CRAFFT: High Resolution FFT Accelerator In Spintronic Computational RAM

Hüsrev Cılasun, Salonik Resch, Zamshed Iqbal Chowdhury, Erin Olson, Masoud Zabihi,
Zhengyang Zhao, Thomas Peterson, Jian-Ping Wang, Sachin S. Sapatnekar, Ulya Karpuzcu
University of Minnesota, Twin Cities
{cilas001, resc0059, chowh005, olso6834, zabih003, zhaox526, pete9290, jpwang, sachin, ukarpuzc}@umn.edu

***Abstract*—High resolution Fast Fourier Transform (FFT) is important for various applications while increased memory access and parallelism requirement limits the traditional hardware. In this work, we explore acceleration opportunities for high resolution FFTs in spintronic computational RAM (CRAM) which supports true in-memory processing semantics. We experiment with Spin-Torque-Transfer (STT) and Spin-Hall-Effect (SHE) based CRAMs in implementing CRAFFT, a high resolution FFT accelerator in memory. For one million point fixed-point FFT, we demonstrate that CRAFFT can provide up to 2.57× speedup and 673× energy reduction. We also provide a proof-of-concept extension to floating-point FFT.**

I. Introduction

Fast Fourier Transform (FFT) is an important algorithm which is extensively used in many different applications with matured implementations. In the literature, FFT has numerous hardware and software based solutions. However, most of the time, such solutions target lower resolution FFT, where the number of points to be processed remains less than several thousands. Ultra-high resolution FFT – where the number of points typically reach millions – by itself is needed for numerous important applications. For example, wireless communication at the heart of emerging IoT and energy harvesting domains requires one million point FFT to enable reliable communication[1] [1]. FFT sizes reach millions of points for radar (e.g., synthetic aperture radar, SAR [2]), sonar and echo-graphy; frequency hopping transmission detection; wide-band spectrum analysis [3]; imaging using radio telescope arrays [4]; and high-resolution medical imaging [5], [6], as well. Although software, FPGA, GPU or ASIC based solutions perform well in small resolutions, due to the space and complexity constraints, increasing resolution requires more parallelism and memory access, which directly translates into higher power consumption and longer execution times, often depleting available budgets. Since the resolution may vary for different applications, the flexibility to change the resolution is also a desired feature.

In-order to address the parallelism and memory access requirements of high resolution FFTs, processing-in memory platforms can be utilized. One very promising reconfigurable in-memory computing substrate is Computational Random Access Memory (CRAM) [7], [8], which is shown to have great potential in accelerating emerging applications with high memory access and massive parallelism demands with significantly lower power consumption, such as binary neural networks [9] and pattern matching or genomics [10], [11]. In this work, we explore acceleration opportunities for ultra high resolution FFTs in spintronic Computational RAM (CRAM), by exploiting its true in-memory processing semantics and reconfigurability for variable resolution support, considering both Spin-Transfer-Torque (STT) and Spin-Hall-Effect (SHE) based implementations. The paper is organized as follows: In Sect.II, we provide preliminary information for Spintronic Computational RAMs as well as the FFT variants to be implemented. Sect.III describes the proposed architecture both for fixed-point and floating-point implementations. Sect.IV covers experimental results. Finally, Sect.V summarizes our findings.

[1]To address local oscillator or Doppler effect related carrier frequency mismatch.

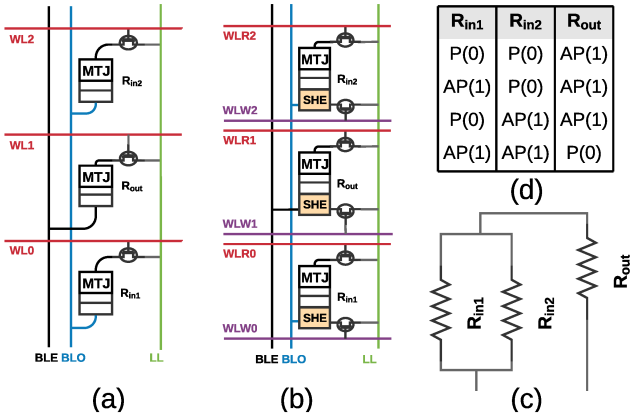

Fig. 1. (a) 1T1M STT-CRAM; (b) 2T1M SHE-CRAM; (c) NAND circuit equivalent; (d) NAND truth table.

II. Background

A. Spintronic CRAM

Spintronic Computational RAM (CRAM) [12] fuses compute and memory functions by performing Boolean gate operations directly within the memory array. No external, peripheral logic is involved for computation as opposed to similar spintronic alternatives such as [13], and no data transfer at the same time. When no computation takes place, the CRAM array becomes equivalent to a spintronic memory array, with the same operational semantics for reads and writes. Different CRAM architectures are proposed so far, targeting different application domains [7], [9]–[11]. In this work, we consider two representatives: one STT- and one SHE-based cell architecture, which deliver the best energy, latency, area trade-offs. In a given column, CRAM can perform only one logic gate at a time, however, all columns –or a reconfigurable subset of them– can perform this same one operation simultaneously. This *reconfigurable parallelism* is the key to CRAM performance.

