Understanding the Design Space of DRAM-Optimized Hardware FFT Accelerators

Berkin Akın, Franz Franchetti, and James C. Hoe
ECE Department, Carnegie Mellon University, Pittsburgh, PA, USA
{bakin, franzf, jhoe}@ece.cmu.edu

Abstract—**As technology scaling is reaching its limits, pointing to the well-known memory and power wall problems, achieving high-performance and energy-efficient systems is becoming a significant challenge. Especially for data-intensive computing, efficient utilization of the memory subsystem is the key to achieve high performance and energy efficiency. We address this challenge in DRAM-optimized hardware accelerators for 1D, 2D and 3D fast Fourier transforms (FFT) on large datasets. When the dataset has to be stored in external DRAM, the main challenge for FFT algorithm design lies in reshaping DRAM-unfriendly memory access patterns to eliminate excessive DRAM row buffer misses. More importantly, these algorithms need to be carefully mapped to the targeted platform's architecture, particularly the memory subsystem, to fully utilize performance and energy efficiency potentials. We use automatic design generation techniques to consider a family of DRAM-optimized FFT algorithms and their hardware implementation design space. In our evaluations, we demonstrate DRAM-optimized accelerator designs over a large tradeoff space given various problem (single/double precision 1D, 2D and 3D FFTs) and hardware platform (off-chip DRAM, 3D-stacked DRAM, ASIC, FPGA, etc.) parameters. We show that generated pareto-optimal designs can yield up to 5.5x energy consumption and order of magnitude memory bandwidth utilization improvements in DRAM, which lead to overall system performance and power efficiency improvements of up to 6x and 6.5x respectively over conventional row-column FFT algorithms.**

I. Introduction

Single and multidimensional Fast Fourier Transforms (FFT) are important computational kernels to accelerate in hardware for scientific data analysis, digital signal processing and high performance computing applications. For large problem sizes, the computation proceeds in stages by transferring portions of the dataset to and from off-chip DRAMs continuously. Standard multi-stage FFT algorithms require large strided DRAM accesses, which inefficiently utilize the DRAM row buffer. There exist novel DRAM-optimized FFT algorithms, which can make full use of each opened DRAM row (e.g. [1], [2]). The core idea of these algorithms is to use a tiled data layout that removes strided memory accesses. Making full use of each opened DRAM row minimizes the number of row buffer misses which substantially reduces the energy consumption and allows achieving close to the theoretical peak memory bandwidth.

Efficient use of the memory subsystem being the key challenge, high-performance and energy-efficient accelerators require well-balanced implementations where none of the overall system components are over-provisioned or under-utilized. The overall system consists of two main components: (i) on-chip computation resources (i.e. arithmetic units and local SRAM) and (ii) main memory (i.e. DRAM). Specifically, (i) throughput and parallelism of computation, and (ii) row buffer locality and bank level parallelism in DRAM are the key aspects of the design space which determine the balance of the overall system. Control over (i) computation and (ii) DRAM utilization is achieved via algorithmic and architectural design parameters (e.g. tile parameters of tiled FFTs, streaming width, frequency, etc.). However, inherent tradeoffs between the algorithmic and architectural parameters lead to rich possibilities in the design space. Different FFT problems on different hardware platforms require special attention and design effort which makes a single efficient solution unachievable. Existing work either focuses on a specific platform/problem ([1], [3], [4]) or targets a generic but simplistic machine model [2]. Further, these works primarily aim high performance implementations. Achieving both high-performance and energy-efficient implementations on various platforms, on the other hand, is only possible through careful mapping of the algorithms to the architectures by analyzing the rich design tradeoffs.

Contribution. This paper evaluates the energy/performance potentials and the key design tradeoffs in mapping DRAM-optimized FFT algorithms (1D, 2D and 3D) to different hardware platforms (e.g. off-chip DRAM, 3D-stacked DRAM, ASIC, FPGA) using automated techniques. We modify Spiral [5] to automatically generate the hardware implementations of the DRAM-optimized FFT algorithms. By using the automatic design generation framework, the designer can tradeoff various architectural and algorithmic parameters to achieve a goal of performance or power efficiency, under a given limited budget of memory bandwidth, power and hardware resources. Studying the designs using synthesis at the Verilog RTL level, we show the rich tradeoff possibilities in the design space and the opportunity for DRAM-specific optimizations to yield an order of magnitude memory bandwidth utilization and 5.5x energy consumption improvements in DRAM, which lead to overall system performance and power efficiency improvements of up to 6x and 6.5x respectively over conventional row-column FFT algorithms.

Paper outline. Section II first surveys the related prior work. Section III offers background on FFTs and on DRAM memory systems. Then, Section IV discusses DRAM-friendly FFT algorithms. Section V provides the architecture and algorithm design space details. Section VI presents our experimental results including (a) power/performance tradeoffs for various problem and platform configurations, (b) DRAM-specific and overall system improvements, (c) actual hardware implementations of the generated accelerators on FPGA. Fi-

This work was sponsored by DARPA under agreement HR0011-13-2-0007. The content, views and conclusions presented in this document do not necessarily reflect the position or the policy of DARPA.

nally, Section VII offers our conclusions.

II. Related Work

There have been many implementations of single and multidimensional FFTs on various platforms. These include software implementations on CPUs [6], GPUs [7], and supercomputers [8], and hardware implementations based on ASIC [9], [4] or FPGA [1]. Further there are studies on design automation frameworks for FFTs. These include hardware generators; kernel level [10], or system level [3], and software generators [5].

Most of these works are limited for the problem sizes where the dataset can be stored on-chip, and do not consider design challenges for the efficient use of DRAM. For example, [9], [10] explore design space tradeoffs for on-chip problem sizes. [3] indeed addresses the memory bandwidth problem but not at a level of detail that includes DRAM row-buffer effects. [2] provides a formal tensor framework for large FFTs targeting a simple machine model and [1] implements a DRAM-optimized tiled algorithm for 2D-FFT on FPGA. Also [4] proposes an implementation using 3D-stacked DRAM. However these works either target a specific platform/problem, or a generic but simplistic machine model. They do not analyze the algorithm and architecture design tradeoffs and their effects on system power/energy and performance in detail. However, achieving high-performance and energy-efficient implementations on various platforms requires careful mapping of the algorithms to the architectures by analyzing the rich design tradeoffs. In this work, we use automated techniques to explore both performance and power/energy potentials of DRAM-optimized FFT accelerators on various hardware platforms.

