3D Tomography parallelization on FPGAs via HLS tools

Maxime Martelli, Nicolas Gac, Alain Mérigot, Cyrille Enderli

3D Tomography back-projection parallelization on FPGAs using OpenCL

Maxime MARTELLI*†‡, Nicolas GAC*, Alain MÉRIGOT†, Cyrille ENDERLI‡
*Laboratoire des Signaux et Systèmes, CentraleSupélec, CNRS, Université Paris Sud,
Université Paris-Saclay, FRANCE
†Laboratoire des Systèmes et Applications des Technologies de l'Information et de l'Énergie,
ENS Cachan, CNRS, Université Paris Sud, Université Paris-Saclay, FRANCE
‡Thales Systèmes Aéroportés S.A., Elancourt, FRANCE

***Abstract*—This paper deals with the evaluation of FPGAs resurgence for hardware acceleration applied to computed tomography on the back-projection operator used in iterative reconstruction algorithms. We focus our attention on the tools developed by FPGAs manufacturers, in particular the Intel FPGA SDK for OpenCL, that promises a new level of hardware abstraction from the developer's perspective, allowing a software-like programming of FPGAs. For this purpose, we start with evaluating different custom OpenCL implementations of the back-projection algorithm. With some clues on memory fetching and coalescing, we then further tune designs to improve performance. Finally, a comparison is made with GPU implementations, and a preliminary conclusion is drawn on FPGAs future in computed tomography.**

***Index Terms*—High-Level Synthesis, FPGA, OpenCL, Optimization, GPU.**

I. Introduction

Moore's law is the observation that the number of transistors in a dense integrated circuit doubles approximately every two years. More like a roadmap, the current tendency shows its pace seizing up, a fact officially acknowledged by the industry. It was common knowledge that physical limitations would eventually block the circuit miniaturization, and the Semiconductor Industry Association announced the effective end of Moores law for 2021 [1]. Circuits miniaturization (up to 10 nanometres thin today) should soon tip over microprocessors from the realm of traditional physics to quantum physics, which governs the probability behaviour of atoms. This technological shift is due to happen at the start of next decade. Until then, traditional chip thickness is expected to reach a ceiling of around 10 nm, stabilizing a high-cadenced progression over the years.

The processor industry, which already dealt in the past with other issues regarding Moores law, like the increase of heat generation correlated with miniaturization, is now facing what looks like a deadlock. However, by considering performance instead of circuit density, the law can be transcended, and a logical evolution is to rethink commonly used architectures. One possible solution is for future computer chips to rely on a granular hardware specialization with reconfigurable fabric at their core, allowing processors to offload specific processing to a suited architecture.

With this in mind, Field Programmable Gate Array (FPGA) is a key technology for the post Moore era. Since the creation of the first mask-programmed gate array with Motorolas XC157 in 1969, FPGAs are widely used for specific needs like embedded [2] and critical [3] systems. Aware of their technological potential, the two main FPGA manufacturers (Xilinx and Altera[1]) developed tools (called SDAccel and Intel FPGA SDK for OpenCL, respectively) for the use of FPGA in software co-processing, in a similar approach as Graphics Processing Units (GPUs) manufacturers did with OpenCL and CUDA toolkits.

Their main advantage is, for programmers, usage of well-known software languages. OpenCL is commonly used for GPU programming, and Intel FPGA SDK for OpenCL claims to allow program optimization by having little to no FPGA experience. This assertion is a potential game changer in the computing field and the mainstream adoption of FPGAs is now at stake. Over the last few years, many research work focused on FPGA implementations with OpenCL in various fields like lossless data compression [4], stencil codes [5], but also on proposing some clues about OpenCL code tuning for FPGAs [6] [7]. In their continuum, our approach is focused on characterizing possible bottlenecks for FPGAs implementations, and memory optimization.

