X-ray tomography reconstruction accelerated on FPGA through High-Level Synthesis tools

Daouda Diakite, Nicolas Gac

X-ray Tomography Reconstruction Accelerated on FPGA through High-Level Synthesis Tools

Daouda Diakite, and Nicolas Gac

***Abstract*— Model-Based Iterative Reconstruction (MBIR) algorithms iteratively use expensive computational operators of forward and backward projections. The irregular memory access pattern of these operators makes them a memory-bound application. Their computation time must be reduced to meet clinical routine constraints. This paper proposes a hardware accelerator architecture based on Field Programmable Gate Arrays (FPGAs) through high-level language, as an alternative to GPU architecture. This acceleration is based on an offline memory access analysis to address the main bottleneck of the algorithm and maximize the data reuse rate. The offline analysis allows for the tuning of the architecture parameters so that they converge to an optimal solution. Then, the Berkeley Roofline model guides our optimization steps by iteratively analyzing the design performance. Our design flow significantly improved the algorithm's computational intensity and overcame the memory bottleneck. Thus, our architecture takes advantage of the FPGA local memory to achieve significant memory bandwidth and efficiently harness the pipeline without stalling the computation. Furthermore, we present the scaling-up strategy from mid-range FPGA to high-end FPGA and any concerns of portability. We used two Intel FPGA devices to implement the algorithm, and then we compared the results with our GPU implementation in terms of speedup and energy efficiency. Our experimental results show that our design has achieved better computational throughput than the works on FPGA architectures reported in the literature.**

***Index Terms*— FPGA, hardware acceleration, high-level synthesis, memory access analysis, roofline model, x-ray computed tomography.**

I. Introduction

X-RAY Computed Tomography (CT) is an imaging technique that initially found its application in the medical field. It has been extended to industrial applications such as non-invasive human body investigation and non-destructive testing of industrial materials. Model-Based Iterative Reconstruction (MBIR) algorithms [1] are proven to produce better image quality at the cost of expensive computational time. To reduce the reconstruction time of CT algorithms, hardware accelerators are required. For the past few years, Graphical Processing Units (GPUs) have been the preferred architecture due to their massively parallel computing pattern [2]. However, Field Programmable Gate Arrays (FPGAs) can be re-considered thanks to their low latency, power efficiency, and accessibility through High-Level Synthesis (HLS) tools provided by leading manufacturers like Intel or Xilinx.

FPGAs based on HLS tools are experiencing great consideration as an acceleration platform for many applications such as high-performance computing [3]–[5] or deep neural networks [6], [7]. The maturity of the HLS tools, making them easier to use, and their many built-in floating-point units (e.g., Digital Signal Processing (DSP) units) in the latest FPGAs explain this interest. These floating-point units provide high design flexibility and are optimized to support high-performance DSP applications in IEEE 754 compliant floating-point single precision. For instance, Intel Stratix 10 [8] and Intel Agilex [9] devices can achieve up to 9 TFLOPS and 20 TFLOPS respectively. FPGAs can express spatial and temporal (fine/coarse-grained) parallelism, making them suitable for algorithms with sequential patterns and high data dependency. The HLS tools have made FPGAs flexible for a wider audience of developers by offering a higher level of abstraction. The HLS compilers such as Xilinx Vivado HLS [10] and Intel HLS compiler [11] use C++ entry to provide a Register Transfer Level (RTL) design for FPGAs. They allow hardware designers to gain development time at a higher abstraction level while guaranteeing control to tune the pipeline characteristics, namely the desired operating frequency, the cycle accuracy, and the Initiation Interval (II)[1]. These HLS tools are IP-driven and more suitable for hardware designers. The FPGA vendors have provided their OpenCL SDK to enable FPGA utilization for software developers. This makes FPGAs utilization possible along with other architectures such as CPU and GPU due to the OpenCL code portability. The OpenCL compiler also allows adjusting the characteristics of the pipeline. However, the designers should alleviate excessive routing constraints. Intel HLS compiler is addressed to hardware designers, while OpenCL SDK responds to software designers by enabling heterogeneous computing. Therefore, the Intel OpenCL SDK is a higher abstraction level tool compared to the Intel HLS compiler and Xilinx Vivado HLS. The OpenCL compiler could benefit from a thorough knowledge of FPGAs architecture and their programming paradigm to generate an efficient pipeline.

HLS tools can significantly reduce development time by allowing the description of an FPGA design through a high-level language, which constitutes their main advantage over Hardware Description Languages (HDL) design. However, using tools at a higher level of abstraction, such as OpenCL, comes with some loss of control over the hardware design. This paper evaluates whether such a tool can guarantee the same level of performance and efficiency as an HDL design. FPGAs are reliable architectures for several applications used through HDL languages to provide high efficiency, and it is essential to preserve this efficiency using HLS tools.

In the past, the various parallelisms on FPGAs were extracted for tomography through the HDL languages requiring strong hardware skills [12]–[15]. This level of abstraction requires a heavy and time-consuming development, based on the complexity of specific algorithms. Hence, the emergence of tools with a high level of abstraction allows a broader audience to use FPGAs through software programming languages like C, C++, or OpenCL. FPGAs with HLS have recently been subject of evaluation in CT algorithms such as Maximum Likelihood Expectation Maximization (MLEM) [16], [17], 3D back-projection [18], [19] or CT data alignment in memory [20]. These works

The authors are with the Université Paris-Saclay, CNRS, CentraleSupélec, Laboratoire des signaux et systèmes, 91190, Gif-sur-Yvette, France. (E-mail: daouda.diakite@l2s.centralesupelec.fr; nicolas.gac@l2s.centralesupelec.fr).

[1]the Initiation Interval represents the number of clock cycles between the launch of consecutive loop iterations.

focus on FPGA-specific optimizations for CT in algorithm architecture co-design purposes. However, other works have based their optimization on the algorithm itself for adaptation to the architecture. The authors of [21] proposed a Ray-driven voxel-tile parallel approach that maximizes the data reuse rate to take advantage of FPGA Block RAM (BRAM). In [22], they also proposed a parallel beam-based reconstruction on FPGA to exploit on-chip BRAM intensively. Both their works [21], [22] use Vivado HLS to synthesize the pipeline. The authors of [23] proposed an FPGA design of iterative reconstruction for Transmission Electron Tomography (TET) by using data locality to optimize memory accesses.

Operators used for reconstruction in tomography are qualified as memory-bound algorithms. Multiple memory models aside (global, constant, ...), Intel FPGA SDK for OpenCL [24] proposes several Load Store Units (LSU) implementations based on the memory access pattern in order to optimize the memory accesses. The compiler implements the LSU that best matches the kernel memory access pattern. However, the programmer needs to help the Altera Offline Compiler (AOC) in the code writing to choose adequate LSU as discussed in [25]. FPGAs DRAM bandwidths are typically insufficient to meet memory-bound applications. Nevertheless, on-chip BRAMs are available and must be as an alternative to achieve maximum throughput. These BRAMs are, however, very small in size for applications with huge data such as CT. A data access strategy must be developed to take advantage of these on-chip blocks without stalling the pipeline.

In this paper, we propose an OpenCL acceleration of the voxel-driven 3D back-projection algorithm on Intel FPGA devices. This design flow is initially based on an offline memory access analysis and then iteratively on a performance analysis represented on a Berkeley Roofline [26]. Our implementation is based on blocks of voxels reconstruction taking into account the date reuse rate, and exploiting the local memory on FPGA using Intel FPGA SDK for OpenCL. We compare the results with other FPGA implementations regarding throughput, execution time, and design efficiency.

The main contributions of this paper are as follows:

- A voxel-driven back-projector architecture designed on FPGA through HLS tools. This algorithm is known to have a high parallelism potential on voxels and a high data reuse rate among neighboring voxels.
- A customized memory access strategy to reach a full computational throughput. Memory optimization first involves the design of prefetcher modules to provide the projection data by reducing global access. The access to the projection data in BRAM memory by the PEs is then elaborated to avoid memory stalls and access conflicts.
- A multi-kernel design has been proposed as an alternative to the single kernel in order to scale the architecture on Stratix 10 and provide a better balance between the compute unit and the prefetcher.
- A comparative study between GPU and FPGA HDL implementations has been performed. The results have also been compared with other FPGA-based HLS works reported in the literature.

This work is an extension of our previous work in [27]. The earlier work focused on an offline memory access analysis on an Intel Arria 10 device. In this work, we deepened this offline analysis to maximize the data reuse and detailed our custom pipeline architecture. We also added the scaling-up strategy to higher-end FPGA devices for higher parallelism. This work enhanced the performance of our design and performed a 3.5× speedup compared to [27]. The design started with a general focus on the architecture to be synthesized onto FPGA by establishing its main stages. The offline memory access analysis follows this architectural study to increase the rate of data reuse by choosing the optimal block size and using the compilation information provided by the HLS tools to ensure that the design fits into the target FPGA. Based on this offline analysis, we begin tuning our architecture and perform the full synthesis to validate this study.

The remainder of this paper is organized as follows: in Section II we present the background of MBIR, the acceleration platform, and the roofline model. We introduce in Section III our BP-Prefetch pipeline architecture and the memory access strategy. In Section IV, we describe our offline analysis for data reuse evaluation and the Computational Intensity (CI) static analysis. We also evaluate our design on Intel FPGA devices by performing first-stage compilations. The experimental results are presented in Section V along with the roofline model of the design. In Section VI presents the comparison to related work and the discussion, then Section VII concludes this work.