III. Background

Fast Fourier transform. Mathematically speaking, n point discrete Fourier transform is defined as the multiplication of n complex element input vector with $n \times n$ complex element DFT matrix to produce n complex element output vector. Computation of DFT by direct matrix-vector multiplication requires O(n^2) arithmetic operations. There exists well-known fast Fourier transform (FFT) algorithms that compute the DFT more efficiently in O($n \log n$) operations [11].

Multidimensional DFTs can also be considered as simple matrix-vector multiplications. Similar to single dimensional DFT, multi dimensional DFTs can be computed efficiently using multidimensional FFT algorithms. For example the well-known row-column algorithm for 2D-DFT can be summarized as follows [11]: Abstracting the input and output vectors as $n \times n$ arrays, firstly n point 1D-FFTs are applied to each of the n rows. Then, taking the results generated by the first stage as inputs, n point 1D-FFTs are applied to each of the n columns. The overall operation is demonstrated in Figure 1(a). Assuming a row-major data mapping to DRAM, first stage results in sequential accesses whereas second stage leads to stride n accesses which is demonstrated in Figure 1(b).

Similarly well-known 3D decomposition algorithm for 3D-DFT is computed as follows: Considering the dataset as $n \times n \times n$ cube on x-y-z space, firstly, n point 1D-FFTs are applied to each of the n^2 stripes in x direction. Then again n point 1D-FFTs are applied to each of the n^2 stripes

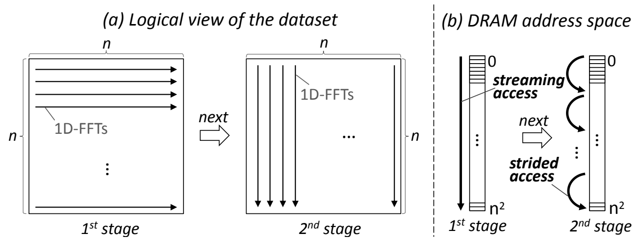


Fig. 1. Overview of row-column 2D-FFT computation.

in y direction and finally the same is done in z direction. Note that similar to the 2D-FFT, each stage takes the results generated by the previous stage as inputs. The overall operation is demonstrated in Figure 2(a). If we assume a sequential data mapping in x-y-z direction of the data cube to the DRAM, first stage corresponds to sequential accesses however second and third stages requires stride n and stride n^2 accesses respectively, which is demonstrated in Figure 2(b).

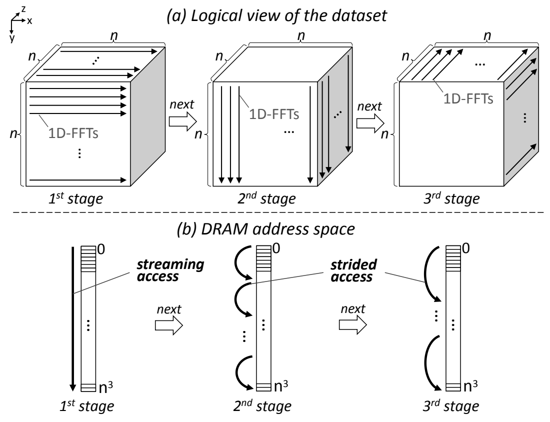


Fig. 2. Overview of 3D-decomposed 3D-FFT computation.

In the above, we have discussed decomposing large 2D and 3D-FFTs into small 1D-FFT stages whose intermediate dataset portions fit in the fast on-chip SRAM. A large 1D-FFT whose dataset do not fit in on-chip SRAM requires similarly decomposing into smaller 1D-FFT kernels and exhibits the memory access pattern behavior discussed for 2D-FFT [11]. For example a 2-stage Cooley-Tukey algorithm for n^2 point 1D-FFT can be summarized as follows: First, n-many n point 1D-FFTs are followed by a data permutation step on n^2 elements. Then, n-many n point 1D-FFTs follow a twiddle factor multiplication step. Here, the data permutation step requires stride n accesses. Hence, from memory access pattern point of view, overall operation is very similar to 2D-FFT computation (see Figure 1(a) and (b)).

DRAM operation. As shown in Figure 3, DRAMs are divided hierarchically into (from top to bottom): rank, chip,

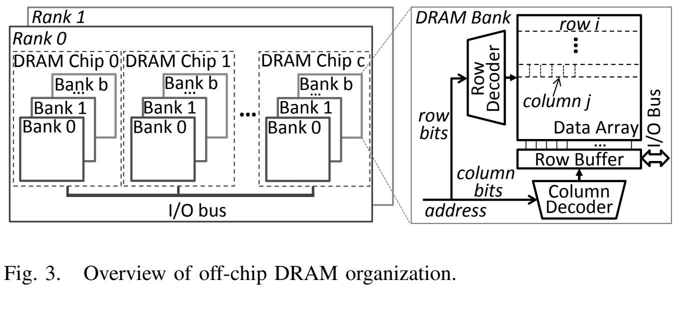


Fig. 3. Overview of off-chip DRAM organization.

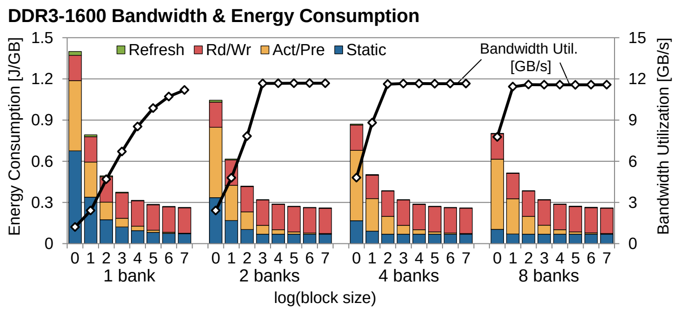


Fig. 4. Locality/parallelism vs. bandwidth, power and energy tradeoffs for DDR3-1600 DRAM (1.5V, single rank, 8 bank, x8 width) [12].

bank, row, and column. DRAM chips within a rank contributes to a portion of the DRAM word; they are accessed parallel in lock-step to form the whole word. Each bank within a DRAM chip has a *row buffer* which is a fast buffer holding the lastly accessed row in the bank. If the accessed bank and row pair are already active, i.e. the referenced row is already in the row buffer, then a *row buffer hit* occurs reducing the access latency considerably. On the other hand, when a different row in the active bank is accessed, a *row buffer miss* occurs. In this case, the DRAM array is *precharged* and the newly referenced row is *activated* in the row buffer, increasing the access latency and energy consumption. Therefore, to minimize the energy consumption and to achieve the maximum bandwidth from DRAM one must minimize the row buffer misses. In other words, one must reference all the data from each opened row before switching to another row by exploiting the spatial locality in the row buffer.

