INTERNATIONAL JOURNAL OF BIOLOGY AND BIOMEDICAL ENGINEERING
DOI: 10.46300/91011.2021.15.33 Volume 15, 2021

# Importance of Some Specifications of Heterogeneous Architectures (CPU+GPU) for 3D Cone-Beam-CT Image Reconstruction Using OpenCL

T. Nouioua[1,2], A. H. Belbachir[2],
[1]Department of Mathematics and Computer Sciences,
Faculty of Exact sciences and sciences of Nature and life,
University of Tebessa,
Tebessa, Algeria
[2]Laboratory of Analysis and Application of Radiation (LAAR),
Engineering Physics Department, Faculty of Physics,
University of Sciences and Technology of Oran Mohamed Boudiaf,
Oran, Algeria

Received: January 22, 2021. Revised: June 21, 2021. Accepted: July 15, 2021. Published: July 20, 2021.

**Abstract— Medical imaging has found an important way for routine daily practice using cone-beam computed tomography to reconstruct a 3D volume image using the Feldkamp-Davis-Kress (FDK) algorithm. This way can minimize the patient's time exposure to X-rays. However, its implementation is very costly in computation time, which constitutes a handicap problem in practice. For this reason, the use of acceleration methods on GPU becomes a real solution. For the acceleration of the FDK algorithm, we have used the GPU on heterogeneous platforms. To take full advantage of the GPU, we have chosen useful features of the GPUs and, we have launched the acceleration of the reconstruction according to some technical criteria, namely the *work-groups* and the *work-items*. We have found that the number of parallel cores, as well as the memory bandwidth, have no effect on runtimes speedup without being rough in the choice of the number of *work-items*, which represents a real challenge to master in order to be able to divide them efficiently into *work-groups* according to the device specifications considered as principal difficulties if we do not study technically the GPU as a hardware device. After an optimized implementation using kernels launched optimally on GPU, we have deduced that the high capacities of the devices must be chosen with a rough optimization of the *work-items* which are divided into several *work-groups* according to the hardware limitations.**

**Keywords—FDK algorithm, GPU, 3D Image Reconstruction, Cone-Beam Computed Tomography-CBCT, Intensive Computing, Reconstruction Acceleration.**

## I. Introduction

IN medical imaging, we are interested in solving a system of equations to find the solution, and this solution is finally the image, in another way we reconstruct data from a set of acquisitions, whether in 2D or 3D. In our work we use different architectures of platforms to accelerate the reconstruction of fully 3D medical images, much more focused on conical CT (Cone Beam Computed Tomography; CBCT) which is expected to be useful in clinical life, it helps to reduce the dose by reducing the X-ray exposure [1, 2], due to its speed time scanning, useful in intensive concert care such as image-guided radiotherapy techniques [3], and tooth implants requiring real-time imaging [4,5], and classification methods of mammography images by Kernel Extreme Learning Machine (KELM) or Kernel Principal Component Analysis (KPCA) techniques [6], and can be useful in Covid image classification by Wavelet Feature Vectors and Neural Network [7].

In the proposed work, we accelerate reconstruction by an analytical method by using heterogeneous architectures (CPU+GPU platforms) using OpenCL with C++. As we will see within our paper, there are some specifications which are very important to take into account with rigour when we want to choose the platform, and mainly the GPU, that will be used to accelerate the algorithm of reconstruction.

Since the most and widely used and implemented algorithm on scanners is the FDK [8], we use GPU to accelerate this method, we present a 3D reconstruction, we produce acceptable images when the acquisition angles are small ( $< 2$ degrees) and using sufficient number of projections and equidistributed around the object (because few numbers of projections or not equidistributed around the object remain to

the limits of the CT and thereafter the CBCT) by using robust 3D algorithm, but very expensive in calculation time and therefore slower to be executed, for this reason we have chosen to accelerate this algorithm on GPU which provides an excellent price/performance ratio for well-suited calculations (including linear algebra calculations) and easier in programming compared to Field Programmable Gate Array (FPGA) which is very complex because it is necessary to describe the entire architecture performing the desired calculation.

The main difficulty is how to divide the *work-items* into several *work-groups* to take full advantage of the parallelization capabilities of the algorithm. As the study of the technical characteristics of the GPU is imperative, the challenge is to have a compromise between the parallelization method and the capabilities of the GPU, since trying to fully occupy the hardware resources of the GPU will result in a loss of computation time, up to a parallel computation time not far from that one of a non-parallelized program.

The rest of this paper is organized as follows. In Section II, a literature review is presented. In Section III, the CBCT geometry and the FDK algorithm principles are overviewed. In Section IV, our strategy for algorithm parallelization using the GPU and the OpenCL environment is presented. In Section V, our implementation of kernel algorithms and the experimentation as well as the obtained results and discussions are presented, in addition to the details of the used datasets. Finally, our conclusion is presented in Section VI.

## II. Literature review

In parallel with the hardware technology evolution, during the last decade, there have been numerous published research works on the use of GPU in accelerating the reconstruction algorithms whether using OpenCL or CUDA methods.

In the work of Bo Wang et al. [9], OpenCL has been used for accelerating the CBCT reconstruction. They reported that the use of OpenCL which is a generic program, reduces the programming effort and considered it as the first truly open and royalty-free programming standard for general-purpose computations on the heterogeneous system which targets multi-core CPUs and latest GPUs.

