# BLADE: A BitLine Accelerator for Devices on the Edge

William Andrew Simon, Yasir Mahmood Qureshi
Alexandre Levisse, Marina Zapater, David Atienza
Embedded Systems Lab (ESL), Swiss Institute of Technology Lausanne (EPFL)
Lausanne, Switzerland
{william.simon,yasir.qureshi,alexandre.levisse,marina.zapater,david.atienza}@epfl.ch

## ABSTRACT

The increasing ubiquity of edge devices in the consumer market, along with their ever more computationally expensive workloads, necessitate corresponding increases in computing power to support such workloads. In-memory computing is attractive in edge devices as it reuses preexisting memory elements, thus limiting area overhead. Additionally, in-SRAM Computing (iSC) efficiently performs computations on spatially local data found in a variety of emerging edge device workloads. We therefore propose, implement, and benchmark BLADE, a BitLine Accelerator for Devices on the Edge. BLADE is an iSC architecture that can perform massive SIMD-like complex operations on hundreds to thousands of operands simultaneously. We implement BLADE in 28nm CMOS and demonstrate its functionality down to 0.6V, lower than any conventional state-of-the-art iSC architecture. We also benchmark BLADE in conjunction with a full Linux software stack in the gem5 architectural simulator, providing a robust demonstration of its performance gain in comparison to an equivalent embedded processor equipped with a NEON SIMD co-processor. We benchmark BLADE with three emerging edge device workloads, namely cryptography, high efficiency video coding, and convolutional neural networks, and demonstrate 4x, 6x, and 3x performance improvement, respectively, in comparison to a baseline CPU/NEON processor at an equivalent power budget.

## KEYWORDS

In-memory processing, In-SRAM processing, Bitline computing, Edge computing.

**ACM Reference Format:**
William Andrew Simon, Yasir Mahmood Qureshi and Alexandre Levisse, Marina Zapater, David Atienza. 2019. BLADE: A BitLine Accelerator for Devices on the Edge . In *Great Lakes Symposium on VLSI 2019 (GLSVLSI '19), May 9–11, 2019, Tysons Corner, VA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3299874.3317979

## 1 INTRODUCTION

As of 2018, 41% of Americans owned a digital home assistant, e.g. Amazon's Alexa or Google's Home, and over 25% owned more than three smart home devices [16]. In the same year, 77% of Americans owned a smartphone, for the first time overtaking PC ownership at just 75% [1]. As edge device adoption accelerates, their compute power must increase correspondingly to meet application demands.

As edge devices are naturally limited in physical size and power, innovations must be developed that boost performance while maintaining low area and power overheads. In this context, in-memory computing presents an attractive solution as it boosts compute power through preexisting hardware elements, while also reducing power consumption due to data movement [3]. Much research in this field focuses on in-DRAM computing to reduce data movement and take advantage of the logic layer available in 3D DRAM stacks [12, 17, 25]. However, the emerging field of in-SRAM Computing (iSC) also shows promise as it takes advantage of SRAM array bitline/wordline structure and the contained data's inherent locality to perform SIMD-like operations without involving the processor [3]. Different iSC architectures have been explored, from simple bitwise logic caches [3] to transposing caches that perform complex operations [14] and large, complex 10T-based arrays that can fully pipeline many operations [4]. However, each of these architectures have drawbacks in terms of computational complexity, size, and performance. Currently there are no iSC architectures designed specifically to address the unique requirements of edge device physical design and workloads.

To fulfill this need, we propose BLADE, a BitLine Accelerator for Devices on the Edge. Its physical and architectural characteristics are ideal for small edge device caches. In particular, BLADE performs complex operations such as `add`, `multiply`, and `greater/less than`, in a massive (i.e. 1024 bitwise/128 8bit) SIMD-like fashion, necessary for increasingly complex workloads. By utilizing conventional 6T bitcells and standard bitline multiplexing, BLADE maintains a low (8%) area overhead. Finally, utilizing local bitlines gives BLADE the most efficient voltage/frequency Pareto curve down to 0.6V of any state-of-the-art iSC architecture.

We validate BLADE at the electrical and system level to demonstrate its correct functionality as well as applicability to emerging edge device workloads. From an electrical standpoint, we implement BLADE in 28nm bulk CMOS technology and validate its reliable operation from 0.6V (415Mhz) to 1V (2.2Ghz) , taking into account process variations and post-layout parasitics and extracting timing and energy characteristics. To demonstrate system-level functionality, we extend the gem5 [8] architectural simulator with BLADE, integrating the previously acquired timing values to create a timing accurate simulator capable of running any workload on a full Linux stack. Our robust electrical simulations combined with a realistic, full software stack allows us to benchmark BLADE acceleration on emerging edge device workloads such as cryptography, video processing, and neural networks, while taking into account system level events such as cache misses, system calls, and architecture specific bottlenecks. BLADE is in fact the first complex operation iSC architecture to be profiled on a full software stack.