In addition to the row buffer locality (RBL), bank level parallelism (BLP) has a significant impact on the DRAM bandwidth and energy utilization. Given that different banks can operate independently, one can overlap the latencies of the row precharge and activate operations with the data transfer on different banks. BLP enables high bandwidth utilization even if the RBL is not fully utilized provided that the accesses are well distributed among banks. However, frequently precharging and activating rows in different banks increase the power and total energy consumed.

Figure 4 demonstrates the impact of RBL/BLP on DRAM bandwidth, power and energy consumption. In this experiment contiguous blocks of data are transferred from DRAM, where adjacent blocks are perfectly distributed among different banks. Therefore, the size of the data block corresponds to the number of elements referenced from an opened row. In Figure 4 we observe that the achieved bandwidth increases with RBL (i.e. size of the data blocks) and/or BLP (i.e. number of banks to which the data blocks are distributed). If the BLP is limited (e.g. accesses are concentrated on a single bank), then RBL must be maximized to reach the maximum bandwidth. On the other hand, if the blocks are well distributed among banks, the maximum bandwidth can be reached with smaller block sizes, but with the cost of additional bank precharge/activate operations. Figure 4 also shows the energy consumption in Joules/GByte which corresponds to the total energy spent in transferring unit GB of data. Both BLP and RBL decrease total static energy by transferring the same amount of data faster, yet RBL is the key to reduce the total activate/precharge energy.

From the DRAM perspective maximizing both BLP and RBL leads to the optimal use, however transferring and operating on large data blocks (to maximize RBL) can be costly from a computational perspective. As we will see, tradeoff possibilities in parallelism (BLP) and locality (RBL) is critical to achieve the best performance per power for the overall system which includes both off-chip DRAM and on-chip computation.

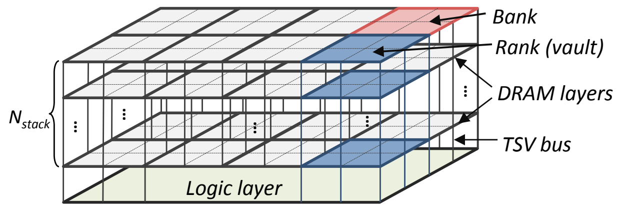


Fig. 5. Overview of the 3D-stacked DRAM organization.

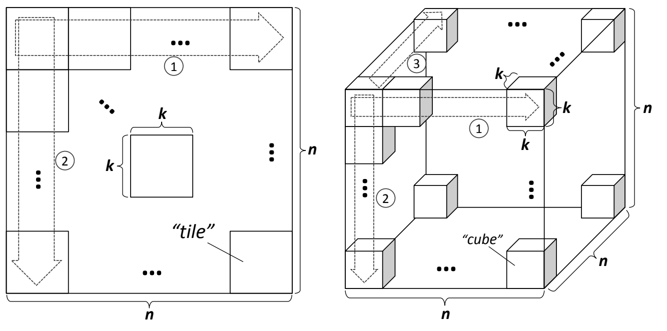
3D-stacked DRAM is an emerging technology where multiple DRAM dies and logic layer are stacked on top and connected by TSVs (through silicon via) [13], [14], [15], [4]. TSVs allow low latency and high bandwidth communication within the stack without I/O pin count concern. Fine-grain rank-level stacking, which allows individual memory banks to be stacked in 3D, enables fully utilizing the internal TSV bandwidth [16], [13]. As shown in Figure 5, fine-grain rank-level stacked 3D-DRAM consists of N_{stack} DRAM layers where each layer has N_{bank} DDR3 DRAM banks, and each bank has its own N_{TSV}-bit data TSV I/O. Vertically stacked banks share their TSV bus and form a vertical rank (or sometimes referred as vault [13]). Hence, the overall system is composed of N_{bank} ranks where every rank has its own N_{TSV}-bit TSV bus and can operate independently.

Internal operation and the structure of 3D-DRAM banks are very similar to the regular DRAMs (see Figure 3) except some of the peripheral circuitry is moved down to the logic layer which enables achieving much better timings [16]. Another interesting distinction of the 3D-stacked DRAM is that the banks within a rank can operate in pseudo-parallel, time multiplexing the shared TSV bus, contributing to the aggregate bandwidth additively by exploiting the high bandwidth TSVs [4]. As a result, row buffer misses in a bank become visible as reduced bandwidth in the corresponding rank which can only be amortized by fully utilizing opened row buffers. Hence, 3D-stacked DRAMs are more vulnerable to the strided access patterns.

IV. DRAM-Optimized FFTs

Strided accesses required by the conventional FFT algorithms lead to precisely the worst-case behavior in DRAM (Figure 1, Figure 2). One needs to change the spatial locality of the memory accesses in order to avoid inefficient strided access patterns by altering the data mapping in memory [1], [2]. We use tiled and cubic memory mapping schemes for FFTs to avoid strided accesses and transfer large contiguous chunks of data from the memory.

As shown in Figure 6, tiled and cubic data layouts are simply block data layouts where the datasets are abstracted as multidimensional data arrays and divided into smaller blocks (i.e. tiles or cubes). For example, in the tiled layout, n^2 element vectors are abstracted as a $n \times n$ element matrix which is divided into $k \times k$ element tiles (Figure 6(a)). Similarly, in the



(a) Tiled representation. (b) Cubic representation.

Fig. 6. Logical view of the dataset for tiled and cubic representation.

