Power-efficient Re-gridding Architecture for Accelerating Non-Uniform Fast Fourier Transform

Umer I. Cheema, Gregory Nash, Rashid Ansari
Department of Electrical and Computer Engineering
University of Illinois at Chicago
Chicago, USA
Email: {ucheem2, gnash2, ransari}@uic.edu

Ashfaq A. Khokhar
Department of Electrical and Computer Engineering
Illinois Institute of Technology
Chicago, USA
Email: ashfaq@iit.edu

Abstract—This paper proposes a novel FPGA-based accelerator for the memory and compute-intense re-gridding process used in computation of Non-uniform Fast Fourier Transform (NuFFT). The re-gridding process interpolates arbitrary sampled data onto a uniform grid using an interpolation kernel function. This re-gridding step is considered one of the most time consuming step in entire NuFFT computation. We propose a memory-efficient technique based on the novel use of customizable hardware components such as FPGA block memory in First-In-First-Out (FIFO) configuration, fill-rate based arbiter, distributed RAM and an array of pipelined single precision floating point multipliers and adders. The proposed architecture exhibits high performance over a wide range of configurations and data-sizes. A speed-up of over 9.6 was achieved when compared with existing FPGA-based technique at a 7 times higher MFLOPS per watt. Compared to GPU based technique, over 6 times higher MFLOPS per watts were achieved.

I. INTRODUCTION

The Discrete Fourier Transform (DFT) can be viewed as the Fourier Transform of a periodic and regularly sampled signal. Non-Uniform Discrete Fourier Transform (NuDFT) is a generalization of the DFT for data that may not be regularly sampled in spatial or temporal domain. This flexibility allows for benefits in situations where sensor placement cannot be guaranteed to be regular or irregular sampling patterns allow more efficient information analysis. Some of the salient applications of NuFFT include Synthetic Aperture Radar (SAR) [1], Computed Tomography (CT) [2] and Magnetic Resonance Imaging (MRI) [3].

Consider a set $S$ of size $M$ consisting of non-equispaced source samples $x$. Let $N$ be the number of equispaced Cartesian grid cells of dimension $d$ belonging to set $I_N$ and let $f_j$ be the complex Fourier co-efficient corresponding to the grid cell $j$. NuDFT can be formally defined as in equation 1 [4].

$$f_k = \sum_{j \in I_N} f_j e^{-2\pi j x_k} \quad (1)$$

Direct computation of NDFT requires $O(MN)$ arithmetic operations. Non-Uniform Fast Fourier Transform (NuFFT) helps computing NuDFT with some approximation in $O(M + N \log N)$ complexity [5]. It uses the Fast Fourier Transform (FFT) in combination with an approximation scheme.

The main steps of the NuFFT algorithm are shown in Fig. 1. The first step is density compensation of the non-uniform samples in the source domain. Density compensation is followed by translating non-uniformly distributed data to a uniform grid using interpolation kernel functions. This re-gridding is performed in the source domain and is also referred to as data-translation, gridding or re-sampling. Re-gridding is followed by domain transformation using FFT (or IFFT). Finally, the transformed data is scaled in the target domain. Depending on application, source samples may be available in arbitrary sampling order. In some cases, the coordinates of source samples can be generated using formulas but it may be the case that source samples are available in random order within a data-file with coordinates specified as a sequence of coordinate tuples [6]. In order to provide a generic solution, we do not assume any particular ordering of source samples in this work. In other words, we assume random sampling in the source domain and hence our approach can be used for any application regardless of sampling trajectory.

Fig. 2 shows the graphical representation of the re-gridding process. Several re-gridding methods have been proposed in the literature that vary in terms of accuracy and associated complexity [7], [8]. In this paper we focus on methods that are based on convolving the non-uniformly spaced source points $S$ with a convolution kernel function $\Phi$. The output of the
Updating the values for all the points bundled together in a subset of $T$ locality by bundling together the points that update the same location in memory accesses. The main idea is to achieve high data-locality and hence reduce the number of memory accesses. The focus in this paper is to accelerate re-gridding process by improving the data-locality and hence reducing the number of memory accesses.

Other works like [6], [10], [5] also use the same gridding algorithm for re-gridding and employ various techniques to improve the efficiency of computation.

As mentioned earlier, the main challenge in accelerating the re-gridding process for an arbitrarily sampled data is to deal with the lack of data locality in memory accesses. Our approach is to accelerate re-gridding process by improvements in terms of an oversampling factor $\alpha$ [9].