II. Background and Motivation

A. Model-Based Iterative Reconstruction

3D CT aims to acquire the internal density f of 3D objects from external measurements g called sinogram or projection data. The 3D object is placed between an X-ray source and a detector plane as illustrated in Fig. 1. The direct model of the problem can be modeled as follows:

$$g = Hf + \epsilon \tag{1}$$

where g represents all the projection data, f the volume to be reconstructed, H the system matrix, and ϵ the additive noise. The reconstruction problem consists of determining the object

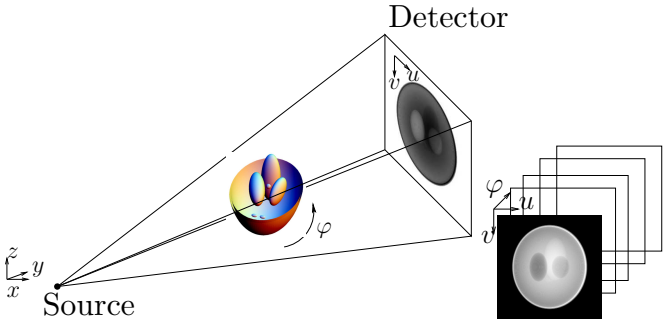

Fig. 1. X-RAY CT Projection. Modified from [27]

of interest f given the measured projection data. The 3D image f is estimated by minimizing a quadratic criterion J_{MC}:

$$\widehat{f} = arg\ min J_{MC} = arg\ min \frac{1}{2}||g - Hf||^2. \tag{2}$$

Iterative methods of reconstruction are used to solve this linear problem by performing several computations of the forward and backward projection using J_{MC} the quadratic error and performing a gradient descent:

$$f^{(n+1)} = f^{(n)} - \alpha.\nabla J_{MC}(f^{(n)}) \tag{3}$$

where $\nabla J_{MC}(f^{(n)})$ is the least square term as follows:

$$\nabla J_{MC}(f^{(n)}) = -2H^t(g - Hf). \tag{4}$$

The solution f is approached using the iterative algorithm until convergence is reached. Starting from an initial image $f^{(0)}$ computed from the measured sinogram, a new volume $f^{(n+1)}$ is calculated from the current estimation $f^{(n)}$, the gradient of the cost function and the function decreasing step size α. The large size of the system matrix is too large, and its storage is impossible. However, the system matrix is required to compute the forward projection Hf and the backward projection $H^t(g - Hf)$. Consequently, the coefficients of H and H^t are computed on the fly by the forward and backward operators. Hence, the storage of the system matrix and the matrix multiplication are avoided.

Backward projection is one of the most time-consuming steps in MBIR CT. The back-projection operator is also used in analytical reconstruction algorithms such as Filtered Back Projection (FBP) [28]. The voxel-driven 3D back-projection used in iterative reconstruction and described in detail in [12] algorithm is given by:

$$f(c) = \int g(u(\varphi, c), v(\varphi, c), \varphi).w(\varphi, c)^2 d\varphi \tag{5}$$

where $c = (x, y, z)$ are the voxel coordinates, (u, v) are the cone beam coordinates, φ is the angular trajectory of the detector and w is the distance weight.

For each voxel (x, y, z), the projection of its contribution is located at a position $(u(x, y, \varphi), v(x, y, z, \varphi))$ on the detector:

$$u(\varphi, c) = x * cos(\varphi) + y * sin(\varphi) \tag{6}$$

$$v(\varphi, c) = x * sin(sin\varphi) - y * cos(\varphi) + z. \tag{7}$$

The contribution on the detector is computed by bi-linear interpolation. In our design, the interpolation is replaced by the nearest neighbor method to reduce resource consumption and computation overhead.

The forward projection is also used in the reconstruction process, but it is not considered in this work.

B. OpenCL HLS

OpenCL is an open-source parallel programming API for heterogeneous processing platforms (CPU, GPU, FPGA). Based on the C99 standard, OpenCL supports both data and task-parallel programming models: Single Work-Item (SWI) and NDRange (NDR) kernels [29]. The first one, called SWI kernel, models the design as a deep pipeline to exploit the parallelism at the finest granularity. The second one, NDR kernel, exploits data parallelism to achieve good design efficiency. An OpenCL application is composed of two programs: a host application and the kernel compiled separately using Just-In-Time (JIT) compilation. The JIT compilation is not supported due to the long-time place and route step for bitstream generation on FPGAs. Therefore the OpenCL kernel is compiled offline using a vendor-specific compiler followed by the full synthesis flow. The Board Support Package (BSP) as shown in Fig. 2, and provided by the board manufacturers, allows programmers to run the kernel executable on the target FPGA. It packages features such as IP Cores, DDR controller, PCIe controller, and DMA drivers to establish communication between the host and the FPGA device. Many Intel FPGA manufacturers provide FPGA with their BSP to quickly design and run Intel device applications using Intel FPGA SDK for OpenCL. OpenCL SDK also provides, besides OpenCL directives, many FPGA-specific optimizations to fully harness the device potential.

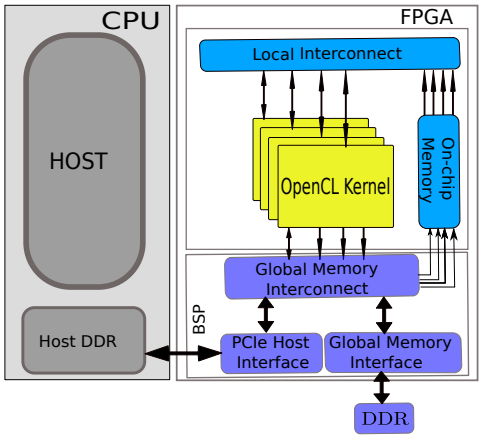

Fig. 2. Intel FPGA SDK for OpenCL platform [27]

The HLS compilers implement different optimization based on their particular Intermediate Representation (IR), such as pipelining or expressing data-level parallelism. However, they suffer from a lack of support for many other unexploited FPGA-specific optimizations for arithmetic operations [30] for instance. Hence, harnessing FPGAs' full potential via HLS tools requires knowledge of their architecture and a significant effort to adapt the application because OpenCL is not performance-portable from one platform to another. This leads to the evaluation of OpenCL optimization techniques on FPGA by various works [3], [4], [19], [31].

Algorithm 1 Kernel OpenCL for 3D back-projector

```
1: for all Voxel_i do
2:     voxel_sum ← 0
3:     #pragma unroll N
4:     for all φ_j do
5:         Compute(u, v)
6:         voxel_sum += projection[u, v, φ_j]
7:     end for
8:     volume[Voxel_i] = voxel_sum
9: end for
```

Algorithm 1 represents the pseudo-code of (5) where $Voxel_i$ corresponds to a voxel with coordinates (x, y, z). The AOC compiler will generate a pipeline to describe this algorithm at the hardware level. As mentioned above, the OpenCL framework is able to express two kinds of parallelism: a single-threaded kernel and a multi-threaded kernel. The algorithm memory access pattern and the available bandwidth make the use of NDR kernel less efficient on FPGAs [18]. Therefore, the algorithm 1 is a pipelined version and will be investigated in this work.

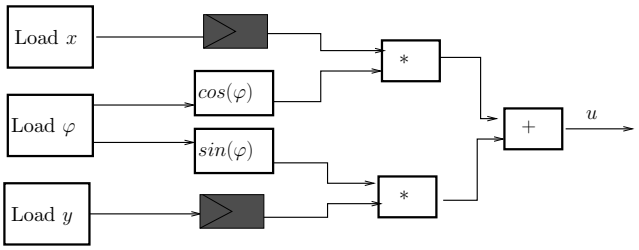

Fig. 3. One part of a back-projection pipeline computing u coordinate in (6)

Fig 3 represents an example of a pipeline architecture that should be generated by the offline compiler for computation of the detector coordinate $u(\varphi, c)$ in (6). The temporal registers inferred after loads of x and y allow the routing of these values along the pipeline until their utilization. The compiler generates such a pipeline for all operations in the design resulting in a deeper pipeline capable of producing a result at each clock cycle. However, HLS tools such as OpenCL do not provide access to that kind of low-level architecture details of the design.

C. FPGA local memory management

We will take advantage of BRAM in order to lighten the global memory bandwidth. FPGA does not support cache hierarchy, as we can observe on general-purpose processors such as CPU or GPU. Nevertheless, one can implement dedicated on-chip memory blocks based on the application memory access pattern. The accesses to local memory are mapped to a physical port. Each local memory has only two read/write ports. It is possible to use double pumping to have twice as many ports. The double pumping allows the compiler to run the memory clock twice the maximum frequency of the kernel. Thus, each local memory object can have at most four ports for simultaneous access. Intel recommends having four or fewer read/write accesses to local memory for stall-free access without arbitration [32].

In order to perform more than four concurrent accesses, the offline compiler will perform memory banking or replication to avoid access conflict. Memory replication consists of replicating the memory object as many as possible to have enough access ports. Each replicate contains exactly the same data.

Once the local memory is well configured with the right number of ports with no arbitration, one could create private copies for this local memory object. The goal of these private copies is to allow concurrent execution across different iterations of a given loop. Unlike replication, each memory copy contains different data for different loop iterations. However, the increase in the number of copies will consume more BRAM resources.

In this work, several memory accesses per cycle will be performed by our architecture to keep occupying the thousands of DSPs available on the FPGA card. To that end, memory replication will be used to avoid access conflict and infer privates copies for concurrent execution of loop iterations.

D. Roofline model for FPGAs

The roofline model first introduced by [26] is a tool for visually and quickly observing the possible limitations of an algorithm relative to theoretical maximum performance on a target architecture. The model is characterized by two key parameters which define two roofs: the device's peak Computational Performance (CP) and the attainable bandwidth. An algorithm's attainable performance is expressed as a function of the CI on the roofline model. The CI corresponds to the number of floating operations performed by memory access, i.e., the algorithm complexity.

The attainable performance for an application is expressed as follows and the model is presented in Fig 4:

$$\text{Attainable Performance}(FLOP/s) = \min(CP, CI \times BW). \quad (8)$$

The CI indicates the algorithm's complexity and represents the ratio of the number of operations to the number of memory accesses.

$$CI = \frac{\#\text{Operations}}{\#\text{Memory access}} \quad (9)$$

The algorithm's offline analysis helped us to obtain the number of operations and the number of memory transactions. Based on the CI, an algorithm is considered as a memory-bound or a compute-bound application.

The work of Williams *et al.* [26], focused on CPU multicore architectures, was extended to the FPGA architectures in [33] through HLS tools and taking into account the resource utilization of the device. The model is both architecture-dependent and application-dependent for FPGAs, thus the authors introduced the scalability parameter determined by the available resources on the target FPGA for pipeline replication. The scalability parameter gives an indication of the replication potential of the elementary pipeline. Therefore, the computational roof is influenced by the scalability factor and is given by the available resources on the target FPGA. The paradigm of the FPGA roofline is even more complex regarding the bandwidth and the computational ceiling. The theoretical limits provided by vendors are far from those obtained after synthesis. We need to use the profiling tools available to collect data, which allows us to have more design information such as effective bandwidth, operating frequency, and runtime.