As a use case, we consider the back-projection (BP) algorithm used in iterative reconstruction[8]. Computed Tomography (CT) is used in multiple domains such as medical imaging [9], quality control [10], or even food characterization[11], and its primary mission is to reconstruct a 3D cartography of an object's parameter. Even though GPUs can solve problems with multiple dimensions and large-scale data thanks to their Single Instruction on Multiple Data (SIMD) architecture, there are some specific algorithmic limitations in tomography like memory bottlenecks that have been unlocked with FPGAs [12] [9]. Therefore, there is an interest to estimate if new FPGA improvements make them competitive relative to GPUs for Computed Tomography.

In this paper our contributions consist firstly in evaluating the impact of new FPGAs tools in tomography, and secondly,

[1]Now Intel FPGA, since Altera's acquisition in December 28^{th}, 2015

in assessing OpenCL code optimization from a software engineer's perspective.

The remainder of this paper is organized as follows: in Section II, we present the BP algorithm to be implemented via OpenCL on FPGAs. Then, we introduce in Section III relevant concepts of the Intel FPGA SDK for OpenCL. Section IV deals with the different optimization implementations, and results are provided and discussed in Section V.

II. 3D Tomography algorithm

Computed Tomography relies on the analysis of a known radiation stream through the considered object to recover said physical characteristic by reversing the matter transport equation, and its algorithm includes a BP that accounts in iterative reconstruction for 50% of the Computed Tomography execution time[13].

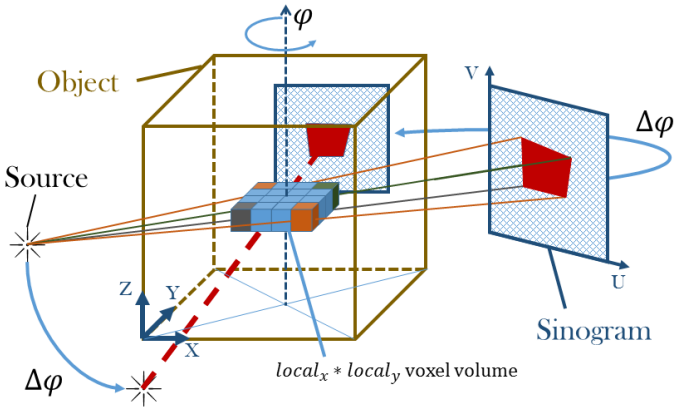

Fig. 1. 3D Computed Tomography Projection.

An X-ray source (Fig.1) revolves around the φ axis at $z = constant$. Radiation emitted from it is attenuated depending on the object local density, and a two-dimensional sensor array records intensity values, for each elementary φ angle. Those values are stored in a 3D matrix along (u,v,φ) in what is called a sinogram $s_{CT}(u, v, \varphi)$. The BP consists, for a given voxel[2] $\vec{c} = (x, y, z)$, in summing up the contribution of every elementary detector (u, v) in line with the source and the considered voxel for every φ value. We then obtain the density $d(\vec{c})$ given as follows :

$$d(\vec{c}) = \int_0^{2\pi} s_{CT}(u(\varphi, \vec{c}), v(\varphi, \vec{c}), \varphi).w(\varphi, \vec{c})^2 \mathrm{d}\varphi \qquad (1)$$

where $(u(\varphi_0, \vec{c}), v(\varphi_0, \vec{c}))$ are the values on the sinogram of the beam passing through $\vec{c}$ for $\varphi = \varphi_0$, and $w(\varphi_0, \vec{c})$ is the distance weight[14].

The sensors distribution being discrete, the integral transforms in a sum for all φ values. This algorithm is particularly suited for SIMD cores, because this sum has to be computed for every voxel of the object, and is best executed on massively parallel architectures. However, for a given voxel, the gathering of sinogram values for each φ iteration follows an irregular pattern, and a way to improve the algorithm is by grouping density computation per $local_x * local_y$ voxel rectangle to take advantage of memory coalescing. This particular point will be discussed in Section IV-B.

[2]Stands for VOlume piXEL.

III. Intel FPGA SDK for OpenCL

A. Architecture

OpenCL is an abstract programming model and the Intel FPGAs corresponding basic architecture is showcased in Fig.2. Adapted to a co-processing implementation on heterogeneous architectures, the framework is designed to easily abstract most hardware considerations. As so, the host program is written in standard C, and communicates with Compute Devices via library routines that handle communication between the host processor and devices kernels[3].