Each CRAM cell features an Magnetic Tunneling Junction (MTJ) as the state element, which has two magnetic layers separated by an insulating layer. Polarity of the first magnetic layer is fixed but the polarity of the second layer can change. When two polarities (do not) align, MTJ is in low-resistance parallel, P (high-resistance anti-parallel, AP) state. P state encodes logic 0; AP state, logic 1. To read the state, passing a small current (not sufficient for switching) through the device suffices. To write, a larger current is necessary to change MTJ state as a function of the current direction. While STT-CRAM cells solely rely on the MTJ, SHE-CRAM cells augment it with a Spin Hall Effect (SHE) channel. This enables separate optimization for reads and writes (which otherwise induce conflicting optimization targets) and thereby results in better energy efficiency, at the expense of a larger cell area.

Fig.1(a) depicts the 1-Transistor 1-Magnet (1T1M) STT-MRAM cell structure for 3 adjacent cells in a column. When

978-1-7281-1085-1/20/$31.00 ©2020 IEEE

Authorized licensed use limited to: University of Minnesota. Downloaded on October 05,2021 at 20:30:12 UTC from IEEE Xplore. Restrictions apply.

operating as memory, rows can be accessed by selecting the respective Word-Line (WL). The design features two bitlines for alternating rows, Bitline-Odd (BLO) or Bitline-Even (BLE), respectively, and a Logic Line (LL). One end of each MTJ is connected to its respective bitline, the other end, over an access transistor, to LL. Reads and writes are performed by adjusting the voltages on the bitlines and LL, which induce enough current to sense (switch) the cell state for reads (writes), respectively.

To compute (i.e., perform a Boolean gate), first the cells in a column to act as inputs or the output to the gate should be activated by setting the respective WLs, such that their MTJs get connected to the LL. Fig.1(c) shows the resulting (voltage-dividing) circuit, where each cell is presented by its resistance, for a 2-input single-output gate. The two ends of the circuit are connected to BLE (input side) and BLO (output side), respectively. For example, to perform NAND, R_{out} would be preset to P (0), and the voltage difference between BLE and BLO adjusted in a way that, R_{out} always switches if at most one of R_{in1} and R_{in2} is AP (1) (but not both). Thereby, the output preset value and the voltage applied between BLO and BLE uniquely define NAND operation, and enforce state changes according to the truth table shown in Fig.1(d). The equivalent (parallel) resistance of R_{in1} and R_{in2} assumes its maximum if both are AP (high resistance state). Hence, the voltage difference should be set in a way that for any value of the equivalent resistance lower than this maximum (i.e., for any combination where at most one of R_{in1} and R_{in2} can be AP), the current through R_{out} is enough to switch it. The lower the equivalent resistance, the higher would be the current through R_{out}, for any given voltage difference between BLO and BLE.

Using this principle (i.e., by determining pairs of preset and voltage values to enforce switching according to different truth tables), different gates such as 2-input AND, COPY, NOT, 3- or 5- input Majority can be implemented. Universal gates (such as NAND) are fully supported, hence, CRAM theoretically can enable any type of computation. As a building block, a full adder can be implemented using the described gates. Depending on the chosen set of gates, full adder implementations exist with 3, 4, or 9 stages. [10]–[12]. The only limitation in this case is that if the inputs of the logic operation are in an even row, then the output should be in an odd row and vice versa.

Fig.1(b) shows three adjacent cells in a column for SHE-CRAM. This is the same technology as Spin-Orbit Torque (SOT) MRAM [14], which will likely replace STT-MRAM. SHE channels are CMOS/MTJ-compatible and fabricated prototypes exist [15]. Integration with CRAM is covered in [8]. The key feature of SHE-CRAM is separation of read and write paths. A separate access transistor –each controlled by a separate wordline, one for write (WLW) and one for read (WLR)– enables either the read or the write path in this case. Each cell can be connected to either the read or the write path, depending on the memory operation or, when computing, depending on whether it is to serve as a gate input or output. Hence, WLRs are activated for gate inputs; and WLW, for the gate output. The operating principle (specifically when it comes to formation of Boolean gates) is very similar to STT-CRAM, except that SHE-CRAM can perform logic and memory writes more energy efficiently.

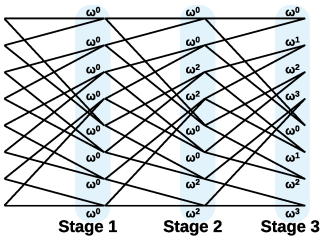

Fig. 2. Signal flow graph.

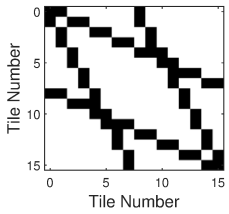

Fig. 3. Connectivity matrix.

B. Singleton's FFT

Although Cooley-Tukey algorithm [16] is the most widely used FFT variant, it does not have a regular memory access pattern, complicating direct mapping to CRAM. Singleton's FFT [17], on the other hand, has a regular memory access pattern that does not vary depending on the stage of computation. Consequently, the near-memory FFT accelerator from [18], as well as the ASIC-based FFT accelerator from [19], use Singleton's FFT as the acceleration target.