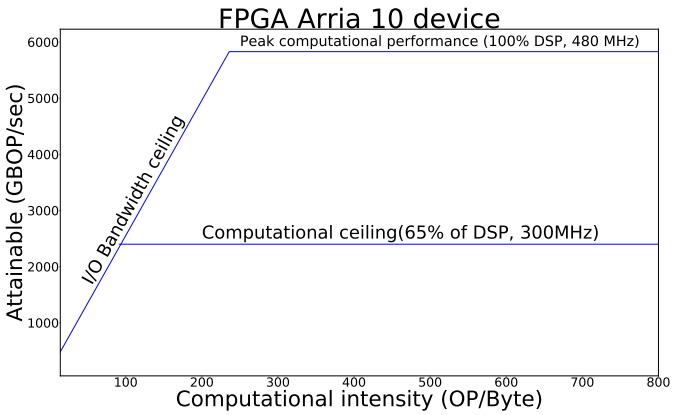

Fig. 4. Roofline representation on Intel Arria 10. Two computational roofs, the higher roof is the theoretical CP and the lower roof is the computational ceiling for a specific application on FPGA

An example of roofline for a design occupying 65% at 300 MHz on an Arria 10 is presented in Fig. 4. The higher roof corresponds to the theoretical computational ceiling when the design uses the full DSP capability running at the maximum frequency. The second roof is the computational ceiling of a design with respect to the actual DSP usage and frequency. This second roof is the maximum attainable performance for the given pipeline. The applications will be plotted on that roofline regarding their CI. Depending on the CI, an application could be bounded by the memory ceiling or the computational ceiling. The goal is to have an application in the compute-bound area and get close to the roof.

III. BP-Prefetch design

3D back-projection is one of the most time-consuming steps in iterative reconstruction. In this section, we explain the memory access strategy used to avoid the main bottleneck of this algorithm. We then describe our BP-Prefetch architecture on FPGA along with all the main stages of the pipeline.

A. Memory access strategy

The detector planes (u, v) are stored in memory following the projection angles (φ). The voxel-driven back-projector accumulates, for each voxel, the elementary contribution of each projection pixel in line with the given voxel under consideration for all angular projections. The reconstruction of a single voxel thus requires sinogram pixels located in different detector planes $(\varphi = 0, \varphi = 1, ..., \varphi = N_\varphi)$. This projection data also called a sinogram, is stored in the long latency global memory of the FPGA board and cannot be fully transferred in FPGA on-chip memory because of its tremendous size. During each voxel back-projection, sinogram pixels accessed in memory are deterministic but discontinuous with high and irregular strides, as illustrated in Fig. 5. The coordinates of these pixels cannot be precomputed in practice because of the unsustainable storage cost it would require. Hence each voxel back-projection implies on-fly projection coordinate computations with irregular jumps in memory, making standard cache mechanisms inefficient in predicting the memory accesses for the next φ angles.

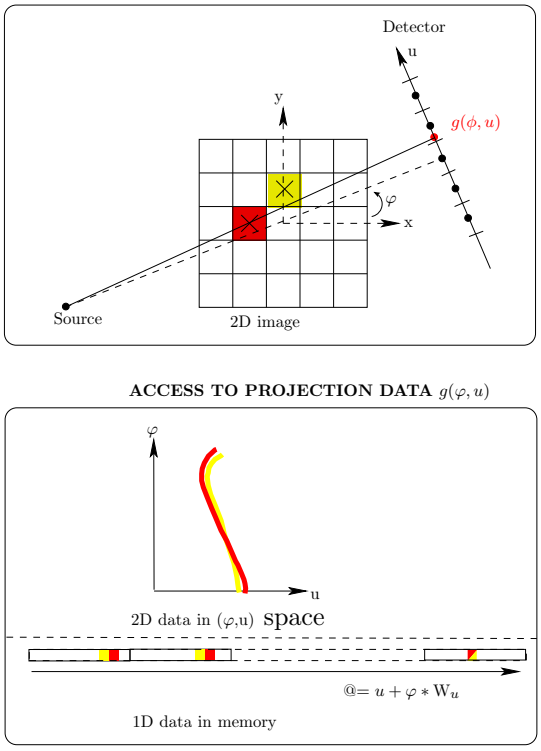

Fig. 5. Memory accesses for the voxel-driven back-projection in the 2D case. The red and yellow neighboring pixels draw close sinusoids in the 2D sinogram. This spatial locality is exploited during a voxel block reconstruction. $\boldsymbol{W_u}$ corresponds to the number of projection pixels

The Intel FPGA automatic on-chip cache implementations are, above all, efficient for contiguous and repetitive global memory access. Regarding non-sequential and random accesses, these automatic caches are inferred by the Intel compiler and are much less relevant for speeding up the application on FPGAs efficiently. Their inference is counterproductive and wastes valuable BRAM resources with a high risk of memory stalling. The acceleration of CT algorithms suffers from this memory stalling and consumption, which has been pointed out by [18] for the 3D back-projection algorithm, thus remaining a significant concern. An efficient prefetching memory strategy must be found facing this memory wall.

B. Reconstruction by voxel blocks

Performing a reconstruction by voxel blocks, i.e., having an inner loop on neighbored voxels on algorithm 1, increases the spatial and temporal locality compared to voxel by voxel reconstruction. The projection of a block (B_x, B_y, B_z) corresponds to a rectangle shape (T_u, T_v) in the detector plane for a given projection angle φ_i as represented in Fig. 6. Inside this sinogram tile, a high data re-utilization exists during a voxel block reconstruction. Indeed, as illustrated in Fig. 5, a single ray may pass through several neighboring voxels in the 3D volume. Therefore, the re-utilization of projection data stored in BRAM can be exploited because all these voxels will read this single projection during the back-projection. Voxels in the same block will access the same sinogram tile for each projection angle. The main concern is to capture the projection data footprint without loss of information and calculate the coordinates of its boundary. For each voxel (x, y, z), its reconstruction depends on its N_φ angular projections. These projections are spatially distant due to their storage in the projection data following the order (u, v, φ).

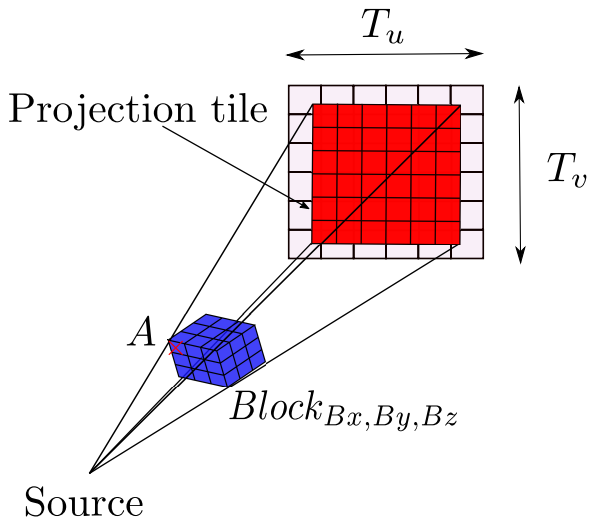

Fig. 6. Voxel block (in blue) and its projection (in red)

During a block reconstruction, prefetching in BRAM the sinogram tile associated with the voxel block allows a high reuse data rate for moderate costs in local memory for the storage of the sinogram tile required at the current φ iteration and the voxel block, but also in computation with the simple sinogram tile shape calculation. In order to identify the projection data footprint, we perform the following operations. The edge voxel A in the bloc is projected in the detector plane (see Fig. 6), and its coordinates (u_A, v_A) are computed. Then the dimension of the projection data footprint is determined by the computation of T_u and T_v depending on block size (B_x, B_y, B_z): $T_u = \sqrt{B_x^2 + B_y^2}$ and $T_v = \sqrt{2} * B_z$. The projection pixels in the located footprint (red rectangular shape in Fig. 6) are accessed with a regular memory access pattern. With this strategy, we gain in the temporal and spatial locality for memory accesses and consequently make these accesses contiguous for coalescence.

The use of FPGA BRAM provides higher bandwidth than the global memory bandwidth. Moreover, the access latency will be considerably reduced with a low stall percentage. This approach's advantage remains in reducing the memory access

to local memory. Because the block size and shape decision is crucial for the data reuse rate and contributes to the DRAM transfer reduction, an offline memory access analysis is presented in Section IV-A.

C. BP-Prefetch architecture

The BP-cache design is the baseline version of the 3D back-projection using burst-coalesced cached LSU (Algorithm 1) presented in [18]. OpenCL optimizations such as loop pipelining and unrolling were applied to this version to leverage the FPGA. The use of Intel automatic cache became a bottleneck when loop unrolling was applied to Algorithm 1. Loop unrolling consists of fully or partially replicating the loop body, and consequently increases the BRAM usage, thus preventing DSPs' maximum use. Usually, loop unrolling does not increase BRAM usage. However, when using Intel's automatic cache with an irregular access pattern, the offline compiler will have trouble coalescing memory accesses. This leads to the generation of a private cache for every global memory access. The cache is implemented using FPGA BRAM, hence several private copies may result in a waste of valuable on-chip memory blocks. In our new BP-Prefetch design, data locality is manually managed. Therefore, unrolling does not affect the BRAM usage critically.

Algorithm 2 Kernel OpenCL for BP-prefetch

```
1:  for all B_k do
2:    #pragma max_concurrency M_Tile
3:    for all φ_j do
4:      //Prefetching projection data
5:      Compute (u_A, v_A)
6:      for all m ∈ [0, T_v] do
7:        #pragma unroll W_prefetch/32
8:        for all n ∈ [0, T_u] do
9:          projections[n,m] = projections_global[u_A, v_A]
10:         u_A ← u_A + 1
11:       end for
12:       v_A ← v_A + 1
13:     end for
14:     //Computing back − projection
15:     #pragma unroll N_PE
16:     for all Voxel_i ∈ B_k do
17:       Compute (u, v)
18:       block_Bk[Voxel_i] += projections[u, v]
19:     end for
20:   end for
21:   volume ← block_Bk
22: end for
```

The architecture of the BP-prefetch design (Algorithm 2 written in pseudo-code[2]) is presented in Fig. 7. The main stages of the architecture are made of a prefetching module and a compute unit. The compute unit is local-memory related and is placed after the prefetching module. The compute unit is based on multiple Processing Element (PE) for parallelism. In the proposed architecture, the critical path consists of reconstructing the block of voxels with the innermost loop over the voxels. The loop body, considered as PE, can be replicated for parallel voxel intensity computation by loop unrolling with a factor N_{PE}. This number of PEs depends on the target FPGA available resources.

[2]The OpenCL kernel source code is fully available at https://github.com/nicolasgac/tomoGPI/ in file src/TomoGPI_lib/src/OCL/backprojection3D_kernel_A10.cl

To express fine-grained parallelism, a PE is designed as a pipeline for good efficiency. Each PE is responsible for the calculation of u, v, and voxel accumulation. The occupancy rate of the compute unit depends heavily on the ability of the prefetcher to provide the data. We inferred multiple copies of the prefetched data for concurrent computation and therefore increased the pipeline occupancy.