*GLSVLSI '19, May 9–11, 2019, Tysons Corner, VA, USA*

ACM ISBN 978-1-4503-6252-8/19/05…$15.00
https://doi.org/10.1145/3299874.3317979

Thus, the contributions of this paper are as follows:

- We propose BLADE, the first iSC architecture designed specifically for edge devices.
- We implement BLADE in 28nm bulk CMOS, demonstrating reliable operation down to 0.6V without performance degradation (416Mhz) - a first for iSC architectures.
- We design a timing-accurate gem5 extension for BLADE, allowing workload profiling on a full software stack.
- We profile BLADE with the primary kernels of three edge device workloads, namely cryptography, image processing, and neural networks, demonstrating 4x, 6x, and 3x performance improvements, respectively, against an identical embedded processor equipped with a NEON SIMD co-processor.

The rest of the paper is structured as follows. Section 2 discusses state-of-the-art iSC architectures. Section 3 describes the edge device workloads we use to profile BLADE. Section 4 explains BLADE's bitline architecture and its optimization to edge devices. Section 5 describes our benchmarking methodology in gem5, and finally Section 6 presents and analyzes benchmark results.

## 2 RELATED WORK - BITLINE COMPUTING

BitLine (BL) computing utilizes SRAM structures, cache or otherwise, to perform computations at a much lower transistor count than a standard logic circuit. Jeloka *et al.* [21] demonstrate that sensing the BL and BL bar of simultaneously activated bitcells results in an `and` or `nor` operation of the bitcell values, respectively. Compute Caches [3] extend the number of operations with the addition of extra logic to support a `xor` operation, as illustrated in Figure 1. With this base BL logic architecture, complex operations can be achieved through various means. Neural Cache [14] performs bit serial complex operations by transposing operands to occupy a single BL and performing series of bitwise operations on them. DRC$^2$ [4] implements carry logic across BLs to facilitate complex operations. Finally, Dong *et al.* [11] rely on deeply depleted channel technology and an unconventional 4T bitcell design to achieve an ultra-low voltage iSC architecture.

While these proposals highlight the potential of iSC, each has drawbacks, both in general and in the context of edge device computing. In particular, DRC$^2$'s architecture deviates significantly from typical cache structure as it utilizes 10T SRAM cells and complex periphery for maximum performance, resulting in what is more an accelerator than a cache. Alternatively, the ultra low voltage (0.3V) iSC architecture presented in [11] relies on unconventional CMOS technology (deeply depleted channel) and modified 4T bitcells, known to suffer from stability issues [9] and disturb risks, as well as showing poor performance with voltage scaling (100Mhz@0.6V).

While such architectures may be beneficial to edge devices, they are in veins of research parallel to this work. Compute Caches and Neural Cache, on the other hand, are reliant on the 6T bitcell architecture presented in [21], which has been demonstrated to function in a frequency/voltage range of 800MHz/1V to 70MHz/0.7V. Neural Cache assumes iSC frequency can be improved up to 2.5GHz, however the cache upon which Neural Cache's profiling is based cannot achieve such frequencies at <0.85V [20], well above the 0.66V wordline voltage limitation presented in the work. Neural Cache also relies on bit serial computation to perform complex operations.

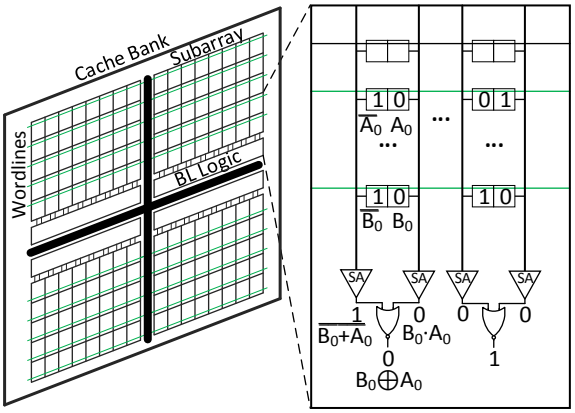


**Figure 1: Cache subarray with AND/NOR/XOR BL computing.**

While this increases the number of parallel operations, the resultant latency from bit serial computation reduces the throughput such that it is comparable to more standard approaches utilizing carry logic, as demonstrated in DRC$^2$ and in this work. Lastly, Neural Cache's reliance on operand transposition reduces the operation granularity at which such an architecture can improve performance; in other words, mingling iSC and CPU operations results in a significant transposition overhead, mitigating performance gains.