Algorithm 1: N-point Singleton's FFT.

input : x_n where $n \in [0, N-1]$ in bit-reversed order
output: y_n where $n \in [0, N-1]$

$s \longleftarrow \log_2 N$;
$\omega \longleftarrow e^{-2\pi i}$;
for $k \in [0, \frac{N}{2} - 1]$ **do**
  $Twiddles[k] \longleftarrow \omega^k$;
end
for $k \in [1, s]$ **do**
  for $j \in [0, \frac{N}{2} - 1]$ **do**
    $CurrentTwiddles[j] \longleftarrow Twiddles[\frac{\lfloor \frac{j}{2^{s-k}} \rfloor}{2^k}]$;
  end
  for $j \in [0, \frac{N}{2} - 1]$ **do**
    $y_j \longleftarrow x_{2j} + x_{2j+1} \times CurrentTwiddles[j]$;
    $y_{j+\frac{N}{2}} \longleftarrow x_{2j} - x_{2j+1} \times CurrentTwiddles[j]$;
    $x \longleftarrow y$;
  end
end

Algorithm 1 summarizes N-point Singleton's FFT, where the signal flow graph for 8-point is given in Fig. 2. Singleton's FFT requires bit-reversed inputs in the beginning but it produces ordered results after the computation is complete. After each stage of computation, odd products of (radix-2) butterflies are mapped to the first half; and even products, to the second half of the output vector. We define z^{th} input and output of a butterfly with x_z and y_z, where $z \in [0, N-1]$. For N-point FFT, provided that

$$\omega = e^{-2\pi i} \tag{1}$$

radix-2 butterfly is defined by equations (2) and (3) as given in [19], where the current stage $k \in [1, s]$ and $j \in [0, \frac{N}{2} - 1]$ for $s = \log_2 N$ stages:

$$y_j = x_{2j} + x_{2j+1}\omega^{\frac{\lfloor \frac{j}{2^{s-k}} \rfloor}{2^k}} \tag{2}$$

$$y_{j+\frac{N}{2}} = x_{2j} - x_{2j+1}\omega^{\frac{\lfloor \frac{j}{2^{s-k}} \rfloor}{2^k}} \tag{3}$$

In each stage, (2) and (3) are calculated for each j where x and y are immediate butterfly inputs and outputs, respectively.

Authorized licensed use limited to: University of Minnesota. Downloaded on October 05,2021 at 20:30:12 UTC from IEEE Xplore. Restrictions apply.

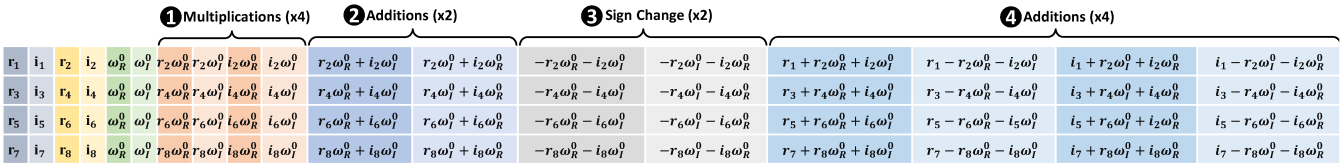

Fig. 4. Transposed and non-interleaved butterflies in a single FFT stage for an 8-point FFT.

III. CRAFFT: High Resolution FFT in CRAM

CRAFFT tailors CRAM to efficiently support Singleton's FFT, with a particular focus on scalability. Specifically, to accommodate very large problem sizes as induced by the target (ultra-high) FFT resolutions, CRAFFT adapts a tiled CRAM design, as depicted in Fig.5. Very large monolithic CRAM arrays would not be feasible in this case due to electrical restrictions on array dimensions. Each tile features a separate tile controller, orchestrated by the global CRAFFT controller. The global controller distributes computational steps, as well as external memory read and write requests to the respective tiles. Thereby *CRAFFT exploits two types of parallelism: the finer grain column-level parallelism and the coarser grain tile-level parallelism.*

CRAFFT features two types of tiles: *A Twiddle tile* and *Compute tiles.* The Twiddle tile keeps real and imaginary parts of precomputed powers of ω per Equations (1), (2), and (3). Compute tiles feature CRAM arrays where the FFT computation takes place. Compute tiles are homogeneous in design, array size and organization, and communicate with the CRAFFT controller via tile controllers. Alongside triggering FFT operations in each butterfly stage, CRAFFT controller reads the twiddle factors from the Twiddle tile and writes them to compute tiles before computation starts in each FFT stage.

We next discuss how CRAFFT implements Alg.1, using the running example from Fig.4 which demonstrates computation in a single (compute) tile for 8-point FFT step-by-step. First, CRAFFT initializes *Twiddle tile* by filling it with precomputed powers of ω (line 4 of Alg.1). Then, for each stage (line 6), CRAFFT controller distributes the current twiddle factors to the twiddle fields in compute tiles (line 8) which are denoted by transposed green rows in Fig.4. We always write x_{2j} and x_{2j+1} to the same column in each compute tile. After performing multiplications and additions (lines 11 and 12, following steps in Fig.4), computation is completed within each column in each compute tile. As the final step in a stage, compute tile controllers read and transmit immediate outputs (y_n) to corresponding tiles where they become immediate inputs (x_n) in the next FFT stage (line 13). Since redistribution of ys (of current stage) to corresponding xs (of next stage) follows a fixed regular pattern (algorithmically, by construction), no index calculation is needed for mapping y_ns.