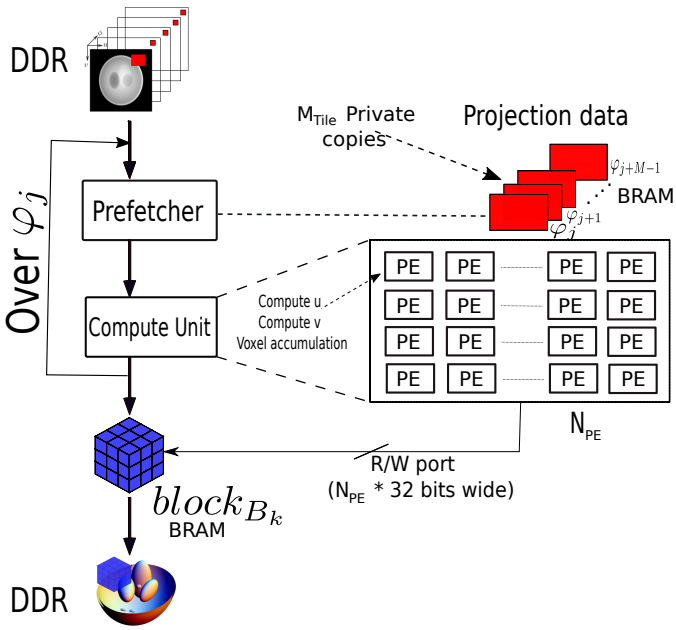

Fig. 7. BP-Prefetch pipeline. Modified from [27]

The N_{PE} PEs must have stall-free access to local memory when reading the projection data. We use memory replication as explained in Section II-C to ensure stall-free access. The memory replication amount depends on the number of PEs N_{PE} in the design. Depending on the resource available, we may also infer M_{Tile} private copies to lunch multiple iterations of loop φ.

The architecture's input is the projection data in the detector plane as shown in Fig. 7. For each projection angle φ_i, we prefetch all projection data required (red rectangle) for the voxels accumulation in the block. We load more data, from global memory, than required to ensure correct reconstruction and take advantage of memory coalescence. After the accumulation over all projections angles φ_i, the reconstructed block of voxels (blue cube) is written back to the volume stored in the global memory.

D. Memory prefetcher module

In the proposed design, all PEs are fed efficiently in data by a prefetcher module. As stated in Section III-B, prefetching the tiles corresponding to the projection of voxel blocks will allow high reuse rates of data. Thus this circuit-level design generated by the compiler from Algorithm 2 (lines 5 to 13) is loading projection tiles from the external global memory to the FPGA's on-chip memory for use by the PEs. The global access is a long latency operation, so we leverage the global bandwidth by making accesses in burst mode thanks to the offline memory analysis. When the kernel and the memory controller run at the same frequency, FPGAs external memory can perform 512-bit access per cycle per memory bank to saturate the bandwidth. With this in mind, the prefetcher module uses memory coalescence by performing multiple accesses simultaneously. Indeed,

the projection pixels to prefetch are located in the same tile, which allows multiple pixels to be merged into wider access from global memory. The number of accesses per cycle is chosen with caution because overusing the bandwidth could lead to worse f_{max} for the kernel. In order to preserve the kernel operating frequency, we perform the memory accesses with respect to the available bandwidth on the target FPGA device.

The FPGA boards used in this work are equipped with two DDR memory banks offering a memory bus 1024-bit wide. The memory access pattern is regular, thanks to our access strategy. The prefetcher module computes the edge coordinates of the projection data footprint based on our offline analysis to identify the data to be fetched, as presented in Section III-B. We use memory coalescence in order to fill the local memory with the projection data and maximize the bandwidth. Therefore, the prefetcher module provides wide access of $\frac{W_{prefetch}}{32}$ bits floating-point values per clock cycle. $W_{prefetch}$ corresponds to the bandwidth allocated to the prefetcher module, which is limited to 512 bits to let some bandwidth to the additional memory operations, such as the global volume updates.

The private copies as mentioned above in Section III-C inferred for concurrent execution helped to hide memory accesses latency in the pipeline, hence maximizing the design throughput.

E. Pipeline Initiation Interval (II)

A compute-intensive application optimization requires a specific focus on loops. The FPGAs excel in the loop pipelining for greater cyclic efficiency, and this is combined with loop unrolling to maximize throughput. The 3D back-projection algorithm has four nested loops zn, yn, xn ($Voxel_i$ coordinates) and φ (see Algorithm 1). The new method (Algorithm 2) allows us to have a specific tiling in our design thanks to the reconstruction by block. The computation of detector coordinates u and v for the image edge could prevent the compiler from achieving the best II for the innermost. The compiler assumes false dependency due to conditional branching scope. We move this computation out of the branching scope and use a temporal register to avoid this false dependency to address this problem. Therefore, the compiler can achieve an II value of one for all the loops in the design. Each PE has to compute the detector coordinates (u, v) and the voxel accumulation.

IV. Architecture Tuning

This section discusses the design scalability on Intel Arria 10 and Stratix 10 devices. In order to take advantage of our design, the offline memory access analysis is performed to choose the block size and shape with optimal data reuse rate. The study of the CI is also presented for the roofline model.

A. Offline Memory Access Analysis

The prefetched projection data depends on the shape of the block of voxels as described in Section III-B. The data reuse rate is computed by the following formula and illustrated in Fig. 8:

$$Data\ reuse = \frac{B_x * B_y * B_z}{\#\text{Memory access I/O}} \tag{10}$$

where the #Memory access I/O represents the number of actual projection pixels used and is obtained by a static analysis of the CPU code.

Memory accesses are no longer irregular thanks to our access strategy presented in Section III-A. The objective is to further reduce access to the external memory of the FPGA by exploiting the data reuse potential. Indeed, a single pixel in projection data is used by several voxels during the back-projection. The offline analysis first locates the footprint of the projection data required for a given voxel block and a given projection angle. Then, the algorithm is run on the CPU, and a counter is assigned to the projection data to identify the pixels that are used during the back-projection of the block for a projection angle. These pixel numbers are used in the calculation of the data reuse rate. This allows us to ignore some of the pixels present in the projection data footprint at the time of the memory accesses, especially those at the edges of the footprint, so as not to fall back into memory accesses with irregular strides. In order to evaluate the data reuse rate, the size of the voxel block and the number of pixels used in the projection data are required, as presented in (10).

The block size and shape decision depends on the BRAM consumption, the $\#Memory\ access\ I/O$, and the data reuse rate. A larger block may require a larger sinogram tile during back-projection. This large-size sinogram tile may consume several BRAM resources. Therefore, a trade-off between the data reuse rate and resource consumption must be done in order to avoid synthesis failure.

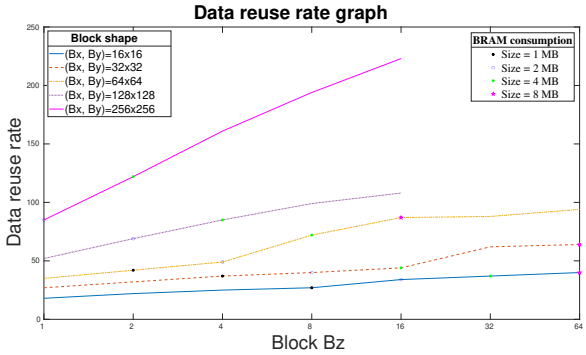

Fig. 8. Data reuse rate with 3 different shapes with B_z variation. Modified from [27]

The data reuse rate regarding different block shapes is illustrated in Fig. 8. The size of the BRAM needed to store the prefetched projection data is mentioned in order to choose to best (B_x, B_y, B_z) configuration. For a fixed number of voxels reconstructed, the requested memory size varies greatly: a (32,32,8) block requires twice as much as BRAM than a (64,64,2) block to store their projection data. This offline memory access analysis helps to manage the memory resource which is critical for this design. Indeed, with the replication and private copies overhead allowing concurrent execution, the BRAM sizes indicated in Fig. 8 have to be multiplied by the replication factor and the number of private copies. Assuming that we have three read ports per replicated memory and M_{Tile} private copies for concurrent execution, the required memory size will be multiplied by $\frac{N_{PE}}{3}$ (the replication factor) and M_{Tile} (number of copies). For instance for a $\frac{N_{PE}}{3} * M_{Tile} * 4$ MB available on BRAM, this offline study points that the highest data reuse rate is obtained for (256,256,2) blocks.

B. Offline Computational Intensity analysis

We use the roofline model to iteratively analyze our algorithm and guide the optimizations. Each algorithm results in a specific roofline for FPGAs. The performance roof is determined by the number of resources consumed and the effective operating

frequency. The dynamic profiler gives the effective (measured) DRAM bandwidth achieved. Table I lists the number of operations Giga Bytes Operations Per Second (GBOPS) and the number of global memory accesses of the 3D back-projection in GB. We determine the CI for different versions of the algorithm. The BP-Cache design is memory-bound with low CI (see Table. I) due to the lack of spatial and temporal data locality in the projection data. The CI of this version is very low, elaborating another strategy to access off-chip memory might substantially increase the CI and allow the use of more DSP slices to improve the performance.

TABLE I
STATIC ANALYSIS OF CI FOR DIFFERENT BLOCKS

Design	Block size	#Operations (GBOPS)	#Memory accesses (GB)	CI
BP-cache	N/A	236	17.2	13.7
BP-prefetch	$64 \times 64 \times 1$	253	1.1	236
	$32 \times 32 \times 16$	253	0.67	377
	$64 \times 64 \times 8$	253	0.6	419
	$128 \times 128 \times 4$	253	0.57	443
	$256 \times 256 \times 2$	253	0.55	456

We can see that CI increases as long as the block size grows until it reaches the maximum data reuse rate, and at the same time, the DRAM transactions are decreased. This variation is because of the data reuse potential of our memory access strategy. The total number of operations remains the same. In contrast, global memory invocations are considerably reduced. Therefore we perform the same number of computations for fewer access to the global memory. The CI of our design is improved by the new strategy and it is no longer in the memory-bound area on the roofline.

The memory operations presented in Table I take into account memory coalescence allowed by loop unrolling. The bytes operations in Table I include arithmetic operations (multiply-accumulate, divide...), and logic comparison performed in our design. Some computations are floating-point related, and other operations involve integer operations.

C. Design on Arria 10

The possible number of PEs on Arria 10 device is $N_{PE} = 64$ PEs in our architecture without exceeding available resources. Each PE must have free access to local memory, which requires a physical port for each memory read. In the case of insufficient ports, the memory requests are made with arbitration which causes a severe performance problem for the pipeline by increasing the II [32]. We perform memory replication for multiple accesses in order to avoid arbitration. Our architecture requires N_{PE} read-ports (32 bits wide) for all the PEs to read projection data in local memory. Thanks to the memory access strategy, global memory accesses are contiguous and coalesced when loading projection data. Hence, only one write-port is needed to write the projection data from global into BRAM memory. The total number of memory replications will depend on the number N_{PE} of PEs in the architecture. Each replicate has four ports due to double pumping (three reads and one write) and contains the same projection data for the block reconstruction. In this configuration, the total replication will be at least $\lfloor \frac{N_{PE}}{3} \rfloor = 22$

In addition, we place $M_{Tile} = 8$ private copies of the local memory for concurrent execution of loop iterations (φ loop in algorithm 2). This reduces the overall latency and increases the pipeline efficiency at the cost of additional BRAM consumption.