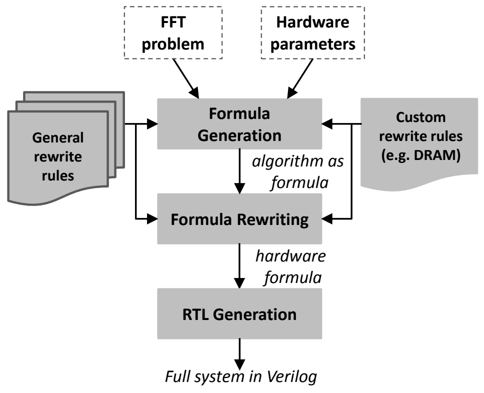


Fig. 7. Overview of the design generator tool.

cubic layout, n^3 element vectors are abstracted as a $n \times n \times n$ element data cube which is divided into $k \times k \times k$ element cubes (Figure 6(b)). The elements within blocks are mapped into consecutive locations in DRAM and blocks are distributed among different DRAM banks. When each logical block is physically mapped to a DRAM row, transferring a block from DRAM corresponds to transferring a whole row from DRAM.

Large size single and multidimensional FFTs when decomposed into smaller stages require strided memory accesses. These FFTs can be performed while avoiding strided accesses by using block data layouts. (i) Remembering that when each tile (cube) is mapped to a DRAM row, transferring tiles (cubes) in any order (i.e. ①, ② and ③ in Figure 6) corresponds to making use of the whole DRAM rows. (ii) Once tiles (cubes) are transferred to on-chip SRAMs, they can be shuffled freely since on-chip SRAMs do not incur any extra penalty depending on the access patterns. Hence, by combining the two properties (i) and (ii), one can perform the memory access pattern schemes required by multi-stage FFT algorithms efficiently. Note that these algorithms require full row or column of tiles (cubes) to be held simultaneously in the fast on-chip memory (SRAM) so that the 1D-FFTs can be done SRAM-resident.

[2] provides a formal derivation of these algorithms targeting a simplistic machine model and [1] proposes a FPGA based 2D-FFT implementation. Both of these works simply maximize locality in the DRAM via large tile sizes for high performance. Large tiles require large on-chip storage which incurs an energy overhead. As we will see later, these DRAM-optimized algorithms need to be specifically fitted to the platform parameters exploiting both locality and parallelism tradeoffs to be high-performance and energy-efficient.

V. Algorithm and Architecture Design Space

In this section we explain our design generation methodology, accelerator architecture and the key algorithm/architecture design space parameters.

A. Design Generator

We use Spiral [5] formula generation and optimization framework to automatically derive details of the DRAM-optimized FFT algorithms and generate hardware implementations. Spiral uses formal representation of FFT algorithms in tensor notation and restructures the algorithms using rewrite rules. We express the DRAM optimizations that are described in Section IV in tensor notation and integrate them into the Spiral framework as rewrite rules. Spiral formally derives the DRAM-optimized 1D, 2D and 3D-FFT algorithms, however the formal details of the algorithm generation is beyond the scope of this paper. Lastly, we build a custom backend that translates the hardware datapath formula to the full system described in Verilog.

The overall view of our compilation flow is shown in Figure 7. First, the input FFT problem is expanded as a formula tagged with hardware parameters. At this point the formula is a high-level generic representation that does not have explicit implications for the hardware. Then this formula is restructured using our custom DRAM optimization extension rules as well as selected Spiral default rules. After this step we reach a final structured formula where each formula construct is labelled with its targeted hardware module. Lastly, the custom backend generates the hardware components for the labelled formula constructs in Verilog targeting the parameterized architecture shown in Figure 8. In addition to the inferred hardware structures, the backend generates necessary wrappers, glue logic and configuration files that will interconnect all the modules and configure the full system.

B. Formula to Hardware

The architecture that we target (shown in Figure 8) is highly parameterized and scalable which can be configured for the given problem/platform parameters.

DRAM controllers. In Figure 8, without loss of generality, we provide a dual channel architecture featuring two DRAM controllers. Throughout the computation, one of the DRAM controllers is used for reading the inputs and the other one is for writing the outputs to DRAM in parallel. When a stage is completed, the two DRAM controllers switch their read/write roles for the next stage repeatedly until the computation is finished. Note that dual channel architecture is an abstraction of the actual underlying hardware, multiple DRAM channels can be bundled together or a single DRAM channel can be split into two to fit in this abstraction.

Depending on the optimizations that are used, a memory mapping scheme is represented in the generated hardware formula. The RTL generator translates this representation and configures the address mapping module in the DRAM controller (see Figure 8).

Local memories. DRAM-optimized algorithms used in this work require holding multiple tiles (cubes) on-chip at once to apply data permutation and 1D-FFT on the local

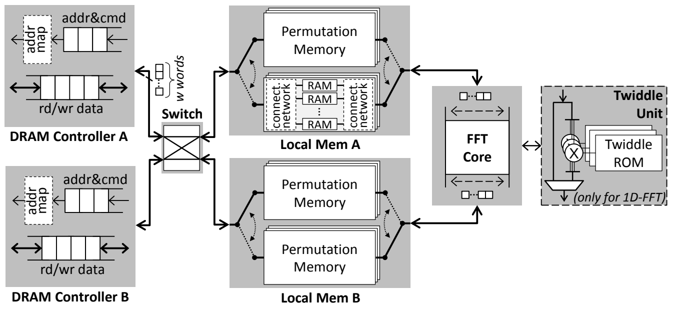


Fig. 8. Overall view of the targeted architecture.

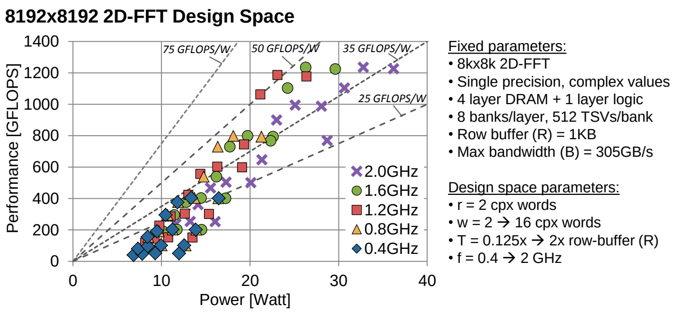


Fig. 9. Design space exploration for 2D-FFT with 3D-stacked DRAM. Isolines represent the constant power efficiency in Gflops/W (see labels).

data. Local memories are local fast buffers constructed from SRAM and connection networks serving for that requirement. Implementing the streaming data permutation operations by using minimal storage and avoiding bank conflicts is a non-trivial task. We use techniques described in [17] to generate the permutation memories automatically.

Local memories also construct the interface between the FFT core and the DRAM controller and allow the decoupling between the two. We employ double-buffering technique in the local memories. Hence, the computation and data transfer operations are overlapped to maximize the system throughput.