In contrast to the above mentioned approaches, BLADE specifically targets edge devices by utilizing a conventional 6T SRAM bitcell design with shared read ports to avoid data corruption, limit area overhead, and avoid frequency reduction while being compatible with any CMOS technology. Further, BLADE uses a conventional carry chain that allows interspersion of iSC and CPU operations at no performance loss, and we have robustly simulated it down to 0.6V to guarantee its operation in conditions typical of edge devices.

## 3 EDGE DEVICE WORKLOADS

As the adoption of edge devices increasingly supplant traditional compute environments such as servers, emerging workloads will increasingly resemble those of traditional servers in function and complexity. While designing BLADE, we focused specifically on designing an architecture optimized for such workloads, and hence chose applications representative of three such fields for profiling.

### 3.1 Cryptography

As the IoT continues to expand, more and more sensitive data will be being transferred between edge level devices and servers [28]. The need for data security and privacy in the midst of this data explosion is such that many processors now include hardware support for common cryptographic functions [6]. Given the ongoing and increasing need for data privacy, as well as emerging technologies such as blockchain, we have included the Secure Hash Algorithm 3 (SHA-3) [13] in our BLADE benchmarks, an integrity algorithm used in data encryption and to detect data tampering.

### 3.2 HEVC Video Processing

Multimedia creation and consumption makes up the largest portion of total mobile app usage, with Youtube being the second most downloaded mobile application at 52.1%, behind Facebook [23].

**Table 1: BLADE operation support and cycle count.**

| Operation | Cycle Count |
|---|---|
| Bitwise (and, nor, xor, not) | 2 |
| Shift | 2*(# shifts) |
| Add | 2 |
| Subtract | 4 |
| Multiplication | 1 + 2*(operand bit width) + 2*3 |
| Greater/Less Than | 10 |
| Copy | 2 |

Video streaming, music, and mobile games all require significant and increasing processing capabilities. Specifically, with the advent of 4K cameras in phones, the ability to efficiently compress video data will become a necessity for video streaming apps such as Twitch and Facebook Live [15]. As such, we choose the HEVC video encoding program Kvazaar [30] as one of our BLADE benchmarks.

### 3.3 Convolutional Neural Networks

Since Alexnet [22] was unveiled in 2012, Convolutional Neural Networks (CNNs) have become the premier neural net for tasks such as image and language processing [10, 26] and object detection [24]. Given their extreme utility, hardware support on mobile platforms for neural network inference and specifically CNNs has become increasingly attractive [18, 19]. We therefore choose to benchmark CNNs on BLADE. Specifically, we implement our testbench on the Arm Compute Library framework (ACL) [2], a suite of functions designed by ARM to optimally utilize its NEON SIMD co-processor.

## 4 EDGE DEVICE OPTIMIZATION

In targeting edge devices, several design choices in terms of architecture design and integration into the cache hierarchy were made to optimize BLADE for the previously presented workloads.

### 4.1 Supporting Edge Level Applications

BLADE performs simple bitwise operations `and`, `nor`, and `xor` in a manner similar to [3] and demonstrated in Figure 1, namely, by activating two wordlines and sensing the resulting discharge at the output of a single ended sense amplifier. However, in order to support emerging edge device workloads, support for complex operations such as addition and multiplication is also necessary. Hence, we extend the simple BL logic with the addition of a xor and two nor gates, as illustrated in Figure 2-b. The output of the xor gate is in fact an addition between the bits of the two activated wordlines, while the output of the nor gates is a carry bit that can be fed to the following BL logic, resulting in a carry ripple adder. The carry line also doubles as a shift if only one wordline is activated.

With the implementation of addition and shift logic, many more complex operations become possible through a series of simple operations. For example, multiplication can be accomplished through a series of additions and shifts. In this work, we accelerate these complex operations through the inclusion of latches after the sense amplifiers and before the writeback logic, allowing pipelining of add+shift stages, as well as an add write-forward line, allowing 2 cycle add+shifts. Alternatively, in a highly area constrained environment, these latches and write-forward lines could be omitted at the cost of higher cycle counts for complex operations. Table 1 lists all currently supported operations and their respective cycle counts, assuming the cache design detailed in Section 4.3. This cache design utilizes a typical 64 byte wordline length, meaning that a minimum of 512/64 bitwise/8bit operations can be performed per in-cache operation, with this number dependent on cache size and geometry.