The relationship between the number of source $|S|$ and target points $|T|$ can also be expressed in terms of $\alpha$: $|T| = \alpha |S|$

The basic procedure for computing the target point array is shown as pseudo-code in Fig. 3. It is based on updating all the target points within a specified distance of the source point using an interpolation kernel function $\Phi$. In this paper, we refer to this specified distance as interpolation threshold $\sigma$. The process of updating the target points is repeated for all the source points in $S$. The final target array $T$ is the translated version of $S$. Usually, $S$ and $T$ are assumed to be available in external memory as they cannot fit in on-chip memory for large problem sizes. Since the source points are read in random order with respect to coordinates, accesses to $T$ for updating corresponding target points lack data-locality. This incurs significant delays due to repeated memory accesses and hence re-gridding is known to be the most time consuming step (over 90%) of the whole NuFFT computation [5].

Other works like [6], [10], [5] also use the same gridding algorithm for re-gridding and employ various techniques to improve the efficiency of computation.

As mentioned earlier, the main challenge in accelerating the re-gridding process for an arbitrarily sampled data is to deal with the lack of data locality in memory accesses. Our focus in this paper is to accelerate re-gridding process by improving the data-locality and hence reducing the number of memory accesses. The main idea is to achieve high data-locality by bundling together the points that update the same subset of $T$ into a block memory based FIFO, accessing the external memory to read the corresponding subset of $T$ and updating the values for all the points bundled together in the FIFO. The novelty of the proposed architecture is in the efficient use of FPGA on-device block memory for storing temporarily the subsets of $S$ and $T$, and the efficient usage of other customizable hardware components such as fill-rate based arbiter, decentralized memory control logic and an array of pipelined single precision floating point multipliers and adders.

High performance implementations for computing NuFFT have been pursued on various platforms including multi-core CPU’s, [3], [1], [2], General Purpose Graphics Processing Units (GPGPU’s) [3] and FPGAs [5]. Since the lack of locality in memory accesses is the major bottleneck in the re-gridding process, most of the techniques are based on efficient utilization of memory bandwidth. Kestur et al. [5] also proposed an FPGA based implementation employing a linked-list based approach to improve the locality of memory accesses and hence the memory bandwidth efficiency.

II. PROPOSED RE-GRIDDING ARCHITECTURE

The proposed re-gridding architecture, called Data-Translator, is based on mapping uniform grid of target points to a number of on-device, block memory based FIFOs. The 2-Dimensional (2D) array $T$ is subdivided into smaller 2D sub-arrays, referred to as tiles in this paper. The choice for the size of tile is made based on the resources available in the device and the maximum burst size allowed by the system bus architecture. Each tile has a corresponding FIFO that is used to group the source points that affect any target point within the tile. The FIFO-to-tile mapping is illustrated in Fig. 4. The source points are read from the external memory and pushed into the FIFO(s) that correspond to the tile(s) they affect. The idea is to group all the source points that affect the same tile into a single FIFO. One FIFO is read at a time and the corresponding tile is loaded into the device from the memory. Once all the source points in the FIFO update the corresponding target points, the tile is written back and another FIFO is selected to be read. The process is repeated till the target points corresponding to each source point in $S$ have been updated.

As shown in Fig. 7, the proposed Re-gridding architecture consists of memory interfaces, multiple block memory based FIFOs, decentralized control logic, read arbiter, address generator and an array of floating-point multipliers/adders (FpMAs). Rest of the section explains the architectural details of individual components and data-flow through the system.
Fig. 4. Mapping between Tiles and point FIFO. The boundaries of the tiles are shown by dotted blue line and tiles are numbered from 1 - 12. Grey dots indicate the target points, crosses indicates the source points and dotted red box indicates the convolution widow for each source point. The FIFOs at the bottom are shown filled with the source points and have the same index as the tile they correspond to.

A. Points Representation

1) Source Points: Each source point $s_i$ is stored as a string of 128-bits. $s_{ix}$ and $s_{iy}$ are the $x$ and $y$ co-ordinate in single precision floating point format as defined in [11]. The co-ordinates may have any value within the specified range. Both real and imaginary part of $s_i$ are also stored in single precision floating point format. The set of source points $S$ is available in external memory with an arbitrary (random) order of source points.

2) Target Points: Set of target points $T$ is also stored in external memory. Since target points are on uniform grid, by storing them in order, the coordinates are no longer required to be stored as part of the string. Target point is a string of 64-bits consisting of real and imaginary part expressed as single precision floating point values.

B. Data-translator Components

1) Memory Interface: The proposed architecture has two memory interfaces. The first interface called Source Interface (SI) reads the source points from the external memory. The second interface, called Target Tile Interface (TTI) is used to read and write target tiles from the external memory.