D. Replication potential on Stratix 10

We also use the Intel Stratix 10 device for our design in order to express more parallelism. The Stratix 10 is a high-end FPGA with thousands of DPS slices and more BRAM resources than the Arria 10 device. Consequently, our pipeline could benefit from more PEs and achieves better throughput. The replication factor N_{PE} of our architecture is fixed to 256 for the Stratix 10 devices. We have two ways of synthesizing this design: as a single kernel pipeline with all the PEs or multikernel model with several compute units running in parallel.

The single kernel represents the architecture presented in Fig. 7 with $N_{PE} = 256$ PEs. The throughput of this version will depend heavily on the ability of the prefetcher module to provide the data to the PEs. By having 256 PEs running in parallel, the number of physical read ports must increase, which leads to local memory replication overhead. In this scenario, all the PEs compute voxels in the same block at a time. The challenge to best leverage the pipeline is to feed it with data and maximize the throughput. The issue is that all 256 PEs are fed with data from the same prefetcher module. The higher number of PEs in the design makes the compute unit more efficient than the prefetcher module. This imbalance between the compute unit and the prefetcher module will result in degrading the pipeline efficiency. In order to balance between the compute unit and the prefetcher module, a multikernel approach has been proposed. We split the 256 PEs into several kernels, where each kernel has its own prefetcher module for more balance. The prefetcher module designed in Section III-D is able to keep 64 PEs sufficiently occupied to ensure high efficiency. However, increasing the number of PEs requires an increase in the capacity of the prefetcher module. Although, as explained in Section III-D, increasing the prefetcher module capacity strongly affects the kernel's operating frequency. We can have multiple prefetcher modules in the design without sacrificing the kernel frequency. Indeed, several prefetcher modules do not affect the kernel frequency because the compiler considers them separately and schedules their use of the memory bus in various clock cycles during the pipeline execution.

We replicate our SWI kernel four times to have multiple kernels in concurrent execution in order to further improve the design efficiency as illustrated in Fig. 9. Therefore we have the same pipeline architecture with four instances, and each kernel has 64 PEs. The kernel replication is quite identical to compute unit replication available in the NDR kernel. However, replicating the kernel consumes additional hardware resources because the whole kernel object is replicated. On Stratix 10, the multikernel model is not recommended as optimization, especially if there is a data exchange between the kernels. Nevertheless, if there are no dependencies between the kernels, the only concern is the resource consumption overhead. The available resources on the Stratix 10 device are enough to support this replication overhead. This is good for our design because the four kernels run concurrently on different portions of the 3D volume without any communication.

V. RESULTS

We evaluated in this section the performance of our design on Intel FPGA devices guided by the roofline analysis. The performance metrics such as the execution time, the pipeline stall, and

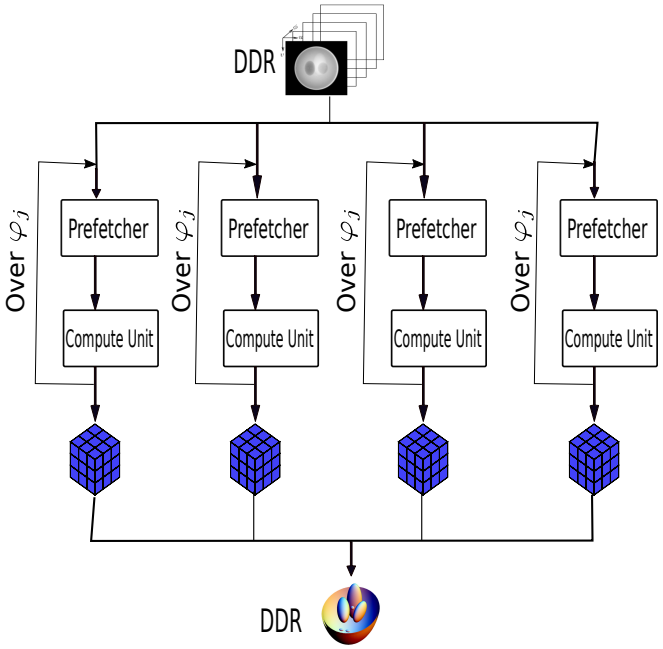

Fig. 9. BP-prefetch multikernel architecture

the occupancy are discussed. We also described the dataset used in this work and the software and hardware platforms. For the sake of reproducibility, the source code of this project is publicly available at https://github.com/nicolasgac/tomoGPI.

A. Experiment setup

1) Dataset: We used in this experiment the 3D Shepp-Logan phantom in a cone-beam X-ray CT system with 256×256 detector cells. The detector pixel size is 14.2 mm in each dimension, and the acquisition produced 256 projections distributed between 0 and 2π. The size of the considered volume is a 256^3 voxels, and the voxel size is $(x = 6.05\ mm, y = 6.05\ mm, z = 5.86\ mm)$. The Focus-Object Distance (FOD) is 98 mm, and the Focus-Detector Distance (FDD) is 230 mm.

2) Software: The Arria 10 device was part of the FLIK platform, which is a compact, all-in-one, and portable accelerator platform for laptops. The OpenCL release was the Intel FPGA SDK for OpenCL version 18.1.2.227 and the BSP 18.1 was used.

The OpenCL release of the Stratix 10 device was the Intel FPGA SDK for OpenCL version 19.1.0.240 with the BSP 19.1. For both devices, their host machine was based on Linux OS.

Each kernel execution is monitored through the Intel FPGA dynamic Profiler for OpenCL. For each kernel, this tool provides, amongst other things, the operating frequency, the execution time, the logic utilization, and the latency, bandwidth, and stall of most memory access.

TABLE II
PLATFORMS USED IN THE EXPERIMENT

Board	Compute resources	Freq (MHz)	External Memory	TFLOPS
CPU i7-3820	-	3600	DDR3	-
GPU Jetson TX2	256 Cuda cores	1465	DDR4	0.75
GPU A100	6912 Cuda cores	1410	HBM2	19.5
FLIK Arria 10	1518 DSPs	480	DDR4	1.45
DE10-Pro Stratix 10	5760 DSPs	800	DDR4	9.2

3) Hardware devices: We used the FLIK and the DE10-Pro boards for this experiment as shown in Table II. The FLIK Arria 10 GX FPGA (10AX115N2F45E1SG), with 1150K logic elements, comes with 8 GB of DDR4-2133 memory, with a maximum frequency of 480 MHz. The FPGA is connected in a PCIe connection (via Thunderbolt 3) to the host system. The FPGA was connected to the host via thunderbolt 3 because the FLIK card is a compact, all-in-one, and portable accelerator platform for laptops. The DE10-Pro board is based on Intel Stratix 10 GX (1SG280HU2F50E2VG) with 32GB arranged in 2 banks of DDR memory. The Stratix 10 device is a high-end FPGA with 5760 DSP slices.

In this experiment we do not consider the data transfer between the host and the device; therefore the considered runtimes do not include memory transfer. However, to speed up the memory transfer, the allocated data must be aligned at least 64 bytes to allow for Direct Memory Access (DMA) transfer. To allocate an aligned memory, the POSIX function *posix_memalign* can be used by the host. In our experience, the aligned memory achieves a better transfer rate than the non-aligned memory in all cases. The AOC compiler allows designers to specify a seed value to relax the routing constraints or solve the timing violations due to the design complexity. After validating the architecture on Stratix 10, we perform a seed sweep to choose the best design configuration that reaches the maximum frequency. The seed sweep permits determining an optimal seed value for a design without any change in the initial pipeline characteristic.

B. Arria 10 results and roofline

Table III shows the results of our implementation using the Arria 10 device. The performance metrics such as operating frequency, execution time, stall percentage, and occupancy are presented in this table. The stall percentage represents the amount of time memory access causes pipeline stalls. In contrast, occupancy represents the percentage of time when a work item (thread) performs a valid memory instruction. The more the occupancy is, the more the compute units are active during the execution. The BP-cache version suffers from a high pipeline stall percentage because of the memory access pattern, making the global bandwidth the main bottleneck. The execution time of this version is not acceptable for CT reconstruction routine. The DSP usage was very low because of high BRAM usage to implement the cache. Therefore, extra compute unit replication was impossible, and the BP-cache design could not achieve higher throughput.

The BP-Prefetch design (Algorithm 2) achieved better performance compared to the BP-Cache version. It overcomes the abovementioned issue by providing a good design with an acceptable memory access pattern. The results show the effect of block shape and size variation. We saw in Fig. 8 that the reuse rate varies with the number of voxels in the block, i.e., high block size results in a high reuse rate. However, the BRAM resource consumption should be considered concerning the available resources on the target FPGA. A block size that consumes at most 4 MB has been used to store the projection data for our architecture configuration, and the data reuse rate is high enough for this amount of memory. The results of our architecture on Arria 10 are presented in Table III for different block sizes and shapes. The stall percentage is very low for all the chosen blocks, and the occupancy is high to ensure good execution time. The change in the data reuse rate is noticed in the increase in the occupancy of the architecture.

TABLE III
BLOCK SIZE VARIATION EFFECT ON THE DESIGN PERFORMANCE ON ARRIA 10

Design		BRAM (%)	DSP %	Stall (%)	Occ (%)	Freq (Mhz)	Time (s)
BP-Cache		72	27	71.27	24.6	150	3.65
BP-Prefetch	$32^2 \times 16$	73	58	0.2	74.7	179.2	0.502
	$64^2 \times 8$	72	63	0.06	84.1	189	0.425
	$128^2 \times 4$	68	63	0.56	90.2	176	0.423
	$256^2 \times 2$	73	62	0.06	94	180	0.396

All loops are successfully pipelined with an II value of 1 in the BP-Prefetch design. There is no memory access conflict, and the data dependencies are handled in one clock cycle. We tune the block sizes and shapes to have the best performance possible for our architecture. We have obtained a better execution time with the $256 \times 256 \times 2$ block, corroborating the static study performed on the data reuse. Compared to the previous BP-cache design, we achieved a speedup of 9.2× at 180 MHz.

