Does Arithmetic Logic Dominate Data Movement ? A Systematic Comparison of Energy-Efficiency for FFT Accelerators

Tung Thanh-Hoang[1], Amirali Shambayati[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

***Abstract*—Data format and data type selections for hardware accelerators design are important since it comes at the energy cost of arithmetic logic. However, these selections can affect the energy cost of data movement, as a large fraction of system energy, due to many memory-limited executions that are common on accelerator-centric heterogeneous systems. In this paper, we perform a systematic comparison to study the energy cost of varying data formats and data types w.r.t. arithmetic logic and data movement for heterogeneous systems in which both compute-intensive (FFT accelerator) and data-intensive accelerators (DLT accelerator) are added. We explore evaluation for a wide range of design processes (e.g., 32nm bulk-CMOS and projected 7nm FinFET) and memory systems (e.g., DDR3 and HMC).**

Our results show that when varying data formats, the energy costs of using floating point over fixed point are 5.3% (DDR3), 6.2% (HMC) for core and 0.8% (DDR3), 1.5% (HMC) for system in 32nm process. These energy costs are even decreasing to only 0.2% and 0.01% for core and system in 7nm FinFET process respectively with DDR3 memory (slightly increasing with HMC). Furthermore, we identify that the core and system energy of systems using fixed point, 16-bit, FFT accelerator is nearly half of using 32-bit which show the high proportion of system energy on the amount of moving data which is dependent on the selection of data type.

I. Introduction

Fast Fourier Transforms (FFT) is one of the most important kernels which has been widely used in many numeric computations ranging from scientific computing to embedded signal processing (e.g., finance, astronomy, quantum, image processing, SAR applications etc). Since FFT computation is data-intensive which requires numerous arithmetic operations thus computing FFT on general-purpose processors (GPP) may incur severe performance and energy inefficiency. This leverages the use of specialized hardware (a.k.a. accelerator) [1] to accelerate the FFT computation which has been proven as the most efficient approach among others (i.e., GPP with SIMD support, GPU or DSP) [2].

In a natural manner, numerical algorithms are usually developed with floating point data format in order to achieve sufficient accuracy (e.g., large dynamic range, easy scaling and truncation), error detection mechanisms (e.g., overflow, underflow, round-off error). However, the implementation of floating point arithmetics come at the costs of performance, power and area. These costs limit the usage of floating point, for example, in embedded domain (e.g., portable, wearable, biomedical devices etc.) and industrial automation [3].

Due to the costs of using floating point, hardware accelerator is often designed with fixed-point data format. Therefore, there are needed to develop efficient floating-to-fixed point conversion algorithms in order to minimize quantization error and this requires high efforts and design time (30-50% of total development cycle of software) [4, 5].

With the advanced process and technology (e.g., FinFET, FDSOI), the relative energy cost of arithmetic logic is dramatically scaled down. On the other hand, technology for memory, especially off-chip memory, is slowly scaled far behind logic [6] due other constraints (cost, reliability etc.). This causes the impact of data movement, as a fraction of system energy, becomes important concern. This issue is a challenging for traditional architectures but it will be extremely critical for future accelerator-based heterogeneous systems. where memory-limited execution is common [7].

The impact of data format and data type selection on the energy of accelerator-integrated systems raise two interesting questions: *1) Does the energy cost of arithmetic logic dominate data movement when varying data formnats? 2) Is the system energy proportional dependence on the amount of moving data when varying data types?*

To answer these question, we analyze the impact *-in system scenario-* of varying data formats (32-bit, floating versus fixed point) and data types (fixed point, 32-bit versus 16-bit) on core[1] and total system energy of accelerator-integrated systems. Our system are supported by a compute-intensive FFT accelerator and a data-intensive Data-Layout-Transformation (DLT) accelerator. We explore systematical energy comparison for varied processes (e.g., 32nm bulk-CMOS and projected 7nm FinFET [8]) and memory systems (e.g., DDR3 and HMC [9]).

Our results suggest that *-in the system perspective-* the relative energy cost using floating point over fixed point is decreasing (even engligible) in accelerated-based heterogeneous system for a wide range of process and memory. Therefore, we can adopt floating point arithmetic to take aforementioned advantages. This estimation also shows conclusively study on the effect of varying data types to the data movement cost which envisions the design of energy-proportional accelerator-based systems.

The specific contributions include:

1) A comparison of core energy for systems using 32-bit, floating and fixed point FFT accelerator that shows the cost of using floating point decreases when acceleration is added. For designs using 32nm process, the floating point FFT requires only 5.3% (DDR3) and 6.2% (HMC) greater energy. Furthermore, as we scale to projected 7nm FinFET process, using floating point only requires 0.3% (DDR3 and HMC) higher energy.
2) A comparison of system energy 32-bit fixed and floating point shows that the relative cost of using floating point FFT begins even smaller, and decreases further as acceleration is added. For the designs in 32nm, the floating point requires 1.5% (DDR3 and HMC) more energy. And for 7nm FinFET process, the floating point penalty shrinks to only 0.01% (DDR3 and HMC) higher energy.
3) Data type selection affects the amount of moving data resulting in the energy cost of data movement. We compare the core energy of fixed point, 16-bit and 32-bit FFT computation. Our evaluation show that using 16-bit fixed point results in 47% (DDR3), 48% (HMC) energy reduction compared to using 32-bit fixed point in 32nm bulk-CMOS and 7nm FinFET process respectively.

[1]Here, core energy refers as the *total energy consumsed by computation* detailed in Section IV.

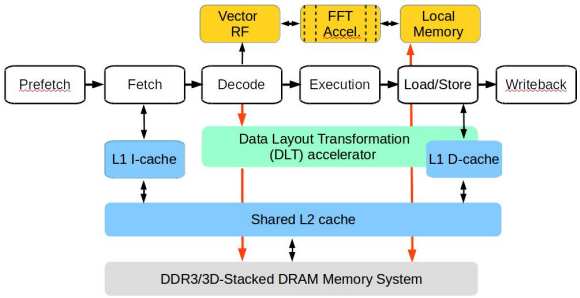


Fig. 1: Architecture of the Accelerated-integrated Systems.