2) Point-FIFOs and Overflow-FIFO: There is a FIFO $f_m$ corresponding to each tile $e_m$ in the target set $T$ as depicted in Fig. 4. The set of FIFOs corresponding to tiles are referred as point-FIFOs. The decision to push an incoming source point $s_i$ into $f_m$ is made using a decentralized control logic. $s_i$ may be pushed into multiple FIFOs and if any FIFO corresponding to $s_i$ is full, $s_i$ is pushed into an extra FIFO called the overflow FIFO. If the number of empty slots in the overflow FIFO become less than the read burst size used by SI, read operation of source points from the external memory is stalled. Overflow FIFO is emptied by repeatedly trying to write the source points contained into the point-FIFOs. If any FIFO corresponding to the source point is still full, the point is written back to the overflow-FIFO.

The total number of point-FIFOs depend on the size of the target frame $T$ and the tile size. If the target set has $|T|$ points and a tile of size $z \times z$ is used, the total number of point-FIFOs will be $|T|/z^2$.

3) Decentralized control logic for the point FIFOs: The source points read from the memory through SI are pushed into the point-FIFOs depending on their co-ordinates and interpolation threshold $\sigma$. Consider the $m$th tile $e_m$ spanning the cartesian region $(x, y)$ where $X_{m_l} < x <= X_{m_h}$ and $Y_{m_l} < y <= Y_{m_h}$. $m$ is called the tile index of tile $e_m$. A source point $s_i$ is pushed into $f_m$, if the following conditions are satisfied:

$$X_{m_l} - \sigma < s_{ix} \leq X_{m_h} + \sigma$$  
$$Y_{m_l} - \sigma < s_{iy} \leq Y_{m_h} + \sigma$$  

The control logic corresponding to each FIFO also keeps track of the Rate-of-Fill (ROF) of the FIFO. The ROF is calculated over length of time. The length of this history is an important design parameter that can be changed. ROF along with Fill-Status (FS) of the FIFO is fed to a centralized arbiter.

4) Read Arbiter: A single point-FIFO is read at a time. The selection of a particular point-FIFO to be read is made by the read arbiter based on FS and ROF of all FIFOs. FIFO that is full is given the highest priority. In case multiple FIFOs are full, the FIFO corresponding to greater tile index is given priority. When none of the FIFO is full, the decision is based on ROF. The FIFO with highest ROF is read first. A register keeps track of the highest ROF amongst the FIFOs. Only the FIFOs having the highest ROF request for read. When multiple
FIFOs have the highest ROF, the tile with the greatest tile index is given priority. The arbiter also controls the select line for the multiplexer at the output of the point FIFOs. Since the arbiter is a centralized control, the hardware complexity directly depends on the number of tiles.

5) **Tile Address Generator:** If the read arbiter decides to read data from a FIFO $f_m$, address generator generates the address of the corresponding tile $e_m$ in the memory and requests the TTI to read $e_m$ from the memory.

6) **Floating point Multipliers and Adders (FpMAs) with associated Random Access Memory (RAM):** Since for a single source point, an array of $(2\sigma) \times (2\sigma)$ target points need to be updated, we use an array of $(2\sigma) \times (2\sigma)$ FpMA units to compute all these values in parallel as shown in Fig. 6. An array of pre-computed values of interpolation kernel function is stored locally in each FpMA unit in the form of a Look-Up-Table (LUT). The size of LUT depends on $\sigma$ and the desired precision. The LUT is addressed using the distance between the target point that is being updated in FpMA and the source point.

a) **Mapping of Target tile points to FpMA units:** Storing the contents of tile in a single RAM on the device restricts the number of simultaneous accesses to the target points. This severely hinders the extraction of parallelism in the computation of target point values. Another way to store the tile on the device is to use registers. This solves the problem of parallel access by using numerous multiplexers but the approach is not efficient with respect to interconnect resources required for large tile sizes.

We propose a distributed approach to store the values of fetched target tile on the device. Each target point is mapped to a single FpMA unit and all target points mapped to the same FpMA unit are stored in a single RAM. The mapping of an $8 \times 8$ tile onto an array of $4 \times 4$ FpMAs is shown in Fig. 7. FpMA units are also indexed as a two dimensional array to make mapping easier and more intuitive. If a target point $t_k$ is at index $(x_{tk}, y_{tk})$ in the tile, the index $(q_x, q_y)$ of the FpMA associated with it is given by the following expressions:

$$ q_x = x_{tk} \mod (2 \times \sigma), \quad q_y = y_{tk} \mod (2 \times \sigma) $$