Holger Scherl et al. [10] have compared the reconstruction in different architectures to the exception of the filtering stage which was done on CPU, and have reported that when using GPU, the known Bandwidth has no effect on time diminution within the acceleration method.

Leeser et al. [11] have used different architectures for accelerating the FDK algorithm using CUDA and OpenCL. They used Apple's FFTW [12] for 128 and 1024 points, and reported that their work is compatible with that one of Fessler on Matlab [13], and that to minimize the consumption of time in calculation it is preferable to transfer all calculation data to the GPU.

G. Yan et al. [14] have used two techniques in their algorithms to accelerate the Feldkamp-Davis-Kress (FDK) algorithm, the cyclic render-to-texture (CRTT) to save the copy time, and the combination of z-axis symmetry and multiple render targets (MRTs) for reducing the computational cost on the geometry mapping.

Dominguez et al. [15] have implemented a parallel version of the FDK algorithm on two different capacities level of GPU using CUDA-C and shown in their study that to have a higher speed up using GPU card one has to choose the highest capacities.

In their recent work, Shunli Z. et al. [16] have proposed a fast method for parallel implementation of the FDK algorithm using CUDA in multi GPU, by optimizing the backprojection operation, and they have considered as the most consuming time part in the algorithm. They have used simulated phantoms, and have mentioned that if the projection size gets larger it would be difficult to the device memory to store the whole projection data and the image would be reconstructed chunk by chunk.

Navid Z. et al. [17], have proposed an analytic modelling MLEM approach for the H-matrix MLEM algorithm by using GPU acceleration to make a high-performance computing software, they have reported that by using 100 iterations the images reconstructed by H-matrix method are less noisy.

Inam et al. [18], have proposed a new parameterizable architecture of optimized CUDA kernels reconstruction on GPU, and have declared having obtained high-speed reconstructions without compromising the image quality.

Valencia et al. [19] have proposed a parallel implementation of an iterative method concerning the MLEM reconstruction algorithm on GPU, and have presented their results and have declared that the MLEM implementations in CUDA using GPUs' capabilities were reliable and fast.

Table I, summarizes some works in the literature of accelerated methods and algorithms on GPU.

## III. Theory

### *A. The Cone-Beam CT geometry*

For cone-beam acquisition geometry, we have a circular orbit of the source-detector, a planar detector where data are collected as shown on Fig. 1. For volumetric CT the efficient acquisition setup is to use a two-dimensional detector [20], where rays form a cone with its base on the detector and its apex on an X-ray source, which naturally produces a cone of rays. This configuration increases the scanning speed and makes better use of the emitted rays, which can be also removed by collimation.

Three-dimensional reconstruction algorithms, in reconstruction method for fully three-dimensional reconstruction approach, are usually based on the three-dimensional Radon transform. In the object space, each plane can be represented by a unique point which is a plane integral in the object domain representing a three-dimensional Radon value. This point is the intersection of the plane and its normal

Table I: Summery of some works of accelerated algorithms on GPU

| Related work | Accelerated algorithm | Method used |
|---|---|---|
| Bo Wang et al. [9] | FDK | OpenCL |
| Scherl et al. [10] | FDK | - |
| Leeser et al. [11] | FDK | OpenCL and CUDA |
| G. Yan et al. [14] | FDK | CRTT, MRT techniques |
| Dominguez et al. [15] | FDK | CUDA-C |
| Shunli Z. et al. [16] | FDK | CUDA on multi GPUs |
| Navid Z. et al. [17] | MLEM ((MPH) SPECT) | - |
| Inam et al. [18] | GRAPPA (MRI) | CUDA |
| Valencia et al. [19] | MLEM | CUDA |

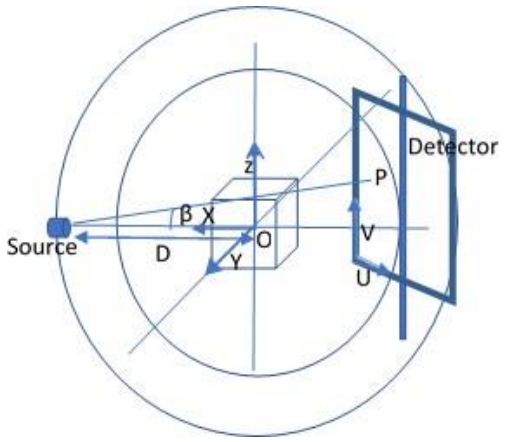


Fig. 1: Cone-Beam Acquisition Geometry

passing the origin. All Radon values placed at the corresponding points represent the three-dimensional Radon space. The Radon values of all planes intersecting the object have to be known to perform an exact reconstruction.

*B. FDK algorithm Principles*

The FDK is an approximate reconstruction algorithm for circular cone-beam tomography and has been used as a standard reconstruction approach for CBCT. Our configuration follows the original form of FDK, and data assumed coming from a planar detector [21].

The image space *f(x,y,z)* in Eq. (1) is obtained by backprojecting the filtered weighted projection according to Eq. (5).

$$f(x,y,z)=\int_0^{2\pi}\frac{D^2}{\hat{D}^2}\tilde{p}(\beta,u,v)d\beta \quad (1)$$

Where

$$\hat{D}=D+x\cos\beta+y\sin\beta \quad (2)$$

$$u=\frac{D}{\hat{D}}(-x\sin\beta+y\cos\beta) \quad (3)$$

$$v=z\frac{D}{\hat{D}} \quad (4)$$

$$\tilde{p}(\beta,u,v)=\left(\frac{D}{\sqrt{D^2+u^2+v^2}}\,p(\beta,u,v)\right)*h(u) \quad (5)$$

$\beta$ is the projection angle, *D (known as gantry rotation or SAD: Source to center Axes Distance)* is defined as distance between source and the origin *O*, *(x,y,z)* are voxel coordinates, *(u,v)* are the detector coordinates, *p(β,u,v)* are the detector acquainted data, *h(u)* is the ramp filter.