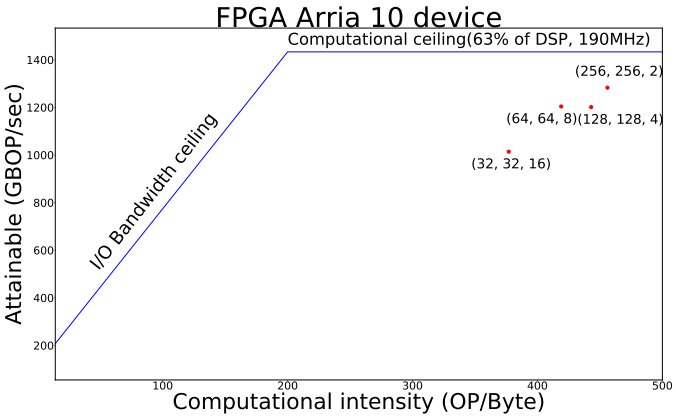

Fig. 10. Roofline of BP-Prefetch with different blocks on Arria 10

The roofline plots are presented in Fig. 10 for four different blocks using the Arria 10 device. The chosen blocks are the ones that provided a high reuse rate, therefore the shapes and sizes are different. These high reuse rates will result in high CI moving the algorithm out of the memory-bound area. Since the algorithm is no longer in the memory-bound zone, the challenge is to get as close as possible to peak performance, i.e., the computational ceiling. This can be problematic because even if the application is not limited by memory on the roofline, it can be prevented from having the maximum performance because of the impact of the memory access latency. Unfortunately, the effect of these memory latencies is not directly visible on the roofline. For these different optimization techniques are used such as the access latencies hiding or the maximization of concurrent executions to occupy the compute units. As shown in the roofline, by increasing the size of our blocks, the CI also increases, allowing us to have a better occupancy rate and get close to the computational ceiling. The computational ceiling of our architecture is not affected by the algorithm's CI because our block size does not affect our design's compute configurations. The BRAM usage is concerned by the block size variation while the DSP consumption is slightly impacted. A 100% occupancy rate would be on the horizontal line of our computational ceiling. However, our best design on Arria 10 has an occupancy of 94% which is close to the computational ceiling.

C. Scalability to Stratix 10

Our design is then implemented and evaluated on Intel Stratix 10 device as a single kernel and multikernel. In the multikernel version, we replicated this design with the same block size on our Stratix 10. Therefore we have four compute units working on four blocks of voxels concurrently. The BSP ensures the OpenCL device bandwidth. In practice, the effective design bandwidth is low compared to the BSP's maximum capability. The bandwidth follow-up is primordial for an acceleration equivalent to the parallelism factor (the ideal speedup). We use the bandwidth to its full capacity without sacrificing operating frequency.

TABLE IV
SINGLE KERNEL VERSUS MULTIKERNEL ON STRATIX 10

Design	Block	BRAM (%)	DSP (%)	Stall (%)	Occ (%)	Freq (MHz)	Time (s)
Single kernel	$64^2 \times 8$	50	49	8.28	61.5	127	0.32
	$128^2 \times 4$	31	50	4.01	60	87	0.47
	$256^2 \times 2$	40	57	3.89	60.2	117.2	0.24
Multikernel	$64^2 \times 8$	33	57	13.91	82.3	172.5	0.12
	$128^2 \times 4$	36	57	1.45	83.1	133	0.84
	$256^2 \times 2$	40	57	2.28	85.1	127	0.51

The single pipeline version contains 256 PEs working on the same block of voxels at the time. Despite the concurrent execution of the loops, the pipeline occupancy rate was not optimal. The prefetcher module could not efficiently feed the pipeline for better throughput. Furthermore, the operating frequency of this version was slightly low, as shown in Table IV due to the complexity of the single compute unit. Compared to the Arria 10 implementation, the Stratix 10 results were not as good as expected, even though the design used a high number of PEs for a single kernel. However, the block $256 \times 256 \times 2$ provided the best execution time with a 1.6× speedup using 4× as much PEs as the Arria 10 design.

The multikernel configuration allows the design to achieve a better operating frequency. This improved frequency enables further saturation of the global memory bandwidth and the occupancy rate of each kernel. The routing constraints are relaxed with this strategy of splitting the compute unit. Another main difference between the two designs is that the multikernel version runs on four blocks in parallel. The volume was divided into four subvolumes reconstructed by the four compute units. The reconstruction is performed block by block within each subvolume. Therefore the multikernel version is preferred over the single kernel, which achieves the best operating frequency and the optimal design occupancy. Despite the high occupancy for blocks $256 \times 256 \times 2$ and $128 \times 128 \times 4$, their execution times are worse than the single kernel version. This is due to a prediction mechanism implemented by the AOC compiler on FPGAs. The computations of the projection data coordinates are performed on the fly, and the hardware tries to predict these computations to optimize the design. By analyzing the dynamic profiler, we notice that the percentage of prediction hit was very low for these two blocks, which means most of the computations performed were invalid. The success of the prediction rate was given by *activity* metric in the dynamic profiler. The best execution time was obtained for the block $64 \times 64 \times 8$ with high occupancy and a good prediction rate. It should be noted that the prediction failed when the number of PEs is lower than the number of voxels per line in the block. This is the reason we do not have the failed prediction for a single kernel because the number of PEs (256) is equal to or greater than the voxels per line of the block (64,

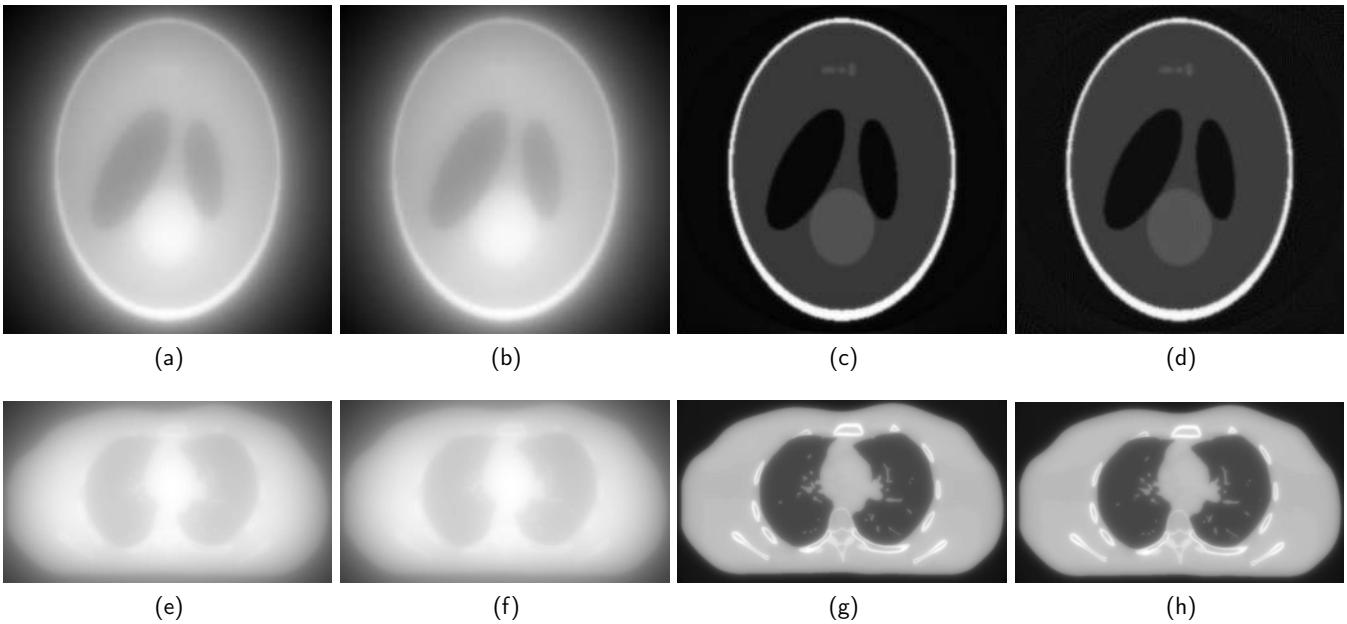

Fig. 11. Shepp-Logan and XCAT phantoms reconstructed image slice 128. The Shepp-Logan images are presented at the top and the XCAT images are at the bottom. Fig. 11a-11b and 11e-11f: The reconstructed image using single back-projection. Fig. 11c-11d and 11g-11h: The reconstructed image after the full MBIR algorithm. Fig. 11a, 11c, 11e, and 11g: Reconstructed by GPU. Fig. 11b, 11d, 11f, and 11h: Reconstructed by FPGA.

128, or 256). For multikernel we had 64 PEs, and for blocks with more than 64 voxels per line, the predictions failed several times. Therefore, a good consistency is required between the chosen block and the number of PEs.

Thus, the multikernel has achieved an execution time of 0.12 s at 172.5 MHz and preserved the pipeline occupancy for each compute unit with a block size of $64 \times 64 \times 8$. However, the stall percentage of the multikernel version is slightly high because we have multiple kernels with multiple prefetcher modules soliciting the global bandwidth. Nevertheless, this increase in memory stall does not impact the overall performance that much, and yet the design achieved significant throughput.

D. Image Accuracy

TABLE V
IMAGE QUALITY EVALUATION BETWEEN GPU AND FPGA RECONSTRUCTED IMAGES ON DIFFERENT DATASETS

Dataset	Algorithm	UQI	CC	NRMSE	SNR
Shepp-Logan	Back-projection	0.999	0.999	0.0162	37.5
	MBIR (100 iterations)	0.996	0.996	0.099	25.9
XCAT	Back-projection	0.999	0.999	0.0659	35.4
	MBIR (100 iterations)	0.999	0.999	0.0457	37.9

We use Shepp-Logan phantom and XCAT phantom [34] test cases to evaluate our design. Slices of the 256^3 volume of Shepp-Logan phantom and $1024^2 \times 256$ on XCAT phantom are illustrated in Fig. 11, reconstructed by GPU and FPGA using single-precision floating-point representation. We used the nearest neighbor method for interpolation on FPGA, while the GPU version used the bi-linear interpolation method. Table V shows the evaluation of the FPGA reconstructed images with respect to GPU ones. The Universal Quality Index (UQI) [35], the Correlation Coefficient (CC), the Normalized Root Mean Squared Error (NRMSE), and the Signal to Noise Ratio (SNR) metrics have been used in order to evaluate our design accuracy compared to the GPU reconstructed image. The FPGA reconstructed image has NRMSE = 0.099, SNR = 25.9, CC = 0.996, and UQI = 0.996 after the full IR algorithm on the Shepp-Logan phantom. We obtained for XCAT phantom NRMSE = 0.045, SNR = 37.9, CC = 0.999, and UQI = 0.999, which are accurate enough for CT application.

