most promising non-volatile memories due to its small cell size ($4F^2$, F is the minimum feature size), excellent scalability (<10nm) [94], fast switching time (<10 ns) [87], and good endurance (up to 10^{12} cycles) [53]. Resistive switching is achieved by applying appropriate voltages on the terminals. Another advantage of RRAM is the CMOS-compatible fabrication process and monolithic 3D integration [46][15]. Therefore, RRAM cells can be organized into stackable and ultra-dense crossbar arrays at no extra area overhead.

3 Hardware Architecture

Fig. 3 depicts the architecture of *L-Si*, comprising a 2D array of *physically identical* tiles. Each tile can be configured into one or a combination of the four modes: 1) heavy-weight compute, 2) light-weight compute, 3) interconnect, and 4) memory, depending on the workload (Fig. 1). It is worth noting that, in addition to the coarse-grained (tile-wise) configuration, *L-Si* also allows for flexible partitioning of resources *within* a tile between computation and routing in a more fine-grained manner (Fig. 4a). Such a combination of coarse-grained and fine-grained controls results in improved utilization and reduced routing pressure over conventional FPGAs (Fig. 4b).

Moreover, compared with FPGA, *L-Si* provides scalable architectural support to assist virtualization. Specifically, the configuration architecture of *L-Si* leverages the monolithically stacked RRAM arrays to implement multiple planes (contexts) of configuration data, facilitating *dynamic*/*fast* runtime reconfiguration (RTR) at low overhead. On the contrary, implementing a multi-context FPGA incurs prohibitive costs due to the required additional configuration storage and a large number of multiplexing circuitry that could otherwise be used for logic or routing [26]. The key architectural advantages of *L-Si* over FPGAs have been summarized in Section 3.6.

In the following subsections, we start with an overview of the *L-Si* architecture and present how to cost effectively implement the multi-context architecture. Then, we describe the detailed operations of the configuration modes and present the configuration process, context switching process and the physical design. Finally, we discuss the key architectural differences between *L-Si* and FPGA.

3.1 Architecture Overview

Despite the rich configuration modes it supports, the tile structure is relatively simple. As shown in Fig. 3, each tile is composed of two basic building blocks: 1) a 3D stacked array of RRAM crossbars and 2) connection nodes.

The crossbar array has a native array structure, where each 1D1R cell (contains 1 diode [45] and 1 RRAM element [92]) is placed at the intersection of a word-line (WL) and a bit-line (BL). We note that the diode (also known as access device) is used to suppress the sneak path leakage. The array itself can be *fully* reused across the four modes of operations, i.e., implementing arbitrary logic functions (heavy-weight compute mode), TCAM function (light-weight compute mode), memory and routing. Multiple crossbar arrays can be stacked monolithically to store multiple contexts of configuration data at no extra area penalty (Fig. 3). The number of stacked layers is determined by the number of contexts to be supported in hardware, and context selection is done by multiplexers. Previously, an 8Mb two-layered crossbar array, which uses the TaO_x-based 1D1R RRAM cell, has been demonstrated by Panasonic [46]. As the focus of this paper is not on technology, details for fabricating RRAM crossbar can be found in [46].

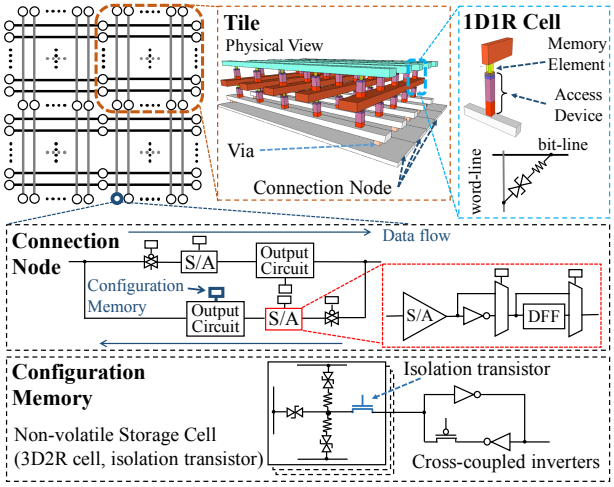

Figure 3. Conceptual diagram of *L-Si* architecture comprised of a 2×2 array of tiles. The key circuit components of a tile are highlighted, including stackable 1D1R crossbar arrays and connection nodes. The structure of the configuration memory is also illustrated, which supports multi-context by stacking multiple non-volatile storage cells.

The connection node is used for connecting WLs (or BLs) of two adjacent crossbar arrays, restoring small analog signals to full-swing digital signals for noise tolerance, and supporting the operations of four configuration modes. As shown in Fig. 3, each connection node consists of sense amplifiers (S/As), output circuits and configuration memories. In particular, as compared with large and power-hungry conventional current or voltage S/As, a compact and low-power RC-based S/A (adapt the design in [54]) is employed to improve noise tolerance. It detects the small analog resistance changes on a WL (or a BL) in one array and generates a full-swing digital output to drive the corresponding WL (or BL) in the adjacent array, and vice versa. The output circuit can be configured to generate different drive voltages to support the four modes of operation, controlled by configuration memories. Each configuration memory comprises of non-volatile storage cells (NV-cells) and a cross-coupled inverter, as shown in Fig. 3. The NV-cells are used for storing multiple contexts of configuration data. The cross-coupled inverter is used to sense and hold the configuration data of one context from the NV-cells during normal computation. Similar to crossbar arrays, NV-cells can also be stacked to support multi-context and context selection is controlled by the isolation transistors (Fig. 3). We note that each connection node is designed with two copies of the circuits connected back to back (Fig. 3) to operate *bidirectionally* and can be disabled if not in use to save power.

3.2 Configuration Modes

Heavy-weight Compute Mode. One tile in this mode can implement arbitrary combinational logic functions *in situ* within the crossbar array without using ALUs. Sequential logic can be implemented by configuring a skippable flip-flop at the output of each S/A in connection nodes (Fig. 3). Here, we take an example to illustrate how to map a combinational logic function $F = AB$ on one of the tile entries (Fig. 5a). Our mechanism exploits the fact that each S/A in the connection node is connected to many RRAM cells in the crossbar array. We show that by simultaneously driving two WLs (or BLs) by logic inputs A and B, and by programming the values of the two cells into the low resistance state, the S/A is forced to perform the *bitwise AND* of the inputs, realizing the function $F = AB$. The rest of the RRAMs connected to the same S/A are programmed into the high resistance state for isolation.

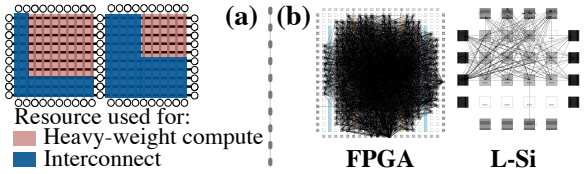

Figure 4. (a) To improve resource utilization, one tile can be partitioned between heavy-weight compute mode and interconnect mode. (b) These flexibilities result in better algorithmic mapping with low routing pressure.

Overall, there are three key features for such crossbar-based logic implementation: 1) Each data entry of a tile can implement a multi-input-single-output logic function (up to 256 inputs). 2) The logic inputs are shared among the logic functions in the same tile. 3) The data flow direction can be controlled at a fine granularity of entry level (instead of tile level). The inputs and outputs can be applied on either the top/bottom or left/right side of a tile. We will show that our compilation framework (Section 4.2) takes advantage of these architectural features at the technology mapping stage to improve mapping quality (Fig. 4b).

Light-Weight Compute Mode. One tile in this mode can be configured as an embedded TCAM block which can be used to implement high-performance parallel search or binarized neural network [25]. The idea is inspired by the fact that a TCAM array based on non-volatile memory [54] has the same physical design as a memory array and thus the crossbar array can be reused to implement the TCAM function as a dedicated light-weight configuration mode.

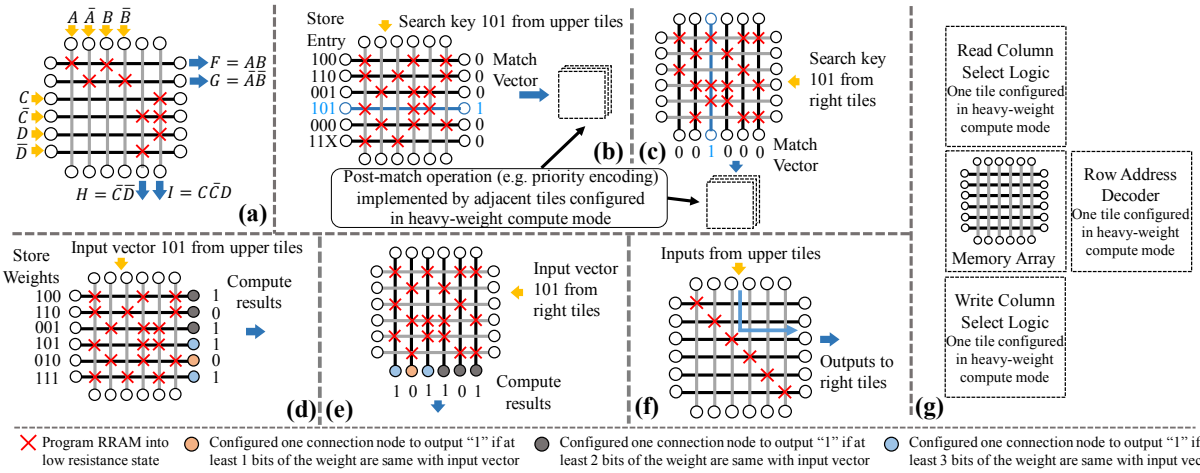

Figure 5. (a) The operation of the heavy-weight compute mode is illustrates and four logic functions are packed and mapped onto one tile. (b)(c) The light-weight compute mode supports the parallel search operation. Data layout in one can be either (b) horizontally or (c) vertically. The data sets stored in (b) and (c) are same, and the matched entry is highlighted in blue. (d)(e) The light-weight compute mode is also used to implement the binarized neural network, and the data layout in one tile can be either horizontally or vertically. (f) The operation of the interconnect mode is illustrated. (g) Four tiles are used to implement a memory block. One of them is configured into Memory Mode as the memory array, and other three tiles are configured in heavy-weight compute mode to perform address decoding.