In a nutshell, each compute tile performs butterfly computation. Tile controllers activate the columns and rows participating in computation. Once real and imaginary parts of the inputs are written into each compute tile in bit-reversed order, each butterfly is computed in a separate column, allowing L different butterflies to be computed in parallel given a tile with L columns. Once butterfly computation is complete, the tile controller reads the output row by row and transmits it to the tiles which are in charge of the next stage. The key here is laying out data and scheduling computations in a way to best exploit CRAFFT's intrinsic column- and tile-level parallelism.

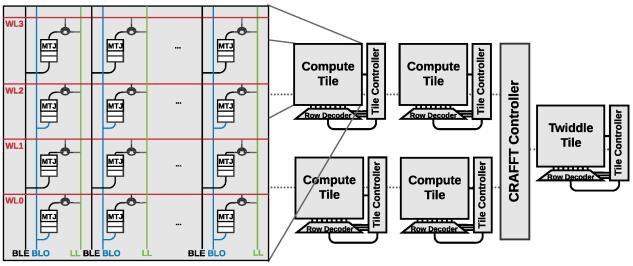

Fig. 5. CRAFFT organization.

Since the order to transmit data to the next stage does not change for Singleton's FFT by construction, CRAFFT does not need specialized routing. Specifically, for N-point FFT, $\frac{N}{2}$ different butterflies are needed. Without loss of generality, we assume that both N and L are powers of two, and that $L \leq \frac{N}{2}$. Assuming $L \times L$ tiles, this renders a total of $\frac{N}{2L}$ compute tiles for N-point FFT. Following the notation from Equations (2) and (3) and using $t \in [0, \frac{N}{2L} - 1]$ as the tile index, we can express CRAFFT's connectivity as

$$t \longleftrightarrow \left\lfloor \frac{t}{2} \right\rfloor ; t+1 \longleftrightarrow \left\lfloor \frac{t}{2} \right\rfloor ; t \longleftrightarrow \left\lfloor \frac{t}{2} \right\rfloor + \frac{N}{4L} ; t+1 \longleftrightarrow \left\lfloor \frac{t}{2} \right\rfloor + \frac{N}{4L} \quad (4)$$

Here, each arrow indicates a connection between the tiles with the respective indices on the left and right sides of the arrow. Identity (4) can be obtained by integer division of the indices from Equations (2) and (3) by $2L$ after replacing j with the limits of tile indices since js are monotonically increasing. For reference, the connectivity matrix for an 16-tile CRAFFT is shown in Fig.3, which clearly shows that no involved routing would be necessary. We next take a closer look into the fixed-point and floating point designs.

A. Fixed-point FFT

1) Design Specifics: In fixed-point arithmetic, fractional arithmetic is performed only using integers, by scaling the numbers by a determined factor which is a power of 2. Any arithmetic operation can be performed using standard Boolean arithmetic in full-adder granularity. It is important to note that, to best exploit CRAFFT's column-level parallelism, we place inputs and outputs in the memory in an *interleaved* fashion, i.e., inputs (outputs) reside in even (odd) cells in a column or vice versa. CRAFFT performs fixed-elementary addition and multiplication operations as described in [9]. For addition and subtractions, we use a ripple-carry full-adder based structure where each full adder is implemented as a cascade of 3 gates [10]. Multiplication relies on a step-by-step implementation of Wallace-Dadda tree adder [20], which renders $N(N-1)$ full-adders for N-bit multiplication.

Similar to [21], the input (x per Alg.1) represents an 16-bit signed integer. Fixed-point complex numbers are represented as a vector limited by unit circle. However, butterfly operations may cause a complex number to grow outside the unit circle. Each butterfly stage can scale the complex number up to $\frac{1}{\sqrt{2}} \approx 0.7$ which can cause overflow if

Authorized licensed use limited to: University of Minnesota. Downloaded on October 05,2021 at 20:30:12 UTC from IEEE Xplore. Restrictions apply.

TABLE I
GATE COUNTS OF IN-MEMORY SINGLE-PRECISION FLOATING-POINT OPERATIONS

Operation	Standard Library	CRAM library
Addition	1041	2882
Multiplication	4262	13795

not addressed [21]. In order to address potential overflow resulting from butterflies without compromising accuracy, we conservatively increase the output size by one bit after each stage.

Fig.4 depicts the memory layout of the butterfly operation (lines 11 and 12 of Alg.1) in a compute tile for 8-point FFT, without loss of generality. Real and imaginary parts of the 16-bit input for $n \in [0, 7]$ are shown with r_n and i_n. In step ❶, real (r_n) and imaginary (i_n) parts of inputs x_n are multiplied by real and imaginary parts of the twiddle factors ω_R and ω_I, which are written into all compute tiles before computation starts. In step ❷, real and imaginary numbers are added. In step ❸, output of step ❷ is negated. In step ❹, real and imaginary numbers are added to obtain the real and imaginary outputs. After rounding (Sect.III-A2), $4 \times (16 + s)$ different row read operations are performed for s^{th} stage. Once reads finish, these results are transmitted to the tiles to perform the next stage.