4) At the system level, the relative energy of varying data types for fixed point, 32-bit over 16-bit data format is negligible across memory systems. We observe the energy cost due to using 16-bit fixed point is almost 50% (DDR3 and HMC) in 32nm bulk-CMOS process and even getting closer in 7nm FinFET process.

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. FFT Accelerator Design and Integration

Our goal is to use a fast and efficient FFT accelerator as a computing kernel of system in order to improve performance and energy efficiency of data-intensive applications (e.g., real-time video, image processing or molecular dynamics, etc.) while supporting high programmability (e.g., easy programming and scaling to compute large FFTs). To fulfill these requirements, the FFT accelerator is tightly-couple integrated into processor pipeline to take the advantage of instruction level integration. By doing so, the processor-integrated FFT accelerator can exploit the high bandwidth of system components (e.g., local memory and vector register file) while mitigating the synchronization cost between the accelerator and processor (this is critical drawback of loosely-coupled integration such as co-processors).

In detail, we integrate a highly optimized FFT computation array into a simple 5-stage RISC processor [10] which contains a wide vector register file. To enable energy efficiency for large FFTs, we use a multi-bank wide-IO local memory. The composition of these elements and the high level architecture are illustrated in Figure 1. The specifications of system components are described as below:

- **RISC core** is a lightweight 5-stage, 32-bit RISC design similar to the MIPS R2000 [10] which can run at 1Ghz to provide general-purpose computations such as 32-bit arithmetic, logic, and control flow operations.
- **Fft64Fl32** is an optimized hardware accelerator generated by using Spiral tool [11]. This accelerator can compute 64 complex value of single-precision, floating point (IEEE 754-1985) with a single instruction. In addition, the output of FFT accelerator is extended to perform parallel multiplication with twiddle factors stored in vector register which is a necessary step to compute large FFTs [12]. In overall, the Fft64Fl32 design includes 1056 32-bit adders and 504 32-bit multipliers. Synthesizing in 32nm process, the Fft64Fl32 accelerator runs at 1Ghz and contain 3.8M gates which is extremely larger than the RISC core.
- **Fft64Fx32** is designed in the same manner with the Fft64Fl32 accelerator to run at 1Ghz with same latency. This design is synthesized in 32nm and contains 2.3M gates.
- **Fft64Fx16** is designed in the same manner with the Fft64Fx32 accelerator to run at 1Ghz with same latency. This design is synthesized in 32nm and contains 1.1M gates.
- **Data Layout Transformation (DLT) accelerator** is a specialized hardware to accelerate *i)* data movement (i.e., gather/scatter stride data) between the local memory and off-chip memory *ii)* transposition of data layout inside the local memory. Data access pattern is compacted into a descriptor which is composed of the number of moving elements, stride of element, and size of element. The DLT accelerator supports sufficient data movement types (compared to Enhanced Direct Memory Access engine [13]) thus has low implementation cost. The simple pipeline of DLT accelerator can be integrated into the RISC core (Figure 1) without performance degradation[2].
- **Local memory** has 1MB size organized as 64 banks (16kB/bank) which can provide total bandwidth up to 512 GB. By buffering data in the local memory and scheduling memory accesses, we can achieve high performance and energy efficiency benefits.
- **Vector register file (VRF)** supports fast streaming data to FFT accelerator instead of using programmable IO or DMA which may incur significant performance overhead. In our evaluation, we use 18x2048b wide VRF which is sufficient to accommodate the selected FFT accelerator.
- **Cache hierarchy** is configured according to the model of low-power Intel Atom processor [14].

A. Software Interface of the FFT Accelerators

We implement processor-integrated FFT accelerator using LISA -*L*anguage for *I*nstruction *S*et *A*rchitcture- [15] which is supported by Synopsys Processor Designer and Synopsis Compiler Designer. The software interfaces of FFT and DLT accelerators are exposed as C-level intrinsic functions for easily programming. Then, we modify benchmarks to adopt these intrinsic functions for performance evaluation. The micro instructions and C-intrinsic functions of FFT and DLT accelerators are described in Table I.

For the FFT accelerator, vector instructions (*LoadLm2Vr* and *StoreVr2Lm*) are used to fast load/store vector register from/to the local memory. A wide IO, low latency interface of the local memory is possible because such kinds of memory is on-chip, small size, and near the core. The FFT computing instructions *Fft64Fl32/Fft64Fx32/Fft64Fx16* can compute 64 samples of 32-bit floating/32-bit fixed/16-bit fixed point respectively, taking 2048-bit vector registers as inputs and others as the destinations.

Regarding accelerating memory access, the combinations of data movement instructions (*DltGather*, *DltScatter*) and memory synchronization instructions (*DltGatherFence*, *DltScatterFence*) allow efficiently gather/scatter stride data between the local memory and off-chip memory (e.g., DDR3 and HMC). These gather and scatter instructions can support on-chip data layout transformation (e.g., transposing data inside the local memory) since the address range of local memory is mapped to the global memory.

B. Execution Phases of Fast Fourier Transform

In order to analyze the impact of varying data formats for the FFT computation in system scenario, we need to separate the execution

[2]The detail design of the DLT accelerator is out of scope of this paper. However, briefly, the DLT accelerator has three simple pipeline stages which are DLT's instruction decoder, fast parallel address generator and read/write memory access unit.

TABLE I: C-intrinsic function and micro instruction to access the FFT and DLT accelerators.