Fig. 5(b)(c) shows an example of performing parallel search. First, the search keys (101) are applied on WLs (or BLs) as search inputs. Then, they are compared with every data entry that is stored in the crossbar array in parallel. Finally, the S/As output a match vector in which a logic "0" indicates a mismatch between the search inputs and the stored entry value while a logic "1" indicates a match. The match vector can be further fed to its adjacent tiles for post processing (Fig. 5b,c).

It is worth noting that the TCAM implementation on *L-Si* provides three main advantages over the other standalone TCAM designs [54][57][12]. 1) It can implement TCAM blocks with different sizes and/or aspect ratios by coalescing adjacent tiles. 2) It allows data to be stored in either a row-wise (Fig. 5b) or column-wise manner (Fig. 5c). 3) It allows users to flexibly define their custom post-match functions such as a priority encoder or a population counter by configuring adjacent tiles into heavy-weight compute mode (Section 3.2). We will show that these unique flexibilites will be exploited by our compilation framework to optimize the mapping of user-defined TCAM blocks.

This mode also provides a native implementation of binarized neural network (BNNs) (Fig. 5d,e), which is equivalent to a TCAM function. Specifically, the binary weights are stored in the crossbar arrays (like the stored data entry in TCAM), while the input vector is applied on the WLs (or BLs) of the tile (like the search inputs). The S/A will implement equivalent count, normalization and activation functions in the neural network.

Interconnect Mode. One tile in this mode implements a fully populated crossbar switch for routing. Compared with the dedicated routing blocks in FPGAs (Section 2.1), this mode can co-exist with the heavy-weight compute mode in the same tile (Fig. 4a). In fact, this mode can also be viewed as a special case of the heavy-weight compute mode in which a tile is configured into a single-input buffer function ($F = A$), as shown in Fig. 5(f). The co-existence of the two modes inside one tile opens up compiler optimization opportunities. We will show that this feature is exploited by our compilation framework (Section 4.2) to adaptively partition the hardware resources (WLs and BLs) between logic and routing based on applications' needs to improve the mapping quality.

Memory Mode. In this mode, four adjacent tiles are used to implement an embedded memory block. As shown in Fig. 5(g), one of them is configured as the memory array. The other three tiles are configured in the heavy-weight compute mode to implement column address decoder, row address decoder and read/write column select logic. The read/write process is the same as that of a conventional scratchpad memory. It appears that using four tiles to implement one fully functional memory block is likely to be inefficient. However, due to the ultra-dense array organization and simplified pitch match between the array and the periphery, the overhead is negligible as shown in our evaluation in Section 5.2.

We note that since all circuits are implemented in a soft-logic style (instead of ASIC), such an on-chip memory implementation on *L-Si* is much more flexible than the embedded hard IP memory blocks on FPGA. Specially, it offers the following flexibility: 1) The size and/or aspect ratios of memory blocks can be configured by coalescing adjacent tiles. 2) The location of the memory blocks can be controlled by users to better exploit locality [27]. 3) The orientation of the memory blocks can vary to allow flexible data layout in either row-wise or column-wise manner. Similar to other modes, our compiler optimization takes advantage of these flexibilities to achieve better mapping quality.

3.3 Configuration Process

In this subsection, we will present the configuration process for *L-Si*. Specifically, as configuration data are stored in both crossbar arrays and configuration memories (in connection nodes), we will describe how to configure each of them.

We first present the configuration process for crossbar arrays. Note that each connection node has additional circuits reserved for configuration, containing one buffer and two multiplexers. They can be controlled to connect WLs (or BLs) in the adjacent crossbar arrays of the same context plane (selected by the multiplexers). Therefore, each context plane of *L-Si logically* only contains *one* crossbar array and can be configured using any write scheme (e.g. "V/3" write scheme) proposed to write crossbar arrays [13]. During configuration, the buffers are used to restore the degraded write voltages due to long-distance routing to mitigate the IR-drop issue. We note that due to the separate data-paths in configuration and computation, when configuring one context plane (i.e. loading one context into *L-Si*), the others will not be disturbed and thus can operate in normal computation mode without interruption.

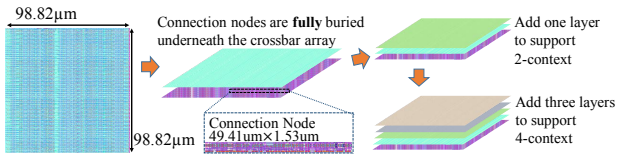

Figure 6. Physical design of one tile (45nm technology).

We then discuss the process of programming configuration memories in the connection nodes. Specifically, the configuration process for configuration memories is divided into two steps, 1) write the NV-cells, 2) sense and hold the configuration data (of one context) from the NV-cells into a cross-coupled inverter. As shown in Fig. 3, one bit of configuration data is stored in one NV-cell by programming the two RRAM cells into complementary resistance states. Before normal computation starts, the isolation transistor is turned on and the configuration bit is sensed by the cross-coupled inverter. As soon as the sensing process finishes, the isolation transistor will be turned off and the cross-coupled inverter holds the sensed value for running applications. Meanwhile, the NV-cells are turned off to save power. Note that, during the first step (writing NV-cells), the isolation transistor remains off to disconnect the NV-cells from the cross-coupled inverter which holds the configuration bit of current running program. Thus, the NV-cells can be written in the background without interrupting the running application. This way, the relatively long programming process of RRAM cells can be effectively hidden, resulting in fast/dynamic runtime reconfiguration.

3.4 Context Switching Process

The context switching process for *L-Si* is relatively straightforward compared to the configuration process. In the following discussion, we will present the switching operations for both crossbar arrays and configuration memories, respectively.

Specifically, a control signal (context ID) is first broadcasted to all tiles. Then, the crossbar array that stores the configuration data will be selected. For the configuration memories, the configuration data of the selected context will be loaded from the corresponding NV-cell into the cross-coupled inverter by turning on the isolation transistor (Fig. 3). We note that such a data loading process is similar to the sense-and-hold configuration process in configuration memories (Section 3.3).

To summarize, such a multi-context design has two main benefits. 1) It enables background loading of configuration data during operation, overlapping computation with configuration. 2) The context switching process does not need to write RRAM cells and thus can switch between stored configurations quickly, dramatically reducing reconfiguration overhead if the next configuration is present in one of the alternate contexts. This fast context switch enables 1) several programming models (e.g. pipeline reconfiguration model [78]) to simplify the programming interface and 2) time-sharing of the hardware resources in a multitasking and resource-sharing environment.

3.5 Physical Design

The physical design of *L-Si* is non-trivial compared with conventional CMOS-based chip, as the layout will need to account for the special monolithic 3D integration of RRAM to achieve area optimization. Specifically, as shown in Fig. 6, we performed two optimizations to save area. 1) We *fully* buried the CMOS circuits (connection nodes) underneath the crossbar array and stacked multiple crossbar arrays between metal layers to implement multi-context *L-Si*. 2) We share the common circuits (e.g. S/A and voltage driver) as much as possible between tiles. After applying these optimizations, the area of one tile is $98.82 \times 98.82 \mu m^2$ at the 45nm CMOS technology node. In addition to providing more accurate measurement of the chip area, the physical design of *L-Si* also provides key parameters (e.g., parasitic capacitance/resistance) for circuit simulation in Section 5.

3.6 Comparison with FPGA

L-Si shares some similarities with FPGAs in its reconfigurable data-flow architecture, but it also radically differs from FPGAs by providing the following features which addressed the architecture limitations of state-of-the-art FPGAs.

Hardware support for light-weight computation. *L-Si* provides native TCAM hardware support by virtue of a dedicated light-weight compute mode, whereas FPGAs need to consume scarce on-chip memory resources to emulate the equivalent TCAM function. Moreover, this mode also provides an efficient implementation of binarized neural networks, which nevertheless requires costly hardware resources (including both logic and on-chip memory) in FPGA for the same purpose. Therefore, search-/data-intensive applications can be performed more efficiently on *L-Si* than FPGAs.

Flexible memory blocks. *L-Si* provides a flexible memory support to better customize the *capacity* and location of on-chip memory, depending on workloads. For instance, one can configure more tiles into the memory mode to achieve high capacity and in close proximity to compute units to better exploit data locality to save power. On the contrary, memory blocks are hard-wired resource in FPGA and thus their capacity and location cannot be changed after manufacturing.

Scalable multi-context support. *L-Si* provides multi-context support in a cost-effective manner by monolithically stacking crossbar arrays and NV-cells (configuration memories). The low cost is not only due to the 3D stacking capability of RRAM technology (no area overhead), but also comes from the architecture itself. More specifically, the crossbar structure in *L-Si* enables better circuit sharing than the CLB structure in FPGA, thereby requiring fewer multiplexers than FPGA to support the same number of contexts (Section 5.5).

Coarse-grained logic implementation As presented in the heavy-weight compute mode (Section 3.2), tiles in *L-Si* supports logic functions with a large number of inputs. To improve tile utilization (the percentage of resources of a tile used for mapping), our compilation optimization (Section 4.2) has taken advantage of this architectural feature and employs a coarse-grained logic implementation, i.e., applications are synthesized into complex logic functions with larger granularity ($\sim$30 inputs). On the contrary, logic implementation in the FPGA is fine-grained, where applications are synthesized into a netlist of simple logic gates ($\leqslant$ 6 inputs), and these gates are mapped onto 6-input lookup tables (LUTs) in an FPGA. As compared with FPGA, the coarse-grained logic implementation in *L-Si* results in shallower logic depth, less routing pressure (Fig. 4b), better tile utilization and thus higher performance and energy efficiency than FPGAs.