The integral in Eq. (1) is replaced by a sum over the projection angles in the discrete case. For each term of the backprojection sum, a two-dimensional interpolation of the filtered projection data is applied.

The FDK algorithm is a highly parallel nature, it can be performed in three processing stages; weighting the input projections, filtering the weighted input projections and back-projecting the filtered projections data into reconstructed volume, where all of them should be performed within the GPU.

By its massively parallel nature, the FDK algorithm finds a way out of the problem of time consumption in its resolution by using GPU as a platform where it can be implemented using the advanced technological development offered in new generations of GPUs. Since modern GPUs have several hundreds of parallel cores and can make calculation in floating-point precision, they become a good alternative candidate to support and best respond to the parallel nature of the FDK algorithm by the fact that in the reconstructed volume all projections are independent and can be calculated

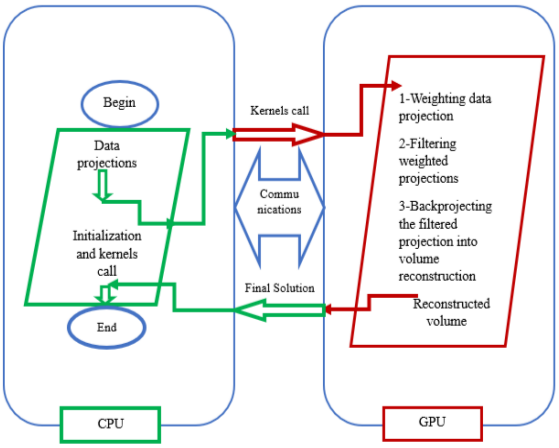


Fig. 2: Illustration how to call Kernels to perform parallel operations.

Table II: Some useful parameter descriptions

| | |
|---|---|
| Global-work-size | Dimension of global-work-items |
| Local-work-size | Dimension of the work-groups |
| Work-items number | Total number of work-items specified for the kernel launch |
| Work-groups number | Total number of work-groups |
| Work-group-size | Number of work-items per work-group |

independently, also each pixel in many cases can be calculated independently.

To reduce programming difficulties, we adopted a heterogeneous platform solution (CPU+GPU) where the OpenCL programming model is used on modern AMD and NVIDIA GPUs platform which both provide a parallel computing architecture. Parallel operations should be programmed as kernel functions that execute on GPU. Sequential operations should be programmed as host functions that execute on the CPU as illustrated on Fig.2. Both host and kernel functions are wrapped and called via a main host function. Communication between the CPU and the GPU can be done via the device's global memory, constant memory or texture memory on the GPU.

## IV. OpenCL and GPU Parallelization Strategy

OpenCL is known as a model where the notion of a host device is the basis of running data as well as tasks in parallel works. Kernel functions executed on the GPU can be executed as a function of multi-dimensional domains of indices named the global *work-size*. This later can be divided into sub-domains called *work-groups*, useful descriptions are given in Table II.

The essential element is called a *work-item*, and individual *work-items* within a group can communicate through global or locally shared memory. For synchronization, in OpenCL there are two domains: *work-items* in a single *work-group* and *command-queue(s)* in a single context. For more a comprehensive description, we refer the readers to [22, 23].

After compilation by the OpenCL environment, a program runs as a kernel in the GPU. A kernel takes input parameters, conducts computations, and outputs the result to device memory where the CPU can read it. The computation is done by thousands of *work-items*, each *work-item* performs the same operation in the kernel, but the input data can be different. With thousands of *work-items* doing similar tasks simultaneously, the computation speed can be considerably improved.

The host code running on the CPU prepares input data and accepts output values from the GPU. The intensive computation task is handled by GPU kernels. The output data is written to global device memory in order to be retrieved by a CPU program.

The strategy of the accelerated program can be explained as follows: The program begins by reading in 3D CBCT parameters and then initializes the output image by the specified data volume size. OpenCL is called and the device and its memory are initialized, then kernels specified for the GPU are built by taking into account that number of *work-items = local-work-size* x *the number of work-group*. data are transferred to the GPU and the execution starts running the weight kernel then the filter kernel and ends by the back-projection kernel, after finishing the resulting image is transferred back to the CPU to be displayed. For the transfer of the data from the CPU to the GPU or the opposite, we use a function called *oclMoveData* which intelligently copies data of the source location with the specified dimension to the destination location with the specified dimension, the dimension of the data must be the same for both source and destination.

## V. Implementation

### A. Kernels algorithms

To perform a parallel version of the FDK algorithm we have chosen the heterogeneous platform, we have selected some characteristics and hardware parameters that we consider relevant for analyzing against other works as shown on Table III.

As stated above, the FDK parallel version is performed in three stages. The weighting stage is not very consuming time, on the contrary, filtering and backprojection stages have both high runtimes. The filtering stage is performed within the FFT kernel presented in an OpenCL kernel to be executed on GPU without any performance (see for example [24]) which will be treated in our future work. The two first stages are presented by kernels shown by algorithms Algorithm 1 and Algorithm 2.

