Performance and Energy Limits of a Processor-integrated FFT Accelerator

Tung Thanh-Hoang[1], Amirali Shambayati[1], Calvin Deutschbein[1], Henry Hoffmann[1], and Andrew A. Chien[1, 2]

[1]Department of Computer Science, University of Chicago, Chicago, Illinois, USA
[2]Argonne National Laboratory, Chicago, Illinois, USA
{hoangt, amirali, hankhoffmann, achien}@cs.uchicago.edu, calvin.excalibur@gmail.com

Abstract—**Accelerators have long been used to improve the performance and energy efficiency of embedded signal processing systems relying on Fast Fourier Transforms (FFTs). We explore the benefits of processor-integrated FFT accelerators, characterizing their performance and energy efficiency for current and future memory architectures. First, we consider designs that deeply integrate an FFT accelerator into a simple 5-stage RISC pipeline and evaluate the performance and energy efficiency for a 32 nm process. Our results indicate that a 64-point processor-integrated FFT accelerator alone can increase performance for a 4K/32k-point 1D-FFT by 7/4-fold respectively. In term of energy efficiency, our 64-point FFT accelerator increases it at least 4-fold. Second, since memory performance is a critical constraint, we evaluate system configuration with 3D-stacked DRAM systems. Our results indicate that energy efficiency bottlenecks can be alleviated, as the 3D-stacked memory reduces energy by nearly 14-fold. When combined with our FFT accelerator, overall energy efficiency for 4k and 32k-point FFTs increases 86-fold and 70-fold respectively. Prospectively, with addition of a data layout transformation engine, cycle count and energy for the data transpose phase can be reduced 10x. Such a step would increase the accelerator benefit at least 10-fold in energy for DDR3 and more than 100-fold in 3D-stacked memory system.**

I. Introduction

Fast Fourier Transform (FFT) performance is critical for a number of embedded systems. Hardware acceleration for the FFT improves both performance and energy efficiency. Not surprisingly a tremendous number of accelerator designs have been proposed over the years since the Cooley-Tukey algorithm was published [4].

Many of the proposed accelerators are designed and evaluated considering only on-chip constraints. These studies show the tremendous potential of FFT accelerator designs, but they neglect the implications of placing the accelerator in the context of the larger system. Specifically, the inputs in many embedded systems come through memory-mapped I/O devices. This structure means that the true performance of the system may be constrained not just by on-chip factors, but also by the ability to move data to the accelerator.

This paper studies the performance and energy limits of FFT acceleration when 1) the FFT accelerator is *integrated* into the processor and 2) the FFT must operate on data that starts in external memory. To begin, we design a *processor-integrated* FFT accelerator that forms another path in the processor pipeline; it is integrated into the bypass networks and register files. This allows programmability to be preserved and the FFT operations to be well integrated with the rest of the computation (including other signal processing kernels and high-level data analysis) [1, 11, 24].

We evaluate the performance of this FFT accelerator in several full-system scenarios. First, we characterize its performance with an on-chip focus, and then with the constraints of a full DDR3 memory system. This study explores the performance and energy penalties incurred by scaling to a main memory. We find that the DDR3 bandwidth limits accelerator performance, and DRAM leakage dominates system energy. These effects limit the performance and energy efficiency of the accelerator.

These observations motivate additional exploration of accelerator performance with a three-dimensional (3D) stacked memory system. Such systems are increasingly available commercially and offer new levels of performance and energy efficiency [14, 17, 22, 33]. In the future, such technologies may become available at low cost. Our findings indicate that the switch from DDR3 to 3D-stacked memory provides tremendous energy savings.

The specific contributions of this paper include:

1) A detailed performance and energy characterization of a processor-integrated FFT accelerator, including breakdown by phase and hardware module. This study shows that up to 20-fold performance and over 10-fold energy improvement in on-chip energy can be achieved (ignoring DRAM).
2) Further performance and energy characterization of that same processor-integrated FFT accelerator, with DDR3 memory system constraints. This study shows that up to 7-fold performance and 4-fold energy improvement are achieved when DDR3 memory system constraints are added, demonstrating the limitations of traditional external memory for FFT acceleration.
3) A final performance and energy characterization of the processor-integrated FFT accelerator with a stacked memory system. Our results show that the increased bandwidth provides little benefit for the FFT sizes we consider, but yields dramatic energy efficiency improvements (91-fold and 52-fold for 4k and 32k-point FFTs respectively). With stacked memory, memory energy is no longer the dominant factor, so the energy efficiency of the accelerator is critical.

The remainder of this paper is organized as follows. Section II describes the processor-integrated FFT accelerator design. Section III describes the methodology and configurations studied. It also explains the rationale for these configurations. The experimental results for these designs are presented in Section IV, followed by a discussion of the related work in Section V. We close with a summary and discussion of future interesting research directions in Section VI.

II. A Processor-Integrated FFT Accelerator

Our goal is to integrate acceleration for FFT flexibly with high level object recognition and classification computations often found in later stages of signal, image, and video analysis pipelines. The former requires a dense array of ALUs and efficient network for communication. The latter requires pointer oriented computation with high level data structures and a general purpose pipeline [1, 24].