Fully exploit RRAM technology. Enabled by the nonvolatile nature of RRAM technology, *L-Si* does not need to load bitstreams from external memory when it is powered on, thereby reducing the configuration times and power. The nonvolatile nature also improves its security as the bistreams are stored *internally*. It eliminates the security vulnerability caused by the external bitstream loading process in FPGA which creates an easily exploitable, non-invasive conduit by which the FPGA's IP can be captured and copied.

Efficient resource partitioning. In *L-Si*, the hardware resources can be flexibly partitioned by the compilation framework between logic and interconnect based on the actual usage (Fig. 4a), leading to better resource utilization than FPGA.

4 System Integration and Compilation Framework

4.1 System Integration

As a hardware accelerator, *L-Si* can be easily integrated into current systems in the same way as FPGAs. For instance, the low cost and low latency Intel QuickPath Interconnect (QPI) [42], IBM Coherent Accelerator Processor Interface (CAPI) [93] and PCIe [2] can be applied to provide a high-performance interconnection for integration. Moreover, *L-Si* can be integrated on the memory bus by leveraging the existing processor-DIMM interface, and host processors can use ordinary memory read/write commands to communicate with *L-Si*. This integration requires moderate modifications to the existing system and has been used in prior work [33][36]. Advanced memory management techniques developed for FPGAs [1][21] can also be applied to *L-Si* in the same way.

4.2 Compilation Framework

Each L-Si chip contains hundreds of thousands of tiles with Gb-scale RRAM that needs to be configured to run an application. Therefore, it is intractable to custom-tailor each memory element for application mapping. To address this issue, we present a compilation framework to facilitate application development for *L-Si* in high-level programming languages and programming frameworks.

Fig.7 depicts the compilation framework for *L-Si*. It comprises a *flexible* front-end that supports a wide range of popular high-level programming languages/frameworks, and a *custom* back-end that can *fully* exploit the low-level architectural features of *L-Si* to achieve optimal code mapping on target hardware (Section 3.6). More specially, the front-end takes an application written in high-level programming languages/frameworks as an input and translates it into synthesizable Verilog RTL code. The back-end further synthesizes the Verilog RTL code into bitstreams, which are used to configure *L-Si*. In the following discussion, we will first describe the front-end and then highlight the key compiler optimization techniques in the back-end design.

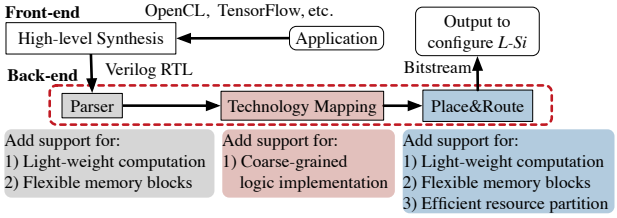

Figure 7. Workflow of the compilation framework. The back-end is modified to support the features provided in *L-Si*.

The front-ends of the framework are reused from those developed for FPGAs, including popular high-level programming languages (e.g. OpenCL [41]) and domain-specific programming frameworks (e.g.,MapReduce [90] and TensorFlow [35]). The reason to select these front-ends is that they all generate a common code representation in Verilog RTL which is fully compatible with our custom back-end. We note that our compilation framework is designed to be **reusable** and **extendable** to other front-ends that output the same intermediate code representation in Verilog RTL.

The custom back-end is adapted from one of the most popular open-source retargetable toolchains — Verilog to Routing (VTR) [59] that has been developed for mapping applications written in Verilog onto FPGA. Nevertheless, we made several major modifications to VTR to account for the fundamental architectural differences between FPGA and *L-Si* for optimal code mapping. As shown in Fig. 7, the back-end contains three stages, i.e., parser, technology mapping and place&route. Specifically, we modify the parser to support the four configuration modes, and modify the technology mapping tool to realize the coarse-grained logic implementation. In addition, the place&route tool in VTR (VPR [60]) is adapted from a custom tool developed in Zha et al. [101] with additional modifications. Specifically, to optimize the placement and routing of the TCAM or BNN modules in *L-Si*, we modify the cost function following the same approach of optimizing the memory

Table 1. Description of the search-intensive benchmark set.

Benchmark	Description	Search key width (bit)	Post-match processing
String Match [72]	Scan a list of encrypted words for occurrences of a set of keys	80	Bitwise OR
Packet Classification [85]	Classify every incoming packet by comparing its header fields against a filter set.	104	Priority encoding
Word Count [72]	Count the occurrences of each unique word in a document.	184	Population count
Similarity Search [80]	Data mining technique for various applications [61][8][51][73] A TCAM-based implementation of similarity search is realized in [80].	288	Priority encoding

placement in [101]. As more details about these three stages can be found in [60], in the following discussion, we only highlight the key compiler modifications as compared with [60].

Compiler support for light-weight computation. As VTR is originally developed for FPGAs, which do not have dedicated configuration support for light-weight computation (Section 3.6), we modify it to add compiler support for such a new feature in *L-Si*. More specifically, we add two new Verilog modules for TCAM and BNN that can be instantiated in the Verilog RTL code. The parser is modified to identify the instantiations of these modules and convert them into a logical netlist. In case the size of some modules exceeds a predefined threshold, the parser will split it into multiple smaller ones to ensure it is physically realizable. The place&route tool then takes the netlist as an input and maps it onto physical tiles. To optimize the placement and reduce the routing cost, the cost function is modified in the same way as [101]. The new cost function is applied to optimize the locations and orientations (row-/column-wise) of TCAM or BNN modules during placement.

Compiler support for flexible memory blocks. Although VTR provides compiler support for mapping memory modules on FPGA, the mapping is subject to the constraint of physical hardware IP blocks with fixed size and location. Here, we modify it to better exploit the flexibility of *L-Si*. Firstly, an additional parsing step, which is needed in VTR to split a large memory module into smaller ones in order to fit into physical BRAM blocks with fixed size and location, is no longer needed in the case of *L-Si*, whose architecture naturally supports flexible size and location. Secondly, two major modifications are made in the place&route tool. 1) Since the location of memory blocks is flexible in *L-Si*, we remove the legalization step in VTR, which is used to legalize the placement of memory modules due to the fixed location of hard IPs. 2) To optimize the placement of memory modules and to reduce routing cost, we apply the cost function proposed in [101].

Compiler support for coarse-grained logic implementation. VTR has a fine-grained logic implementation in the technology mapping stage, which synthesizes the logic netlist into simple logic gates with $\leqslant 6$ inputs. To adapt VTR to *L-Si*, we modify the technology mapping tool and use the cut enumeration with the priority cuts algorithm [64] to pack the simple logic gates into large clusters (e.g. complex logic functions with $\sim$30 inputs), thereby improving the tile utilization.

Compiler support for adaptive resource partitioning. VTR utilizes dedicated hardware (CBs and SBs in Fig. 2) for routing, while in *L-Si*, routing is flexible and can co-exist with the other modes within a tile. As such, our place&route tool exploits the unused portion of a tile to perform routing. More specifically, we apply a technique called Adaptive Resource Partition [101] to partition the hardware resources in one tile between heavy-weight compute mode (logic) and interconnect mode (routing) to achieve high tile utilization.

5 Evaluation

In this section, we evaluate *L-Si* on a set of benchmarks, from traditional FPGA application and emerging search-/data-intensive applications. We also evaluate the effectiveness of the *L-Si* architecture in supporting multi-context, as compared with off-the-shelf FPGAs (single-context) and multi-context FPGAs. In the following subsections, we first describe the experimental setup, then present the evaluation results.

5.1 Experimental Setup

Benchmark Selection. We have taken the following factors into consideration when selecting the benchmarks. 1) They should be sufficiently representative and diverse enough to cover a range of workload characteristics in the form of compute-to-memory access ratio [104], ranging from compute-intensive applications to data-/search-intensive applications. 2) They should account for the demands of potential new applications which may not be amenable for FPGA-like acceleration but can better exploit the flexible resources of *L-Si*.

Based on these factors, three sets of benchmarks are used: 1) traditional FPGA benchmarks, 2) search-intensive benchmarks and 3) binarized neural network benchmarks. The traditional FPGA benchmarks contain benchmarks from the MCNC suite [99], which have been widely used by the FPGA community to evaluate reconfigurable architectures [65]. We note that the benchmarks in this evaluation are originally developed for FPGA under the constraint of limited on-chip memory support and thus are compute-intensive applications with high compute-to-memory access ratio. The search-intensive and binarized neural network benchmarks, on the other hand, are representative of emerging applications with low compute-to-memory access ratio and are mainly used to evaluate the unique flexibilities of *L-Si* in supporting light-weight computation. In the following discussion, we provide more details about the search-intensive and binarized neural network benchmarks.

The search-intensive benchmark set contains five representative workloads obtained from a diverse set of application domains. In these benchmarks, most of the runtime/energy is spent on the search operation, therefore they can be used to evaluate the light-weight compute mode of *L-Si*. In addition, these benchmarks require different post-match processing (e.g. priority encoding and population count) and have different search key widths, thereby covering the different use cases. More details are presented in Table 1.

The binarized neural network benchmark set contains five binary neural network (BNN) designs. A BNN stores weights as 1-bit binary numbers (+1/-1), thereby significantly reducing the size of weights, as well as the computational complexity (perform bit-wise XNOR instead of floating-point multiplication in Convolutional Neural Networks or CNNs) [25]. We note that the primary reason for choosing BNNs over CNNs in our evaluation is its simplicity and efficiency to be implemented on hardware. Although the precision of weights is reduced, BNNs still provide comparable classification accuracy compared to CNNs on several datasets [25][24][89][49]. Due to these advantages, there has been a growing effort devoted to implementing BNN on different platforms, e.g., CPU, GPU, TrueNorth and FPGA [89][105][66][25][24].

Table 2. Topology for BNN benchmarks.