VI. COMPARISON AND DISCUSSION

We compare the results of our design with the state-of-the-art tomography implementations on GPU and FPGA. In this comparison, the overall throughput, design efficiency, power consumption, and resource consumption are assessed.

A. Performance comparison

To fairly compare different implementations with different problem sizes, we use the Giga Updates Per Second (GUPS) indicator, which is unaffected by the size of the problem, using the formula given in [36]:

$$GUPS = \frac{GU}{Time_{kernel}} \quad with \quad GU = \frac{N_{voxel} * N_{acc/voxel}}{1024^3} \tag{11}$$

with N_{voxel} the size of volume and $N_{acc/voxel}$ the number of accumulations per voxel. The GUPS in [36] uses 1024^3 instead of 1000^3, so we use calculate all GUPS according to this formula.

Several works on CT algorithm acceleration on hardware architectures have been reported. These different works use different approaches such as ray-driven, voxel-driven, or separable footprint. Choi *et al.* [21] has proposed a ray-driven voxel-tile parallel approach to accelerate the EM algorithm on FPGA. They exploit the data reuse to leverage the BRAM memory by minimizing global memory access and expressing high parallelism. Gac *et al.* [12] has proposed a pipelined architecture exploiting the spatial and temporal locality of the back-projection algorithm. Their

TABLE VI
PERFORMANCE COMPARISON OF OUR WORK AND OTHER WORKS.

Ref.	Back-projector[a] (geometry, volume)	Arch.	Platform (Year, Freq., Process size)	Time (s)	GUPS	Update /cycle /op[b]	PE	Cycle /update /PE
This work	VD-float with 256 updates/voxel (CT circular, 256^3)	**FPGA**	Arria 10 (2014, 189 Mhz, 20 nm)	0.396	10.1	0.030	64	1.12
			Stratix 10 (2016, 172 Mhz, 14 nm)	0.12	33.3	0.032	256	1.23
		GPU	C2050 (2011, 1.15 Ghz, 40 nm)	0.129	31.1	0.032	448	15.4
			P100 (2016, 1.46 Ghz, 16 nm)	0.017	237	0.027	3584	18.7
			V100 (2017, 1.38 Ghz, 12 nm)	0.011	364	0.028	5120	18.1
			A100 (2020, 1.41 Ghz, 7 nm)	0.009	424	0.023	6912	21.4
			TX2 (2016, 1.46 Ghz, 16 nm)	0.25	16.0	0.023	256	21.8
Chou [36]	VD-float with 360 updates/voxel (CT circular, 512^3)	**GPU**	C1060 (2008, 1.30 Ghz, 55 nm)	2.47	18.2	0.031	240	15.9
Gac [12]	VD-fix with 480 updates/voxel (PET, $128^2 \times 63$)	**FPGA**	Virtex-4 (2004, 200 Mhz, 90 nm)	0.526	0.88	0.025	8	1.70
Choi [21]	RD-fix with 831 updates/voxel (CT helical, $512^2 \times 372$)		Virtex-6 (2009, 100 Mhz, 40 nm)	14.8	5.1	0.052	64	1.17
Wen [20]	RD-fix with 502 updates/voxel (CT helical, $1024^2 \times 128$)		ZCU102 (2018, 300 Mhz, 16 nm)	2.10	29.9	0.024	-	-

[a] *The back-projector studied in our work is Voxel-Driven with floating-point computation (VD-float). Back-projectors studied by Choi [21] and Wen [20] are Ray-Driven with fixed-point computation (RD-fix).*
[b] *op. : hardware operators (adders and multipliers). GPU cores have two FP32 (Floating-point on 32 bits) op. with one multiplier accumulator; DSP Intel FPGA has two FP32 op. with one multiplier accumulator; DSP Xilinx has three fixed-point op. with a pre-adder and a multiplier accumulator.*

method overcomes the memory bottleneck by loop reordering to take advantage of cache memory using HDL language. FPGA designs with HDL, as in [12], are known to provide efficient pipeline architecture, and this same efficiency can also be seen when using HLS tools. An asynchronous beam-based parallelism to accelerate the Mumford-Shah (MS) algorithm with a high data reuse rate and low external memory transactions has been proposed by Zhang *et al.* [22]. Their approach reduces the computational cost of the backward projection to a lightweight operation. Wen *et al.* [20] has proposed a data management strategy to achieve a best reuse rate and exploit the FPGA on-chip memory to accelerate the forward and backward projections. A forward projection parallel architecture on FPGA has been proposed by Kim *et al.* [15]. Their projector architecture was fully pipelined and exploited loop-level parallelism for high performance. In [36], the authors proposed a GPU-based reconstruction algorithm to accelerate the forward and backward projections. Their approach takes advantage of the GPU architectures to perform multiple rays computation with multiple threads while avoiding thread divergence.

Table VI shows that our embedded GPU implementation based on [12] achieved slightly better performance than the Arria 10 design in terms of GUPS. Vivado HLS is commonly used as HLS tool for FPGA accelerations as in [20], [21], although we use Intel FPGA SDK for OpenCL in this work. OpenCL SDK is at a somewhat higher level of abstraction than Vivado HLS, giving the designer more control over the pipeline by using HLS compilers such as Vivado HLS or Intel HLS compiler. Choi *et al.* [21] used the Convey HC-1ex platform based on helical geometry. This platform runs at 100 Mhz of operating frequency with four FPGAs, and the authors' design consumes 1408 DSP slices. Wen *et al.* [20] targeted the Xilinx ZCU102 platform based on an UltraScale FPGA and with an overall DSP utilization of 1476 at 299.97 Mhz.

Our Stratix 10 design with four compute units and a total of 256 PEs achieved an overall throughput of 33.3 GUPS at 172.5 Mhz. We compared the results to our GPU implementation using an embedded GPU and a powerful desktop GPU. The Stratix 10 design outperforms our embedded GPU implementation on the same dataset. However, the GPU A100 was the most efficient platform regarding throughput. The DSP usage of the four kernels is 3282 slices. The results show better performance compared to all the implementations in Table VI. Our design achieves a 6.5× speedup of throughput in terms of GUPS compared to the back-projector of Choi *et al.*. The Convey HC-1ex platform of Choi *et al.* [21] contains four FPGAs. However, The results reported here are for one FPGA in order to make a fair comparison since we use a single FPGA platform. However, their back-projector is a ray-driven approach, and their PE is responsible for ray tracing. Our PE performs a voxel update since the back-projector used in this work is a voxel-driven approach. Moreover, each ray traverses an average of 168 voxels in our dataset, while it traverses 1004 voxels in their dataset. They have a higher potential for data re-utilization in their dataset than ours. Nevertheless, our design exploits more parallelism and concurrent computations to achieve high throughput.

We then evaluated the design efficiency of all the FPGA implementations by comparing the number of updates performed per cycle by each hardware operator (adder and multiplier). Our OpenCL implementation on Arria 10 and Stratix has approximately the same design efficiency as the HLS ones (Table VI). It should be noted that the work on FPGA related in the literature used fixed-point representation to perform the reconstruction while we use floating-point single-precision for our design. Potentially, performances could be improved by a factor of two if Intel DSP is used as two fixed-point Multiplier ACcumulator (MAC) instead of one floating-point MAC.

Finally, we have evaluated the pipeline efficiency of each PE. Our FPGA architectures on Arria 10 and Stratix 10, Gac *et al.* [12] HDL design and Choi *et al.* [21] have an efficiency close to the optimal of one cycle per update per PE. Conversely, GPU cores have an efficiency of around 20 cycles per update. It highlights the strength of FPGA technology which allows the design of customized architectures with a high computation efficiency even if its lower frequency clock and hardware resources density make it, at the end, slower than GPU.

B. Resource consumption

Table VII shows the resource usage of the $64 \times 64 \times 8$ block version. As mentioned above, our design on Arria 10 contains 64 PEs on Arria 10 device and 256 PEs on Stratix 10. The BRAM usage also includes the memory replication overhead in order to support concurrent access within the pipeline. The resources that are consumed by the design on Stratix 10 are shown in Table VII for our four compute units. The table reveals that the extra logic consumption is due to the kernel replication overhead in terms of LUT and DSP. It should be noted that the LUTs are larger in Xilinx devices (6-Input) than in Intel devices (4-Input). Therefore it is the percentage of usage that matters.

TABLE VII
FPGA RESOURCES CONSUMPTION

Reference	FF	LUT	BRAM	DSP
Ours Arria 10	407183 (25%)	184616 (23%)	1967 (73%)	949 (63%)
Ours Stratix 10	1338708 (36%)	604963 (32%)	3898 (33%)	3282 (57%)
Choi *et al.* [21]	1263716 (33%)	1142380 (60%)	3680 (64%)	1408 (40%)
Wen *et al.* [20]	200062 (36%)	235928 (86%)	1352 (74%)	1476 (58%)

Our design consumed more DSP slices than other FPGA designs due to the high level of parallelism exploited by our method. The complexity of our pipeline is not identical to the ray-tracing PE for other works because our approach is based on voxel-driven. Moreover, this work uses single-precision floating-point numbers for the volume image and the projection data, while other works [20], [21] use fixed-point values. Floating-point computations are more expensive than fixed-points regarding latency and hardware resources.

C. Power consumption

Table VIII shows the power consumption of our design on different architectures. Intel Arria 10 and Stratix 10 boards contain a power sensor that reads the actual power consumed by the board using a software API provided by the vendor. We used the Intel power monitor to read the sensor and measure the power consumption of the FLIK Arria 10 board. The *fpgainfo* tool from the Open Programmable Acceleration Engine (OPAE) C toolkit was used to measure the power sensor on Intel Stratix 10 board. Therefore, The power values reported in Table VIII are the actual power consumption of the FPGA board. The reported values correspond to the maximum Thermal Design Power (TDP) of 250W for the GPU boards.

TABLE VIII
DESIGN ENERGY EFFICIENCY

Device	Process (nm)	Power(W)	Time (s)	GUPS/Watt
GPU P100	16	250	0.017	0.95
GPU V100	12	250	0.011	1.45
GPU A100	7	250	0.009	1.70
Jetson TX2	16	12.9	0.25	1.22
Arria 10	20	14.9	0.396	0.68
Stratix 10	14	23.5	0.12	1.42

The embedded Jetson is more energy-efficient than our architecture on Arria 10 FPGA, which was designed using OpenCL for the back-projection algorithm. Conversely, the Stratix device is more efficient than Arria 10 and the embedded GPU. We can say that the Stratix 10 is as energy-efficient as the GPUs except for the A100 device. However, the difference in the process size between Stratix 10 and A100 devices (14 nm versus 7 nm) is quite significant. The FPGA can potentially become more advantageous than GPUs because there is still room to reduce the process size. By exploiting this technological gap, FPGAs can benefit from more computing resources and be more energy efficient.