To address these needs, our design integrates a highly optimized FFT computation array into a simple 5-stage RISC processor [16]. To enable efficient computation on larger FFTs than will fit into a register file, we add a local memory that can be accessed efficiently by the processor, and then compose the RISC pipe with the FFT accelerator using a 2048-bit vector register file. These elements and the high level architecture are depicted in Figure 1.

The basic design and integrated components are given below:

- **RISC Core:** This basic design remains the most energy efficient general-purpose building block for complex data structures and general programs. We employ a simple 5-stage, 32-bit RISC design similar to the MIPS R2000 [16]. This 1GHz pipeline supports only 32-bit arithmetic, logic, and control flow operations.

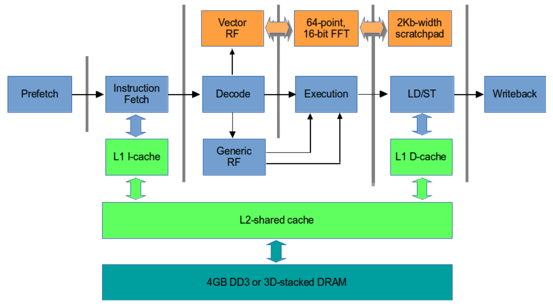


Fig. 1: Block Diagram of the enhanced Processor Pipeline

TABLE I: Core, Accelerator, and Memory contribution to Chip Area

Component	Area (mm^2 32nm)	Percentage
Core	0.05	$<$ 1%
FFT Accelerator	1.30	11%
Scratchpad (1MB)	3.68	33%
L1 I-cache	0.22	2%
L1 D-cache	0.33	3%
L2 Unified	5.73	51%

At 11K gates, the RISC pipeline is remarkably small.

- **64-point FFT Accelerator:** We generate an optimized FFT design using the Spiral hardware generator [19]. This accelerator computes a 64-point FFT, on 16-bit complex fixed point data with a single instruction. The design includes 332 16-bit multipliers, 964 adders, and has a pipeline depth of 26 clocks at 1Ghz. At 77k gates, it is much larger than the RISC core.
- **Scratchpad or Local Memory:** Because FFTs have completely predictable data movement, a local memory gives the most efficient (performance and energy) design. For FFT computations larger than the register files, we stage the needed data from the DRAM into the scratchpad and then carefully schedule access to achieve high performance and energy efficiency.
- **Vector Register Integration:** Accelerators often require a large a data block as input/output. Providing it one word at a time via programmed IO or DMA can incur significant overheads. Instead, we use a large vector register file (see Table IV) to move data efficiently between the FFT accelerators and local memory. This approach reduces instruction overheads for accelerator interfacing by as much as 100-fold.
- **Cache hierarchy:** We configure the cache hierarchy to model the low-power Intel Atom [9],detailed in Table IV

Our design allows both regular RISC instructions and complex FFT accelerator operations to be issued and executed under the same in-order pipeline control with the vector register file being inserted parallel to the traditional 32-bit register file, and the FFT accelerator residing in the execute stage. The area and fraction of chip area for each of these elements is shown in Table I. The FFT accelerator is nearly 25x larger than a RISC core, but still a small fraction of a typical chip area. The area is dominated by the memories, notably the L2 cache and the scratchpad memories.

A. Software Access to Accelerator Capabilities

We implement our processor integrated accelerator using the Synopsys design compiler. The RISC core is designed in LISA [37], a high level architectural design language, and the Synopsys Processor Designer and compiler generator allow for the automatic generation of software interfaces (intrinsics) used at the C source level to access the new FFT instructions.

These instructions and corresponding intrinsics are described in Table II. `BVMM2SM` and `BVSM2MM` move data between scratchpad and the memory hierarchy. `LDSM2VR` and `STVR2SM` load/store the vector

Listing 1: Pseudo code of 32k-point integer FFT based on 4k-point FFT and 64-point FFT accelerator.

```
FFT_4k_int14_1d (direct,*src,*tdf64,*tdf32k,*dst)
//--- Marshal phase
   BVMM2LM: (tdf64, tdf4k, tdf32k, src) -> scratchpad;
//--- Transpose phase
   transpose: src->dst;
//--- Compute phase
   for (i=0; i<8; i++) // Do 4k-FFT for rows
      fft4kint16(direct, &dst[i*4096],
            tdf64, tdf4k,  &dst[i*4096]);
//--- Transpose phase
   transpose: dst->src;
//--- Twiddle-mult phase
   scalar_mult: src * tdf32k;
//--- Do 8-FFT phase
   for (i=0; i<4096; i++) {
      get_tdf: tdf8 // for 8-point FFT
      fft8int16 (direct, &src[i*8], tdf8, &src[i*8]);
   }
//--- Transpose phase
   transpose: src->dst;
//--- Unmarshal phase
   BVSM2LM: dst->main memory;

FFT_4k_int14_1d(direct, *src,*tdf64, *tdf4k, *dst)
 // Load twiddle factor to vector
    LDSM2VR tdf -> vecTdf
//--- Transpose phase
   transpose: src->dst;
//--- Compute phase
   for (i=0; i<4; i++) {
     LDSM2VR dst -> vecRF; // load vec from scratchpad
     FFT64INT16(direct, vecRF, vecTdf, vecRF);
     STVR2SM vecRF -> dst; // store back scratchpad
   }
//--- Transpose phase
   transpose: dst->src;
//--- Twiddle-mult phase
   scalar_mult: src * tdf4kk;
//--- Compute phase
   for (i=0; i<4; i++) {
     LDSM2VR src -> vecRF;
     FFT64INT16(direct, vecRF, vecTdf, vecRF);
     STVR2SM vecRF -> src;
   }
//--- Transpose phase
   transpose: src->dst;
//--- Unmarshal phase
   BVSM2LM: dst->main memory;
```