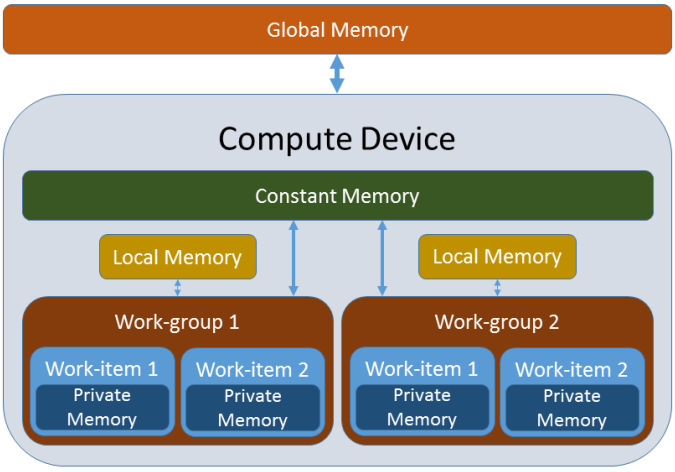

Fig. 2. Intel FPGA SDK for OpenCL Memory architecture.

Each kernel instruction is transformed by the Altera Offline Compiler in a sequence of logic blocks, creating elementary pipelines that are then aggregated to form the kernel pipeline, which we refer to as Compute Unit (CU). There are two OpenCL kernel categories : NDRange (ND) and single work-item (SWI), detailed in Section IV. Generally, Intel FPGA SDK for OpenCL Guides[15] [16] recommends implementing a single work-item kernel in case of loop or memory dependencies, and a NDrange kernel otherwise.

Given the 3-dimensional BP problem, with a dimension of (dim_X, dim_Y, dim_Z) voxels, we can use the OpenCL work-group and work-item partitioning to handle its computation. In this case, a work-item, that is the elementary instantiation of a kernel, is the computation of the volume density of a given (x_0, y_0, z_0) voxel, that is a sum of dim_φ sinogram values as explained in Section II. Multiple work-items can be aggregated as shown in Fig. 3, creating a work-group (equivalent of CUDA blocks). This repartition allows effective

[3]A function implemented on an accelerator device is called a kernel

internal communication and data synchronization between work-items of the same work-group.

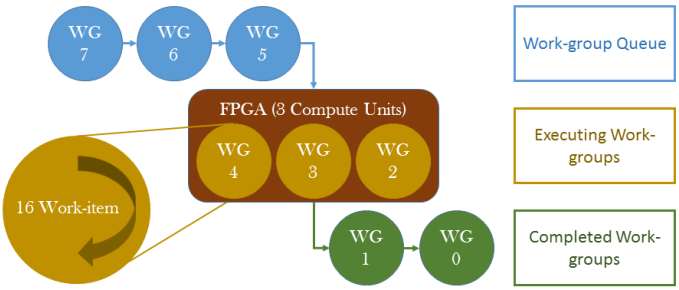

Fig. 3. OpenCL work-group enqueueing mechanism.

B. Memory model

The OpenCL implementation has four memory types : global, constant, local, and private. Their manual handling is a lead for effectively tuning the algorithms. In order to benchmark the different memory latency, we implement a routine program (experiment setup in V-B) executing multiple random reads in the four memory structures. Results are shown in Table I and discussed further on (code on Gitlab [17]).

TABLE I
MEMORY STRUCTURE LATENCY ON AN ALTERA CYCLONE V.

Memory structure	Mean latency (cycles)
Global	178
Constant	45
Local	13
Private	3

1) Global memory: Despite having the highest access latency amongst memory structures, global memory storage within the Intel FPGA SDK for OpenCL can still be efficient, thanks to automatic embedded on-chip caches implementation in Load-Store Units (LSUs) [15]. In case of repetitive global memory access, data will be stored in embedded caches[4] guaranteeing a high memory bandwith and a shorter latency compared to global memory, provided that memory access is not too large. A coalescent memory access through LSU embedded caches are the best way to optimize global memory bandwith.