2) Scaling and Rounding: Since butterflies in the FFT algorithm involve additions and multiplications, overflow is inevitable in the fixed-point implementation. Since full-resolution output is not required for most applications, practical implementations typically opt for conservative scaling to avoid exponential bit growth. Such conservative scaling involves bit reductions between consecutive stages to prevent overflow and to keep intermediate numbers below a specified limit.

In this case, truncation is a practical and costless solution, however, it increases the quantization error significantly. Therefore, standard rounding is more preferable. Provided that rounding discards q bits at each stage, we add 2^{q-1} to each number before quantization, to prevent excessive accuracy loss. Therefore, the only overhead is an n-bit addition for an n-bit number. CRAFFT performs this type of rounding before step ❹ in Fig.4 for both real and imaginary values. Since we apply a conservative bit growth before rounding in each butterfly, our scheme guarantees no overflow. On top of this, the 1-bit incremental bit growth in each stage, as described in Sect.III-A1, prevents further overflow in consecutive stages.

B. Proof-of-concept Extension to IEEE-754 Floating-point FFT

IEEE-754 single-precision floating-point number format stores real numbers in 32 bits which consists of a sign bit, 8 exponent bits, and 23 fraction (mantissa) bits. It allows signed numbers to be represented in a significantly higher dynamic range compared to the fixed-point counterpart of same bit size in expense of increased complexity for arithmetic operations. Floating-point addition entails four steps of operations: First, mantissas of the numbers should be aligned by shifting one number if the exponents are different. Then aligned mantissas are added. The sum is renormalized if binary separator is not placed correctly, by shifting the number up to the first nonzero bit. Finally, the sum is rounded. Floating-point multiplication also consists of four steps. First, exponent bits are added. Then, mantissa fields are multiplied. Finally, renormalization and rounding steps are performed similar to the floating-point addition. Although addition and multiplication are complex, sign change only entails the inversion of the sign bit.

As a proof-of-concept, we implement support for single precision floating-point as follows: We first synthesize addition and multiplication operations into a standard gate library from Verilog code. Then we directly translate this into CRAFFT's correspondents for NOT, NAND and AND gates, without any CRAFFT specific optimization. The gate breakdown of operations is given in Table I. Overall scheme of floating-point computations is similar to the fixed-point implementation (Sect.III-A). One main difference is that bit size does not change in consecutive multiplication, addition, sign change, and addition stages for single precision floating point numbers. Twiddle factors are also stored as single precision numbers. Therefore extra scaling and rounding operations are not needed. That said, we do not address floating-point exceptions in this proof-of-concept design.

As we will cover shortly, this proof-of-concept design results in uncompetitive execution time per operation. Even in its unoptimized form, however, CRAFFT's massive parallelism helps in masking this overhead.

TABLE II
CONFIGURATION PARAMETERS

Parameter	**STT-M**	**SHE-M**	**STT-F**	**SHE-F**
Tile size (1M)	16 MB	16 MB	64 MB	64 MB
Array dimension	256	256	1024	1024
Compute Tile Count (1M)	2048	2048	512	512
Bulk preset current limit	30 mA		30 mA	
P State Resistance	3.15 $k\Omega$		7.34$k\Omega$	
AP State Resistance	7.34 $k\Omega$		76.39$k\Omega$	
Switching Time	3 ns [22]		1 ns	
Switching Current	40 μA [22]		3 μA	
Twiddle Tile Size (1M)	1 MB (fixed-point) - 2 MB (floating-point)			

IV. EVALUATION

To evaluate CRAFFT, we use an in-house simulator which is capable of performing functional verification as well as energy and timing calculation. We extract the peripheral circuitry overhead from NVSim [23]. NVSim estimates subarray overhead resulting from charge/precharge, row decoder, sense amplifier, and bitlines (as well as array overheads resulting from array read/write operations such as predecoder overhead). NVSim's subarrays correspond to our tiles. We also take the time and energy overhead of tile controllers into consideration. The configuration parameters are given in Table II. STT-M and SHE-M configurations reflect modern technology, and stem from very conservative assumptions for switching time and current. STT-F and SHE-F configurations are future-term technology projections.

We assume that in the beginning of the computation, the reverse-bit ordered input is written to compute tiles by the CRAFFT controller. Then the controller configures the tiles with twiddle factors and triggers the tile controllers to initiate butterfly computation. Butterfly operations are executed as described in Section III. After real and imaginary outputs are calculated, each tile controller simultaneously reads and broadcasts the tile outputs, following the connectivity captured by Equation (4). Corresponding tiles receive and write the inputs of the next FFT stage.

CRAFFT is not limited to ultra-high resolution FFT. Thanks to its reconfigurability, CRAFFT can also support smaller FFT sizes efficiently. This is achieved only by reconfiguring the *Twiddle tile* accordingly. CRAFFT controller then disables unused tiles to preserve energy.

We optimize execution time by experimenting with different tile sizes for the four different configurations (STT-M,

Authorized licensed use limited to: University of Minnesota. Downloaded on October 05,2021 at 20:30:12 UTC from IEEE Xplore. Restrictions apply.