The backprojection stage is performed after the processing of the weighting and filtering stages, but in a different way where the slices of all projections are considered to be processed, if we consider *nu* and *nv* are number of pixels in a slice *(u,v)* and *np* is the number of projections, we scan all projection angles *np* and all pixels *nu* x *nv* in the buffer to be padded by zeros, and then after we scan slices over *Z* axis of

Table III: Summary of different GPUs characteristics and parameters

| Parameters | NVIDIA GTX 1060 | NVIDIA 425M | Radeon HD 8670M | Tesla C2075 | Radeon HD 7970 | Quadro 2000 | NVIDIA Geforce 9400 | NVIDIA GTX280 |
|---|---|---|---|---|---|---|---|---|
| Parallel cores | 1280 | 96 | 320 | 448 | 1600 | 192 | 16 | 240 |
| Bus Width (bits) | 192 | 128 | 64 | 384 | 256 | 128 | 64 | 512 |
| Band Width (GB/S) | 192.2 | 25.6 | 4.8 | 144 | 128 | 80.19 | 9.6 | 141.7 |
| Memory (GB) | 3 | 1 | 2 | 6 | 3 | 1 | 0.125 | 4 |
| System Memory (GB) | 8 | 8 | 8 | 32 | 32 | - | - | - |
| Memory Clock (Mhz) | 2002 | 800 | 800 | 750 | 1375 | 640 | 600 | 1107 |
| Related Work | This paper | This paper | This paper | [11] | [11] | [15] | [15] | [9] |

```
begin Kernel
kernel void weight(Params)
for each pixel in a single projection do
    generate a weight value for that pixel;
    for each projection do
        apply the current weight to the appropriate pixel;
    end
end
return;
end Kernel
```

**Algorithm 1**: Weight Kernel

```
begin Kernel
kernel void filter( Params )
for each 2D projections do
    for each row in the projection do
        convert to the frequency domain;
        //Frequency Domain;
        fft1024(parameters);
        Apply the filter to the weighted projection data;
        //Time domain;
        Convert back to time domain and scale;
        fft1024(parameters);
    end
end
return;
end Kernel
```

**Algorithm 2**: Filter Kernel

total number *nz* to process each pixel in the output slice *nx* x *ny* and we apply the output projection data to all projection angles, we scan all slices, and we perform each pixel for all projections to obtain final image *nx* x *ny* x *nz* which simply indicates that kernels process a 3D image and it is the final render volume to transfer to the host where it is displayed, see (Algorithm 3).

```
begin Kernel
kernel void backProj( Params )
// Projections in buffer padded with zeros
for each output image slice do
    for each pixel in the slice do
        apply projection data to output image over each projection
        angle;
        apply a bi-linear interpolation;
    end
end
return;
end Kernel
```

**Algorithm 3**: Backprojection Kernel

### *B. Experimentation and Results*

In this sub-section, we present a global view on how we evaluate our results according to the hardware used and specify some technical specifications on the choice of the platform to use for the acceleration of the reconstruction method.

It is recommended to start the experiment with a good hardware of high capacities. For the implementation of the acceleration method, one has to be rough in the way with which a heterogeneous platform will be handled and to ensure a fast data communication between host and device. It is also necessary to guarantee a rapid implementation of the kernels which are responsible for executing the reconstruction quickly and optimally, by a good choice of the groups of *work-items* which are the elements to be well structured to benefit from a considerable gain in runtime.

*1) Used Datasets*

In the beginning, we use data of mathematical phantoms of size 128×128×128 and size 256×256×256 which have been tested on different architectures and we record different runtimes of each configuration, see (Table IV).

In the second step of our experiments we use different data of real Phantoms and their modified configurations that we processed to have different sizes:

1. Sample-data1 (sample data in [25]), tested in [9], we have 320 projections of size 192×256 and source-to-detector-distance SDD = 155 cm and source-to-axis-distance SAD = 100 cm.
2. Sample-data2 (sample data in [26]), we have 211 projections of size 256×200 and source-to-detector-distance SDD = 153.6 cm and source-to-axis-distance SAD = 110 cm.
3. Sample-data3 (for the sample-data1), we have 320 projections of 400×400 to get after reconstruction a volume of size 256×256×256.
4. Sample-data4 (for the sample-data2 ), we have 211 projections of 400×400 to get after reconstruction a volume of size 128×128×128.

*2) Results and Discussions*

For the first step where we used simulated datasets, by seeing (Table IV) we notice that a higher number of parallel cores has no effect on the speedup of the reconstruction time without considering the Bandwidth, the Bus Width and other capacities. This can be seen by comparing, for example, NVIDIA 425M and Radeon HD 8670M where the first has 96 parallel cores with a memory frequency of 800 MHz, while the second one has 320 parallel cores with the same memory frequency but a significantly lower computation time with respect to the first one.

Moreover, one has to confirm that memory system has to be considerably large if we want to process huge sizes as 512x512x512 for example; this means that the parallelization method has no effect on the runtime speed without acceptable GPU capacities and high system capacities.

For best results, we have chosen the size of the *local-work-size* and the number of *work-groups* properly so as not to have all the private and local memory of the device used continuously which limits the number of *work-items* to be executed in parallel. Such an idea is related to the fact that *local-work-size* cannot be bigger than the specified dimension through CL_DEVICE_MAX_WORK_ITEM_SIZES, and the cumulative *work-group-size* which is the product of all dimensions *x, y, z* must be less or equal to the resulting number.