2) Constant, local, and private memory: Constant memory is equivalent to on-chip global read-only memory. Access port allocation size and width can be manually tuned to a maximum defined by the hardware manufacturer. However, latency improves substantially with the number of ports accessing the constant memory, thus simultaneous constant memory access per work-item must be narrowed to the strict minimum.

The main difference between local and private memory is their accessibility within a work-group. A private variable is stored in registers, and accessible only to one work-item, whereas a local one is visible to all work-items of a work-group. Altera Offline Compiler automatically implements local and private memories depending on the underlying access patterns, but users can also allocate and use those two memory types.

[4]Direct-mapped 64 bytes cache with a 12 cycles mean latency.

IV. OPENCL 3D BACK-PROJECTION IMPLEMENTATIONS

The main FPGA advantage is its ability to be programmed for task or data parallelism. The Intel FPGA SDK for OpenCL allows both programming models, whose implementations are showcased further on (code on Gitlab [17]).

A. Task parallelism - Single Work-Item

Single work-item is referred to as Task Parallel Programming. Similar to the sequential model of a mono-threaded CPU program, there is no data repartition across work-items[5]. The Altera Offline Compiler optimization for this programming model is based on two core concepts : memory handling, and Shif-Register Pattern (SRP). Because the kernel is mono-threaded, the high-throughput is achieved by ensuring that at any moment, multiple instructions of the same kernel are processed concurrently at every pipeline step.

A key difficulty for single work-item implementations are loop handling, because the Altera Offline Compiler default behaviour is to have each loop iteration executed sequentially, thus drastically reducing the kernel throughput.

The baseline model for this article is the basic translation of the equation described in Section II as a single kernel on FPGAs, in a sequential CPU-like programming, and without any optimization.

From this implementation, a first optimization is to improve streaming throughput. By default, when a kernel needs to access an array, it allocates memory resources for efficient reads and writes. When an array access pattern matches with a streaming pattern, implementation can be modified to integrate a Shift-Register Pattern.

From lines 10 to 12 of Algorithm 1, we implement this Shift Register Pattern mechanism. Initially, α and β arrays are stored in global memory, and for every voxel, they are sequentially accessed. For every successive (x, y, z, φ) iteration, the implemented shift-register pattern shifts the data contained in the shift-register pattern array in a loop pattern. A streaming pipeline is generated through all the loops and shifted for each iteration, instead of a costly memory mechanism. This optimization allows the compiler to extract the parallelism between each loop iteration, effectively reducing the execution time as shown with the SWI+SRP+LM kernel version in Section V-C2. From a software developer perspective, those three lines seem counter-intuitive. In reality, the Altera Offline Compiler recognize the shit-register pattern and implements it as a cascade of flip flops, sharing the same clock.

B. Data parallelism - NDRangeKernel

Data parallel implementations on OpenCL strongly depends on the underlying hardware architecture. GPUs have SIMD architectures, whereas FPGAs can be reprogrammed as such,

[5]Single work-item architecture is as in Fig.2, but with only one work-group and one work-item per work group

Input: $\alpha[dim_\varphi]$, $\beta[dim_\varphi]$,
sinogram[dim_U*dim_V*dim_φ]
Output: volume, 3D array of reconstructed volume

```
1  int2 SRP[dim_φ];
2  for φ = 0 to dim_φ - 1 do
3      SRP[φ] = (α[φ], β[φ]);
4  for zn = 0 to dim_Z - 1 do
5      for yn = 0 to dim_Y - 1 do
6          for xn = 0 to dim_X - 1 do
7              voxel_sum = 0;
8              #pragma unroll;
9              for φ = 0 to dim_φ - 1 do
10                 SRP[dim_φ - 1] = SRP[0];
11                 for i = 0 to dim_φ - 2 do
12                     SRP[i] = SRP[i+1];
                   /* Calculate (U_n, V_n) from
                      SRP[φ]                    */
13                 voxel_sum += sinogram[U_n, V_n, φ];
14             volume[xn,yn,zn] = voxel_sum;
```

Algorithm 1: Shift Register Pattern Single work-item optimization

but works best with iterative-bound architectures. The Altera Offline Compiler instantiates an iterative loop in order to sequentially execute each work-group and, if the design allows it, multiple compute units can be implemented, allowing some work-groups to be launched in parallel.