FFT Core. We generate the streaming 1D-FFT core automatically using [10]. Control over the FFT core parameters (radix r, streaming width w) allows us to adjust the computation throughput according to the data transfer bandwidth to create balanced designs.

1D-FFT algorithm for large problem sizes requires a separate twiddle multiplication step when decomposed as discussed in Section III. The FFT core is augmented by a separate twiddle factor multiplication unit for that case as shown in Figure 8. Note that for large problem sizes, twiddle factor ROM sizes becomes too large to fit on-chip. We use logic-in-memory units based on [18] to compress and interpolate twiddle factors for large 1D-FFTs.

C. Design Space Parameters

There are several ways of architecting a system to achieve a given goal of performance or power efficiency for a given FFT problem and hardware resources. Control over the adjustable design parameters allows walking within the space of various design possibilities. Some of the important design parameters can be categorized as follows:

- Throughput: FFT radix (r), streaming width (w), frequency (f).
- Bandwidth: Type of tiling (2D, 3D), tile size (T)
- Algorithm: Algorithmic choices provided by Spiral (A)

In addition to the adjustable design parameters, there are also given problem and platform constraints:

- Problem: FFT type (1D, 2D, 3D), size (n), precision (p)
- Platform: DRAM banks (b), row buffer size (R), max bandwidth (B), max power (P), max on-chip SRAM (S)

As we will see, exploring such a design space constructed by the given problem/platform constraints and design parameters is extremely difficult considering the relations between them and the costs associated with adjusting every parameter. An automated design generation and exploration is essential to cover this wide design space and decide the most efficient parameters.

VI. Evaluation

Experimental setup. Before going into the details of design space exploration we first explain our experimental methodology. We use a modified version of Spiral to generate designs for a given problem. Then, we synthesize the generated HDL targeting a commercial 32nm standard cell library using Synopsis Design Compiler following the standard ASIC synthesis flow. In addition to the standard ASIC synthesis flow, for non-HDL components, we use tools such as: CACTI 6.5 for on-chip RAMs and ROMs [19], McPAT for DRAM memory controllers [20], and DesignWare for single and double precision floating point units [21]. We target both off-chip DDR3-DRAM and 3D-stacked DRAM as main memory. For off-chip DRAM we use DRAMSim2 [22] and Micron Power Calculator [23] to estimate DRAM performance and power consumption. For the 3D-stacked DRAM model we use CACTI-3DD [16]. Lastly, for the overall performance estimation, we use a custom performance model backed up by cycle-accurate simulation. All of the tools are integrated resulting in an automatic push-button end-to-end design generation and exploration tool.

Exploration. First, we present an example design space for a selected 2D-FFT hardware implementation using 3D-stacked DRAM. Given problem parameters are 8192×8192 point complex single-precision ($2 \times 32 - bits$ per complex word) 2D-FFT, and the platform parameters are 4 layer, 8 banks/layer, 512 TSVs/bank, 8192 bit row buffer ($N_{\text{stack}} = 4, N_{\text{bank}} = 8, N_{\text{TSV}} = 512$, $R = $ 1KB) fine-grain rank-level 3D-stacked DRAM. Further we set $S = $ 8MB and $P = $ 40W.

Problem and platform parameters are the basic inputs to the generator. Spiral handles the algorithmic design choices (A) and prunes the algorithms which are determined to be suboptimal at the formula level. Then, for the given input configuration, it generates several hardware instances varying the design space parameters. A subset of the design space is shown in Figure 9 in terms of performance (GFLOPS) and power consumption (Watt) for various streaming width ($w = 2, 4, 8, 16$), frequency ($f = 0.4\text{GHz} \rightarrow 2\text{GHz}$), radix ($r = 2$) and tile size ($T = 0.125\times \rightarrow 2 \times$ row buffer size) parameters.

Figure 9 also provides isolines for constant power efficiency, in other words achieved performance per consumed power (GFLOPS/Watt). We observe that there are multiple design points on the constant power efficiency lines, which suggests that the same power efficiency can be achieved

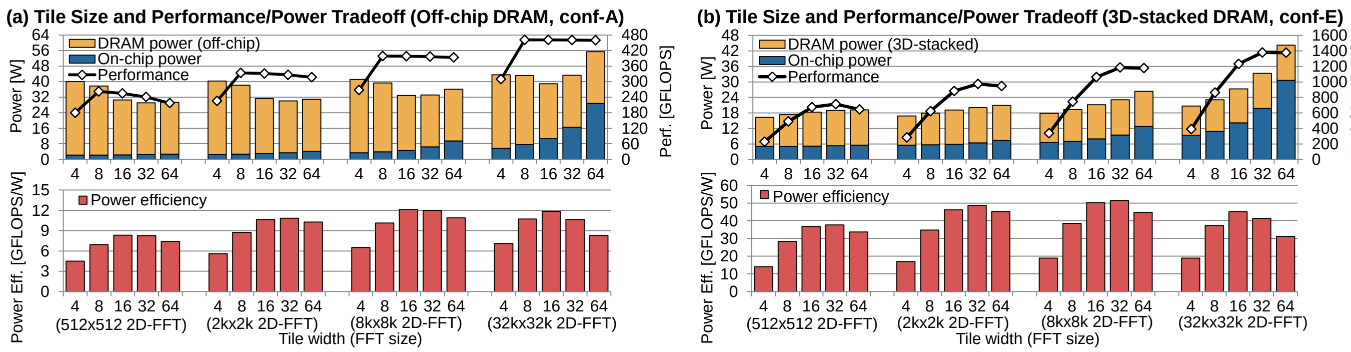


Fig. 10. Effect of tile size on performance and power efficiency for off-chip and 3D-stacked DRAM systems. (Rest of the parameters are fixed.)

with different parameter combinations. There are also several suboptimal designs that are behind the pareto frontier. Hence, it is not obvious which parameter combinations will yield the most efficient system at the design time. This highlights the complexity of the design space.