b) **Architecture of FpMA:** Each FpMA unit has a pipelined floating point multiplier along with a floating point adder to perform the multiply and accumulate operation. The values of interpolation kernel functions are addressed from the LUT based on the distance between the target point being updated and the source point. The proposed architecture is based on pipelined floating point multiplier by Altera [12]. The depth of pipeline for floating-point multiplier is 5 whereas the floating point adder is single stage. A register pipeline is used to synchronize the timing of source co-ordinates with multiplier output. A new source point is available every clock cycle and the whole process of multiplication and accumulation is pipelined. An interface with TTI is provided to read a new tile from memory and write back the fetched tile. The architecture of FpMA is shown in Fig. 8. For simplicity, Fig. 8 does not show separate multipliers and adders for the real and complex part of the values.

The hardware complexity of an FpMA cell depends on the size of the tile and $\sigma$. For a tile of size $z \times z$, the number...
of memory locations in the RAM of each FpMA would be \((\sigma^2)/(4 \times \sigma^2)\).

C. Data-flow through the System

The source points are written to external memory by an initializing bus master in the format described earlier. All the target points are initialized to zero. SI reads source points from external memory in a burst memory access mode. During each cycle of burst, a new source point \(s_i\) is read and pushed into appropriate FIFO(s) by the decentralized control logic. If any point-FIFO corresponding to \(s_i\) is full, \(s_i\) is pushed into the Overflow-FIFO. Based on FS and ROF of FIFOs, read arbiter selects the FIFO to be read. Address generator generates the address of the selected tile. TTI requests the bus to read the specified tile from the memory. The contents of the fetched tiles are distributed over the RAMs associated with FpMA units. Each target point has a single FpMA unit associated with it and all the target points associated with an FpMA unit are stored in the local RAM. A source point is read from the FIFO at each clock cycle and all the target points within the fetched tile that are \(\sigma\) distance from the source point are updated in parallel. SI and TTI access the memory through a shared bus with the bursts from both interfaces interleaved. This implies that the source points are read out from the FIFOs in between the FIFO-fill operation. This minimizes the chance of any FIFO getting full. The arbitration share for each memory interface on the system bus can be modified to give preference to a particular one. If SI is given a higher preference, the overall FIFO fill-rate is greater then the rate of popping points out of FIFOs. This means grouping more points together at the expense of higher probability of any FIFO getting filled.

III. Experimental Setup

1) Data Translator: Data-translator is described using Verilog Hardware Description Language (HDL). Synthesis and fitting is done using Altera Quartus II 13.1 for Stratix IV EP4SGX230KF40C2 device. Simulations are performed on Modelsim Altera 10.0d. Data translator is integrated to a DDR2 memory controller using two standard Avalon Bus Master Interfaces [13]: SI and TTI. SI is 128-bit wide and reads source points from the memory in the form of bursts. The upper limit on burst size is imposed by Avalon Specifications [13]. A maximum of 1024 points are read as a single burst and for larger number of points, multiple burst transactions are done. The Target interface is 2048-bit wide and is used to read/write the target points from/to the DDR2 memory in the form of bursts. Maximum allowed burst size is one of the factors in deciding the size of the tile. For a 2048-bit wide bus, the maximum tile size that can fit in a single burst is \(64 \times 64\). Considering the delays associated with bus-competition, the benefits associated with bigger tiles may not scale if multiple bursts are needed per tile. Nevertheless, having tile size greater than burst size would be an interesting study as part of future work. The array of floating point multiplier and adder of the FpMA units are generated using Altera Mega-wizard [12].

The initializing bus master, SI and TTI communicate with memory through the Avalon Bus Interconnect that has built-in bus arbitration support. The arbitration shares amongst the masters are programmable. For our experiments, equal arbitration share is given to the three bus masters in the system. The source points are generated using a MATLAB script. The co-ordinates for the source points are generated randomly within the specified range. This mimics the arbitrary sampling order. The real and imaginary parts of source point values are also generated randomly. Each co-ordinate as well as real and imaginary parts is represented in single precision floating point representation. The power analysis of the proposed architecture is done using Altera’s Powerplay power analysis tool [14].