Therefore, the main difference compared to task parallelism is the handling of shared local memory within a work-group. The shift-register pattern optimization (Section IV-A) of the projection coefficient α and β can no longer be implemented because of the work-group repartition, and the main challenge of the BP algorithm is to effectively access the sinogram array. The FPGA architecture inherent inadequacy to data parallelism can be curtailed if memory pre-fetching is efficient.

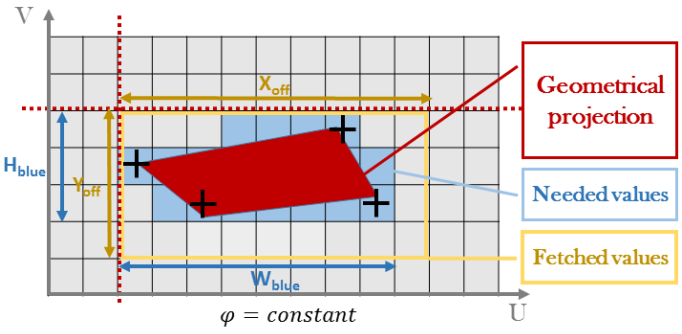

Fig. 4. Sinogram memory fetching pattern optimization.

As explained in Section II, (U, V, φ) is the projection of a voxel over the detector matrix. For a given φ, a projection of a $(local_x, local_y, 1)$ voxel rectangle looks like the red pattern of Fig. 4, the four ends of the volume rectangle corresponding to the four black points. To compute the volume density of the initial voxel rectangle, the kernel needs to access blue cells of the sinogram, and its access pattern cannot be predicted by the Altera Offline Compiler. Defining (W_{blue}, H_{blue}) as the Cartesian width and height of the blue region, the geometry of the 3D back-projection problem guarantees the following :

$$W_{blue} < \sqrt{local_x^2 + local_y^2},\ H_{blue} <= 4 \qquad (2)$$

In our implementation, $local_x = local_y = 16$. By choosing a local array dimension of $X_{off} * Y_{off}$ = 24*4, we are assured to fetch all necessary sinogram cells needed for the work-group computation. Therefore, the implemented fetching algorithm (Algorithm 2) first calculates the top-left coordinates[6] of a matching rectangle (intersection of the red dotted lines), and then evenly distributes the fetching of the values inside the constant-size yellow rectangle. To guarantee a well-ordered execution, work-item synchronization is mandatory, and achieved at lines 4, 5, and 6.

Additionally, α and β arrays are stored in constant memory, in order to reduce memory latency through all work-groups.

Input: constant $\alpha[dim_\varphi]$, constant $\beta[dim_\varphi]$,
sinogram[dim_U*dim_V*dim_φ]
Output: volume, 3D array of reconstructed volume

```
1  local int local_sinogram[X_off * Y_off];
   /* Recovery of work-item
      characteristics                        */
2  voxel_sum = 0;
3  for φ = 0 to dim_φ - 1 do
       /* Calculate U_n, V_n coordinates     */
       /* Dispatch min, max coordinates
          computation between local
          work-items                         */
4      barrier(CLK_LOCAL_MEM_FENCE);
       /* Global sinogram fetching by local
          work-items                         */
5      barrier(CLK_LOCAL_MEM_FENCE);
6      voxel_sum += local_sinogram[local_U_n,local_V_n];
7  volume[xn,yn,zn] = voxel_sum;
```

Algorithm 2: NDRange kernel fetching optimization

The drawback of this algorithm is that there are more memory fetching than needed, that is, in Fig. 4, blue values are always included in the yellow rectangle area, but its major advantage is that each line fetching is coalesced and shared between work-items of a work-group, thus improving burst read access and potentially reducing memory stalls.