The *global-work-size* can be chosen as big as we want, but it has the constraint to be a multiple of the *local-work-size*. Furthermore, a *work-group* has to execute on one compute unit and this compute unit might not be fully used, because a compute unit might execute as many *work-groups* as possible for performance, and we have to specify *local-work-size* because if we do not, and if configured to be determined by the OpenCL implementation, that we do not advise to do, we cannot know how our work is divided into *work-groups* and we do not have guarantee that the *work-group-size* will be optimal.

To have a precise justification, one has to know how the runtime of the backprojection changes with the size of the *local-work-size*, for this reason, we use Sample-data3 and Sample-data4 to plot the runtime of the backprojection according to different sizes of the *local-work-size* see (Fig. 3 and Fig. 4).

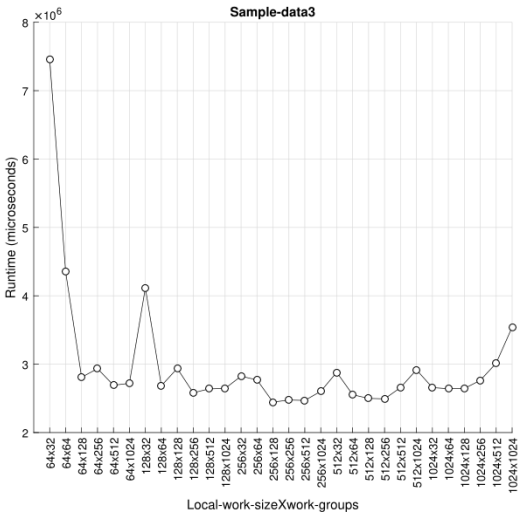


Fig. 3: Runtimes of a set of local-work-item of Sample-data3

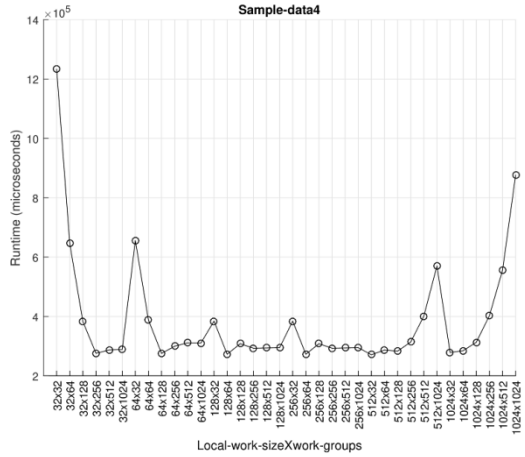


Fig. 4: Runtimes of a set of local-work-item of Sample-data4

We can see the symmetry on both graphs (on *X axix* like 64x128 with 128x64, 128x1024 with 1024x128...), this can be explained by the fact that the number of *work-items* specified for the kernel launch is the product result of *local-work-size* and *work-groups*, but the runtimes will not necessarily be equal, and this is explained by how the *work-items* are executed in parallel, and to have an optimal method, we have to choose a *work-group-size* lower than the value defined by CL_DEVICE_MAX_WORK_GROUP_SIZE which defines the maximum size of a *work-group*, and due to the limited hardware resources we have to choose a scalar for *local-work-size* less than the maximum value allowed by the device, because CL_DEVICE_LOCAL_MEM_SIZE increases with the *local-work-size*.

The peak values represent high values which are qualified by a very high time consumption, and it contradicts with the aim of method acceleration which wants to benefit from a significant time gain, for this reason, it is necessary to avoid

Table IV: Different GPUs runtimes by using mathematical phantoms (seconds)

| GPU | Runtimes for size 128 × 128 × 128 | | | | Runtimes for size 256 × 256 × 256 | | | |
|---|---|---|---|---|---|---|---|---|
| | Weighting | Filtering | Back projection | Total | Weighting | Filtering | Back projection | Total |
| NVIDIA GTX 1060 | 0.032 | 0.374 | 0.138 | 0.544 | 0.063 | 0.531 | 0.789 | 1.383 |
| NVIDIA 425M | 0.101 | 1.529 | 1.589 | 3.219 | 0.129 | 1.956 | 2.841 | 4.926 |
| Radeon HD 8670M | 0.28 | 2.1 | 10.462 | 12.842 | 0.28 | 2.67 | 78.83 | 81.78 |

choosing small values of *work-groups* compared to the size of *local-work-size*, or to choose the maximum values (1024).

For more program optimization, we compare theoretical peak memory bandwidth and effective one (or observed) which can be calculated using Eq. (6), and as we can find in [27] that if the effective bandwidth is much lower than the peak memory bandwidth, optimization efforts should increase the effective bandwidth.

$$BW_{Effective} = \frac{(R_B + W_B)/10^9}{t} \quad (6)$$

Where: $R_B$ is the number of bytes read per kernel and $W_B$ is the number of bytes written per kernel, and $t$ is the elapsed time given in seconds.

We use the GPU NVIDIA GTX 1060 considered the most efficient in our work, and we optimize our method according to values of peak memory bandwidth presented in Table V, and we calculate effective bandwidth for memory copy from Host to Device using a set of *work-items* that we have chosen for both sample-data3 and sample-data4 that we report in Table VI, and to see how much the value of effective bandwidth is close to the value of the peak memory bandwidth, we calculate the percentage through the ratio between effective bandwidth and peak memory bandwidth, see Table VII.