To further elaborate on the design space tradeoffs, we first illustrate the effects of the tile size in tiled FFT algorithms (see Section IV) on the performance and power efficiency for different system configurations in Figure 10 (see Table I for memory configurations). (i) DRAM and (ii) on-chip resources are two main components of the overall system: (i) Increasing the tile size improves the spatial locality in DRAM accesses through efficient use of the row buffer. Efficient use of the row buffer leads to minimal activate/precharge operations which improve the bandwidth utilization and energy efficiency in DRAM, and consequently improve overall system performance and energy efficiency. (ii) On the other hand, from the on-chip resources viewpoint, local memory needs to be large enough to hold larger tiles which increases power consumption. Additionally, larger local memory needs to be filled and emptied in the beginning and at the end of the overall computation pipeline which decrease the performance. Conflicting tradeoffs in determining the tile size construct an optimization problem.

Moreover, different platform and problem configurations have different optimization curves. For example, power consumption of smaller problem size configurations are heavily dominated by the DRAM power consumption hence it is more desirable to improve the DRAM power consumption with larger tiles. Whereas larger problem size configurations prefer smaller tile sizes to save the on-chip power consumption (see Figure 10). Different platform configurations can also have different tradeoff relations. For example, BLP allows overlapping row buffer miss delays with data transfer on different banks. Therefore one can maximize the performance even if the RBL is not fully utilized, with the extra energy cost of activate/precharge operations in DRAM (see Figure 10(a)). However, in 3D-stacked DRAM, banks within a rank contributes additively to the aggregate bandwidth exploiting the high bandwidth TSV bus as discussed in Section III. Therefore, unlike off-chip DRAMs, low RBL utilization (i.e. small tiles) reduces the performance significantly as shown in Figure 10(b).

Determining the memory subsystem configuration, particularly tile size in our case, is a key component in achieving high performance and energy efficiency, but computation configuration poses a great importance as well to achieve a balanced design. In a balanced design, computation throughput needs to match the estimated bandwidth utilization. One can pick different parameter combinations that will achieve the same throughput matching the given DRAM bandwidth. In Figure 11, given a fixed fixed tile size and platform/problem parameters (i.e. fixed DRAM bandwidth), we demonstrate design instances with various frequency and streaming width parameters. As highlighted in Figure 11 the most efficient parameter combinations show variations for different problem/platform configurations.

TABLE I. MAIN MEMORY CONFIGURATIONS.

Name	Configuration (off-chip DRAM) Chan/Bank/R(Kb)/width/Type	tCL-tRCD-tRP (ns)	Max BW (GB/s)
conf-A	8 / 8 / 64 / x8 / DDR3-1600	13.8-13.8-13.8	102.4
conf-B	8 / 8 / 64 / x8 / DDR3-1333	15.0-15.0-15.0	85.36
conf-C	8 / 8 / 64 / x8 / DDR3-800	12.5-12.5-12.5	51.2
Name	Configuration (3D-stacked DRAM) $N_{\text{stack}}/N_{\text{bank}}/N_{\text{TSV}}/R$(Kb)/Tech(nm)	tCL-tRCD-tRP-tTSV (ns)	Max BW (GB/s)
conf-D	4 / 8 / 256 / 8 / 32	12.2-7.89-16.8-0.68	178.2
conf-E	4 / 8 / 512 / 8 / 32	12.2-7.89-16.8-0.68	305.3
conf-F	4 / 8 / 512 / 8 / 45	14.2-9.23-19.1-0.92	246.7
conf-G	4 / 8 / 256 / 32 / 32	11.8-21.9-16.4-0.68	229.5

Finding the best system configuration given the problem/platform constraints establishes an optimization problem. Simply chasing the highest performance or lowest power consumption is not sufficient to get the most efficient system. As it is shown in Figure 10 and Figure 11, careful study of the design space is necessary to understand the tradeoffs. It is difficult to determine the crossover points where one of the dependent parameters becomes more favorable to the others and there is no structured way of finding the best parameter combinations. This highlights the importance of an automatic design generation and exploration system–it would be extremely difficult to complete such design exploration by hand.

Pareto-optimal designs. We automatically generate pareto-optimal DRAM-optimized implementations and evaluate their performance and energy/power efficiency for the main memory as well as for the overall system. Our experiments include various main memory configurations which are shown in Table I. Firstly, the overall system performance and power efficiency comparison of the naive baseline and the DRAM optimized implementations for 1D, 2D and 3D-FFTs of various sizes with regular off-chip DDR3 DRAMs and 3D-stacked DRAMs are demonstrated in Figure 12. DRAM-optimized implementations are generated by Spiral for the given problem and platform parameters. The same on-chip hardware resources

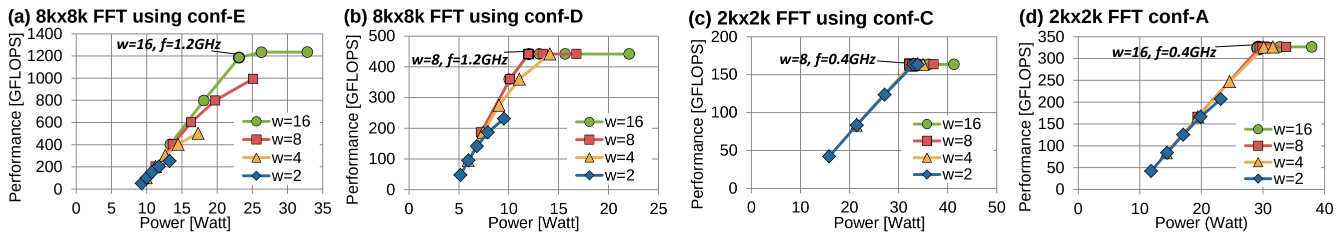


Fig. 11. Frequency (f) and streaming width (w) effects on power and performance for various problem/platform configurations (fixed tile size). Parameter combinations for the best design (GFLOPS/W) are labelled.

and the same memory configurations are provided both for the baseline and the DRAM-optimized implementations. However, baseline implementations have naive unoptimized DRAM access patterns in contrast to DRAM-friendly access patterns of the optimized implementations. Further, DRAM-optimized implementations are "pareto-optimal" such that they are specifically fitted to the target platform for the best performance per power (i.e. GFLOPS/Watt) by the design generator tool. Although the best designs in terms of power efficiency vary depending on the problem and platform configurations as shown in Figure 12, generated DRAM-optimized designs can achieve up to 6.5x and 6x improvements in overall system performance and power efficiency respectively over naive baseline implementations. Also, to provide a point of reference, modern GPUs and CPUs can achieve only a few GFLOPS/W for the problem sizes that we are concerned with [1], [24]. For example, cuFFT 5.0 on the recent Nvidia K20X reaches approximately 2.2 GFLOPS/W where machine peak is 16.8 GFLOPS/W [25]. On the other hand, our DRAM-optimized hardware accelerators achieve up to nearly 50 GFLOPS/W (see Figure 12).