V. Results and Discussion

A. Theoretical attainable performance

FPGA as a hardware platform can implement almost any type of instruction in a custom pipeline, and its architecture is therefore algorithm dependent. A major difficulty relies on defining proper FPGA performance, especially with HLS

[6]Or bottom-right, depending on the array boundaries.

tools. However, when it comes to pipeline parallelism, the optimal implementation guarantees one general operation per cycle per input data. For a given design, the number of normalized stream (NS) corresponds to the number of simultaneous handled input. For 3D back-projection, where the elementary data item is a voxel, can be defined the global optimal execution time of a design (3) and the maximal throughput (4).

$$T_{sim} = \frac{N_{voxel}}{f_{max} * NS} \tag{3}$$

$$D_{opt} = 1\,voxel/cycle/normalized\,stream \tag{4}$$

B. Experiment setup

The FPGA board used for testing our implementations is a DE1-SoC coming with 1 GB of DDR3 memory and an Altera Cyclone V chip that integrates both a dual core ARM Cortex A9 processor and the FPGA fabric, with a maximum FPGA frequency of 305 MHz. All versions were compiled and synthesized using the Intel FPGA SDK for OpenCL 16.0. The GPU used for comparison is a Nvidia Titan X Pascal, with CUDA 8.0 toolkit.

The considered volume is a 256^3 voxel cube, with 256 angles variations. Each kernel execution is monitored through the Altera OpenCL Profiler. For each kernel, this tool provides amongst other things the operating frequency, the execution time, the logic utilization, and the latency, bandwith and stall of most memory access.

In the following subsection, we suggest a benchmark for this specific BP algorithm, to further discuss the execution results shown on Table II. Kernel implementations match the different optimizations previously discussed in Section IV.

C. OpenCL optimization for FPGA

1) Performance benchmarking: In practice, theoretical performance of Section V-A is unattainable. The Altera Offline Compiler memory management reduces the maximum operating frequency for kernels, and use an irreducible percentage of logical elements for kernel enqueueing. To evaluate the minimum logical footprint of our algorithm, we construct the ND+Backbone kernel by removing all memory access in the code, with only the algorithm backbone remaining. This gives us two important information. Firstly, the logic utilization cannot be shrunk more than 21% on the DE1-SoC board, and secondly, the measured mean frequency for NDRange kernels is of 140 MHz.

Also, because of the specificity of each design, every kernel implemented uses a varying percentage of the total available logic elements. In order to accurately compare those designs, we propose a linear extrapolation of the execution time for a given logic utilization to a full-chip usage. To validate this assertion, we replicated the initial design (ND+Naive) into a design with two Compute Units (ND+CU, cf Table II) on the same chip. Each compute unit of this new kernel handles half of the total voxels. Results show that the time estimated with our linear model (Normalized Execution Time (NET)) is slightly slower than the actual replicated kernel execution time (ND+2CU Raw Execution Time (RET)). This is mostly due to the memory footprint of two compute units being smaller than twice the logic utilization used for a single one thanks to the Altera Offline Compiler optimizations. Overall, it validates our extrapolation as a good empirical model, and we can safely use the Normalized Execution Time to compare performances.

2) Single work-item and NDRange kernels: As discussed in Section IV-A, we implemented two different single work-item kernels. The first one (SWI+Naive) is the baseline algorithm for this article with no optimization, and the second one implements a shift-register pattern to reduce memory footprint and increase streaming efficiency. This simple improvement gives a normalized 4.5 speedup, underlining the importance of a comprehensive approach to the algorithm data access pattern for performance optimization.

With a single work-item there is no notion of shared memory within a work-group. Therefore, it has less logical footprint than a NDrange kernel. However, with an execution time of 67.5 s, the kernel mean frequency is of 63.6 MHz, compared to a maximum operating frequency of 140 MHz per normalized stream. Improving a single work-item kernel is closely related to optimizing memory handling and data streaming effectiveness, in order to increase kernel frequency.

The ND+Naive kernel is the GPU-like version of the BP algorithm, with no memory optimization. From this version, we implemented the replicated kernel (ND+2CU) already presented in Section V-C1 and the memory fetching kernel described in Section IV-B.