Binarized CNN	Topology Description
BNN1	3x3conv-3x3conv-2x2pool (output depth 64) 3x3conv-3x3conv-2x2pool (output depth 128) 3x3conv-3x3conv-2x2pool (output depth 256) two FC layers with 512 neurons
BNN2	3x3conv-3x3conv-2x2pool (output depth 128) 3x3conv-3x3conv-2x2pool (output depth 256) 3x3conv-3x3conv-2x2pool (output depth 512) two FC layers with 1024 neurons
Binarized MLP	**Neurons in each layer**
BNN3	784-256-256-256-10
BNN4	784-1024-1024-1024-10
BNN5	784-2048-2048-2048-10

conv: convolution layer. pool: max pooling layer. FC: fully connected layer.

Among the five BNN benchmarks, two are binarized convolutional neural networks (BNN1 and BNN2), while the rest are multilayer perceptrons (BNN3, BNN4 and BNN5) with three hidden layers, as listed in Table 2. Their reported error rates are comparable to their non-binarized counterparts [89][25]. In these BNN benchmarks, the key processing units (perform XNOR, count and normalize operations) are highly pipelined and optimized for performance. Additionally, we assume the training is done offline and only evaluate the performance of inference as most prior works did.

Baseline. FPGA is used as a baseline since it is the most popular commercially available reconfigurable architecture. More importantly, *L-Si* shares some similarities to FPGAs in its morphable data-flow architecture, despite a number of radical differences (Section 3.6).

Moreover, the goal of this evaluation is to provide insights in comparing two architectures (*L-Si*, FPGA) paired with two technologies (RRAM, SRAM). To ensure the benefits that we gained in *L-Si* are not simply due to the advance in technology, we also include two more cases (RRAM-based FPGAs and SRAM-based *L-Si*) in our evaluation. We note that commercially off-the-shelf FPGAs are SRAM-based FPGAs and will be chosen to be our baseline. The RRAM-based FPGA is a drop-in replacement for a SRAM-based FPGA while maintaining the same architecture. Similarly, in the SRAM-based *L-Si*, the RRAMs are replaced with SRAMs. In the rest of the discussion, we also refer to L-Si as RRAM-based *L-Si* to distinguish it from SRAM-based *L-Si*.

Simulation Setup. For *L-Si*, 1) the applications are mapped using our compilation framework (Section 4.2), 2) the delay and power consumption are obtained from the HSPICE simulation (45nm PTM HP model [5]), and 3) the area is measured based on our custom physical design (Section 3.5). More specifically, in the HSPICE simulation, two Verilog-A modules are created to simulate the behavior of the TaO_x RRAM device [92] and the diode [45]. The characteristics of RRAM devices are (1) $R_{on}/R_{off} = 5k\Omega/\ 100k\Omega$ and (2) 1.8V@100ns/-1V@100ns pulse for SET/RESET [91]. The turn on voltage of the diode is $0.4V$ [45]. In the physical design, the size of the crossbar array is chosen to be 256×256. For FPGA, the mapping of benchmarks is generated by the VTR tool set. The area, delay and power are estimated based on the models of SRAM-based FPGAs provided by VTR.

5.2 Traditional FPGA Benchmarks

In this evaluation, the traditional FPGA benchmark set is applied to evaluate the performance of *L-Si*. The SRAM-based FPGA architecture is chosen as the baseline, and the performances of other architectures are normalized to it.

Area: RRAM-based *L-Si* achieves **81**% area savings compared to SRAM-based FPGAs (Fig. 8). We also observe that it consumes **31**% less area than RRAM-based FPGAs, which implies that this area reduction is not simply due to a drop-in replacement for SRAMs with dense RRAMs. It is also worth noting that among the four architectures, SRAM-based *L-Si* has the largest area cost, indicating that *L-Si* is more amenable to pair with the RRAM technology. Our evaluation confirmed the compilation framework can effectively utilize tiles, and the average utilization of tiles is above 70%.

Delay: The delay results are presented in Fig. 8. For all benchmarks, RRAM-based *L-Si* outperforms the other three architectures on average. The improvement in delay mainly comes from the coarse-grained logic implementation of *L-Si*, which has also been confirmed in our experiments. Specifically, the technology mapping stage generates net lists with much shallower depth, resulting in a reduced number of logic gates on the critical paths in *L-Si*. Therefore, the delay of *L-Si* is **52**% less than that of SRAM-based FPGAs, on average.

The delays of the SRAM-based *L-Si* and RRAM-based FPGA increase by 14% and 18%, respectively, compared to the SRAM-based FPGA. We note that the delay becomes worse in SRAM-based *L-Si* due to the fact that the increase in unit delay per tile caused by longer wires (larger RC constant) is much more significant than the decrease in logic depth. The minor increase in delay for the RRAM-based FPGA is because of the longer sensing time of the RRAM array due to its higher R_{on} compared to that of a MOSFET.

Another advantage of *L-Si* is that it makes better use of the routing resources as compared with FPGAs, due to its coarse-grained logic implementation and flexible resource partitioning between logic and routing. Overall, the area consumed by routing in *L-Si* is only **15**% of the total used area, compared to **58**% in the SRAM-based FPGA (Fig. 8).

Energy Efficiency: The energy efficiency results are presented in Fig. 8. On average, RRAM-based *L-Si* achieves **86**%

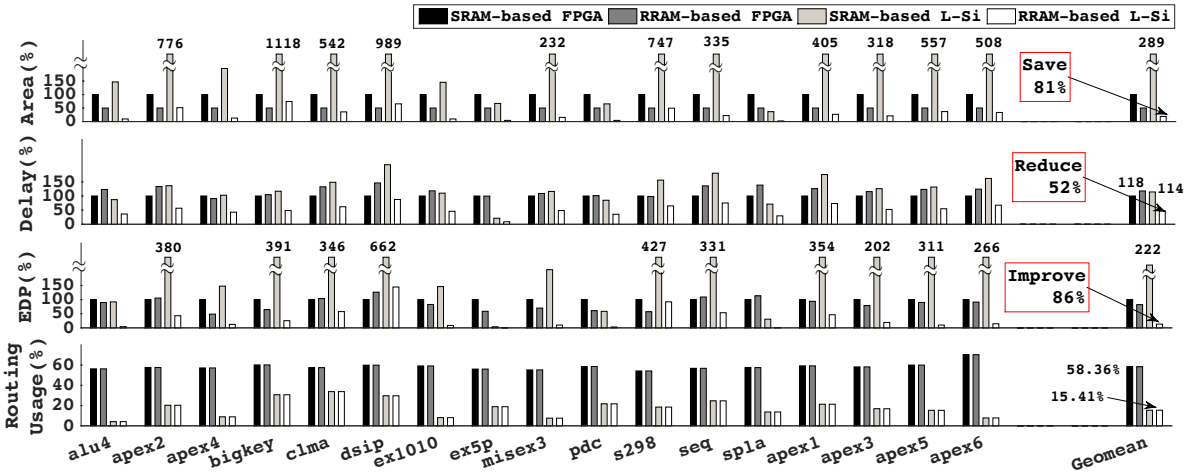

Figure 8. From top to bottom are Area, delay, energy efficiency (energy-delay product, EDP) and routing usage results. Results of the SRAM-based FPGA are used as baseline, and other results are normalized to them. The routing usage is the ratio between routing area and total used area when mapping benchmark circuits. In *L-Si*, it is obtained by first calculating the ratio between routing area and total used area (routing+logic) of each tile and averaging across all tiles.

improvement in energy efficiency compared to the SRAM-based FPGA. This improvement is mainly due to the coarse-grained logic implementation of *L-Si*. Not only does the coarse-grained logic implementation lead to a smaller delay, but it also consumes less hardware resources used for routing and thus reduces energy consumption on data transfer.

5.3 Search-intensive Applications

In this subsection, the search-intensive benchmark set is applied to evaluate *L-Si*. All results are normalized to the SRAM-based FPGA baseline. Overall, across all search-intensive workloads, we observe that the area/throughput/power improvement in *L-Si* are substantially higher than that in the FPGA baseline as well as the other two architectures (RRAM-based FPGA and SRAM-based *L-Si*), as compared with the evaluation results for traditional FPGA benchmarks in Section 5.2. Thus, we confirm the effectiveness of the new light-weight compute mode of *L-Si* in accelerating search-intensive applications. In the following discussion, we present the detailed evaluation results for area, throughput and power for these benchmarks.

Area: On the five evaluated benchmarks, *L-Si* achieves a **99.6**% average area reduction compared with the FPGA baseline (SRAM-based FPGA) (Fig. 9 top). It is also interesting to observe that the SRAM-based *L-Si* even consumes 44.5% less area than the RRAM-based FPGA, indicating that the benefits gained from the light-weight compute mode prevails the area overhead imposed by the larger cell size of SRAM.

Throughput: On average, the RRAM-based *L-Si* achieves **1.43**× improvement in search throughput over the FPGA baseline. Across the benchmarks, we observe a trend that the improvement in search throughput increases with a wider search key. This is mainly because a wide/long search key needs to be split into multiple smaller ones and fed to a local search unit that is implemented using a large amount of logic and BRAM resources in FPGA [44]. The search outputs from these local search units need to be collected globally and processed to generate a final result. Such a distributed implementation consumes more routing resources, thereby increasing the search delay and reducing the throughput. On the contrary, *L-Si* is capable of processing wide search keys locally and directly by coalescing adjacent tiles configured in the light-weight compute mode. Therefore, the search delay and throughput of *L-Si* is *insensitive* to the search key width.

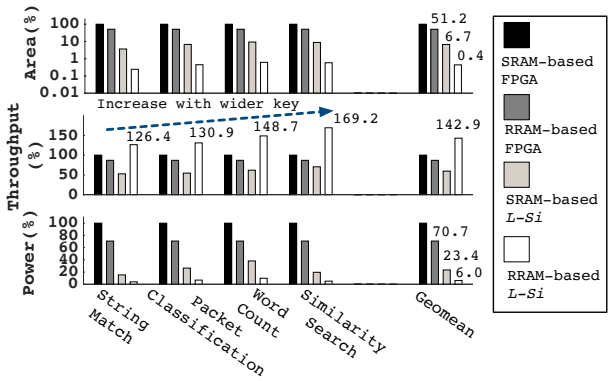

Figure 9. The area saving (top), throughput improvement (middle) and power reduction (bottom) are presented. All results are normalized to that of SRAM-based FPGA. The area result is plotted in logarithmic scale.