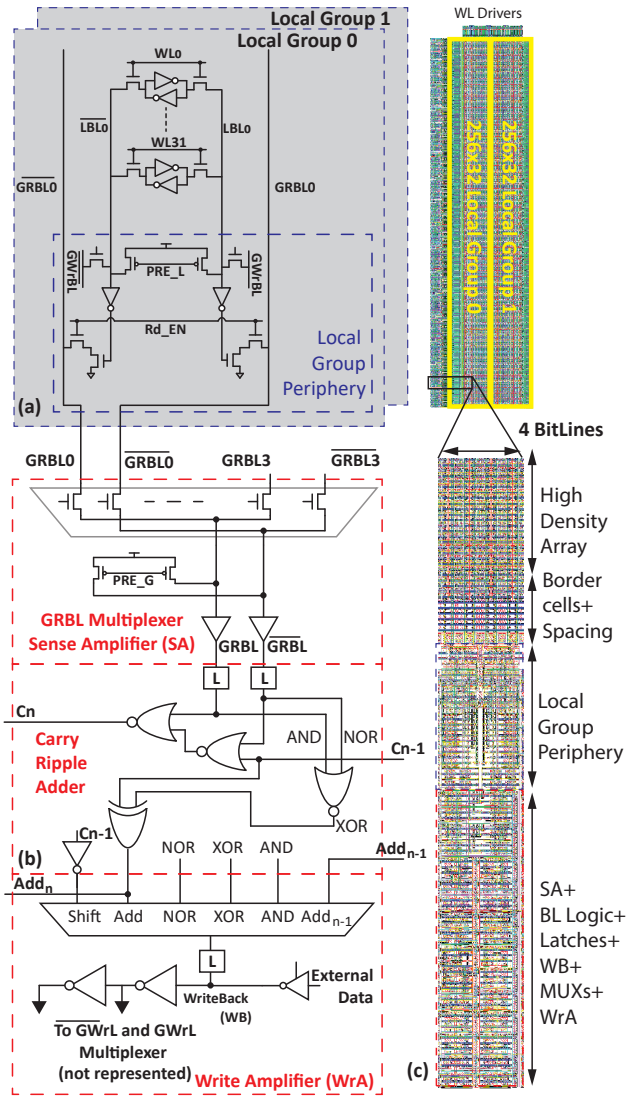


**Figure 2: Layout view of a 256×64 BLADE memory with detailed layout and schematic diagram of the memory periphery.**

### 4.2 Optimized for Small Caches

Several architectural decisions were made in order to optimize BLADE particularly for the small caches found in edge devices. First, we choose to integrate BLADE into the L1 cache in order to simplify cache coherency. Any data requests by the BLADE controller are serviced as standard reads by the cache controller, and similarly, any evictions from the CPU or due to snoop protocol are serviced by the cache controller, with the BLADE controller being notified if relevant data has been evicted.

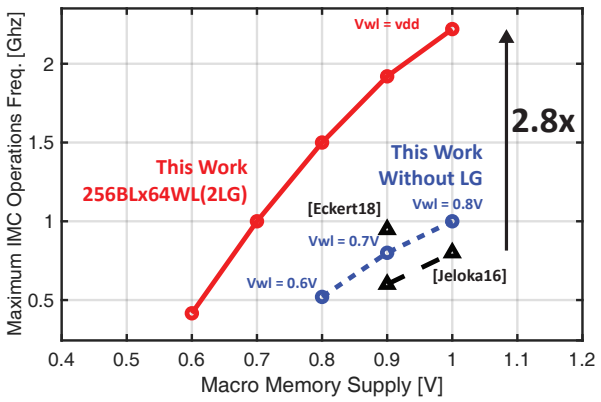


**Figure 3: Maximum frequency of bitwise operations versus memory macro supply voltage at 28nm CMOS.**

**Table 2: Worst case energy values of basic and BLADE operations in a 256×64 array.**

| Operation | Active Energy (fJ) | Leakage Energy (fJ) |
|---|---|---|
| Rd | 23.9 | 88.9 |
| Wr | 25.9 | |
| Bitwise Operation | 23.8 | |
| 8 bit add | 207.4 | |

Second, we implement the BL logic block underneath a BL multiplexer, as illustrated in Figure 2. This reduces area overhead as BL logic is shared between multiple BLs. As set ways are interleaved, BLADE operands are always placed in ways 0 and 1 in order to reduce BLADE controller complexity. Multiplexing also preserves the ability to read cache and tag arrays simultaneously for faster read/write times, known as parallel-tag data access and common in L1 caches. Implementation under the BL multiplexer reduces the number of simultaneous operations that can be performed per cycle, however we believe that the area overhead of putting complex BL logic underneath every BL would make such a design infeasible.