Table V: Peak memory bandwidth (GB/s)

| NVIDIA GTX 1060 | | |
|---|---|---|
| Device to Host Bandwidth | Host to Device Bandwidth | Device to Device Bandwidth |
| 12.61 | 12.19 | 141.65 |

Here we do confirm that bandwidth has no effect on runtime speedup, but on the other hand, effective bandwidth should be close to peak memory bandwidth for optimization considerations, because bandwidth represents the data transferring rate, and can affect the program performance if it is affected when for example, the programmer has not chosen properly where data should be stored and how can it be extracted or accessed.

Table VI: Calculated bandwidth (GB/s)

| NVIDIA GTX 1060 | | |
|---|---|---|
| Work-items-size | Effective Bandwith (Sample-data4) | Effective Bandwidth (Sample-data3 |
| 1048576 | 8.32 | 8.42 |
| 524288 | 8.34 | 8.60 |
| 262144 | 8.53 | 8.64 |
| 131072 | 8.64 | 8.53 |
| 65536 | 8.60 | 8.34 |
| 32768 | 8.42 | 8.32 |

Table VII: Ratio in %: How much effective and theoretical peak bandwidth are close

| NVIDIA GTX 1060 | | |
|---|---|---|
| Work-items-size | Bandwith Ratio (Sample-data4) (%) | Bandwith Ratio (Sample-data3) (%) |
| 1048576 | 68.25 | 69.07 |
| 524288 | 68.41 | 70.54 |
| 262144 | 69.97 | 70.88 |
| 131072 | 70.88 | 69.97 |
| 65536 | 70.54 | 68.41 |
| 32768 | 69.07 | 68.25 |

Table VIII shows some more indications that the higher number of parallel cores does not affect the speedup of the reconstruction without having a device with higher capacities. One can see that with the same dataset volume in the reconstruction procedure for NVIDIA 425M and Radeon HD8670M respectively, the later one has higher parallel cores number but a slower execution time, while the first one has faster execution time with lower parallel cores number and the same memory frequency but has a higher bus width and bandwidth.

Table VIII: Summary of comparison of differents GPUs runtimes (seconds)

| Data size | Runtime | NVIDIA GTX 1060 | NVIDIA 425M | Radeon HD 8670M | Tesla C2075 | HD 7970 | Quadro 2000 | Geforce 9400 | NVIDIA GTX 280 |
|---|---|---|---|---|---|---|---|---|---|
| 60*50*72 | Backprojection time<br>total time | - | - | - | 0.01<br>0.30 | 0.01<br>0.30 | - | - | - |
| 320*192*256 | Filtering time<br>Backprojection time<br>Total time | 0.453<br>0.447<br>0.970 | 0.985<br>11.296<br>12.374 | 2.37<br>106.838<br>109.478 | - | - | - | - | - |
| 211*256*200 | Filtering time<br>Backprojection time<br>Total time | 0.488<br>0.261<br>0.814 | 0.984<br>4<br>5.078 | 2.122<br>48.840<br>51.242 | - | - | - | - | - |
| 512*512*768 | Backprojection time<br>Total time | - | - | - | 49.44<br>60.45 | 16.01<br>28.02 | - | - | - |
| 256*256*128 | Total time | - | - | - | - | - | - | - | 14.227 |
| 10486*10000 | Reconstruction time | - | - | - | - | - | 57.4 | 400.4 | - |
| 1232*10000 | Reconstruction time | - | - | - | - | - | 506 | 3503 | - |
| 320*400*400 | Filtering time<br>Backprojection time<br>Total time | 1.875<br>2.88<br>5.041 | 4.719<br>40.968<br>46.171 | 8.232<br>425.676<br>434.548 | - | - | - | - | - |
| 211*400*400 | Filtering time<br>Backprojection time<br>Total time | 1.27<br>0.30<br>1.789 | 3.093<br>4.688<br>8.110 | 5.461<br>62.15<br>68.050 | - | - | - | - | - |
| Related work | | This paper | This paper | This paper | [11] | [11] | [15] | [15] | [9] |

Result in [9] is the total reconstruction time Result in [15] is the reconstruction time after n/m (input/output) projections

By comparison with other works (see Table VIII) we record different runtimes, even we do not compare with the same dimensions, but we can have a technical point of view on runtimes according to the given configuration.

As a first observation, we can notice when comparing the total runtime of NVIDIA GTX 280 [9] and NVIDIA GTX 1060, that the total runtime of NVIDIA GTX 280 is higher than that one of NVIDIA GTX 1060, and without taking into account the major difference in number of parallel cores in favour of the NVIDIA GTX 1060 we can see that the memory frequency which is nearly doubled and again in favour of NVIDIA GTX 1060 gives significant runtime gain.

GPU hardware limitations conduct the user to perform the acceleration method according to a technical choice, for example, the maximum of *work-groups* to be handled is limited to 1024 but it is clear in our experiments that choosing less than this value is much more benefic for runtime speedup, our experiments show that for 64x128 or 128x64 of *local-work-size* is much more reasonable to have a good runtime speedup than to 1024x1024 or 512x512, which represents the number of *work-items* to perform the job, and programs run better with a *local-work-size* smaller than the maximum allowed by the device it means more complex the kernel and the more private variable it has, the more registers it will need and by consequence the more time it consumed.

We can choose the size of the *global-work-size* as we want as far as that does not exceed one of the device's global resource consumption limits and changing the *global-work-size* does not increase program parallelism because this remains to the occupation of all device's compute units, and somewhere one has to reduce parallelism to more closely much the device capabilities, and avoiding to have memory consuming conflicts.