What is noticeable is that the Altera Offline Compiler top priority is to guarantee no kernel stall. With approximately an execution time of 30 s, and a 140 MHz operating frequency for all NDrange kernels with one normalized stream, this means that we execute one (x, y, z, φ) computation per clock cycle. Also, Normalized Execution Time for the memory fetched kernel (ND+LM+MF) has a 1.4 speedup compared to the ND+Naive iteration, thanks to a reduced logical footprint.

From a software developer's perspective, the Altera Offline Compiler is quite effective. Firstly, a program can be implemented on FPGAs using two different kernel types suited for task or data parallelism, and this generic characteristic can be used over a wide range of algorithm implementations on FPGAs. Secondly, the automatic optimizations are focused on performance, and guarantees a filled pipeline. However, this automatization comes with two main drawbacks : a bigger memory footprint, and a reduced kernel frequency. Therefore, effective optimization tracks is to reduce the logical footprint and increase operating frequency, allowing more kernel replication.

D. GPU versus FPGA, consumption and performance

The DE1-SoC chip is a low-value product. For an adequate comparison to high-end GPUs, we compiled the ND+LM+MF kernel with 17 replicated CU targeting an Arria 10 FPGA,

TABLE II
RAW AND NORMALIZED EXECUTION TIME OF VARIOUS KERNEL OPTIMIZATIONS ON THE CYCLONE V SOC.

Kernel version	Logic utilization (%)	RET(s)	NET(s)
SWI+Naive	49	222.9	109.2
SWI+SRP+LM	36	67.5	24.3
ND+Naive	55	32.26	17.7
ND+2CU	96	16.9	16.2
ND+LM+MF	40	31.3	12.5
ND+Backbone	21	30.8	6.47

using 98% of its logic elements. Within Quartus, we obtained the kernel operable frequency (260 MHz), and using the PowerPlay Early Power Estimator, we get the power of the design (2.27 W). Because we didn't have access to a SX660 Arria 10 card, we estimated the execution time by dividing the Raw execution time of the ND+LM+MF kernel on a Cyclone V by the number of replication fitting on an Arria 10, and further multiplying it by the ratio of both designs operable frequency ($t_{Arria\ 10} = \frac{31.3}{17} * \frac{140}{260} = 0.991\ s$).

We compare the extrapolated execution time on an Arria 10 to the raw execution time measured on a Titan X Pascal GPU (11 TFLOPS), and to the extrapolated execution time on an embedded Jetson TK2 GPU (1.4 TFLOPS). In terms of raw performance, FPGA is merely comparable to GPUs due, as discussed in Section II to the BP algorithm being appropriate for data parallel architectures. As shown in Table III, even if an Arria 10 OpenCL implementation has a better performance per watt than on a Titan X, the same program is faster on a Jetson TK2 while consuming less.

TABLE III
POWER AND ENERGY CONSUMPTION OF BP OPTIMIZATION ON GPUS AND FPGAS.

Device	Power (W)	Execution time (ms)	Energy (mWh)
Titan X Pascal	250	12	0.83
Jetson TK2	15	94	0.39
Intel Arria 10	2.27	991	0.63

Even though performances were largely improved on FPGAs compared to the naive version, the inadequacy between algorithm and architecture remains a major obstacle for implementing this type of algorithm on FPGAs.

VI. CONCLUSIONS AND PERSPECTIVES

In this paper we presented different FPGA optimizations using the Intel FPGA SDK for OpenCL. We achieved to port a CPU code on a FPGA, with an overall speedup of 8.74 between the naive and the best optimized kernel on a Cyclone V chip. What first spring to mind is that the developer must be aware of its program specificity, and some hardware knowledge is required to fully harness the power of OpenCL on FPGAs. Even more, memory management is at the core of OpenCL implementation, and, on FPGAs, reducing kernels logical footprint is key for further optimizations. Despite those improvements, FPGA is lagging behind GPU implementations partly due to a mismatch between the algorithm and FPGA architecture. We observed that the algorithm backbone was using a significant percentage of the total available logic (ND+Backbone cf Table II)), and this overhead, caused by the OpenCL handling, limits larger improvements.