## 4.3 BLADE at Low Voltage

The ability to operate at low voltage is important in power-constrained edge level devices. Unfortunately, cache voltage scaling in iSC architectures is extremely limited due to the risk of data corruption [21]. Data corruption results from the fact that activating two wordlines simultaneously shorts the wordlines' bitcells along their BLs, leading to the possibility of bitcell flipping. The architectures presented in [3, 14] overcome this issue by reducing wordline voltage (underdrive). However, this solution comes at a cost of significantly reduced compute frequency and precludes low voltage operation.

In order to meet the wide voltage range requirements of edge devices while maintaining high frequency, BLADE reuses Local BitLines (LBLs), already reported in many cache architectures [27], to perform reliable, high frequency iSC operations. Each wordline is contained inside a Local Group (LG), which contains a variable number of wordlines and dedicated read/write periphery. All LGs in a subarray are connected in parallel to global read/write BLs(GRBL/GWrL), and are isolated from other LGs, preventing

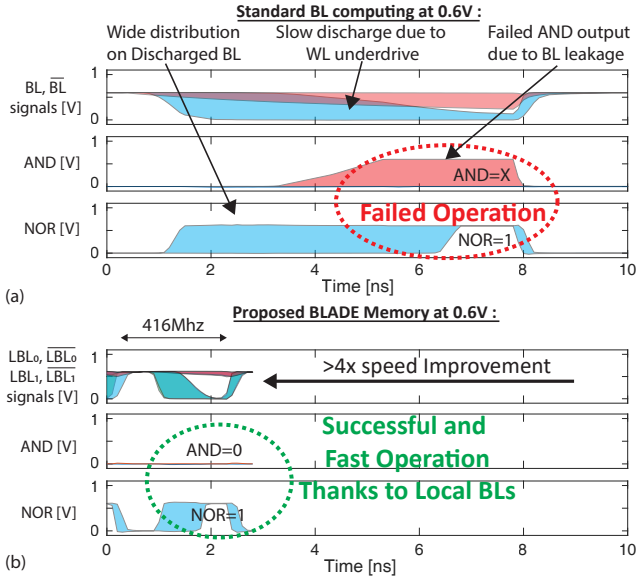


**Figure 4: Transient simulations of bitwise AND, NOR operations at 0.6V on (a) previously published architectures and (b) the proposed BLADE architecture. Thanks to its Local BL organization, BLADE does not require WL underdrive and can achieve fast operation at low voltage while standard architectures are not operational due to BL leakage.**

bitcell shorting or flipping. Figure 3 plots voltage/frequency points for BLADE, both with and without LGs, as well as those of other works. BLADE's voltage/frequency Pareto curve is the most efficient of any conventional state-of-the-art iSC architecture, with a voltage/frequency range between 0.6V/416MHz-1V/2.2GHz.

We implement BLADE in 28nm high performance CMOS technology PDK from TSMC and simulate BLADE's critical paths, considering post-layout parasitics and CMOS variations (10,000 monte-carlo runs at 300K). In this work, we simulate a 256 BL ×64WL memory array with 2 LGs (32WL per LG). This array has an area efficiency of 55.7%, of which the BL logic accounts for 8%. Figure 4 compares the operation of the BLADE memory operating at 0.6V with the same architecture while not using LGs. Beyond showing over 4× speed improvement, it illustrates how BL leakage due to underdriven wordlines preclude correct iSC functionality for standard iSC architectures at 0.6V. We also measure the energy consumption of basic cache operations at a cache frequency of 2.2GHz, as seen in Table 2.

It should be noted that the use of LGs results in a small area overhead [27] and an additional data placement constraint, namely that operands cannot share the same local BL in order to achieve frequencies shown on the red curve in Figure 3. Other placement constraints are discussed in [3], and can be overcome via various architecture-specific solutions. We believe that the nearly 3x frequency gain provided by local BLs outweighs its costs in area and programming complexity. Further, as local BLs are already present in many cache architectures, primarily to improve speed and read stability [27], BLADE in fact improves area efficiency by providing a secondary usage of existing hardware.