registers from/to the scratchpad memory. A wide interface is possible because that memory is on-chip, small, and near the core. A single instruction, `FFT64INT16` initiates a 64-point FFT operation, taking a 2048-bit vector register as input and another as the destination.

B. Structure of a Fast Fourier Transform

We describe the FFT computation (see Listing 1), introducing standard terminology for each phase. These terms both explain specific software and hardware optimizations and the performance results presented in Sections III and IV.

The key phases of the FFT running on our design our:

- **Marshal/Demarshal:** Move the data to and from the scratchpad memory to setup the efficient FFT.
- **Computation:** The ops and data movement for the butterfly operations used to create larger FFTs
- **Twiddle Multiply:** Multiply the FFT data by the twiddle factors.
- **8-FFT:** The software FFT needed to compose intermediate results for the 32K FFT.
- **Transpose:** The larger-scale data movement to compose the computation sub-elements together.
- **Vector Load/Store:** Move the data from the scratchpad (local memory) into the vector registers.

TABLE II: New instruction to access and integrate the FFT accelerator

Instruction	Intrinsic	Description
FFT64INT16	FFT64INT16(direct, *srcAddr, *twdAddr, *dstAddr)	Compute 64-point FFT, each point consists of two 16-bit integer values.
LDSM2VR	LDSM2VR(*srcAddr, vrDst)	Load 256 bytes from scratchpad memory to vector register.
STVR2SM	STVR2SM(vrSrc, *dstAddr)	Store 256-byte vector register to scratchpad memory.
BVMM2SM	BVMM2SM(*mmAddr, *smAddr, nByte)	Moving nByte data from main memory to scratchpad memory.
BVSM2MM	BVSM2MM(*smAddr, *mmAddr, nByte)	Moving nByte data from scratchpad memory to main memory.

TABLE III: Summary of System Configurations

Configuration Name	Baseline	Accel-F/S/M	Stacked	StAccel-F/S/M	fastStacked	fastStAccel-F/S/M
FFT Accelerator	No	Yes	No	Yes	No	Yes
Memory	DDR3	DDR3	3D-slow	3D-slow	3D-fast	3D-fast

III. Methodology and Experiments

This section describes the processor and memory configurations as well as the simulation tools used in our experimental study.

A. System Configurations

Table III summarizes the system configurations studied. The differences are in the presence of the FFT accelerator and the memory system. All of the systems studied include a 1 MB scratchpad memory and the basic RISC core elements shown in Table IV.

TABLE IV: RISC Core Attributes

Parameter	Value
Core type	In-order, 5-stage pipeline
ISA	MIPS-like ISA
Vector register file	2048b x 16 registers
Wide IO scratchpad	64 banks and 2048b wide (each bank 4Bx16K)
Cache hierarchy	L1-I: 32KB, 2-cycle latency
	L1-D: 24KB, 2-cycle latency
	Shared L2: 512KB, 10-cycle latency

B. Memory Configurations

Each system configuration includes one of the three following memory configurations.

DDR3. This represents a single DDR3 bus running at 666.67 MHz, typical for a low-power embedded system. The controller supports 8 DRAM devices (2 Gb/device) for a system capacity of 2GB and a peak memory bandwidth of 10.6 GB/s.

3D-Stacked Slow. A slow variant of the Hybrid Memory Cube (HMC) architecture [22], where there is only one lane and the speed of signaling in that lane is reduced from 10 GHz to 2.5 GHz to match the DDR3 bandwidth. Because of the use of surface mount and die stacking for the DRAM chips, this memory is more energy efficient.

3D-Stacked Fast A full-speed HMC design, but with only one lane (four would be typical). The signalling speed is the specified 10 GHz, netting a bandwidth of nearly 40 GB/s (4x the DDR3 performance), but still one-fourth the anticipated HMC bandwidth. This memory is the most energy efficient and highest performance.

C. Processor and Memory Simulation

The functional and timing model of core-processor are designed by using LISA language which can either be run directly on an instruction-level simulator or be compiled to synthesizable Verilog code using the commercial Synopsis Processor Design tool [28]. We use the instruction level simulation for counts, but a detailed timing and energy simulator involving the elements below for all other performance measurements.