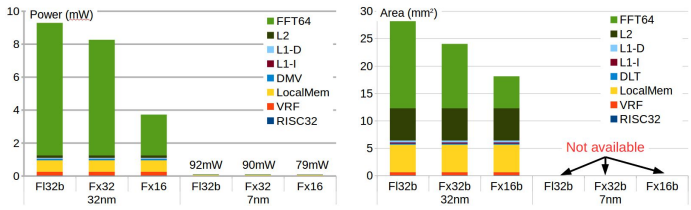
Instruction	Intrinsic	Functional description
Fft64Fl32	Fft64Fl32(Opt, *srcAddr0, *srcAddr1, *tdmAddr0, *tdmAddr1)	Compute 64-sample FFT, each point consists of two 32-bit single-precision floating point values. The FFT output can be multiplied with 64 complex floating point twiddle factors. The selection for computing direction (e.g., forward or inverse) and twiddle multiply of FFT is configured via *Opt*.
Fft64Fx32	Fft64Fx32(Opt, *srcAddr0, *srcAddr1, *tdmAddr0, *tdmAddr1)	Provide the same function as *Fft64Fx32* except data type is fixed point 32-bit.
Fft64Fx16	Fft64Fx16(Opt, *srcAddr, *tdmAddr)	Provide the same function as *Fft64Fx32* except data type is fixed point 16-bit.
LoadLm2Vr	LoadLm2Vr(vrDst, *srcAddr)	Load 256 bytes from local memory memory to vector register.
StoreVr2Lm	StoreVr2Lm(vrSrc, *dstAddr)	Store 256-byte vector register to local memory memory.
DltFormDesc	int32 DltFormDesc(number_of_elemments, stride, size_of_element)	Form 32-bit descriptor of DLT for data movement.
DltGather	DltGather(*mmAddr, *lmAddr, desc)	Gather data described by *desc* from main memory to local memory memory.
DltScatter	DltScatter(*lmAddr, *mmAddr, desc)	Scatter data described by *desc* from local memory memory to main memory.
DltGatherFence	DltGatherFence()	Memory synchronization used to check the completion of all previous gather instructions.
DltScatterFence	DltScatterFence()	Memory synchronization used to check the completion of all previous scatter instructions.

phases of FFT computation. We first describe the code example of a 4kx4k-sample 2D-FFT computation in Listing 1 which is factorized by using 4k-sample 1D-FFT computations and accelerator instructions. Then, we introduce standard terminologies for execution phases of FFT computation. These terms both explain specific software-hardware co-optimizations and the performance results presented in Sections III and IV respectively.

Here, we explain the execution phases of FFT computation:

- **Marshal/De-marshal**: Move data between the local memory and off-chip memory to setup efficient FFT computation.
- **Compute**: The ops and data movement for the butterfly operations inside FFT routine including multiply the FFT outputs with the twiddle factors (if needed) in order to create large FFTs.
- **Transpose**: Rearrange data layout stored in local memory at different scales to compute large FFTs by using the FFT accelerator.
- **Vector LD/ST**: Move the data between the local memory and the vector registers.

C. Hardware Implementation of FFT Accelerators



(a) Power (Watt) (b) Area (mm^2)

Fig. 2: Chip power and area breakdown in 32nm and 7nm process.

The RISC core and DLT accelerator are designed in LISA then compiled to Verilog by using Synopsis HDL Generator [16] while the Verilog of FFT accelerator is generated by Spiral tool [11] for high performance. We use Synopsis Design Compiler to synthesize the Verilog code with 32nm and 7nm standard cell library [8]. The power and area of caches and local memory are evaluated using Cacti 6.5 [17] in 32nm then extrapolated for 7nm process. Evaluation result is shown in Figure 2.

First, Figure 2a shows the total power of designs which contains the FFT accelerators designed using varying data formats and data types. Since the FFT accelerators are composed of a large number of arithmetic units thus they are obviously dominant of total chip power by Fft64Fx32 87% (32nm), 92% (7nm); Fft64Fx32 85% (32nm), 91% (7nm); Fft64Fx32 66% (32nm), 79% (7nm). When varying data formats, the Fft64Fl32 accelerator consumes 1.15x (32nm), 1.09x (7nm) higher power than Fft64Fx32 due to the power cost of using floating over fixed point for arithmetic units. For varying data types, comparing with the Fft64Fx16 accelerator, the Fft64Fix32 accelerator incurs 2.84x (32nm) and 3.86x (7nm) power overhead due to requiring wider arithmetics and complicated wiring.

Figure 2b shows the total area of on-chip components of evaluated designs. Although the area of FFT accelerators may not dominate other system components but they still take a significant fraction of total area, particularly, Fft64Fx32 57% (32nm), Fft64Fx32 49% (32nm), Fft64Fx32 18% (32nm)[3]. Comparing the area of Fft64Fl32 with Fft64Fx32 accelerator, the area cost of using floating over fixed point data format is 1.37x (32nm). While the cost of using 32-bit fixed point (Fft64Fx32) over 16-bit fixed point data type (Fft64Fx16) is about 2x (32nm)[3].

III. Methodology and Experiment

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

A. System Configuration

Table II summarizes the system configurations studied. The differences are in the presence of the accelerator (FFT, DLT), memory system (DDR3, HMC). All of the systems studied include a 1MB local memory and the basic RISC core elements. In addition, each system configuration is evaluated with two design processes (bulk-CMOS 32nm, projected FinFET 7nm).

- **Baseline** is the RISC processor without any accelerator support.
- **FFT-LdSt** integrates either 64-sample, 32-bit floating/32-bit fixed/16-bit fixed point FFT accelerator, 256B vector register file, and a wide IO local memory which supports 1-cycle vector load/store instructions. The address range of local memory is mapped to global address space thus it content can be accessed by using normal Load/Store (*LdSt*) instructions.
- **FFT-DLT** is extension of FFT-LdSt in that DLT accelerator is added to accelerate data movement between the main memory and local memory or data transposition inside the local memory.

[3]Cell area in 7nm is unvailable due to non-existing design rules for layout.

TABLE II: System configurations.

Configuration	Baseline	FFT-LdSt	FFT-DLT	Baseline	FFT-LdSt	FFT-DLT
FFT accelerator	No	Yes	Yes	No	Yes	Yes
DLT accelerator	No	No	Yes	No	No	Yes
Memory	DDR3	DDR3	DDR3	Stacked	Stacked	Stacked
Data format & type	Float32b/Fixed32b/Fixed16b					
Process	Use both 32nm and 7nm					

B. Memory Configuration

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

DDR3: This represents a single DDR3 bus running at 667 Mhz, typically used in embedded systems. 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.

HMC: It is a full-speed Hybrid Memory Cube (HMC) design running at 1.25 Ghz, but with only one lane (four would be typical) [9]. The signaling 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 is the most bandwidth and energy efficient memory system.

TABLE III: Simulation Platform Configuration.