Improvements particularly in the main memory is the core of our framework. In Figure 13, pairs of bars compare DRAM-optimized (opt) and naive baseline (nai) implementations for various memory configurations (see Table I). The results show that the DRAM-optimized accelerators for 1D, 2D and 3D FFTs can achieve up to 7.9x higher bandwidth utilization and 5.5x lower energy consumption in main memory for off-chip DRAM, and up to 40x higher bandwidth utilization and 4.9x lower energy consumption in main memory for 3D-stacked DRAM. As discussed in Section III, due to their vulnerability to the strided access patterns, 3D-stacked DRAMs achieve better improvement through optimized access patterns. Another interesting point is that the generated 1D, 2D and 3D FFT designs have very similar efficient tiled access patterns which allow them to achieve a bandwidth and energy efficiency near theoretical maximum. Consequently, in Figure 13 we observe that optimized implementation for different FFTs with the same memory configuration reach almost the same DRAM bandwidth and energy consumption.

In Figure 13, the bars representing DRAM energy consumption are broken into segments as refresh, read/write, activate/precharge and static energy consumption. An important observation is that the activate/precharge energy is significantly reduced, almost eliminated, compared to the baseline. Effective usage of DRAM row buffer in our DRAM-optimized implementation leads to very low row buffer miss rate which significantly reduces the total activate/precharge energy consumption. As expected, read/write energy stays the same since the same amount of data is read/written in both naive and

TABLE II. PERFORMANCE RESULTS FROM ALTERA DE4.

FFT	Prec. (bits)	TP (GFLOPS)	Perf (% of TP) (GFLOPS)	Model Est. (Error) (GFLOPS)
256×256	32	32	23.25 (72.6%)	23.15 (-0.41%)
512×512	32	36	29.23 (81.2%)	29.31 (+0.27%)
$1k \times 1k$	32	40	34.42 (86.0%)	34.74 (+0.94%)
$2k \times 2k$	32	44	38.35 (87.2%)	39.54 (+3.10%)
$4k \times 4k$	32	48	42.10 (87.7%)	43.89 (+4.25%)
256×256	64	16	12.47 (77.9%)	12.53 (+0.48%)
512×512	64	18	14.97 (83.2%)	15.10 (+0.85%)
$1k \times 1k$	64	20	17.19 (86.0%)	17.40 (+1.19%)
$2k \times 2k$	64	22	19.23 (87.4%)	19.50 (+1.40%)
$128 \times 128 \times 128$	32	28	23.40 (83.6%)	23.69 (+1.22%)
$256 \times 256 \times 256$	32	32	26.89 (84.0%)	27.24 (+1.30%)
$512 \times 512 \times 512$	32	36	30.30 (84.2%)	30.70 (+1.32%)

optimized implementations. DRAM-optimized systems have higher bandwidth utilization which allows them to finish the data transfer quicker, saving total refresh and static energy consumption.

Complete design flow. We demonstrate our complete design flow by starting with high level problem and hardware platform specifications, then generating designs automatically and finally implementing and running them on an actual hardware. We target the Altera DE4 FPGA platform featuring a Stratix IV FPGA (4SGX530) interfaced to 2 channels of DDR2-800 DRAMs. Table II demonstrates the results from example implementations including single and double precision various size 2D and 3D FFTs. Under the column TP (theoretical peak), for each FFT problem, we report the performance numbers for an idealized platform with perfect main memory that has the bounded bandwidth (same as DE4) but no row buffer miss penalty or refresh penalty, and infinitely fast on-chip processing. We also provide the achieved performance in percentage of TP performance to give an absolute sense of quality.

Performance model verification. We now verify the performance model used in the design space exploration against the results from the actual hardware implementations. For performance estimations, DE4 platform parameters and FFT problem parameters are input to performance model. Performance estimation results from the model are given in Table II under column "model est.". When the performance estimations and the actual hardware results are compared, the error is found to be less than %4.25 for the designs that are generated and implemented on DE4.

VII. CONCLUSION

Main memory bandwidth is a common performance bottleneck for single and multi dimensional FFT hardware accelerators. To address that problem, prior work propose DRAM-optimized FFTs that reshapes DRAM-unfriendly access pat-

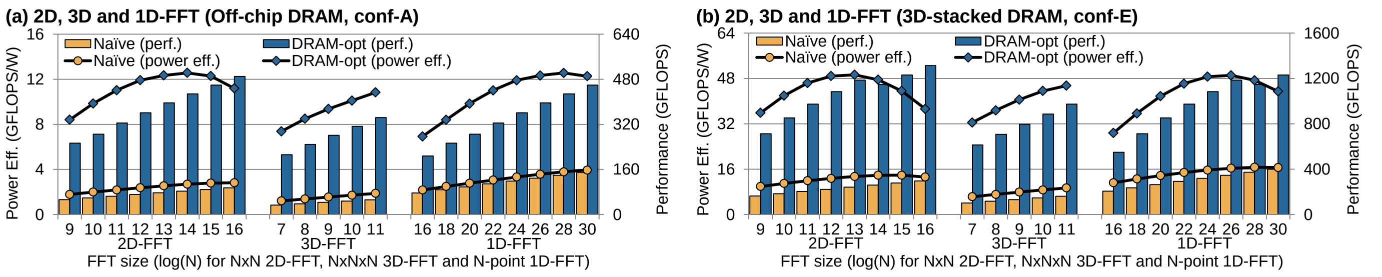


Fig. 12. Overall system performance and power efficiency comparison between naive and DRAM-optimized implementations for 1D, 2D and 3D FFTs using memory configurations conf-A and conf-D respectively.