For the cache hierarchy, we calibrate using power numbers from Cacti 6.5 [30]. We use a functional and timing model from the MarssX86 simulator [20] that we have integrated with the Synopsys simulation tools. This gives us reasonable execution speed and accurate results.

Regarding to DRAM systems, we use cycle-accurate DRAMSim2 tool [26] to model the traditional DDR3 and modern 3D-stacked DRAM systems. Both cache hierarchy and main memory system are integrated with core-processor through LISA-wrapper, enabling full integrated simulations for performance and energy.

IV. Experimental Results

A. Performance and Energy Breakdown for Baseline

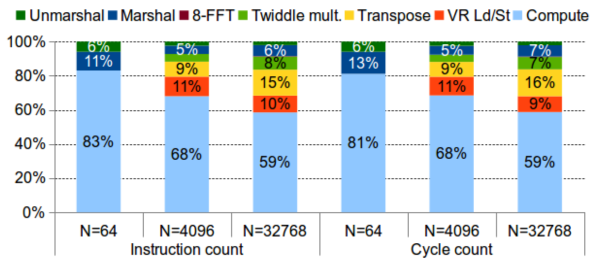


Fig. 2: Instruction and cycle counts of FFTs running on Baseline.

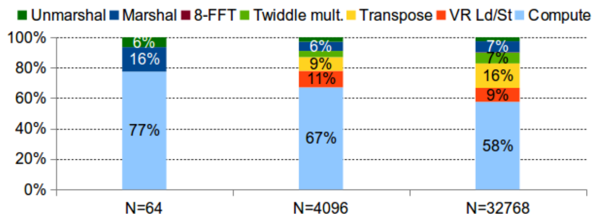


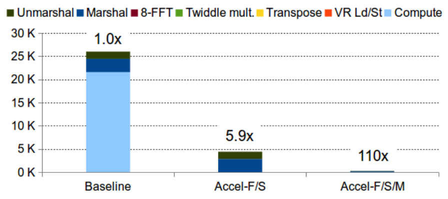
Fig. 3: Energy breakdown of FFTs running on Baseline.

Figures 2 and 3 show the properties of our model workload on the baseline architecture. First we examine the instruction and cycle counts. Naturally as the size of the FFT increases, the total amount of work grows as $O(NlogN)$, but the communication work grows faster, changing the balance significantly. For the larger FFTs more instructions and cycles are spent in data movement. With the core computation effort declining from 83% to 59% share for instructions and from 81% to 59% in cycles. Thus, Amdahl's law will be an increasingly restrictive limit on benefit achievable by the FFT accelerator. The energy graph shows a similar trend, with compute energy declining as a fraction for large FFTs and the data movement energy rising correspondingly.

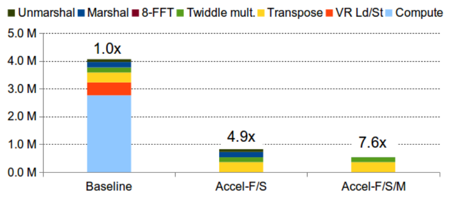
B. Effect of Accelerator

This section shows how the performance and energy efficiency of the FFT is improved with hardware support. We evaluate three FFT designs: simple 64-point FFT, full scale 4k-point FFT and odd size 32k-point FFT. For each, we measure *instruction count reduction, cycle count reduction, and energy reduction*. Three designs are considered in our evaluation:

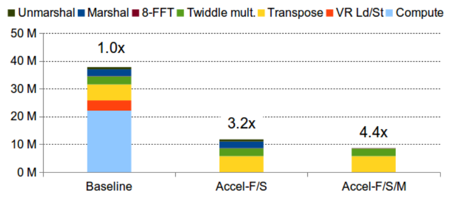
- **Baseline** is the RISC processor without FFT acceleration.
- **Accel-F/S** integrates a 64-point FFT accelerator, 256B vector register file, and a wide IO scratchpad memory to support 1-cycle vector load/store instruction used in VR Load/Store stage.