Power: For all benchmarks, *L-Si* outperforms the other three architectures in power consumption, and achieves a **94.0**% average power reduction compared with the FPGA baseline. The power saving is mainly due to the more efficient mapping of search operations on *L-Si*, which dominate the power consumption in these applications.

5.4 Neural Network Benchmarks

In this subsection, we further evaluate the effectiveness of *L-Si* in accelerating neural network workloads (specifically BNNs) which are data-intensive. The evaluation results for FPGAs and *L-Si* on BNNs are presented in Fig. 10. For *L-Si*,

on average, it achieves **52.3**× speedup, **113.9**× reduction in energy consumption, and **81**% area reduction compared with the FPGA baseline (SRAM-based FPGA). This improvement mainly comes from two sources. First, in *L-Si*, the weights are stored and processed (bit-wise XNOR) *in situ* inside the same RRAM crossbar, therefore eliminating the frequent memory access to fetch key neural network parameters (e.g. weights) in FPGA. The second reason is that the count, normalization and activation operations are all performed in connection nodes using simple analog circuits, i.e., S/A (as discussed in Section 3.2), whereas FPGA needs complex logic i.e. adders and comparators to perform these operations, resulting in larger delay and energy consumption. Moreover, as the on-chip memory capacity is fixed in FPGA but can be flexibly configured by users in *L-Si*, *L-Si* is expected to achieve even more improvement in performance and energy efficiency than FPGA as the size of the neural network grows exceeding the capacity of the FPGA's on-chip memory.

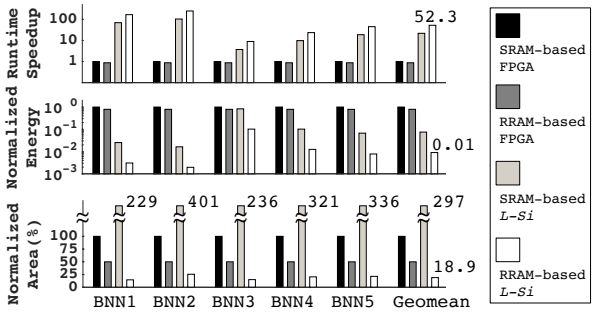

Figure 10. The runtime speedup (top), energy consumption (middle) and area (bottom) results are presented. All results are normalized to that of SRAM-based FPGA.

We note that a BNN implementation on GPU, TrueNorth and FPGA has also been evaluated in prior work [25][66][89], and FPGA achieves the best performance among these three architectures. Since *L-Si* achieves better performance than FPGA, and the same BNNs are used in our evaluation, this indicates that *L-Si* is more advantageous than these architectures in accelerating BNN workloads.

5.5 Multi-Context Architecture

In this subsection, we evaluate the effectiveness of *L-Si* architecture in supporting multi-context, as compared with the off-the-shelf FPGA (single-context) and multi-context FPGA. Note that we limit the scope of our evaluation to be context switching delay and die area, as these two are known obstacles for virtualizing FPGA [9].

Table 3. Architectural parameters to support virtualization.

Architecture	Multi-context Support	Context switching delay
Off-the-shelf FPGA	No	$17.52ms \sim 164.2ms$
L-Si	Yes	$\sim 10ns$

The key parameters of the *L-Si* and off-the-shelf FPGA are listed in Table 3. To ensure a fair comparison, we choose the latest Virtex UltraScale FPGAs from Xilinx [98] as our baseline. Since it is a single-context device, the context switching delay in FPGA is equivalent to its configuration time. The best-/worst-case context switching delay is estimated by fully exploiting the device features of Virtex UltraScale FPGAs including configuration rates and partial reconfiguration [96]. We show that the context switching delay in *L-Si* is at least *six orders of magnitude* shorter than that the latest FPGA devices.

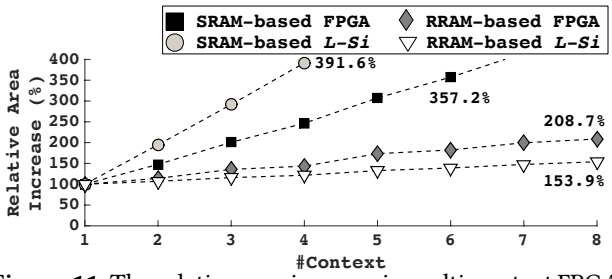

Figure 11. The relative area increase in multi-context FPGAs and *L-Si* for supporting more contexts. The relative area increase is calculated by normalizing the area of a multi-context architecture to that of its single-context version.

To evaluate the cost benefit of *L-Si* architecture in supporting multi-context, we further compare *L-Si* with multi-context FPGA (including RRAM-based and SRAM-based). Specifically, we measure the percentage increase in the die area when adding more contexts to the architecture for the four cases. Our evaluation shows that although these two types of architectures (*L-Si* vs. FPGA) have a comparable context switching delay, the relative area increase in *L-Si* is significantly smaller than that in multi-context FPGAs for implementing the same number of hardware contexts. We further observe that the relative area increase of RRAM-based *L-Si* is also smaller than that of the RRAM-based multi-context FPGA, indicating the area saving is not only from the RRAM technology but from the architecture. The key insight is that the crossbar structure in *L-Si* leads to better sharing of the multiplexers used for selecting contexts than the LUT structure in FPGA. For instance, under the same die area, the number of multiplexers needed in *L-Si* to support multi-contexts is **55.6**% fewer than that in FPGA. Nevertheless, the same area of *L-Si* can map larger applications than *L-Si* as discussed in Section 5.2-5.4.

6 Related Works

6.1 RRAM-based FPGA

Extensive research efforts have been devoted to investigate novel FPGA architectures based on non-volatile memory technologies [18][34][40][56][22][23]. In these works, non-volatile memory cells (e.g. RRAMs) are used as a direct drop-in replacement to either 1) the SRAM cells in CLBs; 2) the programmable switch circuits (CBs and SBs) comprising pass gates and the SRAM-based configuration memory in the routing fabric; or 3) the SRAM cells in BRAMs. Benefiting from the dense cell and non-volatility, these implementations reduce the chip area and power consumption *without* changing the basic architecture of FPGAs. Compared with these works, *L-Si* presents a fundamentally new architecture tailored to the RRAM technology that fully exploits the great potentials of the RRAM technology and the ultra-dense crossbar structure. A comprehensive comparison between RRAM-based FPGA and *L-Si* has already been discussed in Section 5.

6.2 Multi-context FPGA

Multi-context FPGA architectures [26][11][88] have been proposed by the research community but have not yet gained a market foothold. The reason for this is that these architectures typically incur a large area overhead, thereby reducing the available reconfigurable hardware resources for computation. Several circuit/architecture techniques have been proposed to reduce the area overhead but either at the cost of reduced routability [6] or increased fabrication cost [47]. Several recent works proposed to reduce the area overhead by replacing large SRAM cells with denser RRAM cells [40][84][83][52]. However, similar to other RRAM-based FPGAs in Section 6.1, the benefit was only from technology advancement, in contrast to *L-Si*, which exploits the merits of both technology and architecture (i.e., better hardware reuse and sharing).

Despite the cost barriers to adopt multi-context FPGA in practice, there has been a rich literature of developing reconfiguration management techniques which can take advantage of the fast context switching capability of multi-context FPGAs to minimize the reconfiguration overhead. These techniques include configuration scheduling [71][77][63], configuration prefetching [62][76] and configuration caching [55][82]. Note that, these techniques, though developed for multi-context FPGAs, can be *equally* applicable and beneficial to our multi-context *L-Si* architecture.

6.3 Neural Networks Accelerators

CMOS-based designs. A number of domain-specific neural network (NN) accelerators have been proposed recently. DianNao series [16][17][58][29] combine fine-grained parallel processing with distributed on-chip memory to accelerate neural network applications. However, the performance of these accelerators is bounded by on-chip memory. A few other NN accelerators have been developed to address this bottleneck through several innovations. SC-DCNN [75] applies stochastic computing, and replaces a multiplier with a simple AND gate to save chip area. EIE [39] investigates the sparsity of weights to save on-chip memory usage and performs sparse-matrix multiplication on compressed NNs. Minerva [74] proposes an algorithm/architecture co-design approach to reduce storage requirements and compute complexity by tuning the range and precision of each data signal and pruning neurons whose results are close to zero. It also exploits aggressive voltage scaling for SRAM to reduce power. Eyeriss [19] applies a novel dataflow (*row-stationary*) to maximize data reuse and minimize data movement. Finally, Neurocube [48] utilizes 3D stacking memory to increase the on-chip memory capacity/bandwidth. *Compared with L-Si, these accelerators are domain-specific and do not provide flexibility to accelerate diverse applications beyond neural networks, nor do they exploit the potentials of the RRAM technology.*

RRAM-based designs. Several interesting RRAM-based NN accelerators have been proposed recently [20][79][7][81] and demonstrate great promise in improving performance and energy efficiency. However, they are *domain-specific* accelerators, targeting neural network applications, whereas *L-Si* is a *general-purpose* architecture and can run any application. In addition, *L-Si* is much less complex by design, as it maximally reuses a crossbar to implement various functions (logic, memory, search and routing). In contrast, RRAM-based NN accelerators use a crossbar to only implement the dot-product operation, and require adding ALUs and control logic outside the memory array to implement other functions. They also need complex mixed-signal circuits e.g. A/D and D/A to support the dot-product operation which increases design complexity and area overhead (~60% in PRIME [20]).

Finally, the comparison among GPU, TrueNorth [31] and *L-Si* has already been discussed in Section 5.4.