**Table 3: Simulator Parameters**

| | |
|---|---|
| Processor | 2GHz, 4 stage pipeline, ARMv8 ISA<br>in-order core, 7 entry LSQ |
| NEON<br>Co-processor | 128 bit registers<br>16 parallel 8-bit operations |
| L1-I Cache | 32kB, 4-way, 1 cycle access |
| L1-D Cache | 32kB, 4-way, 1 cycle access, BLADE extension |
| L2 Cache | mostly-exclusive, 1MB, 4-way, 6 cycle access |
| Memory | DDR3 2133MHz, 4GB |
| BLADE | Max 1024/128 bitwise/8bit simultaneous operations |

# 5 BENCHMARK SETUP

## 5.1 Benchmarking BLADE in gem5

In order to validate the performance gain of this work at a system level, we implement a timing accurate model of BLADE in the architectural simulator gem5 [8]. This is accomplished by first laying out a BLADE SRAM array as discussed in Section 4.3 and extracting critical path timing values. We then extend gem5's L1 cache with a BLADE controller and BL logic, integrating the previously extracted timing values to guarantee cycle accurate simulation. The simulator is calibrated with the parameters outlined in Table 3, which emulates the ARMv8 A53 in-order core found on the ARM Juno development board [7], and running an Ubuntu 18.04 LTS software environment that demonstrates less than 4% timing inaccuracy on profiling tests compared to physical hardware. The simulator also supports operations utilizing a NEON SIMD co-processor [5]. NEON is a well-established and documented SIMD accelerator present on many edge devices, therefore we choose to evaluate the performance of BLADE against NEON. gem5 does not provide reliable system level energy or power estimations, so we do not include such estimations in our performance results beyond individual instruction energy consumption presented in Section 4.3. However, previously presented work demonstrates how iSC architectures reliably reduce energy consumption by reducing instruction processing and eliminating data movement [3]. Such a claim can be grasped intuitively by considering that performing an equivalent number of operations on an array of operands via BLADE, which can perform 128 simultaneous operations, vs NEON, which can perform 16 8 bit operations simultaneously, will result in significantly reduced memory access and data movement, and hence reduced energy consumption.

## 5.2 Benchmarking Methodology

In order to benchmark BLADE's performance on the target applications, we profile each application and identify its most compute intensive kernel. Each kernel has been highly optimized for use on the NEON, providing a robust comparison for BLADE. The kernels in question are the block permutation kernel of SHA-3, the FIR filter of Kvazaar, and the convolutional layer of an ACL CNN; these kernels consume 94%, 35%, and 90% of total application runtimes, respectively. We therefore extract each kernel and run a series of benchmarks, sweeping size parameters related to each kernel. For the FIR filter and convolutional kernels, we also vary cache associativity and cache size, respectively, to demonstrate interesting trends that apply to iSC architectures in general, as detailed in Section 6.

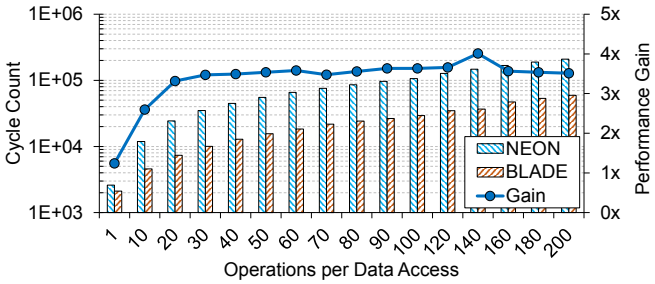


**Figure 5: Computation time of NEON and BLADE on bitwise operations.**

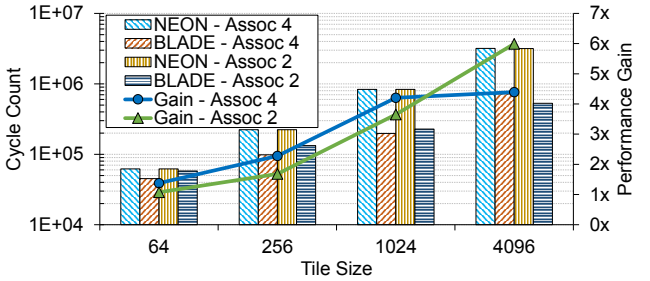


**Figure 6: Computation time of NEON and BLADE on FIR operations.**

# 6 BENCHMARK RESULTS

## 6.1 Bitwise Operations

The block permutation kernel of SHA-3 involves performing a large number of bitwise operations, namely `xor`, `shift`, and `and`, on each byte of information in a series of up to 24 rounds to permute it to an encrypted character sequence. Such a kernel allows us to explore the performance of BLADE with respect to the number of operations performed on data per data access. We therefore design a kernel in which varying amounts of bitwise computations are performed on data per access from memory.