SHE-M, STT-F, and SHE-F, respectively). Since For STT-based designs, a lot of cycles are required to perform bulk preset for large tile sizes, increasing the execution time. However, if the tile size is too small, the design may need to be distributed to many more tiles. In this case, extra preset cycles would be needed to reuse the portion of tile where no future data dependency exists. Another factor is that large tile sizes incur higher read/write latency. We determined that the square array dimensions from Table II give the minimum execution time both for fixed-point and the floating point implementations.

Another possible design point is an unrolled design where each stage is calculated in different tiles. This design can eliminate twiddle factor distribution overhead as there would be no need to reconfigure twiddle factors in a non-volatile memory. This design also conforms itself very well to pipelining, therefore has the potential to increase the throughput significantly. However, such an unrolled design comes with a very high price tag in terms of area footprint since the required number of tiles would be multiplied by the number of stages in FFT. Therefore, it is not considered in this study.

TABLE III
FIXED-POINT FFT COMPARISON FOR 1M POINT FFT

Implementation	Time (μs)	Throughput (MSPS)	Energy (μJ)	EDP (nJs)
SHE-F	194.781	5383.368	21.86	4.26
SHE-M	701.565	1494.624	2526	1772
STT-F	194.781	5383.368	185.3	36.1
STT-M	701.565	1494.624	9922	6961
FPGA [21]	4290	233	14700	63236
12 FPGAs [4]	500	2091	3.17×10^{6}*	1.585×10^{9}*

*Estimated conservatively *only* using static power consumption of the given FPGAs as the actual figure is not reported.

Fixed-point simulation results for one million point FFT are summarized in Table III, where execution time for smaller FFT sizes is given in Fig.6. SHE-F configuration achieves 22× speedup compared to the fastest single-chip Virtex Ultrascale FPGA implementation in [21]. The speed-up of SHE-F drops to 2.57× – which still represents a significant improvement – when compared to [4] where 12 FPGAs are utilized with parallel processing. Although this parallel FPGA baseline does not report energy, only using the static power consumption of given FPGAs (8 Virtex-4 LX60 and 4 Virtex-4 LX80), we estimate 3.71 J per 1M-point FFT operation, which is excessively high compared to CRAFFT implementations (Table III). As shown in Fig.6, the execution time benefit persist for smaller FFT sizes, as well, however, not so much at the other end of the spectrum, for very small FFT. This is because our execution time overheads such as twiddle distribution and tile communication become more dominant, yet CRAFFT still retains a feasible execution time.

Fig.8 provides the energy breakdown of different operations. We observe that addition and multiplication operations dominate the overall energy consumption. As expected, in the STT-based technologies, a significant portion of the energy is spent on preset operation.

We next analyze the accuracy in terms of Signal-to-Quantization-Noise Ratio (SQNR). SQNR represents the logarithmic ratio of ideal FFT of the signal to error resulting from quantization:

$$\text{SQNR (dB)} = 10\log_{10}\left(\frac{\text{E}\{|X_{original}|^2\}}{\text{E}\{|X_{quantized} - X_{original}|^2\}}\right) \quad (5)$$

Here E depicts expectation; $X_{original}$ and $X_{quantized}$, the ideal case and the hardware implementation, respectively.

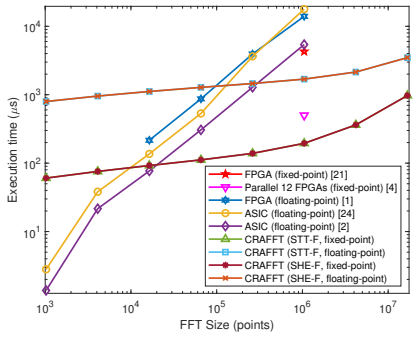

Fig. 6. Fixed-point and floating point execution times vs. various baselines.

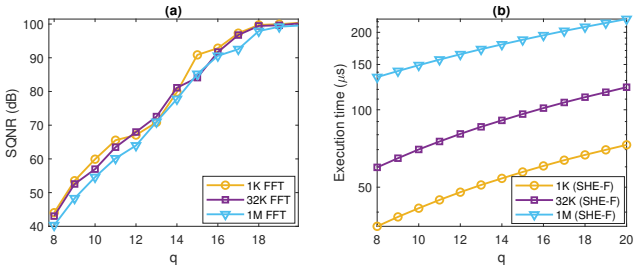

Fig. 7. (a) SQNR vs. twiddle factor bit size (q); (b) Execution time vs. q for three different FFT sizes.

Using 16-bit signed twiddle factors, we obtain an SQNR of 90.6dB compared to the standard double precision implementation of MATLAB. Fig. 7(a) and 7(b) capture the effect of twiddle factor bit size, q, considering different FFT sizes. The effect of q is somehow limited due to the fixed 16-bit input size, however, it can improve SQNR at expense of increased execution time as well as energy consumption up to a saturation point. This is because input bit size is fixed (6dB per bit, 96dB for 16 bits). However, FFT SQNR can slightly exceed 96 dB because we increase the bit size in each stage and each butterfly introduces a small amount of bit growth.

We conclude the evaluation with a feasibility analysis of the proof-of-concept floating-point implementation. Table IV compares our results to single-precision FPGA and ASIC implementations from the literature for different FFT sizes. For one million point FFT, we observe that the SHE-F configuration achieves 3.2× speedup with 1.93× more energy consumption. This result still points to a great potential, as our current design, tightly tailored for fixed-point FFT, does not optimize floating-point operations. Still, CRAFFT's energy-delay product is 1.66× lower in this case. This single-precision implementation achieves 135dB SQNR compared to the double-precision MATLAB implementation.