6.4 Other Architectures

Automata processor [28] is another interesting architecture that leverages the memory array to accelerate pattern matching (e.g. regular expression search). It requires ALUs added to the memory array to facilitate the match operation and thus is a processing *near* (not inside) memory architecture. Additionally, *L-Si* inherently differs from ACDIMM [37], RRAM-based TCAM [106] and associative processor [100] since it is a dataflow architecture, while the others are based on SIMD. Finally, *L-Si* is also fundamentally different from the coarse-grained reconfigurable architecture (CGRA) which uses ALUs as basic building blocks to trade off gate-level reconfigurability for efficiency. CGRA, largely shares the same limitations as FPGA in providing configuration support for on-chip memory and light-weight computation (TCAM) [86].

7 Conclusion

This paper proposes a reconfigurable memory-oriented computing fabric (*L-Si*) enabled by RRAM technology and its ultra-dense crossbar structure. *L-Si* effectively addresses the limitations of state-of-the-art FPGAs by introducing a new building block and a new architectural organization. Our evaluation confirms its effectiveness in accelerating emerging search-/data-intensive applications and lowering the cost barrier of adding multi-context support in reconfigurable architecture to assist virtualization, compared with off-the-shelf FPGAs. With technology scaling of RRAM, we expect *L-Si* to gain more benefits as tile design can be simplified due to reduced write delay and voltage in scaled RRAM.

In general, we believe *L-Si* is a promising *complement* to existing reconfigurable architectures (FPGA, CGRA, etc.) and effectively extends the concept of reconfigurable computing from computation towards the memory domain. It can potentially open up rich research opportunities in driving new memory-oriented reconfigurable architectures, programming model/compiler tools, developing new search-/data-intensive applications which traditionally were not considered to be amenable to FPGA-like accelerations, and developing new virtualization techniques and performing comprehensive system evaluations. Future works on these topics are planned.

Acknowledgements

We appreciate the insightful comments and feedback from Prof. Shan Lu (U. Chicago), Prof. David Wood (UW-Madison), Prof. Mark Hill (UW-Madison), Prof. Onur Mutlu (ETH Zurich/CMU) and the anonymous reviewers. This work was supported by DARPA Young Faculty Award D16AP00122.

References

[1] Michael Adler, Kermin E Fleming, Angshuman Parashar, Michael Pellauer, and Joel Emer. 2011. Leap scratchpads: automatic memory and cache management for reconfigurable logic. In *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays*. ACM, 25–28.

[2] Jasmin Ajanovic. 2008. PCI Express*(PCIe*) 3.0 Accelerator Features. *Intel Corporation* (2008), 10.

[3] Amazon. 2016. Amazon EC2 F1 Instances. https://aws.amazon.com/ec2/instance-types/f1/. (2016).

[4] Mikhail Asiatici, Nithin George, Kizheppatt Vipin, Suhaib A Fahmy, and Paolo Ienne. 2017. Virtualized execution runtime for FPGA accelerators in the cloud. *Ieee Access* 5 (2017), 1900–1910.

[5] ASU. [n. d.]. Predictive Technology Model (PTM). http://ptm.asu.edu/. ([n. d.]).

[6] V Baena-Lecuyer, MA Aguirre, A Torralba, Leopoldo García Franquelo, and Julio Faura. 1999. Decoder-driven switching matrices in multi-context fpgas: area reduction and their effect on routability. In *Circuits and Systems, 1999. ISCAS'99. Proceedings of the 1999 IEEE International Symposium on*, Vol. 1. IEEE, 463–466.

[7] Mahdi Nazm Bojnordi and Engin Ipek. 2016. Memristive boltzmann machine: A hardware accelerator for combinatorial optimization and deep learning. In *High Performance Computer Architecture (HPCA), 2016 IEEE International Symposium on*. IEEE, 1–13.

[8] Jeremy Buhler. 2001. Efficient large-scale sequence comparison by locality-sensitive hashing. *Bioinformatics* 17, 5 (2001), 419–428.

[9] Stuart Byma, J Gregory Steffan, Hadi Bannazadeh, Alberto Leon Garcia, and Paul Chow. 2014. Fpgas in the cloud: Booting virtualized hardware accelerators with openstack. In *Field-Programmable Custom Computing Machines (FCCM), 2014 IEEE 22nd Annual International Symposium on*. IEEE, 109–116.

[10] Adrian M Caulfield, Eric S Chung, Andrew Putnam, Hari Angepat, Jeremy Fowers, Michael Haselman, Stephen Heil, Matt Humphrey, Puneet Kaur, Joo-Young Kim, et al. 2016. A cloud-scale acceleration architecture. In *Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on*. IEEE, 1–13.

[11] Douglas Chang and Malgorzata Marek-Sadowska. 1999. Partitioning sequential circuits on dynamically reconfigurable FPGAs. *IEEE Trans. Comput.* 48, 6 (1999), 565–578.

[12] Meng-Fan Chang, Chien-Chen Lin, Albert Lee, Chia-Chen Kuo, Geng-Hau Yang, Hsiang-Jen Tsai, Tien-Fu Chen, Shyh-Shyuan Sheu, Pei-Ling Tseng, Heng-Yuan Lee, et al. 2015. 17.5 A 3T1R nonvolatile TCAM using MLC ReRAM with Sub-1ns search time. In *Solid-State Circuits Conference-(ISSCC), 2015 IEEE International*. IEEE, 1–3.

[13] An Chen. 2013. A comprehensive crossbar array model with solutions for line resistance and nonlinear device characteristics. *IEEE Transactions on Electron Devices* 60, 4 (2013), 1318–1326.

[14] Fei Chen, Yi Shan, Yu Zhang, Yu Wang, Hubertus Franke, Xiaotao Chang, and Kun Wang. 2014. Enabling FPGAs in the cloud. In *Proceedings of the 11th ACM Conference on Computing Frontiers*. ACM, 3.

[15] Hong-Yu Chen, Stefano Brivio, Che-Chia Chang, Jacopo Frascaroli, Tuo-Hung Hou, Boris Hudec, Ming Liu, Hangbing Lv, Gabriel Molas, Joon Sohn, et al. 2017. Resistive random access memory (RRAM) technology: From material, device, selector, 3D integration to bottom-up fabrication. *Journal of Electroceramics* (2017), 1–18.

[16] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. 2014. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. In *ACM Sigplan Notices*, Vol. 49. ACM, 269–284.

[17] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, et al. 2014. Dadiannao: A machine-learning supercomputer. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 609–622.

[18] Yi-Chung Chen, Wenhua Wang, Hai Li, and Wei Zhang. 2012. Non-volatile 3D stacking RRAM-based FPGA. In *Field Programmable Logic and Applications (FPL), 2012 22nd International Conference on*. IEEE, 367–372.

[19] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. 2016. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*. IEEE, 367–379.

[20] Ping Chi, Shuangchen Li, Cong Xu, Tao Zhang, Jishen Zhao, Yongpan Liu, Yu Wang, and Yuan Xie. 2016. PRIME: A novel processing-in-memory architecture for neural network computation in reram-based main memory. In *Proceedings of the 43rd International Symposium on Computer Architecture*. IEEE Press, 27–39.

[21] Eric S Chung, James C Hoe, and Ken Mai. 2011. CoRAM: an in-fabric memory architecture for FPGA-based computing. In *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays*. ACM, 97–106.

[22] Jason Cong and Bingjun Xiao. 2011. mrFPGA: A novel FPGA architecture with memristor-based reconfiguration. In *Proceedings of the 2011 IEEE/ACM International Symposium on Nanoscale Architectures*. IEEE Computer Society, 1–8.

[23] Jason Cong and Bingjun Xiao. 2014. FPGA-RPI: A novel FPGA architecture with RRAM-based programmable interconnects. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 22, 4 (2014), 864–877.

[24] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. 2015. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in Neural Information Processing Systems*. 3123–3131.

[25] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830* (2016).

[26] André DeHon. 1996. DPGA utilization and application. In *Proceedings of the 1996 ACM fourth international symposium on Field-programmable gate arrays*. ACM, 115–121.

[27] André M. DeHon. 2013. Location, Location, Location: The Role of Spatial Locality in Asymptotic Energy Minimization. In *FPGA*. ACM, New York, NY, USA, 137–146. https://doi.org/10.1145/2435264.2435291

[28] Paul Dlugosch, Dave Brown, Paul Glendenning, Michael Leventhal, and Harold Noyes. 2014. An efficient and scalable semiconductor architecture for parallel automata processing. *IEEE Transactions on Parallel and Distributed Systems* 25, 12 (2014), 3088–3098.

[29] Zidong Du, Robert Fasthuber, Tianshi Chen, Paolo Ienne, Ling Li, Tao Luo, Xiaobing Feng, Yunji Chen, and Olivier Temam. 2015. ShiDianNao: Shifting vision processing closer to the sensor. In *ACM SIGARCH Computer Architecture News*, Vol. 43. ACM, 92–104.

[30] James P Durbano and Fernando E Ortiz. 2004. FPGA-based acceleration of the 3D finite-difference time-domain method. In *Field-Programmable Custom Computing Machines, 2004. FCCM 2004. 12th Annual IEEE Symposium on*. IEEE, 156–163.

[31] Steven K Esser, Paul A Merolla, John V Arthur, Andrew S Cassidy, Rathinakumar Appuswamy, Alexander Andreopoulos, David J Berg, Jeffrey L McKinstry, Timothy Melano, Davis R Barch, et al. 2016. Convolutional networks for fast, energy-efficient neuromorphic computing. *Proceedings of the National Academy of Sciences* (2016), 201604850.

[32] Suhaib A Fahmy, Kizheppatt Vipin, and Shanker Shreejith. 2015. Virtualized FPGA accelerators for efficient cloud computing. In *Cloud Computing Technology and Science (CloudCom), 2015 IEEE 7th International Conference on*. IEEE, 430–435.

[33] Amin Farmahini-Farahani, Jung Ho Ahn, Katherine Morrow, and Nam Sung Kim. 2015. Drama: An architecture for accelerated processing near memory. *IEEE Computer Architecture Letters* 14, 1 (2015), 26–29.