Parameter	Value
Core type	In-order, 5-stage pipeline
ISA	MIPS-like ISA
Vector register file	2048b x 18 registers
Wide IO local memory	64 banks and 2048b wide
	(each bank provides 32bx16k entry)
Cache hierarchy	L1-I: 32KB, 2-cycle latency
	L1-D: 24KB, 2-cycle latency
	Shared L2: 512KB, 10-cycle latency

C. Processor, Memory Hierarchy and Benchmark Simulation

Processor: The functional and timing models of core-processor are designed by using LISA language which can be run directly on an instruction-level simulator or be compiled to RLT Verilog using the commercial Synopsis Processor Design tool [16]. We use the instruction level simulation for instruction counts, but detailed timing and energy simulations are conducted for all measurements.

Memory: For on-chip cache hierarchy, we calibrate using power numbers from Cacti 6.5 [17]. We use the functional and timing models of cache hierarchy from the MarssX86 simulator [18] that we have integrated with the Synopsys tools. This gives us a reasonable execution speed and accurate evaluation results. Regarding main memory system, we use cycle-accurate DRAMSim2 tool [19] to evaluate the traditional DDR3 and extend it to model the 3D-stacked DRAM-based HMC memory [20]. Both cache and main memory systems are integrated with RISC processor through LISA-wrapper, enabling full integrated cycle-accurate simulation for performance and energy. Table III shows the configuration of system simulation.

Benchmark: We use a large 4kx4k-sample FFT in which each sample is a complex number of 32-bit floating/32-bit fixed point/16-bit fixed point values. Since the required storage of input and immediate data are not able to fit in 1MB local memory thus we applied tiled-base implementation for large FFT computation (see Listing 1). Evaluation results are collected with respect to execution phases described in Section II-B.

D. Process and Technology for System Implementation

The hardware implementation of accelerator and core processor are evaluated using TSMC-based 32nm bulk-CMOS and projected 7nm FinFET process [8]. To minimize leakage power, we select high threshold-voltage libraries characterized at nominal operating conditions 1.05V-25C (32nm) and 0.3V-25C (7nm). For caches and local memory in 32nm process, we use Cacti 6.5 [17] to estimate energy-per-access and area footprint, then extrapolate for 7nm FinFET process. We assume that at 32nm process, core processor and accelerators run at 1Ghz while in 7nm process they are run at 4Ghz to compromise with the clock rate of off-chip memory.

Listing 1: Pseudo code of 32-bit, floating and fixed point 4kx4k-sample 2D-FFT based on the C-intrinsic functions of accelerators.

```
FFT32b_4k_1d (Opt, *src, *tdm4k, *dst) {
// Allocate Src, Dst, and twidle factors in the local memory
// Form necessary descriptors for DLT accelerator
desc = DltFormDesc(64, 64, 8);
// Marshal phase: move data from DRAM to LocalMem;
DltGather(lm_src,   src,   desc);
DltGather(lm_tdm4k, tdm4k, desc);
DltGatherFence();
// Transpose phase
for (i=0; i<128; i+=2)
  DltGather(lm_dst+i*64, lm_src+i, desc);
DltGatherFence();
// First computing 64 times of 64-sample FFT
 for (i=0; i < 8192; i+=1024) {
  // Vector LdSt
   for (i=0; i<16; i++) LoadLm2Vr(lm_dst+i*64, Vr[i])
  // Compute
   for (i=0; i<16; i+=2) {
    LoadLm2Vr(lm_tdm4k+i*64,    Vr[16]);
    LoadLm2Vr(lm_tdm4k+i*64+64, Vr[17]);
    Fft64F132/Fft64Fx32 (Opt,Vr[i],Vr[i+1], Vr[16],Vr[17]);
   }
  // Vector LdSt
   for (i=0; i<16; i++) StoreVr2Lm(Vr[i], lm_src+i*64);
 }
// Transpose phase
 for (i=0; i<128; i+=2)
   DltScatter(lm_dst+i, lm_src+i*64, desc);
 DltScatterFence();
// Second computing 64 times of 64-sample FFT ...
}

Tile_FFT32b_4kx4k_2d (Opt, *src, *tdm4k, *dst) {
// Allocate Src, Dst, and twidle factors in the local memory
// Form necessary descriptors for DLT accelerator
desc1 = DltFormDesc( 512,   1, 64);
desc2 = DltFormDesc(4096,   1, 64);
desc3 = DltFormDesc(4096,   8,  8);
desc4 = DltFormDesc(4096, 512, 64);
// Marshal phase: move data from DRAM to LocalMem;
DltGather(lm_tdm4k, tdm4k, desc);
DltGatherFence();
// First computing 4096 times of 4096-sample FFT
for (k = 0; k < 8192; k+=16) {
 // Marshal phase excuted for single tile
   DltGather(lm_src, src+k*4096, desc2);
   DltGatherFence();
 // Compute phase
  for (j=0; j<16; j+=2)
   FFT32b_4k_1d(Opt, lm_src+j*4096, lm_tdm4k, lm_dst+j
       *4096);
 // Transpose for block to avoid exceed local memory
     address
  for (j=0; j < 16; j+=2)
    DltScatter(lm_src + j, lm_dst + j*4096, desc3);
  DltScatterFence();
 // Demarshal phase: move data from LocalMem to DRAM;
  DltScatter(dst + k, lm_src, desc4);
  DltScatterFence(); // Done tile computation
}
// Second computing 4096 times of 4096-sample FFT ...
}
```

IV. Experimental Results

We start this section by defining *evaluation metrics such as core and system energy* via the energy of aforementioned execution phases.

$$E_{Core} = E_{Compute} + E_{VectorLd/St} + E_{Transpose}$$
$$E_{System} = E_{Core} + E_{Marshal} + E_{De-marshal}$$

Then we show, in Section IV-A, how varying data formats (32-

bit, floating and fixed point) affects the cost energy (excluding data movement) and total system energy of the accelerator-integrated designs. Then, we analyze the impact of varying data types (fixed point, 32-bit and 16-bit) on the energy cost of data movement in Section IV-B.

A. *Energy Impact of Varying Data Formats*

1) Core energy cost of using 32-bit, floating over fixed point: In this section, we demonstrate that the core energy cost of using floating over fixed point data format (same 32-bit length) in 4kx4k 2D-FFT computation keeps decreasing in advanced process when the accelerators are added.