In this way, FPGAs resurgence for tomography is not due to happen unless manufacturers integrate dedicated graphical cores within FPGAs. However, their rise is most likely through OpenCL thanks to its capacity to allow data and task parallelization with the same syntax while exploiting software developers' expertise. In the near future, it is therefore viable to think about heterogeneous architectures where algorithms will be segmented depending on their inherent specificity, and each elementary partition executed on FPGA, GPU, or CPU devices, while using the same language: OpenCL.

VII. ACKNOWLEDGMENTS

We would like to thank Nicolas Heemeryck and Mickael Seznec for their preliminary work on CUDA to OpenCL portability.

REFERENCES

[1] "The International Technology Roadmap For Semiconductors 2.0," *Semiconductor Industry Association*, 2015.
[2] Philip Garcia, Katherine Compton, Michael Schulte, Emily Blem, and Wenyin Fu, "An Overview of Reconfigurable Hardware in Embedded Systems," *EURASIP Journal on Embedded Systems*, 2006.
[3] Marek Wegrzyn, "FPGA-Based Logic Controllers for Safety Critical Systems," *IFAC Conference on New Technologies for Computer Control*, 2001.
[4] Mohamed S. Abdelfattah, Andrei Hagiescu, and Deshanand Singh, "Gzip on a chip : High Performance Lossless Data Compression on FPGAs using OpenCL," *International Workshop on OpenCL*, 2014.
[5] Qi Jia and Huiynag Zhou, "Tuning Stencil Codes in OpenCL for FPGAs," *International Conference Computer Design*, 2016.
[6] Zeke Wang, Bingsheng He, Wei Zhang, and Shunning Jiang, "A Performance Analysis Framework for Optimizing OpenCL Applications on FPGAs," *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 114–125, 2016.
[7] Kavya Shagrithaya, Krzysztof Kepa, and Peter Athanas, "Enabling Development of OpenCL Applications on FPGA platforms," *Conference on Application-Specific Systems, Architectures and Processors*, 2013.
[8] P. E.Kinahan et al., "Emission tomography : the fundamentals of PET and SPECT, chapter Analytic image reconstruction methods. Elsevier Academic Press," 2004.
[9] Nicolas Gac, Stephane Mancini, Michel Desvignes, and Dominique Houzet, "High Speed 3D Tomography on CPU, GPU, and FPGA," *EURASIP journal on Embedded Systems*, 2008.
[10] S. L. Vasilev, A. V. Artemev, V. N. Bakulin, and S. A. Yurgenson, "Testing loaded samples using X-ray computed tomography," *Russian Journal of Nondestructive Testing*, 2016.
[11] M. Vidhya, N. Varadharaju, Z. John Kennedy, D. Amirtham, and D. Manohar Jesudas, "Applications of X-Ray Computed Tomography in Food Processing," *RSNA*, 2015.
[12] M. Leeser et al., "Parallel-beam backprojection: an FPGA implementation optimized for medical imaging.," *VLSISIgnal Processing Systems*, vol. 39, no. 3, pp. 295–311, 2005.
[13] Lucas L. Geyer, U. Joseph Schoepf, Felix G. Meinel, John W. Nance, Gorka Bastarrika, Jonathon A. Leipsic, Narinder S. Paul, Marco Rengo, PhD Andrea Laghi, and Carlo N. De Cecco, "State of the Art: Iterative CT Reconstruction Techniques," *Journal of Food Processing & Technology*, 2015.
[14] Hongbing Lu, Jui-His Cheng, Guoping Han, Lihong Li, and Zhengrong Liang, "A 3D distance-weighted Wiener filter for Poisson noise reduction in sinogram space for SPECT imaging," *Medical Imaging, Physics of Medical Imaging*, 2001.
[15] "Intel FPGA SDK for OpenCL Best Practices Guide," *Intel*, 2017.
[16] "Intel FPGA SDK for OpenCL Programming Guide," *Intel*, 2017.
[17] "Gitlab link," https://gitlab.com/maxGit/tomography-back-projection.git.