V. CONCLUSION

In this work, we proposed CRAFFT, a spintronic Computational RAM (CRAM) based FFT accelerator, which can efficiently handle ultra-high resolution FFT at scale. At the same time, CRAFFT is fully reconfigurable to support smaller size FFT. We covered two different MTJ based technologies with four different configurations. While CRAFFT is optimized for fixed-point, we also present a floating-point implementation as an exploratory design point.

For the fixed-point case, CRAFFT is 22× and 2.57× faster than fastest known single-chip and parallel implementations from the literature, while consuming significantly less energy. For floating point, on the other hand, fastest CRAFFT

Authorized licensed use limited to: University of Minnesota. Downloaded on October 05,2021 at 20:30:12 UTC from IEEE Xplore. Restrictions apply.

TABLE IV
FLOATING-POINT FFT COMPARISONS

FFT Size	Metric	SHE-F	SHE-M	STT-F	STT-M	ASIC [2]	ASIC [24]	FPGA [25]	FPGA [1]
4K	Time (μs)	955.72	2855.2	955.72	2855.2	21.6	-	-	-
	Throughput (MSPS)	4.29	1.43	4.29	1.43	18.96	-	-	-
	Energy (μJ)	1.521	142.9	5.05	336.8	9.720	-	-	-
	EDP (nJs)	1.45	408	4.83	961.69	0.03	-	-	-
16K	Time (μs)	1116.1	3336.9	1116.1	3336.9	76.9	-	-	216
	Throughput (MSPS)	14.68	4.91	14.68	4.91	203.1	-	-	-
	Energy (μJ)	7.098	666.88	23.587	1571.8	4.23	-	-	6998*
	EDP (nJs)	7.92	2225.3	26.33	5.245	0.33	-	-	15.12*
64K	Time (μs)	1279.92	3836.96	1279.92	3836.96	306.3	-	-	869
	Throughput (MSPS)	51.20	17.08	51.20	17.08	204.05	-	-	-
	Energy (μJ)	32.45	3048.59	107.83	7185.59	18.29	-	-	281.56*
	EDP (nJs)	41.53	11697.3	138.01	27570.7	5.6	-	-	244.672*
256K	Time (μs)	1457.6	4410.8	1457.6	4410.8	1285.4	3636.8	-	3939
	Throughput (MSPS)	179.85	59.43	179.85	59.43	194.49	-	-	-
	Energy (μJ)	146.02	13718.65	485.22	32335.01	78.67	1316.5	-	1276.2*
	EDP (nJs)	212.83	60510.1	707.25	142623.4	101	4787.84	-	5027.1*
1M	Time (μs)	1690.5	5279.5	1690.5	5279.5	5407.5	17790	-	13971
	Throughput (MSPS)	620.267	198.612	620.267	198.612	184.96	-	457	-
	Energy (μJ)	648.96	60971.8	2156.55	143711.8	336.3	6439.98	-	4526*
	EDP (nJs)	1097.1	321902.7	3645.7	758731.3	1820	114567	-	63241*
4M	Time (μs)	2144.6	7327.9	2144.6	7327.9	-	-	-	-
	Throughput (MSPS)	1955.70	572.37	1955.70	572.37	-	-	217	-
	Energy (μJ)	2855.4	268276.1	9488.8	632333.2	-	-	-	-
	EDP (nJs)	6123.9	1965906	20350.3	4633687	-	-	-	-

*Estimated conservatively *only* using static power consumption of the given FPGAs as the actual figure is not reported.

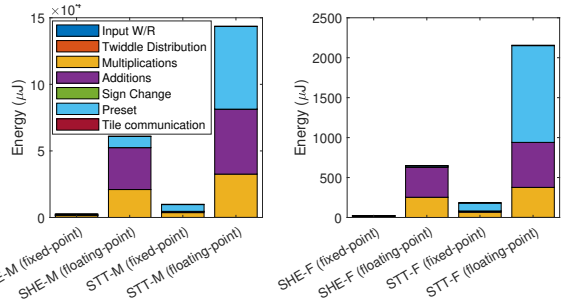

Fig. 8. Energy breakdown of fixed-point and floating-point FFT.

configuration achieves 3.2× speedup while consuming 1.93× more energy than the fastest ASIC implementation from the literature. Overall, this unoptimized CRAFFT configuration still achieves 1.66× more energy efficiency (EDP).

ACKNOWLEDGMENT

This work was supported in part by NSF grant no. SPX-1725420.

REFERENCES

[1] J. Meng, H. Wang, P. Ye, L. Guo, Y. Zhao, W. Wei, and H. Zeng, "Carrier frequency offset estimation based on twice fft matrix algorithm," in *2019 IEEE I2MTC*, IEEE, 2019.

[2] X. Chen, Y. Lei, Z. Lu, and S. Chen, "A variable-size fft hardware accelerator based on matrix transposition," *IEEE Tran. VLSI Systems*, vol. 26, no. 10, 2018.