In 32nm process, Figures 3a and 3b show the absolute and relative core energy respectively. For the *Baseline* designs, the energy cost of *Compute* phase due to arithmetic logic is significant compared to other phases, therefore the core energy of using floating point is 21.6% (DDR3) and 21.7% (HMC) higher than using fixed point. When only the FFT accelerator is added (in *FFT-LdSt* designs) to improve performance and reduce the energy consumption of *Compute* and *Vector LD/ST* phases, the energy of *Transpose* phases becomes superior and dominates core energy for both data formats. As a result, the core energy cost between 32-bit, floating and fixed point is very small as 1.05% (DDR3) and 1.06% (HMC), because they are dominated by the same amount of energy consumed for data transposition. On the other hand, when both FFT and DLT accelerators are used to simultaneously accelerate computation and data movement, the energy cost of varying data formats becomes visible as 5.3% (DDR3) and 6.2% (HMC).

In 7nm FinFET process, arithmetic logic is more energy efficient than 32nm process [21], therefore the energy cost of varying data formats is significantly decreased. As shown in Figures 4a and 4b, the core energy cost of varying data format in the *Baseline* designs is less than 1.45% (DDR3) and 1.48% (HMC). This cost decreases to lower than 0.3% (almost for DDR3 and HMC) when the FFT and DLT accelerators are added.

2) System energy cost of using 32-bit, floating over fixed point: By taking into account the energy impact of off-chip memory, we verify that the system energy cost of using floating over fixed point data format (same 32-bit length) is further decreasing compared to the core energy.

For 32nm process, Figures 5a and 5b report the absolute and relative system energy respectively. Not surprisingly, the system energy cost of using floating over fixed point is even smaller and keeps decreasing from the *Baseline* designs to accelerator-based designs (e.g., *FFT-LdSt* and *FFT-DLT*) because the energy consumption of data movement can overwhelm the computation. We observed that, in *FFT-DLT* design, using floating data format just incurs 0.8% (DDR3) and 1.5% (HMC)[4] system energy overhead compared to using fixed point.

For projected 7nm FinFET process, our result indicates that the system energy cost between 32-bit, floating point and fixed point is *negligible* ($< 0.01\%$ almost same for DDR3 and HMC memory in the FFT-DLT designs (see in Figures 6a and 6b). This observation can conclude that the energy cost of data movement, as a large fraction of system energy, can dominate arithmetic logic and the decision to use floating point instead of fixed point is feasible, at least it does not come at the high cost of system energy while taking numerous other advantages (e.g., easy scaling, high accuracy).

[4]Since HMC is 7x-9x more energy efficient than DDR3 memory [22] thus its impact on system energy is less than DDR3 that results in higher energy cost with HMC.

B. *Energy Impact of Varying Data Types*

1) Core energy cost of using fixed point, 16-bit over 32-bit: To study the impact of varying data types on core energy, we evaluate 4kx4k 2D-FFT computation in which the FFT accelerator is design to support fixed point, 16-bit and 32-bit. Recall that fixed point, 16-bit and 32-bit FFT accelerators can be designed with same latency thus they can show the same performance for FFT computation.

Figures 7a and 7b show the absolute and relative core energy of using fixed point, 16-bit over 32-bit in 32nm process, for both DDR3 and HMC memory systems. The result suggests that the selection of data type dominates amount of moving data thus has significant impact on core energy. For instance, the relative core energy of varying data types in *Baseline* designs is about 73% (DDR3 and HMC) because both compute and data movement are affected by data type (i.e., wider arithmetic, large amount of moving data). Moving to the *FFT-LdSt* designs in that computation energy is reduced by the FFT accelerators and the rest of core energy is dominated by data movement. Therefore, the energy cost of varying data formats is decreased to 55% (DDR3 and HMC) in *FFT-LdSt* designs. In the most energy-efficient *FFT-DLT* designs, although the energy consumption of data movement is reduced using the DLT accelerator, its remainder still dominates core energy. Definitely, the core energy would be affected by the amount of data moving (by almost factor of two) and our evaluation showed that energy cost of uisng fixed point, 32-bit is 47% (DDR3) and 48% (HMC) higher than using 16-bit in 32nm processes.

Exploring the core energy comparison for projected 7nm FinFET process, we observed that using fixed point, 32-bit has 49% (DDR3 and HMC) higher energy than using 16-bit. These results are slightly higher than ones in 32nm process because logic, in 7nm process, is highly energy efficient which make the energy cost of the data movement more prominent than computation (see Figures 8a and 8b).

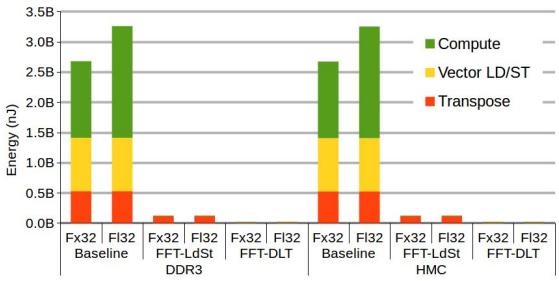
2) System energy cost of using fixed point, 16-bit over 32-bit: In order to have a complete judgment for the effects of varying data types to data movement, we evaluate the system energy of fixed point, 32-bit and 16-bit.

In 32nm process, when both accelerators are added into *FFT-DLT* designs, the energy cost of data movement superiors the computation and dominates system energy. The estimation shows around using fixed point, 32-bit results in 50% increasing of system energy compared to using 16-bit when both the FFT and DLT accelerators are added (Figures 9a and 9b).