[34] Pierre-Emmanuel Gaillardon, Davide Sacchetto, Shashikanth Bobba, Yusuf Leblebici, and Giovanni De Micheli. 2012. GMS: Generic memristive structure for non-volatile FPGAs. In *VLSI and System-on-Chip, 2012 (VLSI-SoC), IEEE/IFIP 20th International Conference on*. IEEE, 94–98.

[35] Yijin Guan, Hao Liang, Ningyi Xu, Wenqiang Wang, Shaoshuai Shi, Xi Chen, Guangyu Sun, Wei Zhang, and Jason Cong. 2017. FP-DNN: An Automated Framework for Mapping Deep Neural Networks onto FPGAs with RTL-HLS Hybrid Templates. In *Field-Programmable Custom Computing Machines (FCCM), 2017 IEEE 25th Annual International Symposium on*. IEEE, 152–159.

[36] Qing Guo, Xiaochen Guo, Yuxin Bai, and Engin Ipek. 2011. A resistive TCAM accelerator for data-intensive computing. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 339–350.

[37] Qing Guo, Xiaochen Guo, Ravi Patel, Engin Ipek, and Eby G Friedman. 2013. Ac-dimm: associative computing with stt-mram. In *ACM SIGARCH Computer Architecture News*, Vol. 41. ACM, 189–200.

[38] Robert J Halstead, Bharat Sukhwani, Hong Min, Mathew Thoennes, Parijat Dube, Sameh Asaad, and Balakrishna Iyer. 2013. Accelerating join operation for relational databases with FPGAs. In *Field-Programmable Custom Computing Machines (FCCM), 2013 IEEE 21st Annual International Symposium on*. IEEE, 17–20.

[39] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. 2016. EIE: efficient inference engine on compressed deep neural network. In *Proceedings of the 43rd International Symposium on Computer Architecture*. IEEE Press, 243–254.

[40] Kejie Huang, Yajun Ha, Rong Zhao, Akash Kumar, and Yong Lian. 2014. A low active leakage and high reliability phase change memory (PCM) based non-volatile FPGA storage element. *IEEE Transactions on Circuits and Systems I: Regular Papers* 61, 9 (2014), 2605–2613.

[41] Intel [n. d.]. Intel FPGA SDK for OpenCL. https://www.altera.com/products/design-software/embedded-software-developers/opencl/overview.html. ([n. d.]).

[42] Intel. [n. d.]. Intel QuickPath Interconnect. http://www.intel.com/content/www/us/en/io/quickpath-technology/quickpath-technology-general.html. ([n. d.]).

[43] Intel. 2017. Intel Collaborates with Alibaba Cloud to Help Customers Accelerate Business Applications. (2017).

[44] W. Jiang. 2013. Scalable Ternary Content Addressable Memory implementation using FPGAs. In *ANCS*. 71–82. https://doi.org/10.1109/ANCS.2013.6665177

[45] Sung Hyun Jo, Tanmay Kumar, Sundar Narayanan, Wei D Lu, and Hagop Nazarian. 2014. 3D-stackable crossbar resistive memory based on field assisted superlinear threshold (FAST) selector. In *Electron Devices Meeting (IEDM), 2014 IEEE International*. IEEE, 6–7.

[46] Akifumi Kawahara, Ryotaro Azuma, Yuuichirou Ikeda, Ken Kawai, Yoshikazu Katoh, Yukio Hayakawa, Kiyotaka Tsuji, Shinichi Yoneda, Atsushi Himeno, Kazuhiko Shimakawa, et al. 2013. An 8 Mb multi-layered cross-point ReRAM macro with 443 MB/s write throughput. *IEEE Journal of Solid-State Circuits* 48, 1 (2013), 178–185.

[47] Daisuke Kawakami, Yuichro Shibata, and Hideharu Amano. 2001. A prototype chip of multicontext FPGA with DRAM for Virtual Hardware. In *Proceedings of the 2001 Asia and South Pacific Design Automation Conference*. ACM, 17–18.

[48] Duckhwan Kim, Jaeha Kung, Sek Chai, Sudhakar Yalamanchili, and Saibal Mukhopadhyay. 2016. Neurocube: A programmable digital neuromorphic architecture with high-density 3D memory. In *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*. IEEE, 380–392.

[49] Minje Kim and Paris Smaragdis. 2016. Bitwise neural networks. *arXiv preprint arXiv:1601.06071* (2016).

[50] Oliver Knodel and Rainer G Spallek. 2015. RC3E: provision and management of reconfigurable hardware accelerators in a cloud environment. *arXiv preprint arXiv:1508.06843* (2015).

[51] Brian Kulis and Kristen Grauman. 2009. Kernelized locality-sensitive hashing for scalable image search. In *Computer Vision, 2009 IEEE 12th International Conference on*. IEEE, 2130–2137.

[52] Yahya Lakys, Weisheng Zhao, Jacques-Olivier Klein, and Claude Chappert. 2012. MRAM crossbar based configurable logic block. In *Circuits and Systems (ISCAS), 2012 IEEE International Symposium on*. IEEE, 2945–2948.

[53] Myoung-Jae Lee et al. 2011. A fast, high-endurance and scalable non-volatile memory device made from asymmetric Ta2O5-x/TaO2-x bilayer structures. *Nature materials* 10, 8 (2011), 625–630.

[54] Jing Li, Robert K Montoye, Masatoshi Ishii, and Leland Chang. 2014. 1 Mb 0.41 μm^2 2T-2R cell nonvolatile TCAM with two-bit encoding and clocked self-referenced sensing. *IEEE Journal of Solid-State Circuits* 49, 4 (2014), 896–907.

[55] Zhiyuan Li, Katherine Compton, and Scott Hauck. 2000. Configuration caching management techniques for reconfigurable computing. In *Field-Programmable Custom Computing Machines, 2000 IEEE Symposium on*. IEEE, 22–36.

[56] Young Yang Liauw, Zhiping Zhang, Wanki Kim, Abbas El Gamal, and S Simon Wong. 2012. Nonvolatile 3D-FPGA with monolithically stacked RRAM-based configuration memory. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2012 IEEE International*. IEEE, 406–408.

[57] Chien-Chen Lin, Jui-Yu Hung, Wen-Zhang Lin, Chieh-Pu Lo, Yen-Ning Chiang, Hsiang-Jen Tsai, Geng-Hau Yang, Ya-Chin King, Chrong Jung Lin, Tien-Fu Chen, et al. 2016. 7.4 A 256b-wordlength ReRAM-based TCAM with 1ns search-time and 14× improvement in wordlength-energyefficiency-density product using 2.5 T1R cell. In *Solid-State Circuits Conference (ISSCC), 2016 IEEE International*. IEEE, 136–137.

[58] Daofu Liu, Tianshi Chen, Shaoli Liu, Jinhong Zhou, Shengyuan Zhou, Olivier Teman, Xiaobing Feng, Xuehai Zhou, and Yunji Chen. 2015. Pudiannao: A polyvalent machine learning accelerator. In *ACM SIGARCH Computer Architecture News*, Vol. 43. ACM, 369–381.

[59] Jason Luu et al. 2014. VTR 7.0: Next generation architecture and CAD system for FPGAs. *TRETS* 7, 2 (2014), 6.

[60] Jason Luu, Ian Kuon, Peter Jamieson, Ted Campbell, Andy Ye, Wei Mark Fang, Kenneth Kent, and Jonathan Rose. 2011. VPR 5.0: FPGA cad and architecture exploration tools with single-driver routing, heterogeneity and process scaling. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 4, 4 (2011), 32.

[61] Qin Lv, William Josephson, Zhe Wang, Moses Charikar, and Kai Li. 2007. Multi-probe LSH: efficient indexing for high-dimensional similarity search. In *Proceedings of the 33rd international conference on Very large data bases*. VLDB Endowment, 950–961.

[62] Rafael Maestre, Milagros Fernandez, Fadi J Kurdahi, Nader Bagherzadeh, and Hartej Singh. 2000. Configuration management in multi-context reconfigurable systems for simultaneous performance and power optimizations. In *Proceedings of the 13th international symposium on System synthesis*. IEEE Computer Society, 107–113.

[63] Rafael Maestre, Fadi J Kurdahi, Milagros Fernández, Roman Hermida, Nader Bagherzadeh, and Hartej Singh. 2001. A framework for reconfigurable computing: task scheduling and context management. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 9, 6 (2001), 858–873.

[64] Alan Mishchenko, Sungmin Cho, Satrajit Chatterjee, and Robert Brayton. 2007. Combinational and sequential mapping with priority cuts. In *Proceedings of the 2007 IEEE/ACM international conference on Computer-aided design*. IEEE Press, 354–361.

[65] Raphael Njuguna. 2008. A survey of FPGA benchmarks. *Project Report, November* 24 (2008).

[66] Eriko Nurvitadhi, David Sheffield, Jaewoong Sim, Asit Mishra, Ganesh Venkatesh, and Debbie Marr. 2016. Accelerating Binarized Neural Networks: Comparison of FPGA, CPU, GPU, and ASIC. *Proc. ICFPT* (2016).

[67] Corey B Olson, Maria Kim, Cooper Clauson, Boris Kogon, Carl Ebeling, Scott Hauck, and Walter L Ruzzo. 2012. Hardware acceleration of short read mapping. In *Field-Programmable Custom Computing Machines (FCCM), 2012 IEEE 20th Annual International Symposium on*. IEEE, 161–168.

[68] Jian Ouyang, Shiding Lin, Wei Qi, Yong Wang, Bo Yu, and Song Jiang. 2014. SDA: Software-defined accelerator for large-scale DNN systems. In *Hot Chips 26 Symposium (HCS), 2014 IEEE*. IEEE, 1–23.

[69] Kostas Pagiamtzis and Ali Sheikholeslami. 2006. Content-addressable memory (CAM) circuits and architectures: A tutorial and survey. *IEEE Journal of Solid-State Circuits* 41, 3 (2006), 712–727.