VII. Conclusion

This paper presents a hardware acceleration of the back-projection algorithm for CT reconstruction using FPGA local memory efficiently. A fully OpenCL-based custom pipeline architecture using memory prefetching to reduce global memory transactions is designed to accelerate the algorithm. The prefetching of the projection data into the FPGA local memory, allowed by the offline memory access analysis, permits us to leverage the global memory bandwidth. Our architecture performs better throughput based on an efficient pipeline with no stall on Intel FPGA devices for the back-projection algorithm. Furthermore, we present a multikernel approach to maximize the Stratix 10 design throughput and operating frequency. We have ensured the bandwidth follow-up by the prefetcher module to keep occupying the compute units as much as possible. The systematic use of the Berkeley roofline model highlighted the optimization steps in our development. We achieved 0.6× and 2.1× throughput against our embedded GPU implementation on Arria 10 and Stratix 10, respectively. FPGAs could be more competitive against GPUs by reducing the process size and gaining computational power and energy efficiency. Indeed, experimental results show that the FPGA-based OpenCL design is as efficient as the HDL design using the same algorithm with reduced development time. Our design with OpenCL outperforms the related CT reconstructions using Vivado HLS on FPGA. The proposed method can be applied to the ray-driven projector for a full iterative reconstruction routine. We can exploit the data reuse potential between multiple rays computation, which is part of our future work.

References

[1] J.-B. Thibault, K. D. Sauer, C. A. Bouman, and J. Hsieh, "A three-dimensional statistical approach to improved image quality for multislice helical CT," vol. 34, no. 11, pp. 4526–4544, 2007. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1118/1.2789499

[2] F. Xu and K. Mueller, "Real-time 3d computed tomographic reconstruction using commodity graphics hardware," vol. 52, no. 12, pp. 3405–3419, 2007. [Online]. Available: https://iopscience.iop.org/article/10.1088/0031-9155/52/12/006

[3] F. B. Muslim, L. Ma, M. Roozmeh, and L. Lavagno, "Efficient FPGA implementation of OpenCL high-performance computing applications via high-level synthesis," in *IEEE Access*, vol. 5, 2017, pp. 2747–2762.

[4] H. R. Zohouri, N. Maruyama, A. Smith, M. Matsuda, and S. Matsuoka, "Evaluating and optimizing OpenCL kernels for high performance computing with FPGAs," in *SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2016, pp. 409–420, ISSN: 2167-4337.

[5] M. A. Mansoori and M. R. Casu, "Efficient FPGA implementation of PCA algorithm for large data using hls," in *PRIME Conference*, 2019.

[6] E. Nurvitadhi, G. Venkatesh, J. Sim, D. Marr, R. Huang, J. Ong Gee Hock, Y. T. Liew, K. Srivatsan, D. Moss, S. Subhaschandra, and G. Boudoukh, "Can FPGAs beat GPUs in accelerating next-generation deep neural networks?" in *Proceedings of the 2017 ACM/SIGDA International Symposium*

on Field-Programmable Gate Arrays, ser. FPGA '17. Association for Computing Machinery, 2017, pp. 5–14. [Online]. Available: https://doi.org/10.1145/3020078.3021740
[7] S. Zhang, Y. Wu, C. Men, H. He, and K. Liang, "Research on OpenCL optimization for FPGA deep learning application," in *PLoS ONE*, vol. 14, no. 10, 2019. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6786543/
[8] Intel Corporation, "Intel stratix 10 device," https://www.intel.fr/content/www/fr/fr/products/programmable/fpga/stratix-10.html.
[9] "Intel agilex device," https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/wp/intel-agilex-fpgas-deliver-game-changing-combination-wp.pdf.
[10] Xilinx Corporation, "Vivado design suite - vivadohls," http://www.xilinx.com/products/design-tools/vivado/index.htm.
[11] Intel Corporation, "Intel high level synthesis compiler," https://www.intel.fr/content/www/fr/fr/software/programmable/quartus-prime/hls-compiler.html.
[12] N. Gac, S. Mancini, M. Desvignes, and D. Houzet, "High speed 3D tomography on CPU, GPU, and FPGA," *EURASIP Emb Sys*, 2008.
[13] F. Pfanner, M. Knaup, and M. Kachelriess, "High performance parallel backprojection on FPGA," in *11th international meeting on Fully three-dimensional image reconstruction in radiology and nuclear medicine*, 2011. [Online]. Available: https://www.osti.gov/etdeweb/biblio/22124816
[14] S. Coric, M. Leeser, E. Miller, and M. Trepanier, "Parallel-beam backprojection: an FPGA implementation optimized for medical imaging," in *Proceedings of the 2002 ACM/SIGDA tenth international symposium on Field-programmable gate arrays*, ser. FPGA '02. New York, NY, USA: Association for Computing Machinery, Feb. 2002, pp. 217–226. [Online]. Available: https://doi.org/10.1145/503048.503080
[15] J. K. Kim, J. A. Fessler, and Z. Zhang, "Forward-projection architecture for fast iterative image reconstruction in CT," *IEEE Trans Sign Process*, 2012.
[16] A. Cilardo, "Evaluating reconfigurable hardware for accelerating industrial CT," in *2020 IEEE 7th International Conference on Industrial Engineering and Applications (ICIEA)*, 2020, pp. 93–97.
[17] M. Ravi, A. Sewa, T. G. Shashidhar, and S. S. S. Sanagapati, "FPGA as a hardware accelerator for computation intensive mlem medical image reconstruction," *IEEE Access*, 2019.
[18] D. Diakite, M. Martelli, and N. Gac, "An OpenCL pipeline implementation on intel FPGA for 3D backprojection," in *6th International Conference on Image Formation in X-Ray Computed Tomography*, 2020. [Online]. Available: https://hal-centralesupelec.archives-ouvertes.fr/hal-02500994
[19] M. Martelli, N. Gac, A. Mérigot, and C. Enderli, "3D Tomography back-projection parallelization on Intel FPGAs using OpenCL," *Journal of Signal Processing Systems*, 2018. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01831884
[20] S. Wen and G. Luo, "FPGA-accelerated automatic alignment for three-dimensional tomography," in *2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2020, pp. 172–176, ISSN: 2576-2621.
[21] Y.-k. Choi and J. Cong, "Acceleration of EM-based 3D CT reconstruction using FPGA," in *IEEE Transactions on Biomedical Circuits and Systems*, vol. 10, no. 3, 2016, pp. 754–767.
[22] W. Zhang, L. Qiao, W. Hsu, Y. Cui, M. Jiang, and G. Luo, "FPGA acceleration for 3D low-dose tomographic reconstruction," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2020, pp. 1–1.
[23] L. Qiao, G. Luo, W. Zhang, and M. Jiang, "FPGA-accelerated iterative reconstruction for transmission electron tomography," in *2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2021, pp. 152–156, ISSN: 2576-2621.
[24] "Intel fpga sdk for opencl pro edition: Programming guide 2019," .
[25] M. A. Dávila-Guzmán, R. G. Tejero, M. Villarroya-Gaudó, and D. S. Gracia, "Analytical model of memory-bound applications compiled with high level synthesis," in *2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2020. [Online]. Available: http://arxiv.org/abs/2003.13054
[26] S. Williams, A. Waterman, and D. Patterson, "Roofline: an insightful visual performance model for multicore architectures," in *Communications of the ACM*, vol. 52, no. 4, 2009, pp. 65–76. [Online]. Available: https://dl.acm.org/doi/10.1145/1498765.1498785
[27] D. Diakite, N. Gac, and M. Martelli, "OpenCL FPGA optimization guided by memory accesses and roofline model analysis applied to tomography acceleration," in *2021 31st International Conference on Field-Programmable Logic and Applications (FPL)*, pp. 109–114, ISSN: 1946-1488.
[28] L. A. Feldkamp, L. C. Davis, and J. W. Kress, "Practical cone-beam algorithm," *Josa a*, vol. 1, no. 6, pp. 612–619, 1984.
[29] K. O. W. Group, "The opencl specification: Version 1.2," https://www.khronos.org/registry/OpenCL/specs/opencl-1.2.pdf.
[30] Y. Uguen, F. De Dinechin, V. Lezaud, and S. Derrien, "Application-specific arithmetic in high-level synthesis tools," in *ACM Transactions on Architecture and Code Optimization*, 2019. [Online]. Available: https://hal.archives-ouvertes.fr/hal-02423363
[31] K. Shata, M. K. Elteir, and A. A. EL-Zoghabi, "Optimized implementation of OpenCL kernels on FPGAs," in *Journal of Systems Architecture*, vol. 97, 2019, pp. 491–505. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1383762118303151
[32] "Intel FPGA SDK for OpenCL pro edition: Best practices guide," p. 193.
[33] B. da Silva, A. Braeken, E. H. D'Hollander, and A. Touhafi, "Performance modeling for FPGAs: Extending the roofline model with high-level synthesis tools," in *International Journal of Reconfigurable Computing*, 2013, Research Article. [Online]. Available: https://www.hindawi.com/journals/ijrc/2013/428078/
[34] W. P. Segars, M. Mahesh, T. J. Beck, E. C. Frey, and B. M. Tsui, "Realistic ct simulation using the 4d xcat phantom," *Medical physics*, vol. 35, no. 8, pp. 3800–3808, 2008.
[35] Z. Wang and A. C. Bovik, "A universal image quality index," *IEEE signal processing letters*, vol. 9, no. 3, pp. 81–84, 2002.
[36] C.-Y. Chou and Y.-Y. Chuo, "A fast forward projection using multithreads for multirays on GPUs in medical image reconstruction," *Journal of Medical Physics Research and Practice*, 2011.

Daouda Diakite Daouda Diakite received the M.S. degree (2019) in embedded systems and information processing and the Ph.D. degree (2022) in Signal and Image Processing Sciences both from Université Paris-Saclay. His Ph.D. work at the L2S focused on algorithm-architecture co-design and hardware acceleration on FPGAs using high-level synthesis tools. He is currently an R&D consultant in high-performance computing in a private company.

Nicolas Gac After an M.Sc. (2003) and a Ph.D. (2008) degrees, both in signal processing, from the Institut National Polytechnique de Grenoble, Nicolas Gac has joined in 2009 the L2S (Laboratory of Signal and Systems) where is currently an associate professor at University of Paris-Saclay. At the Inverse Problems Group, his research interest is focused on algorithm-architecture co-design applied mainly to tomography reconstruction and radio astronomy on multi-GPU servers or FPGA boards.