[3] F. Schlottau, M. Colice, K. Wagner, and W. R. Babbitt, "Spectral hole burning for wideband, high-resolution radio-frequency spectrum analysis," *Optics letters*, vol. 30, no. 22, 2005.

[4] T. Kamazaki *et al.*, "Digital spectro-correlator system for the atacama compact array of the atacama large millimeter/submillimeter array," *Publications of the Astronomical Society of Japan*, vol. 64, no. 2, 2012.

[5] F. Han, L. Li, K. Wang, F. Feng, H. Pan, B. Zhang, G. He, and J. Lin, "An ultra-long fft architecture implemented in a reconfigurable application specified processor," *IEICE Electronics Express*, vol. 13, no. 13, 2016.

[6] A. Agarwal, H. Hassanieh, O. Abari, E. Hamed, D. Katabi, *et al.*, "High-throughput implementation of a million-point sparse fourier transform," in *2014 24th FPL*, IEEE, 2014.

[7] M. Zabihi, Z. Chowdhury, Z. Zhao, U. R. Karpuzcu, J.-P. Wang, and S. Sapatnekar, "In-memory processing on the spintronic cram: From hardware design to application mapping," *IEEE Tran. on Computers*, 2018.

[8] M. Zabihi, Z. Zhao, D. Mahendra, Z. I. Chowdhury, S. Resch, T. Peterson, U. R. Karpuzcu, J.-P. Wang, and S. S. Sapatnekar, "Using spin-hall mtjs to build an energy-efficient in-memory computation platform," in *20th ISQED*, IEEE, 2019.

[9] S. Resch, S. K. Khatamifard, Z. I. Chowdhury, M. Zabihi, Z. Zhao, J.-P. Wang, S. S. Sapatnekar, and U. R. Karpuzcu, "Pimball: Binary neural networks in spintronic memory," *ACM TACO*, vol. 16, no. 4, 2019.

[10] Z. Chowdhury, J. D. Harms, S. K. Khatamifard, M. Zabihi, Y. Lv, A. P. Lyle, S. S. Sapatnekar, U. R. Karpuzcu, and J. Wang, "Efficient in-memory processing using spintronics," *IEEE CAL*, vol. 17, no. 1, 2018.

[11] Z. I. Chowdhury, S. K. Khatamifard, Z. Zhao, M. Zabihi, S. Resch, M. Razaviyayn, J. Wang, S. Sapatnekar, and U. R. Karpuzcu, "Spintronic in-memory pattern matching using computational ram (cram)," *IEEE JXCDC*, 2019.

[12] J.-P. Wang and J. D. Harms, "General structure for computational random access memory (cram)," Dec. 29 2015. US Patent 9,224,447.

[13] S. Li, C. Xu, Q. Zou, J. Zhao, Y. Lu, and Y. Xie, "Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories," in *Proceedings of the DAC*, 2016.

[14] F. Oboril *et al.*, "Evaluation of hybrid memory technologies using sot-mram for on-chip cache hierarchy," *IEEE TCAD*, vol. 34, no. 3, 2015.

[15] K. Garello *et al.*, "SOT-MRAM 300mm integration for low power and ultrafast embedded memories," in *IEEE Symp. on VLSI Circuits*, IEEE, 2018.

[16] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex fourier series," *Math. of computation*, vol. 19, no. 90, 1965.

[17] R. Singleton, "A method for computing the fast fourier transform with auxiliary memory and limited high-speed storage," *IEEE Tran. on Audio and Electroacoustics*, vol. 15, no. 2, 1967.

[18] H. E. Yantir, W. Guo, A. M. Eltawil, F. J. Kurdahi, and K. N. Salama, "An ultra-area-efficient 1024-point in-memory fft processor," *Micromachines*, vol. 10, no. 8, 2019.

[19] Z. Liu, Y. Song, T. Ikenaga, and S. Goto, "A vlsi array processing oriented fast fourier transform algorithm and hardware implementation," *IEICE Tran.s on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 88, no. 12, 2005.

[20] P. Behrooz, "Computer arithmetic: Algorithms and hardware designs," *Oxford University Press*, vol. 19, 2000.

[21] H. Kanders, T. Mellqvist, M. Garrido, K. Palmkvist, and O. Gustafsson, "A 1 million-point fft on a single fpga," *IEEE Trans. on Circuits and Systems I: Regular Papers*, vol. 66, Oct 2019.

[22] D. Saida *et al.*, "Sub-3 ns pulse with sub-100 μa switching of 1x–2x nm perpendicular mtj for high-performance embedded stt-mram towards sub-20 nm cmos," in *2016 IEEE Symp. on VLSI Technology*, IEEE, 2016.

[23] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi, "Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *IEEE TCAD*, vol. 31, July 2012.

[24] L. Guo, Y. Tang, Y. Lei, Y. Dou, and J. Zhou, "Transpose-free variable-size fft accelerator based on-chip sram," *IEICE Electronics Express*, 2014.

[25] "Fft (1d) off-chip design example." https://www.intel.com/content/www/us/en/programmable/support/support-resources/design-examples/design-software/opencl/fft-1d-offchip.html. Accessed: 2019-11-02.

Authorized licensed use limited to: University of Minnesota. Downloaded on October 05,2021 at 20:30:12 UTC from IEEE Xplore. Restrictions apply.