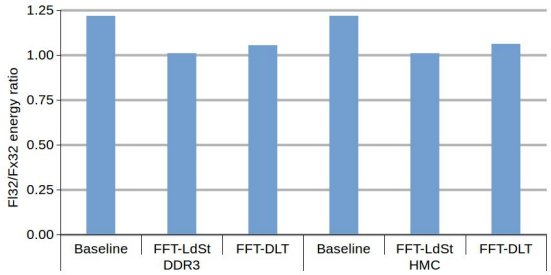
In 7nm process, we found that the system energy cost between fixed point, 16-bit and 32-bit is relatively closed to 50% (DDR3 and HMC) which is dominated by data movement via *Marshal* and *Demarshal* phases. Our comprehensive study for system energy suggests that varying data types is more effective (and/or sensitive) to the energy cost of data movement than arithmetic logic (Figures 10a and 10b). This also suggest that by minimizing amount of moving data and moving data more efficiently can result in highly energy-proportional accelerator-based systems.

V. Related Work

Fast Fourier Transform (FFT) is indispensable computing kernel for a variety of image and signal processing applications. There has been many efforts to implement FFT compute in various architectures such as general purpose processor (Intel-AVX [23]), compute-intensive (GPU [24]), reconfigurable system (FPGA [25]) and hardware accelerator (ASIC [26]) which target to high throughput, energy

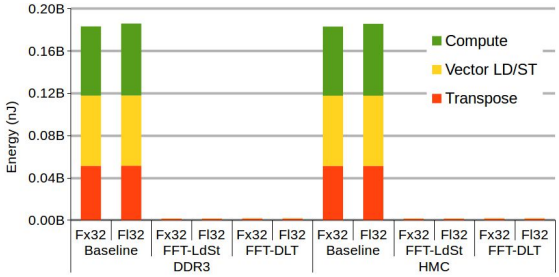


(a) Core energy (nJ) of 32-bit, floating and fixed point.

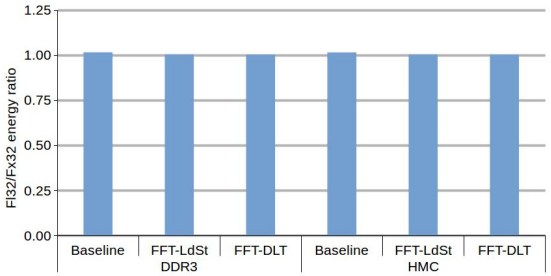


(b) Relative core energy of 32-bit, floating and fixed point.

Fig. 3: **32nm process, CORE** energy of 32-bit, floating and fixed point.

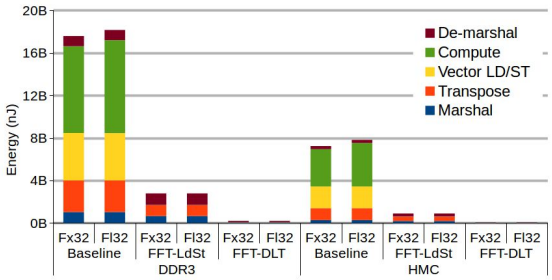


(a) Core energy (nJ) of 32-bit, floating and fixed point.

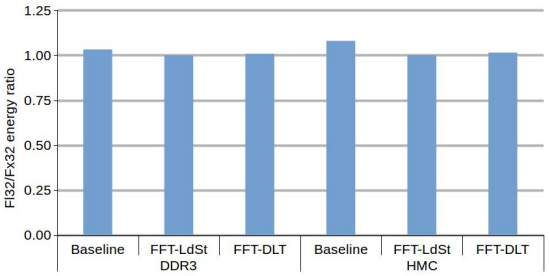


(b) Relative core energy of 32-bit, floating and fixed point.

Fig. 4: **7nm process, CORE** energy of 32-bit, floating and fixed point.

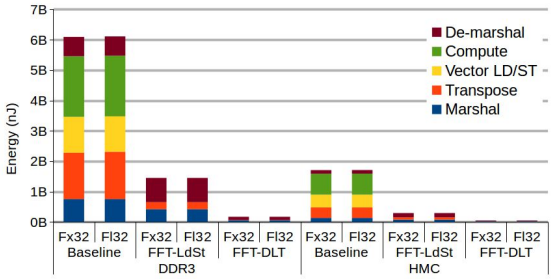


(a) System energy (nJ) of 32-bit, floating and fixed point.

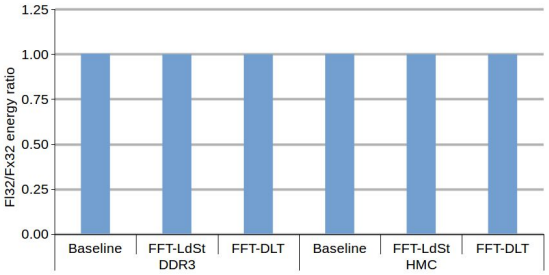


(b) Relative system energy of 32-bit, floating and fixed point.

Fig. 5: **32nm process, SYSTEM** energy of 32-bit, floating and fixed point.

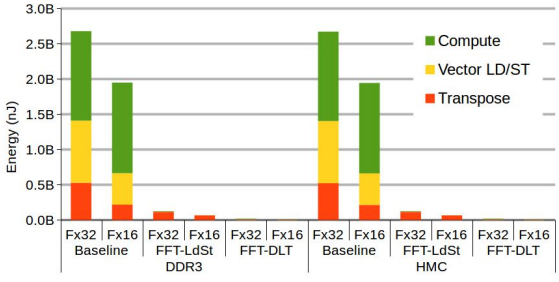


(a) System ernergy (nJ) of 32-bit, floating and fixed point.

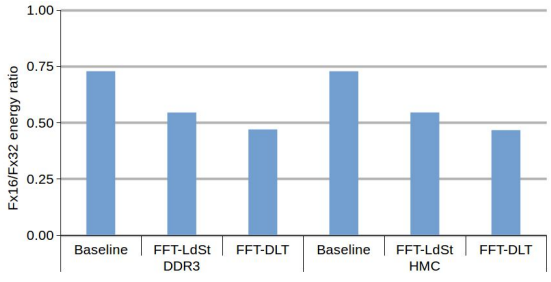


(b) Relative system energy of 32-bit, floating and fixed point.

Fig. 6: **7nm process, SYSTEM** energy of 32-bit, floating and fixed point.