As we deduce by the experiments, the memory frequency is one of the important specifications, the higher the frequency the smaller the transfer time the more the gain in runtime speedup.

For the number of parallel cores, it depends on the device capacities, the more the higher the capacities with a higher

number of parallel cores, the higher the speedup we have. This can be clarified by comparing the NVIDIA 425M, the Radeon HD8670M, and the NVIDIA GTX 1060, where the first one has a lower number of parallel cores than the second one but with a faster runtime. While the third one, which is the NVIDIA GTX 1060, has the most higher number of parallel cores, the highest device capacities, and precisely the fastest runtime of the three cards.

The parallelism method has no effect on runtime speedup without kernel performance, it is necessary to provide the three stages of the FDK parallel version in a manner that will not consume more time than it is necessary and this depends on the implementation strategy, and it is clear that when we have large data size it will be important to perform even the parallelism inside the FFT code because higher it is the number of FFT points the higher is the data to compute and higher it will consume time.

The known peak memory bandwidth has no effect on runtime speedup, but for more program performance it is necessary to have the value of the effective bandwidth close to that one of the theoretical peak memory bandwidth, for our experiments we have a performance of a value around *70\%* whatever the size of the *work-items* used, compared to [27] where the performance value is between *76\%* and *82\%* using different GPUs.

System memory has to be significantly large if size of data to be reconstructed is important with memory management, thing which has not been treated in this work, gives a higher time speedup.

As our experiments give better results with the NVIDIA GTX 1060, we have chosen to use it as a device of comparison with other works.

### C. Applications of this study

Since the introduction of CBCT in medical imaging, the major concern has been to try to get a better quality result for the image while minimising the image reconstruction time and the X-ray exposure time.

For fully 3D reconstruction, CBCT is considered as an important solution providing a 3D image in a fast reconstruction time due to its geometry, and taking into account the results of both datasets 1 and 2, we can say that the acceleration of the FDK method can be useful in several specialities, such as Orthopedics radiology, Dento-maxillofacial radiology and Image-guided Radiotherapy.

## VI. Conclusion

In this work, we have presented an implementation for accelerating a parallel version of the FDK algorithm on several heterogeneous architectures with different capacities. We have presented what the effect of GPU capabilities will be and how to take advantage of the technical capabilities of the GPU to achieve better computational time-saving. In comparison to existing work, we have tried to draw attention to the technical characteristics of GPUs and to the fact that it is necessary to rigorously manipulate the items known as *work-items*, to benefit from a gain in computation time and to have a parallel program optimization.

For the implementation, we have chosen to use OpenCL because it gives the programmer flexibility and ease of use of the GPU for better performance. Our experiments show that the FDK algorithm has a parallel nature, so the choice of its acceleration on GPU is justified. According to the results of our experiments, as the reconstruction size gets larger, the filtering stages consumes more time, which is more beneficial for the reconstruction speedup if we parallelize process within the filtering kernel.

The number of the *local-work-size* has been chosen for higher performance of the parallelized code, and specifying the number of *work-items* to be used in the kernel launch is the best choice than it is configured to be specified by the OpenCL implementation.

Limitations of the device capacities conduct the programmer to reduce, somewhere, parallelism to more closely much the device capacities. Having a device with a higher number of parallel cores does not guarantee to have a higher execution speed as stated above, it depends on the device capacities, hence the obligation to choose a device with higher capacities.

## Acknowledgment

This work was supported by GDSRTD (General Direction of Scientific Research and Technological Development, Ministry of Higher Education and Scientific research, Algeria).

## Conflict of Interests

Conflict of Interests: none

## References

[1] Yu, Lifeng Liu, Xin Leng, Shuai Koer, James Ramirez-Giraldo, Juan Qu, Mingliang Christner, Jodie Fletcher, Joel McCollough, Cynthia. Radiation dose reduction in computed tomography: techniques and future perspective. Imaging in medicine. 2012. 1. 65-84. doi :10.2217/iim.09.5.

[2] Cynthia H McCollough, Andrew N Primak, Natalie Braunc and James Koer. Strategies for Reducing Radiation Dose in CT. Radiol Clin North Am. 2009. 47. 27 . doi : 10.1016/j.rcl.2008.10.006.

[3] Ying Song, Weikang Zhang, Hong Zhang, Qiang Wang, Qing Xiao, Zhibing Li, Xing Wei, Jialu Lai, XuetaoWang, Wan Li, Quan Zhong, Pan Gong, Renming Zhong and Jun Zhao. Low-dose cone-beam CT (LD-CBCT) reconstruction for image-guided radiation therapy (IGRT) by three-dimensional dual dictionary learning, 2020, Radiation Oncology, https://doi.org/10.1186/s13014-020-01630-3

[4] Marion Lahutte-Auboin, Amir Ait-Ameur, Virgine Decat and Laurent Hauret . Dental Implant Imaging: How CT Scan Became a Help to Surgery. Implant DentistryA Rapidly Evolving Practice. Ilser Turkyilmaz. IntechOpen. 267-286. 2011. InTech Europe. 2. 2nd.

[5] B. Rajkumar, Lalit C. Boruah,Vishesh Gupta. Use Of C-Arm Ct Real Time Imaging System in Endodontics-clinical Report . International Journal of Scientic Research. 2015. 07. 4. 712-714.