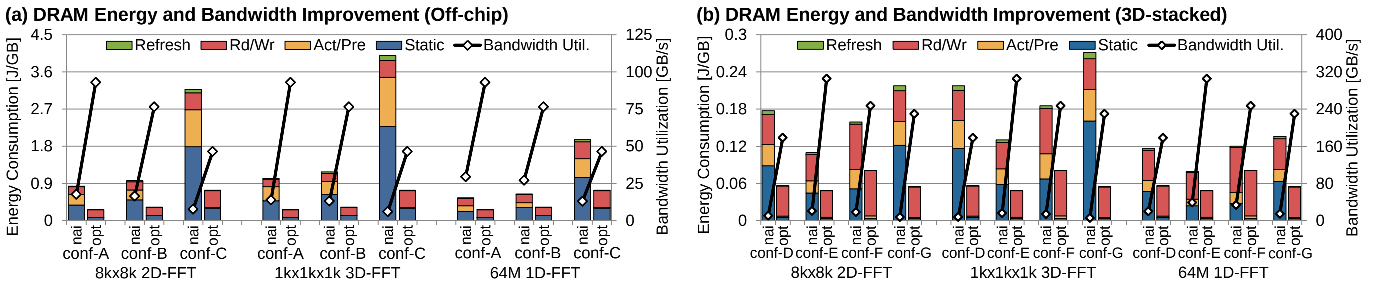


Fig. 13. DRAM energy and bandwidth utilization for naive (nai) and DRAM-optimized (opt) implementations of selected FFTs and memory configurations.

terns to eliminate row buffer misses. However, these solutions either tuned for a specific problem and platform, or target a simplistic machine model. Achieving high-performance and energy-efficient implementations on different platforms, on the other hand, requires careful fitting of these DRAM-optimized FFT algorithms to the targeted architectures. Given the rich implementation possibilities and interdependent design parameters, there is no structured way of determining the best configuration. This paper evaluates the design space tradeoffs and energy/performance potentials of the DRAM-optimized FFT accelerators using automated techniques for 1D, 2D and 3D FFTs on different platforms (off-chip DRAM, 3D-stacked DRAM, ASIC, FPGA). Our experimental results show that generated pareto-optimal designs offer significant improvements in performance and power efficiency.

References

[1] B. Akin *et al.*, "Memory bandwidth efficient two-dimensional fast Fourier transform algorithm and implementation for large problem sizes," in *Proc. of the 20th IEEE Symp. on Field-Programmable Custom Computing Machines (FCCM)*, 2012, pp. 188–191.

[2] B. Akin *et al.*, "FFTs with near-optimal memory access through block data layouts," in *Proc. IEEE Intl. Conf. Acoustics Speech and Signal Processing (ICASSP)*, 2014.

[3] C.-L. Yu *et al.*, "Multidimensional DFT IP generator for FPGA platforms," *IEEE Transactions on Circuits and Systems*, vol. 58, no. 4, pp. 755–764, 2010.

[4] Q. Zhu *et al.*, "A 3d-stacked logic-in-memory accelerator for application-specific data intensive computing," in *3D Systems Integration Conference (3DIC), 2013 IEEE International*, Oct 2013, pp. 1–7.

[5] M. Püschel *et al.*, "SPIRAL: Code generation for DSP transforms," *Proc. of IEEE, special issue on "Program Generation, Optimization, and Adaptation"*, vol. 93, no. 2, pp. 232–275, 2005.

[6] M. Frigo *et al.*, "The design and implementation of FFTW3," *Proceedings of the IEEE, Special issue on "Program Generation, Optimization, and Platform Adaptation"*, vol. 93, no. 2, pp. 216–231, 2005.

[7] N. K. Govindaraju *et al.*, "High performance discrete Fourier transforms on graphics processors," in *Proc. of the ACM/IEEE Conference on Supercomputing (SC)*, 2008, pp. 2:1–2:12.

[8] M. Eleftheriou *et al.*, "Scalable framework for 3D FFTs on the Blue Gene/L supercomputer: Implementation and early performance measurements," *IBM Journal of Research and Development*, vol. 49, no. 2.3, pp. 457–464, 2005.

[9] A. Pedram *et al.*, "Transforming a linear algebra core to an FFT accelerator," in *Proc. of IEEE Int. Conference on Application-Specific Systems, Architectures and Processors (ASAP)*, 2013, pp. 175–184.

[10] P. A. Milder *et al.*, "Computer generation of hardware for linear digital signal processing transforms," *ACM Transactions on Design Automation of Electronic Systems*, vol. 17, no. 2, 2012.

[11] C. Van Loan, *Computational frameworks for the fast Fourier transform*. SIAM, 1992.

[12] "DDR3-1600 dram datasheet, MT41J256M4, Micron," http://www.micron.com/parts/dram/ddr3-sdram.

[13] J. T. Pawlowski, "Hybrid memory cube (HMC)," in *Hotchips*, 2011.

[14] G. H. Loh, "3d-stacked memory architectures for multi-core processors," in *Proc. of the 35th Annual International Symposium on Computer Architecture, (ISCA)*, 2008, pp. 453–464.

[15] C. Weis *et al.*, "Exploration and optimization of 3-d integrated dram subsystems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 4, pp. 597–610, April 2013.

[16] K. Chen *et al.*, "CACTI-3DD: Architecture-level modeling for 3D die-stacked DRAM main memory," in *Design, Automation Test in Europe (DATE)*, 2012, pp. 33–38.

[17] P. A. Milder *et al.*, "Automatic generation of streaming datapaths for arbitrary fixed permutations," in *Design, Automation and Test in Europe (DATE)*, 2009, pp. 1118–1123.

[18] Q. Zhu *et al.*, "Design automation framework for application-specific logic-in-memory blocks," in *Proc. of IEEE Int. Conference on Application-Specific Systems, Architectures and Processors (ASAP)*, 2012, pp. 125–132.

[19] "CACTI 6.5, HP labs," http://www.hpl.hp.com/research/cacti/.

[20] "McPAT 1.0, HP labs," http://www.hpl.hp.com/research/mcpat/.

[21] "DesignWare library, Synopsys," http://www.synopsys.com/dw.

[22] P. Rosenfeld *et al.*, "Dramsim2: A cycle accurate memory system simulator," *IEEE Comp. Arch. Letters*, vol. 10, no. 1, pp. 16–19, 2011.

[23] "DDR3 sdram system-power calculator, Micron," http://www.micron.com/products/support/power-calc.

[24] E. S. Chung *et al.*, "Single-chip heterogeneous computing: Does the future include custom logic, FPGAs, and GPGPUs?" in *Proc. of the 43th IEEE/ACM Intl. Symp. on Microarchitecture (MICRO)*, 2010.

[25] "CUDA toolkit 5.0 performance report, Nvidia," https://developer.nvidia.com/cuda-math-library, Jan 2013.