IV. Results and Discussions

The system was simulated using Modelsim Altera 10.0d. The simulation models for the Avalon bus, memory controller and DDR2 memory were generated using Altera Qsys software. The simulation gives highly precise results since the latency in the simulation models may differ at most by two clock cycles compared to the design on board [15]. Performance of the proposed architecture is evaluated by simulating and computing the number of clock cycles required. Using a clock frequency of 50 MHz, which is chosen to be less then the achieved maximum frequency of 62.9 MHz, computation time for the translation process is calculated. Based on this computed time throughput is calculated in terms of frames per second (fps). For all the experiments, over-sampling factor \(\alpha\) is taken to be equal to 4. This implies that for target point set \(T\) of size \(256 \times 256\), the sizes of corresponding \(S\) will be \(128 \times 128\). Table I shows the throughput for various sizes of tile, convolution window and set \(T\). Taking larger tile size meant larger part of \(T\) was available on the device at a time and more number of source points are grouped together in a single FIFO. This improves the locality of memory access and hence results in a higher throughput. Taking larger tile size for a fixed \(|T|\) results in less number of point-FIFOs and hence reduction in the complexity of Multiplexer and the centralized arbiter. Because of fewer number of FIFOs, size of an individual point-FIFO can be increased to accommodate more points. Larger convolution window (greater \(\sigma\)) results in more points pushed into multiple FIFOs and hence higher computation time. For a four times bigger convolution window, throughput was only reduced by 5%. But it has to noted that bigger convolution window results in more number of FpMA units and hence higher hardware complexity.

Other parameters like memory bandwidth utilization, power consumption, Mega Floating point Operations per Second (MFLOPS) and MFLOPS per watt were also computed. Memory bandwidth utilization was computed by taking the ratio of time the memory interface was active, to the overall computation time for the re-gridding process. For a target set \(T\) of size \(256 \times 256\) and a tile size of \(64 \times 64\), we have achieved 65.37% memory bandwidth utilization, while spending 27.02 watts to reap 750.42 MFLOPS.
TABLE I

<table>
<thead>
<tr>
<th>Size of Target set</th>
<th>$\sigma$</th>
<th>Tile Size</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>256 x 256</td>
<td>2</td>
<td>16 x 16</td>
<td>380.38 fps</td>
</tr>
<tr>
<td>256 x 256</td>
<td>2</td>
<td>32 x 32</td>
<td>691.33 fps</td>
</tr>
<tr>
<td>256 x 256</td>
<td>4</td>
<td>64 x 64</td>
<td>654.57 fps</td>
</tr>
<tr>
<td>256 x 256</td>
<td>2</td>
<td>64 x 64</td>
<td>715.66 fps</td>
</tr>
<tr>
<td>128 x 128</td>
<td>2</td>
<td>64 x 64</td>
<td>3081.98 fps</td>
</tr>
</tbody>
</table>

Fig. 9. Performance comparison of the proposed techniques with an FPGA [5] and GPU [16] based technique for a frame size of 256 $\times$ 256.

A. Comparison with Related Work

Kestur et al. [5] also proposed an FPGA based implementation of the re-gridding process. They implemented the gridding method and used a linked-list based technique to map the arbitrary samples dynamically. The results show that the FPGA based implementation achieved superior per-Watt performance compared to CPU and GPU but the throughput was quite low compared to the GPU version [16]. The major improvement we have made over [5] is better memory bandwidth utilization. By using an overflow-FIFO and allowing efficient sharing of memory bandwidth between SI and TTI through ROF based arbitration technique, each source point is read only once from the memory. We compare the throughput and power consumption of our proposed technique against FPGA [5] and GPU [16] implementations for a target set of size 256 $\times$ 256. [5] reports the results for a convolution window of size 4. We use $\sigma$ equal to 2 which also gives a convolution window of size 4.

It can be seen from Fig. 9 that compared to the throughput of FPGA based technique proposed in [5], our proposed architecture achieves a 9.6 times speed up but has a 1.35 times higher power consumption. When compared in terms of MFLOPS per watt, our architecture achieves 7 times more MFLOPS per watt of power. Compared to the GPU based technique [16], the throughput of our proposed technique is marginally higher but the power consumption is 5.9 times lower.

V. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed a novel power efficient architecture to accelerate the re-gridding process of NuFFT. Arbitrary sampled array of non-uniformly distributed source points was translated into a grid of uniformly spaced target points. This was achieved by accessing the memory efficiently using novel hardware components such as block memory based FIFOs, fill-rate based point-arbiter, decentralized memory control logic and an array of FpMA units. A speed up of over 9.6 was achieved compared to a published FPGA based technique at a 7 times higher MFLOPS per watt. A throughput marginally better than GPU was achieved at 5.9 times lower power. The achieved operating frequency is low because we are using Altera based single cycle Floating point adder that allows maximum operating frequency of 60 MHz. As part of our future work, we plan to take the proposed architecture to hardware and target bigger frame sizes by improving the design of the hardware intensive components like multiplexer and centralized arbiter.

ACKNOWLEDGMENTS

Authors would like to thank Altera for the DE4 donation and Quartus II licenses.

REFERENCES