[6] Bacha Sawssen, Taouali Okba, Liouane Noureeddine, A Mammographic Images Classi_cation Technique via the Gaussian Radial Basis Kernel ELM and KPCA, Int. J. of Applied Mathematics, Computational Science and Systems Engineering, 2020, Volume 2, pp. 92-98

[7] Stella Vetova, Covid Image Classifcation using Wavelet Feature Vectors and NN, EngineeringWorld, 2021 Volume 3, pp. 38-42.

[8] L. A. Feldkamp, L. C. Davis, J. W. Kress. Practical cone-beam algorithm. J. Opt. Soc. Am. A. 1984. 1. 612-619.

[9] Bo Wang and Lei Zhu and Kebin Jia and Jie Zheng. Accelerated cone beam CT reconstruction based on OpenCL 2010 International Conference on Image Analysis and Signal Processing. 2010. 291-295.

[10] Holger Scherl , Markus Kowarschik , Hannes G. Hofmann and Benjamin Keck and Joachim Hornegger. Evaluation of state-of-the-art hardware architectures for fast cone-beam CT reconstruction. Parallel Computing. 2012. 38. 3. 111 - 124.

[11] Leeser Miriam , Mukherjee Saoni, Brock, James. Fast reconstruction of 3D volumes from 2D CT projection data with GPUs. BMC research notes. 2014. 7. 582.

[12] Sample code FFT OpenCl. https://developer.apple.com/library/archive/samplecode/ OpenCLFFT/links.htm.

[13] Michigan Image Reconstruction Toolbox (MIRT). http://web.eecs.umich.edu/fessler/ code/index.html.

[14] Yan, Guorui Tian, Jie Zhu, Shuping Dai, Yakang Qin, Chenghu. Fast cone-beam CT image reconstruction using GPU hardware. Journal of X-Ray Science and Technology. 2008. 16. 225-234.

[15] J S Dominguez, L F de Oliveira, N A Junior and J T de Assis. Using Graphics Processing Units to Parallelize the FDK Algorithm for Tomographic Image Reconstruction. 2012.

[16] Zhang Shunli, Geng and Guohua Zhao, Jian. Fast parallel image reconstruction for cone-beam FDK algorithm. Concurrency and Computation: Practice and Experience. 31. 10. 4697. 10.1002/cpe.4697. https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.4697. 2019.

[17] Navid Zeraatkar, Benjamin Auer, Kesava Kalluri , Lars R. Furenlid , Philip H. Kuo , Michael A. King, GPU-accelerated generic analytic simulation and image reconstruction platform for multi-pinhole SPECT systems, 15th International Meeting on Fully Three-Dimensional Image Reconstruction in Radiology and Nuclear Medicine, International Society for Optics and Photonics, SPIE, 196 - 199, 2019, 10.1117/12.2534523, https://doi.org/10.1117/12.2534523.

[18] Inam, Omair and Qureshi, Mahmood and Akram, Hamza and Omer, Hammad and Laraib, Zoia, 2019 IEEE 2nd International Conference on Information and Computer Technologies (ICICT), Accelerating Parallel Magnetic Resonance Image Reconstruction on Graphics Processing Units Using CUDA, 2019, 109-113, 10.1109/INFOCT.2019.8710946.

[19] Valencia Pérez, T. A., Hernández López, J. M., Moreno-Barbosa, E., de Celis Alonso, B., Palomino Merino, M. R., & Castaño Meneses, V. M. (2020). Efficient CT Image Reconstruction in a GPU Parallel Environment. *Tomography (Ann Arbor, Mich.)*, *6*(1), 44–53. https://doi.org/10.18383/j.tom.2020.00011

[20] J. Daniel Bourland. Image-Guided Radiation Therapy. CRC Press. isbn = 978-1-4398-0274-8,978-1-4398-0273-1. 2012. Imaging in Medical Diagnosis and Therapy. 1. http://gen.lib.rus.ec/book/index.php?md5=7537C24 F975D2FC603CF9930674402CC.

[21] Turbell, Henrik. Cone-beam reconstruction using filtered backprojection . Linkoping Universitet. 2001.

[22] Munshi, Aaftab and Gaster, Benedict and Mattson, Tim and Fung, James and Ginsburg, Dan. Opencl Programming Guide. Addison-Wisley. 2011. 04. 01 Boylston Street, Suite 900Boston, MA 02116. 0321749642.

[23] Kaeli David, Mistry Perhaad Schaa, Dana Zhang D.P. Heterogeneous Computing with OpenCL 2.0: Third Edition. 2015. 01. 1-307.

[24] N. K. Govindaraju and B. Loyd and Y. Dotsenko and B. Smith and J. Manferdelli. High performance discrete Fourier transforms on graphics processors in SC '08: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing. 2008. 1-12. doi :10.1109/SC.2008.5213922. 2167-4337. Nov.

[25] Rezvani, N. Aruliah, D. Jackson, K. Moseley, D. Siewerdsen and J. OSCaR: An opensource cone-beam CT reconstruction tool for imaging research. 2007. 06. 34. Medical Physics – MED PHYS. doi : 10.1118/1.2760393.

[26] 3D Cone beam CT (CBCT) projection backprojection FDK, iterative reconstruction Matlab examples. http://www.mathworks.com/matlabcentral/_leexchange/35548 - 3d - cone - beam - ct - cbct - matlab - examples.

[27] Fatica Massimiliano and Ruetsch Gregory. Performance Measurement and Metrics. 2014. 12. 31-42.