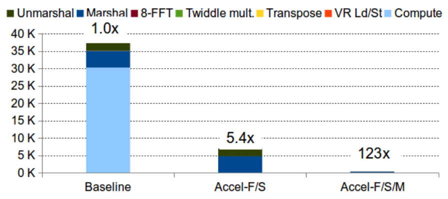
(a) N = 64, Instructions



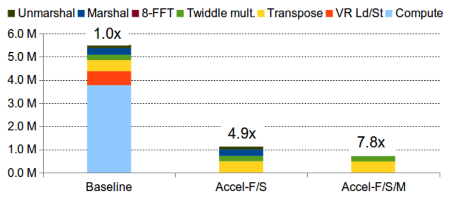
(b) N = 4096, Instructions



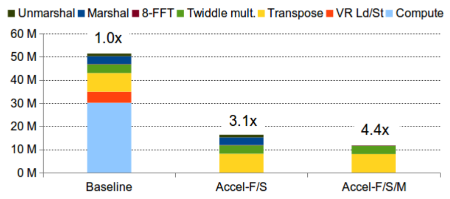
(c) N = 32768, Instructions



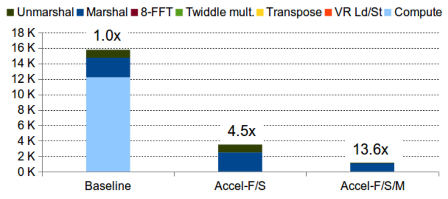
(d) N = 64, Cycles



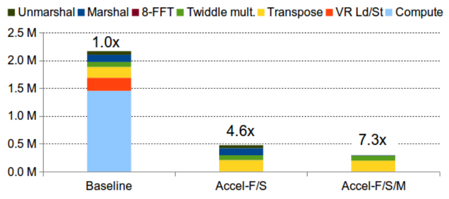
(e) N = 4096, Cycles



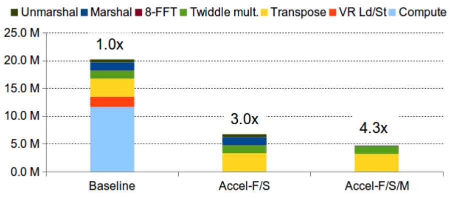
(f) N = 32768, Cycles



(g) N = 64, Energy (nJ)



(h) N = 4096, Energy (nJ)



(i) N = 32768, Energy (nJ)

Fig. 4: Instruction count, cycle count, and energy of various accelerator designs.

- **Accel-F/S/M** extends Accel-F/S with support to directly load 64B from main memory to the scratchpad.

To understand the contributions each addition makes to performance, cycles, and energy, we add them one at a time. In all cases, the system uses DDR3 memory.

Starting with the 64-point FFTs (see Figures 4a, 4d, 4g), we can see that the compute accelerator effectively eliminates more than 80% of the instructions and cycles, producing speedups of over 5x. Subsequent addition of the scratchpad memory and vector register adds little because the 64 point FFTs data movement is mostly contained within the compute accelerator. Energy improvement is slightly less, about 4.4x. However, with the hwardware support for moving a data block from main memory to local memory for the Marshal/Unmarshal phase, both cycle count and energy improvment are extremely increased to 123x and 13.6x respectively.

Moving to 4096-point FFTs (see Figures 4b, 4e, 4h), the benefits of the compute accelerator, as the fraction of compute is smaller. Instruction and cycle benefits of 3.3x, and matching benefits of 3.0x energy reduction are achieved. The smaller compute shifts the balance toward data movement, and as a result the scratchpad memory produces significant additional benefit which reduces instruction and cycles counts by another factor of 1.4x to a cumulative 4.9x improvement in the Accel-F/S design. When Marshal/Unmarshal phase are accelerated in Accel-F/S/M, we can obtain 7.8x and 7.3x improvment with respect to cycle count and energy efficiency.