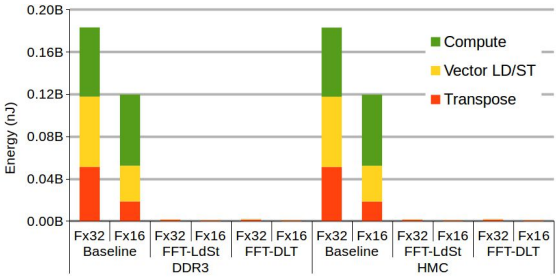


(a) Core energy (nJ) of fixed point, 16-bit and 32-bit.

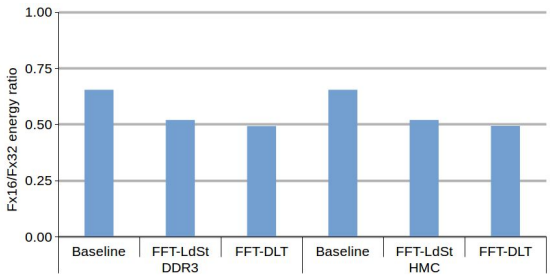


(b) Relative core energy of fixed point, 16-bit and 32-bit.

Fig. 7: **32nm process, CORE** energy of fixed point, 16-bit and 32-bit.

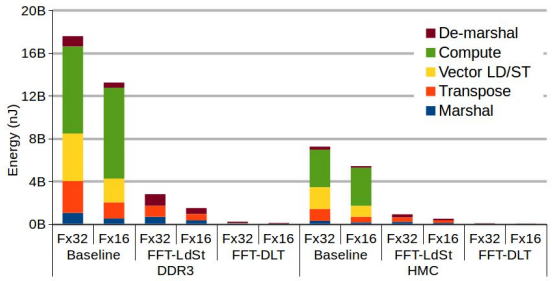


(a) Core energy (nJ) of fixed point, 16-bit and 32-bit.

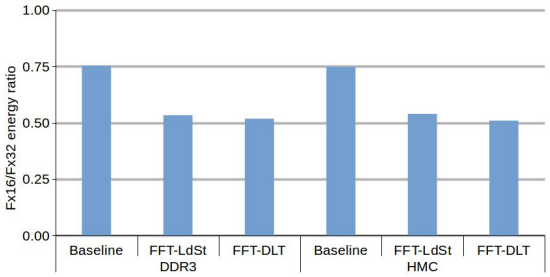


(b) Relative core energy of fixed point, 16-bit and 32-bit.

Fig. 8: **7nm process, CORE** energy of fixed point, 16-bit and 32-bit.

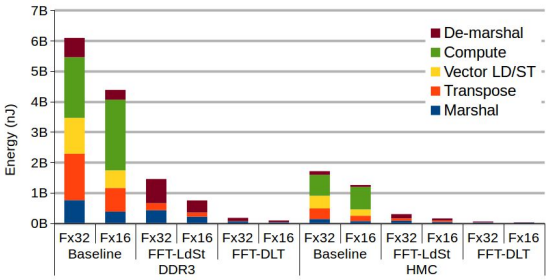


(a) System energy (nJ) of 32-bit and 16-bit fixed point.

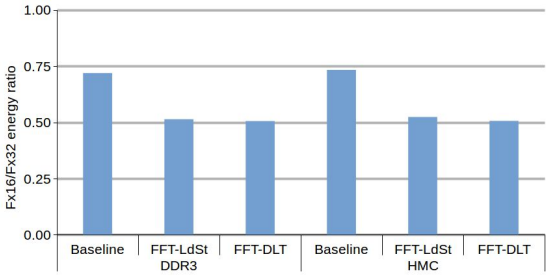


(b) Relative system energy of fixed point, 16-bit and 32-bit.

Fig. 9: **32nm process, SYSTEM** energy of fixed point, 32-bit and 16-bit.



(a) Energy (nJ) of 32-bit and 16-bit fixed point.



(b) Relative system energy of fixed point, 16-bit and 32-bit.

Fig. 10: **7nm process, SYSTEM** energy of fixed point, 16-bit and 32-bit.

efficiency or flexibility (compute arbitrary size of FFT). To maintain aforementioned features for FFT compute, a competent approach is to integrate the FFT –optimized and fixed size ASIC– accelerators into general purpose processors and support a sufficient (and highly programmable) instruction set to efficiently compute large FFT [7, 27]. For an overview, we recommend the work done by Chung et al. [28] which comprehensively surveys the performance and energy efficiency of FFT running on a number of architectures from general purpose processors to custom ASICs.

The performance gap between processors and memory is being increased due to uncompromising advances of CMOS and memory technology while many applications (e.g., large FFT) tends to be memory-intensive. This explicit means that the performance and energy cost of data movement are very critical to system energy efficiency in next generations of CMOS technology. In fact, the energy cost due to accessing memory has been addressed as a key challenging for large FFT computation which incurs a number of non-locality addresses resulting in inefficient memory access [29, 30]. Several works have been proposed to reduce this cost, for instance, in [31, 32], a custom logic is added into logic dies of memory system to support data transformation operation of FFT computation. This work definitely incurs design cost to change logic layer of 3D-stacked memory and low programmability for general-purpose data movement. Ren et al. [33] simply exploit extra memory controllers to increase memory bandwidth and hide data movement latency by computation time but it comes at the hardware cost of DRAM interface design. In [34, 35], a comprehensive study has been conducted in that scratchpad memory (local memory) hierarchy can be designed and optimized to buffer data (compute on on-chip memory) and in-situ re-arrange data layout for energy-efficiency.

Our design exploits the advantages of local memory and carefully considers the data communication between FFT compute and memory components. Furthermore, we leverage numerous features of system components such as high bandwidth, fast access multi-bank local memory, wide vector register, the data-intensive DLT and compute-intensive FFT accelerators to extremely improve system energy efficiency. All of these components are properly designed and integrated into processor pipeline for highly programmable to combat with the energy cost of data movement, data re-arrangement and computation of FFT application.

Another landscape of design FFT is to develop the efficient representations of data format and data type. The advantages of using fixed point (e.g., high clock rate, low (silicon) cost etc.) has been shown as the wide existence of fixed point processing DSP chips and embedded processors [3, 36]. To design fixed point processor, there is a large number of works on developing floating-to-fixed point conversion algorithms to minimize bit-width under of error constraint which turns out significant reduction of power and area of hardware

[37, 38].