Figure 5 illustrates the performance gains of BLADE over NEON for operation counts between 1 and 200 per data access for 4096 bytes of data. At lower numbers of operations, function time is dominated by memory access, resulting in ~1x acceleration for BLADE. However, as the number of bitwise operations per data access increases, BLADE acceleration improves correspondingly, saturating at ~3.5x speed-up. This saturation is reached at 30 operations per access, well under the >400 operations per byte that SHA3-256 performs, indicating that at this point compute time becomes dominant over access time. Overall, the maximum performance gain over a NEON co-processor is demonstrated to be 4x.

## 6.2 FIR Filter

In order to benchmark Kvazaar's FIR filter function, we extract and run the function in isolation on the available sub-image tile sizes according the HEVC standards [29]. The function utilizes 4 separate 8-tap FIR filters by which the input image is filtered in a horizontal and then vertical manner, resulting in 16 filtered outputs. We also vary cache geometry, specifically associativity, for this benchmark in order to illustrate a general trend relating to iSC architectures, namely that of operation efficiency in relation to cache geometry.

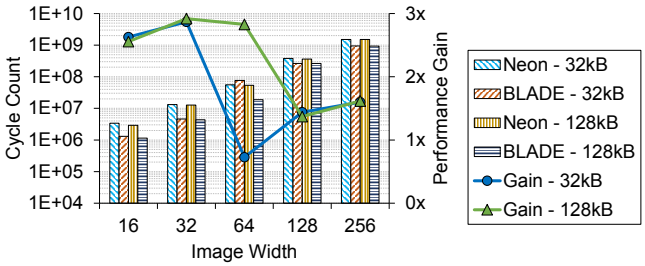


**Figure 7: Computation time of NEON and BLADE on convolution operations.**

Benchmark results are illustrated in Figure 6. This figure demonstrates that, for tile sizes up to 1024 pixels, a cache with an associativity of 4 provides slightly higher acceleration when compared to an equivalently sized cache with an associativity of 2. This results from the fact that BLADE operations in the shallower 4-way cache cause less evictions of relevant data, reducing redundant memory accesses. However, at a tile size of 4096, the 2-way cache dominates in performance, as it is twice as deep as the 4-way cache and can therefore perform twice the number of parallel operations. Maximum performance gain over NEON is demonstrated to be 6x.

## 6.3 Convolution

Finally, we also benchmark the convolutional layer of a CNN implemented in ACL. Our convolutional layer benchmark has 32 input planes and 32 output planes, and performs 3x3 convolution at stride 1 with a padding of 1 to maintain equivalent dimensions. Input/output data is stored in 32 bit fixed point notation, while weights are stored in 8 bit fixed point notation. We convolve input layers with widths varying from 16 to 256 pixels at 2 different cache sizes, 32kB and 128kB, demonstrating a trend relating cache size to iSC efficiency.

Figure 7 illustrates benchmark results for the convolutional layer. As can be seen, each cache size provides similar acceleration over the NEON processor. However, there are sharp performance drop offs at tile widths of 64 and 128 for the 32kB and 128kB caches, respectively. This is a result of the fact that with an output width of 32, the resultant output planes fit entirely within the L1 cache, simplifying the kernel loop design and substantially reducing data movement overhead. At an output width of 64 however, output planes no longer fit within the cache, necessitating a more complex kernel loop to meet operand locality constraints as discussed in Section 4.3, and resulting in a high data movement overhead. By increasing L1 cache size to 128kB, larger layer sizes can be supported at higher speeds. Similar results could be obtained by moving BLADE to lower cache hierarchies; however, this would come at a trade-off of increased coherency complexity if the lower level caches were shared. Overall, a maximum performance gain of 3x over NEON is demonstrated for the convolutional layer of a CNN.

# 7 CONCLUSION

As edge device workloads become more computationally complex, innovative architectures will be necessary to maintain high performance in area and energy constrained environments. In this work we have presented BLADE, an iSC architecture that reuses the preexisting L1 cache with a small (8%) area overhead to perform massively parallel computation at the best voltage/frequency ratios (0.6V/415MHz-1V/2.2GHz) of any conventional iSC architecture. BLADE is demonstrated to outperform an equivalent CPU/NEON processor by significant margins (4x, 6x, 3x) across a variety of cache geometries for a variety of emerging workloads while maintaining or reducing energy consumption. Our results demonstrate that iSC architectures are a promising avenue of research to meet the increasing computational needs of edge device workloads.

# ACKNOWLEDGMENTS

This work has been partially supported by the EC H2020 RECIPE (GA No. 801137) project, and the ERC Consolidator Grant COMPUSAPIEN (GA No. 725657).