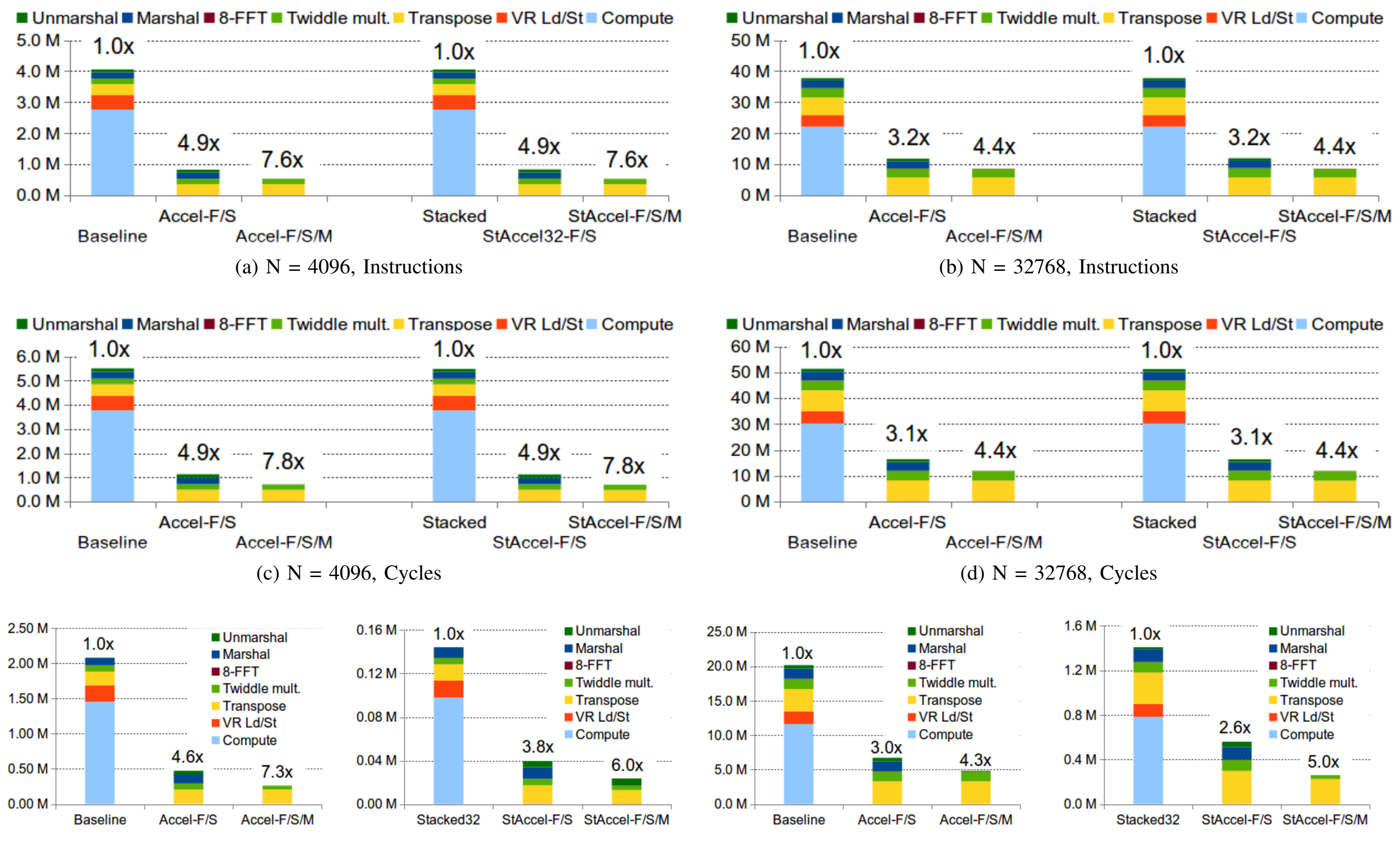
Finally at the largest FFTs we considered, 32768-point, the trend continues (see Figures 4c, 4f, 4i). The further shifting balance of FFT from computing and data movement limits the compute accelerator's benefits to 3.2x. A detailed look suggests that the data movement has complicated structure, including transpose, twiddle, and other data reshuffling that are not so simply addressed. Reducing these elements further is a good topic for further architecture research. In the 32k-FFT, hardware support for Marshal/Unmarshal phase contributes about 1.2x to the total of 4.4x cycle count reduction which turns out 4.3x energy efficency.

C. *Effect of Stacked Memory*

Next we examine the impact of future memory technologies, particularly 3D-stacked memories that both reduce energy and increase bandwidth. Figures 5a, 5c, 5b, and 5d show results for our 3D-fast configuration that supports 40GB/s bandwidth and dramatically lower energy per bit compared to DDR3.[1] Here we omit the 64-point results both in the interest of space, and because they are little affected by changes in the memory system. As expected, stacked memory has little effect on FFT performance or instruction count across the board. This is because the 4k and 32k-point FFTs are not main memory bandwidth limited – the entire data set fits in the on-chip scratchpad.

However, the energy results, shown in Figures 5f and 5h, tell quite a different story. The 3D-stacked memory reduces energy dramatically. Adding the computation accelerator for DDR3 memory system can give 7.3x and 4.3x benefits for 4k and 32k-point FFT (Accel-F/S/M) respectively. However, adding the 3D-stacked memory alone to the baseline (no compute accelerator) reduced overall energy by 14x. And combining the stacked memory with the accelerator gave extraordinary energy efficiency improvements of 86x and 70x for the 4k and 32k-point FFTs respectively. Overall, this suggests that processor-integrated accelerators will not only benefit from stacked memory in energy efficiency, that the system improvements arising

[1]The results for the 3D-slow configuration that matched bandwidths with the DDR3 configuration did not give appreciably different performance than the 3D-fast, and gave similar energy benefits.



(a) N = 4096, Instructions (b) N = 32768, Instructions

(c) N = 4096, Cycles (d) N = 32768, Cycles

(e) N = 4096, Energy (nJ) (f) N = 4096, Energy (nJ) (g) N = 32768, Energy (nJ) (h) N = 32768, Energy (nJ)

Fig. 5: Instruction count, cycle count, and energy of various accelerator designs.

from such memory will increase the importance of accelerators by shifting the system energy balance back towards the logic.

It should be noted that, for the Accel-F/S/M design, a majority of the cycle count is occupied by the RISC core performing a data transpose in the scratchpad memory. If this data is transposed with data-layout-transformation (DLT) accelerator that can generate addresses and access memory in parallel, both cycle and energy dissipation could be dramatically reduced. Note that the scratchpad has low access energy and multi-bank structure. If we added such as DLT to this system, at least 10x energy improvement for both 4k and 32-point FFT running on DDR3 memory can be achieved. In a 3D-stacked memory system, the overall energy benefit would easily exceed 100x.

V. Related Work

Fast Fourier Transforms (FFTs) are essential for many numeric computations from scientific computing to embedded signal processing. This calculation is so ubiquitous that it has been optimized for a number of different environments from multicores [6] to graphics processing units [31]. Due to power and temperature limits on the scalability of multicore processors, the general purpose computing community has also recently turned to the use of accelerators to increase performance in the face of physical constraints [5, 32]. Chung et al survey FFT performance on a number of architectures from general purpose to custom ASIC [3].