On the other hand, floating point format is more naturally used than the fixed point data type (e.g., in high-level modeling system such as MATLAB, LabView). Because floating point support high dynamic range, tunable accuracy capability and error tolerant programs thus it can eliminate design effort paid for developing complex conversion algorithms. As analyzed in [4, 5], about 30-50% of design time is due to optimizing floating-point to fixed-point conversion algorithm.

In addition, there are previous works have shown that using fixed point would turn out higher energy efficiency than using floating point in FFT compute [39, 40] but that claims only address the energy cost of hardware logic rather than taking system perspectives in consideration. Our evaluation shows that using floating and taking its advantages over fixed point in accelerator-centric systems is highly potential because both relative core and system energy cost are small indeed while the area cost can be mitigated, for example, by utilizing *Dark silicon* [41].

VI. Summary and Future Work

We have studied the energy cost of using varying data formats and data types in the processor-integrated FFT accelerator which can support efficient integration of low-level and high level signal, image, and video processing. Our results demonstrate that the energy cost of using floating over fixed point for core and system are negligible in systems which are implemented in advanced CMOS process and combined with energy-efficient memory. when –compute intensive– FFT and –data intensive– DLT accelerators are added. It may imply that using floating point instead of fixed point is feasible in order to achieve accuracy, error-tolerance, and reduce design time for software development.

Furthermore, we identify that by coupling data and compute-intensive accelerators, the system energy is more dependent on the slection of data type than data format. Thus it suggests to develop short floating point data format (e.g., IEEE incompatible floating point) in order to minimize amount of moving data (improve system efficiency) while exploiting the advantage of floating point.

Future directions include exploration of transpose hardware support to further increase performance and energy benefits. Following research directions are possible: adding 2D/3D gather/scatter instructions for DLT, supporting gather/scatter between local memory and vector registers to eliminate vector Ld/St, and adding multiple DLT accelerators for multicore heterogeneous systems.

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 acknowledge Peter Milder (StonyBook), the Spiral team (CMU) whose provide FFT generator used for the 64-point accelerator block and Synopsis Inc. for generous university program support.

References

[1] C. C. et al. “A taxonomy of accelerator architectures, their programming models”. In: *J. of IBM Research and Development* (2010).

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

[3] G. Frantz et al. *Comparing Fixed- and Floating-Point DSPs*. Tech. rep. Texas Instruments, 2004.

[4] M. Clark et al. *Accelerating Fixed-Point Design for MB-OFDM UWB Systems*. Tech. rep. CommsDesign, 2005.

[5] T. Grotk et al. “Evaluation of HW/SW Tradeoffs Using Behavioral Synthesis”. In: 1996.

[6] O. Mutlu. “Memory Scaling: A Systems Architecture Perspective”. In: *Technical talk at MEMCON*. 2013.

[7] H. T. Tung et al. “Performance and Energy Limits of a Processor-integrated FFT Accelerator”. In: *HPEC*. 2014.

[8] USC-SPORT. *DARPA Empower Project: 7nm FinFET Cell Library*. Download link http://sportlab.usc.edu/downloads/packages/. 2014.

[9] J. T. Pawlowski. “Hybrid Memory Cube (HMC)”. In: *HotChips*. 2011.

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

[11] P. Milder et al. “Computer Generation of Hardware for Linear Digital Signal Processing Transforms”. In: *ACM Trans. Des. Autom. Electron. Syst.* 17.2 (Apr. 2012).

[12] J. W. Cooley et al. “An algorithm for the machine calculation of complex Fourier series”. In: (1965).

[13] *TMS320C6748/46/42 and OMAP-L138 Processor Enhanced Direct Memory Access (EDMA3) Controller*. Tech. rep. Texas Instruments, 2010.

[14] 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).

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

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

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

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

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

[20] H. M. C. Consortium. *Hybrid Memory Cube Specification 1.1*. Tech. rep. 2014.

[21] Q. Xie et al. “Performance Comparisons between 7nm FinFET and Conventional Bulk CMOS Standard Cell Libraries”. In: *TCAS-II* (2015).

[22] B. Shekhar et al. “Exascale Computing A fact or a fiction ?” In: *Technical talk at IPDPS*. 2013.

[23] S. Umberto et al. *Signal Processing on Intel®Architecture: Performance Analysis using Intel®Performance Primitives*. Tech. rep. Intel.

[24] N. Govindaraju et al. “High Performance Discrete Fourier Transforms on Graphics Processors”. In: *SC*. 2008.

[25] S. Langemeyer et al. “A FPGA architecture for real-time processing of variable-length FFTS”. In: *ICASSP*. 2011.

[26] K. Maharatna et al. “A 64-point Fourier transform chip for high-speed wireless LAN application using OFDM”. In: *JSSC* (2004).

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

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

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

[30] 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).

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

[32] B. Akin et al. “HAMLeT: Hardware Accelerated Memory Layout Transform within 3D-stacked DRAM”. In: *HPEC*. 2004.

[33] C. Ren et al. “Energy Optimizations for FPGA-based 2-D FFT Architecture”. In: *HPEC*. 2014.

[34] K. Seager et al. “Efficient HPC Data Motion via Scratchpad Memory”. In: *DISCS*. 2012.

[35] L. Carrington et al. *Thrify: UCSD/SDSC Final Report: Energy Efficient Execution of Large-scale HPC Applications*. Tech. rep. Department of Energy, 2014.

[36] *Analog Devices 16/32-bit fixed-point Blackfin digital signal processors*. Tech. rep. Analog Devices.

[37] C. Shi et al. “Automated fixed-point data-type optimization tool for signal processing and communication systems”. In: *DAC*. 2004.

[38] C. Yang et al. “New quantization error assessment methodology for fixed-point pipeline FFT processor design”. In: *SoCC*. 2014.

[39] R. Koutsoyannis et al. “Improving fixed-point accuracy of FFT cores in O-OFDM systems”. In: *ICASSP*. 2012.
[40] G. Gokul et al. “Area, and Power Performance Analysis of a Floating-point based Application on FPGAs”. In: *HPEC*. 2013.
[41] H. Esmaeilzadeh et al. “Dark silicon and the end of multicore scaling”. In: *ISCA*. 2011.