# REFERENCES

[1] 2018. http://www.pewinternet.org/fact-sheet/mobile/
[2] 2018. https://developer.arm.com/technologies/compute-library
[3] Shaizeen Aga, S. Jeloka, et al. 2017. Compute Caches. In *HPCA*. 481–492.
[4] Kaya Can Akyel, H. P. Charles, et al. 2016. DRC2: Dynamically Reconfigurable Computing Circuit based on memory architecture. In *ICRC*. 1–8.
[5] ARM. 2009. Introducing NEON.
[6] ARM. 2014. ARM Cortex-A53 MPCore Processor Cryptography Extension.
[7] ARM. 2015. ARM Versatile Express Juno r2 Development Platform.
[8] Nathan Binkert, B. Beckmann, et al. 2011. The Gem5 Simulator. *SIGARCH Comput. Archit. News* 39, 2 (Aug. 2011), 1–7.
[9] Reda Boumchedda, J. Noel, et al. 2017. High-Density 4T SRAM Bitcell in 14-nm 3-D CoolCube Technology Exploiting Assist Techniques. *VLSI* (2017), 2296–2306.
[10] Ronan Collobert and J. Weston. 2008. A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning. In *ICML*. 160–167.
[11] Qing Dong, S. Jeloka, et al. 2018. A 4 + 2T SRAM for Searching and In-Memory Computing With 0.3-V $V_{DDmin}$. *JSSC* 53, 4 (April 2018), 1006–1015.
[12] Mario Drumond, A. Daglis, et al. 2017. The Mondrian Data Engine. In *ISCA 44*. 639–651.
[13] Morris J. Dworkin. 2015. SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. *FIPS* (2015).
[14] Charles Eckert, X. Wang, et al. 2018. Neural Cache: Bit-Serial In-Cache Acceleration of Deep Neural Networks. *CoRR* abs/1805.03718 (2018).
[15] Chang Ge, N. Wang, et al. 2018. QoE-Assured 4K HTTP Live Streaming via Transient Segment Holding at Mobile Edge. *J-SAC* 36, 8 (Aug 2018), 1816–1830.
[16] GfK. 2018. GfK Smart Home study 2018.
[17] Saugata Ghose, K. Hsieh, et al. 2018. Enabling the Adoption of Processing-in-Memory: Challenges, Mechanisms, Future Research Directions. *CoRR* (2018).
[18] Vinayak Gokhale, J. Jin, et al. 2014. A 240 G-ops/s Mobile Coprocessor for Deep Neural Networks. In *CVPR*.
[19] Song Han, X. Liu, et al. 2016. EIE: Efficient Inference Engine on Compressed Deep Neural Network. In *ISCA 43*. 243–254.
[20] Min Huang, M. Mehalel, et al. 2013. An Energy Efficient 32-nm 20-MB Shared On-Die L3 Cache for Intel Xeon Processor E5 Family. *JSSC* (2013), 1954–1962.
[21] Supreet Jeloka, N.B. Akesh, et al. 2016. A 28 nm Configurable Memory (TCAM/BCAM/SRAM) Using Push-Rule 6T Bit Cell Enabling Logic-in-Memory. *JSSC* 51, 4 (2016), 1009–1021.
[22] Alex Krizhevsky, I. Sutskever, et al. 2017. ImageNet Classification with Deep Convolutional Neural Networks. *Commun. ACM* 60, 6 (May 2017), 84–90.
[23] Adam Lella. 2014. Comscore Reports June 2014 U.S. Smartphone Subscriber Market Share.
[24] Shaoqing Ren, K. He, et al. 2015. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In *NIPS 28*. 91–99.
[25] Vivek Seshadri, D. Lee, et al. 2017. Ambit: In-memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology. In *MICRO*. 273–287.
[26] Karen Simonyan and A. Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. *CoRR* abs/1409.1556 (2015).
[27] Mahmut Sinangil, H. Mair, et al. 2011. A 28nm high-density 6T SRAM with optimized peripheral-assist circuits for operation down to 0.6V. *ISSCC* (2011).
[28] Saurabh Singh, P. Sharma, et al. 2017. Advanced lightweight encryption algorithms for IoT devices: survey, challenges and solutions. *JAIHC* (2017), 1–18.
[29] Gary J. Sullivan, J. Ohm, et al. 2012. Overview of the High Efficiency Video Coding (HEVC) Standard. *TCSVT* 22, 12 (Dec 2012), 1649–1668.
[30] Marko Viitanen, A. Koivula, et al. 2016. Kvazaar: Open-Source HEVC/H.265 Encoder. In *ACMMM 24 (MM '16)*. 1179–1182.