Of course, specialized processors have long been used to improve FFT energy efficiency for embedded applications. For example, He and Torkelson propose a processor design for computing 1024 point FFTs [10]. Additionally, a number of FFT processor designs have been proposed to increase the efficiency of OFDM processing [7, 21, 25, 34, 35]. These types of FFTs are optimized for energy efficiency on the relatively small number of samples required to process OFDM data. Still other designs propose FFT-specific processors that are more general and support arbitrary sizes [12, 15]. The need for these specialized FFT designs is so prolific that Nordin et al created a system that automatically generates customized FFT IPs [19].

These FFT processors are extremely energy efficient. For accelerating FFTs in more general environments, designs have been proposed to integrate FFT accelerators into general purpose processors. MorphoSys provides a general framework for integrating programmable logic into a processor pipeline [27]. Garzia et al couple a coarse-grain reconfigurable array (CGRA) specialized for an FFT with a general purpose core [8]. In this design, the core communicates with the CGRA through load and store operations. In contrast, Hussain et al integrate an FFT accelerator directly into the instruction set architecture (ISA) of a general purpose core [13]. As an alternative, Pedram et al integrate an FFT (and linear algebra) accelerator as a separate core in a multicore design [23]. These kinds of integrated FFT designs have been adopted by industry; e.g., Texas Instruments has digital signal processors with integrated hardware FFT accelerators [18].

The processor-integrated FFT core proposed in this paper is similar to prior integrated FFTs, but has some unique differences. The proposed design is integrated into the ISA of a general purpose core (see Table II), similar to that of Hussain et al [13]. However, our design does not use a separate memory array to communicate with the FFT accelerator. Instead, our FFT accelerator interfaces with the rest of the processor design through the vector register file. This integration allows the design to support additional accelerators that interface with each other through the same vector register file.

As accelerators improve performance and reduce the time spent in computation, the burden placed on the memory system will increase. Three-dimensional (3D) stacked memory architectures have been proposed as one method to address this memory "wall" [17, 22, 33]. Memory can be especially challenging for the FFT because as the size increases, the number of conflicting addresses increases as well [2, 12]. Several researchers have proposed directly integrating custom logic into memory to accelerate FFTs while mitigating potential memory bandwidth bottlenecks. Zhu et al integrate application-specific customized logic into a 3D memory stack [36]. Thorolfsson et al propose a customized 3D FFT design where the FFT processor is one layer in a memory stack [29]. In contrast, the study in this paper does not integrate the FFT into a memory stack, but investigates the effects of using 3D stacked memory, similar to the Hybrid Memory Cube[14, 22], as alternative to DDR3 DRAM as a backing store for our processor-integrated FFT design.

VI. Summary and Future Work

We have studied the performance and energy efficiency of a processor-integrated FFT accelerator, designed to support efficient integration of low-level and high level signal, image, and video processing. Our results show that an integrated accelerator can deliver major performance improvements, and be coupled efficiently into a general purpose core, enabling flexible programmed use. While benefits are high for FFTs that fit on chip, when a conventional DDR3 memory hierarchy is added, the performance benefits are more limited. Considering a 3D-stacked memory which is available now changes the situation dramatically. The much lower access energy exposes the benefits of the FFT accelerator, enabling much larger benefits. In short, in stacked DRAM systems, these processor integrated accelerators will have a major performance impact.

Future directions include exploration of transpose hardware support to further increase performance and energy benefits. Two interesting directions are possible – adding to the FFT accelerator or exploiting the vector units. Another direction is the exploration of mixed applications (including low level and high level processing), and their use to evaluate our processor integrated FFT accelerator.

VII. Acknowledgment

Funding of this work was provided in part by the Defense Advanced Research Projects Agency under award HR0011-13-2-0014 and the NSF under award NSF OCI-10-57921. We thank Peter Milder and the Spiral team whose FFT generator was used for the 64-point accelerator block. We thank Synopsis for generous university program support.

References

[1] M. Amduka et al. *Performance Modeling of a UAV System with an Integrated Radar-Tracker using Petri Nets*. Tech. rep. Lockheed Martin Advanced Technology Laboratories.

[2] J. Baek et al. "New address generation scheme for memory-based FFT processor using multiple radix-2 butterflies". In: *ISOCC*. 2008.

[3] E. Chung et al. "Single-Chip Heterogeneous Computing: Does the Future Include Custom Logic, FPGAs, and GPGPUs?" In: *MICRO*. 2010.

[4] J. W. Cooley et al. "An algorithm for the machine calculation of complex Fourier series". In: *Mathematics of Computation* 19 (1965), pp. 297–301. ISSN: 0025–5718.

[5] H. Esmaeilzadeh et al. "Dark silicon and the end of multicore scaling". In: *ISCA*. 2011.

[6] F. Franchetti et al. "Discrete fourier transform on multicore". In: *Signal Processing Magazine, IEEE* 26.6 (2009).