[70] Andrew Putnam, Adrian M Caulfield, Eric S Chung, Derek Chiou, Kypros Constantinides, John Demme, Hadi Esmaeilzadeh, Jeremy Fowers, Gopi Prashanth Gopal, Jan Gray, et al. 2014. A reconfigurable fabric for accelerating large-scale datacenter services. In *Computer Architecture (ISCA), 2014 ACM/IEEE 41st International Symposium on*. IEEE, 13–24.

[71] Yang Qu, Juha-Pekka Soininen, and Jari Nurmi. 2007. Static scheduling techniques for dependent tasks on dynamically reconfigurable devices.

Journal of Systems Architecture 53, 11 (2007), 861–876.

[72] Colby Ranger, Ramanan Raghuraman, Arun Penmetsa, Gary Bradski, and Christos Kozyrakis. 2007. Evaluating mapreduce for multi-core and multiprocessor systems. In *High Performance Computer Architecture, 2007. HPCA 2007. IEEE 13th International Symposium on*. Ieee, 13–24.

[73] Deepak Ravichandran, Patrick Pantel, and Eduard Hovy. 2005. Randomized algorithms and nlp: using locality sensitive hash function for high speed noun clustering. In *Proceedings of the 43rd annual meeting on association for computational linguistics*. Association for Computational Linguistics, 622–629.

[74] Brandon Reagen, Paul Whatmough, Robert Adolf, Saketh Rama, Hyunkwang Lee, Sae Kyu Lee, José Miguel Hernández-Lobato, Gu-Yeon Wei, and David Brooks. 2016. Minerva: Enabling low-power, highly-accurate deep neural network accelerators. In *Proceedings of the 43rd International Symposium on Computer Architecture*. IEEE Press, 267–278.

[75] Ao Ren, Ji Li, Zhe Li, Caiwen Ding, Xuehai Qian, Qinru Qiu, Bo Yuan, and Yanzhi Wang. 2016. Sc-dcnn: highly-scalable deep convolutional neural network using stochastic computing. *arXiv preprint arXiv:1611.05939* (2016).

[76] Javier Resano, Daniel Mozos, and Francky Catthoor. 2005. A hybrid prefetch scheduling heuristic to minimize at run-time the reconfiguration overhead of dynamically reconfigurable hardware. In *Proceedings of the conference on Design, Automation and Test in Europe-Volume 1*. IEEE Computer Society, 106–111.

[77] Javier Resano, Daniel Mozos, Diederik Verkest, Francky Catthoor, and Serge Vernalde. 2004. Specific scheduling support to minimize the reconfiguration overhead of dynamically reconfigurable hardware. In *Design Automation Conference, 2004. Proceedings. 41st*. IEEE, 119–124.

[78] Herman Schmit, David Whelihan, Andrew Tsai, Matthew Moe, Benjamin Levine, and R Reed Taylor. 2002. PipeRench: A virtualized programmable datapath in 0.18 micron technology. In *Custom Integrated Circuits Conference, 2002. Proceedings of the IEEE 2002*. IEEE, 63–66.

[79] Ali Shafiee, Anirban Nag, Naveen Muralimanohar, Rajeev Balasubramonian, John Paul Strachan, Miao Hu, R Stanley Williams, and Vivek Srikumar. 2016. ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. In *Proceedings of the 43rd International Symposium on Computer Architecture*. IEEE Press, 14–26.

[80] Rajendra Shinde, Ashish Goel, Pankaj Gupta, and Debojyoti Dutta. 2010. Similarity search and locality sensitive hashing using ternary content addressable memories. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. ACM, 375–386.

[81] Linghao Song, Xuehai Qian, Hai Li, and Yiran Chen. 2017. PipeLayer: A pipelined ReRAM-based accelerator for deep learning. In *High Performance Computer Architecture (HPCA), 2017 IEEE International Symposium on*. IEEE.

[82] Suraj Sudhir, Suman Nath, and Seth Copen Goldstein. 2001. Configuration caching and swapping. In *FPL*, Vol. 1. Springer, 192–202.

[83] Daisuke Suzuki and Takahiro Hanyu. 2015. Nonvolatile field-programmable gate array using 2-transistor–1-MTJ-cell-based multi-context array for power and area efficient dynamically reconfigurable logic. *Japanese Journal of Applied Physics* 54, 4S (2015), 04DE01.

[84] Kosuke Tatsumura, Masato Oda, and Shinichi Yasuda. 2014. A pure-CMOS nonvolatile multi-context configuration memory for dynamically reconfigurable FPGAs. In *Field-Programmable Technology (FPT), 2014 International Conference on*. IEEE, 215–222.

[85] David E Taylor and Jonathan S Turner. 2007. Classbench: A packet classification benchmark. *IEEE/ACM Transactions on Networking (TON)* 15, 3 (2007), 499–511.

[86] Michael Bedford Taylor, Jason Kim, Jason Miller, David Wentzlaff, Fae Ghodrat, Ben Greenwald, Henry Hoffman, Paul Johnson, Jae-Wook Lee, Walter Lee, et al. 2002. The raw microprocessor: A computational fabric for software circuits and general-purpose programs. *IEEE micro* 22, 2 (2002), 25–35.

[87] Antonio C Torrezan et al. 2011. Sub-nanosecond switching of a tantalum oxide memristor. *Nanotechnology* 22, 48 (2011), 485203.

[88] Steven Trimberger, Dean Carberry, Anders Johnson, and Jennifer Wong. 1997. A time-multiplexed FPGA. In *Field-Programmable Custom Computing Machines, 1997. Proceedings., the 5th Annual IEEE Symposium on*. IEEE, 22–28.

[89] Yaman Umuroglu, Nicholas J Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers. 2016. FINN: A Framework for Fast, Scalable Binarized Neural Network Inference. *arXiv preprint arXiv:1612.07119* (2016).

[90] Zeke Wang, Shuhao Zhang, Bingsheng He, and Wei Zhang. 2016. Melia: A mapreduce framework on opencl-based fpgas. *IEEE Transactions on Parallel and Distributed Systems* 27, 12 (2016), 3547–3560.

[91] Z. Wei, Y. Kanzawa, K. Arita, Y. Katoh, K. Kawai, S. Muraoka, S. Mitani, S. Fujii, K. Katayama, M. Iijima, T. Mikawa, T. Ninomiya, R. Miyanaga, Y. Kawashima, K. Tsuji, A. Himeno, T. Okada, R. Azuma, K. Shimakawa, H. Sugaya, T. Takagi, R. Yasuhara, K. Horiba, H. Kumigashira, and M. Oshima. 2008. Highly reliable TaOx ReRAM and direct evidence of redox reaction mechanism. In *2008 IEEE International Electron Devices Meeting*. 1–4. https://doi.org/10.1109/IEDM.2008.4796676

[92] Z. Wei, T. Ninomiya, S. Muraoka, K. Katayama, R. Yasuhara, and T. Mikawa. 2014. Switching and reliability mechanisms for ReRAM. In *IEEE International Interconnect Technology Conference*. 349–352. https://doi.org/10.1109/IITC.2014.6831832

[93] Bruce Wile. 2014. Coherent accelerator processor interface (CAPI) for POWER8 systems. *IBM White Paper* (2014).

[94] H-S Philip Wong et al. 2012. Metal–oxide RRAM. *Proc. IEEE* 100, 6 (2012), 1951–1970.

[95] Xilinx. [n. d.]. 7 Series FPGAs Configuration User Guide. https://www.xilinx.com/support/documentation/user_guides/ug470_7Series_Config.pdf. ([n. d.]).

[96] Xilinx. [n. d.]. Vivado Design Suite User Guide: Partial Reconfiguration. https://www.xilinx.com/support/documentation/sw_manuals/xilinx2015_4/ug909-vivado-partial-reconfiguration.pdf. ([n. d.]).

[97] Xilinx. 2016. 7 Series FPGAs Configurable Logic Block User Guide. (2016). https://www.xilinx.com/support/documentation/user_guides/ug474_7Series_CLB.pdf

[98] Xilinx. 2017. UltraScale Architecture and Product Data Sheet: Overview. (2017). https://www.xilinx.com/support/documentation/data_sheets/ds890-ultrascale-overview.pdf

[99] Saeyang Yang. 1991. *Logic synthesis and optimization benchmarks user guide: version 3.0*. MCNC.

[100] Leonid Yavits, Shahar Kvatinsky, Amir Morad, and Ran Ginosar. 2015. Resistive associative processor. *IEEE Computer Architecture Letters* 14 (2015), 148–151.

[101] Yue Zha and Jing Li. 2016. Reconfigurable in-memory computing with resistive memory crossbar. In *Computer-Aided Design (ICCAD), 2016 IEEE/ACM International Conference on*. IEEE, 1–8.

[102] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. 2015. Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks. In *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '15)*. ACM, New York, NY, USA, 161–170. https://doi.org/10.1145/2684746.2689060

[103] Jialiang Zhang and Jing Li. 2017. Improving the Performance of OpenCL-based FPGA Accelerator for Convolutional Neural Network. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '17)*. ACM, New York, NY, USA, 25–34. https://doi.org/10.1145/3020078.3021698

[104] Jialiang Zhang and Jing Li. 2017. Improving the Performance of OpenCL-based FPGA Accelerator for Convolutional Neural Network. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 25–34.

[105] Ritchie Zhao, Weinan Song, Wentao Zhang, Tianwei Xing, Jeng-Hau Lin, Mani Srivastava, Rajesh Gupta, and Zhiru Zhang. 2017. Accelerating Binarized Convolutional Neural Networks with Software-Programmable FPGAs. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '17)*. ACM, New York, NY, USA, 15–24. https://doi.org/10.1145/3020078.3021741

[106] Le Zheng, Sangho Shin, Scott Lloyd, Maya Gokhale, Kyungmin Kim, and Sung-Mo Kang. 2016. RRAM-based TCAMs for pattern search. In *Circuits and Systems (ISCAS), 2016 IEEE International Symposium on*. IEEE, 1382–1385.