[7] B. Fu et al. "An Area Efficient FFT/IFFT Processor for MIMO-OFDM WLAN 802.11N". In: *J. Signal Process. Syst.* 56.1 (July 2009).

[8] F. Garzia et al. "Control Techniques for Coupling a Coarse-Grain Reconfigurable Array with a Generic RISC Core". In: *FPL*. 2010.

[9] G. Gerosa et al. "A Sub-2 W Low Power IA Processor for Mobile Internet Devices in 45 nm High-k Metal Gate CMOS". In: *Solid-State Circuits, IEEE J. of* 44.1 (2009).

[10] S. He et al. "Design and implementation of a 1024-point pipeline FFT processor". In: *CICC*. 1998.

[11] H. Hoffmann et al. "Selecting Spatiotemporal Patterns for Development of Parallel Applications". In: *Parallel and Distributed Systems, IEEE Trans. on* 23.10 (2012).

[12] C.-F. Hsiao et al. "A Generalized Mixed-Radix Algorithm for Memory-Based FFT Processors". In: *Circuits and Systems II: Express Briefs, IEEE Trans. on* 57.1 (2010).

[13] W. Hussain et al. "A Reconfigurable Application-specific Instruction-set Processor for Fast Fourier Transform processing". In: *ASAP*. 2013.

[14] J. Jeddeloh et al. "Hybrid memory cube new DRAM architecture increases density and performance". In: *VLSI Technology (VLSIT), 2012 Symp. on*. 2012.

[15] B. Jo et al. "New continuous-flow mixed-radix (CFMR) FFT Processor using novel in-place strategy". In: *Circuits and Systems I: Regular Papers, IEEE Trans. on* 52.5 (2005).

[16] G. Kane. *MIPS R2000 RISC Architecture*. Englewood Cliffs, NJ, USA: Prentice Hall, 1987.

[17] G. Loh. "3D-Stacked Memory Architectures for Multi-core Processors". In: *ISCA*. 2008.

[18] M. McKeown. *FFT Implementation on the TMS320VC5505, TMS320C5505, and TMS320C5515 DSPs*. 2013.

[19] G. Nordin et al. "Automatic Generation of Customized Discrete Fourier Transform IPs". In: *DAC*. 2005, pp. 471–474.

[20] A. Patel et al. "MARSS: A Full System Simulator for Multicore x86 CPUs". In: *DAC*. 2011.

[21] T. Patyk et al. "Low-power application-specific FFT processor for LTE applications". In: *SAMOS*. 2013.

[22] J. T. Pawlowski. "Hybrid memory cube (HMC)". In: *Hot Chips 23*. 2011.

[23] A. Pedram et al. "Transforming a linear algebra core to an FFT accelerator". In: *ASAP*. 2013.

[24] A. Reuther. *Preliminary Design Review: GMTI Processing for the PCA Integrated Radar-Tracker Application*. Tech. rep. ESC-TR-2003-070. MIT Lincoln Laboratory, 2004.

[25] D. Revanna et al. "A scalable FFT processor architecture for OFDM based communication systems". In: *SAMOS*. 2013.

[26] P. Rosenfeld et al. "DRAMSim2: A Cycle Accurate Memory System Simulator". In: *Computer Architecture Letters* 10.1 (2011).

[27] H. Singh et al. "MorphoSys: an integrated reconfigurable system for data-parallel and computation-intensive applications". In: *Computers, IEEE Trans. on* 49.5 (2000).

[28] Synopsys. *Automating the Design and Implementation of Application-Specific Processors (ASIP)*. Online document, http://www.synopsys.com/Systems/BlockDesign/processorDev.

[29] T. Thorolfsson et al. "A Low Power 3D Integrated FFT Engine Using Hypercube Memory Division". In: *ISLPED*. 2009.

[30] S. Thoziyoor et al. "A Comprehensive Memory Modeling Tool and Its Application to the Design and Analysis of Future Memory Hierarchies". In: *ISCA*. 2008.

[31] Y. Ukidave et al. "Quantifying the energy efficiency of FFT on heterogeneous platforms". In: *ISPASS*. 2013.

[32] G. Venkatesh et al. "Conservation cores: reducing the energy of mature computations". In: *ASPLOS*. 2010.

[33] D. H. Woo et al. "An optimized 3D-stacked memory architecture by exploiting excessive, high-density TSV bandwidth". In: *HPCA*. 2010.

[34] K.-J. Yang et al. "MDC FFT/IFFT Processor with Variable Length for MIMO-OFDM Systems". In: *IEEE Trans. Very Large Scale Integr. Syst.* 21.4 (2013).

[35] C. Yu et al. "A low-power 64-point pipeline FFT/IFFT processor for OFDM applications". In: *Consumer Electronics, IEEE Trans. on* 57.1 (2011).

[36] Q. Zhu et al. "A 3D-stacked logic-in-memory accelerator for application-specific data intensive computing". In: *3DIC*. 2013.

[37] V. Zivojnovic et al. "LISA-machine description language and generic machine model for HW/SW co-design". In: *VLSI Signal Processing*. 1996.