GPU-based Image Reconstruction for Real Time X-Ray Fluoroscopy with a Regular Detector Grid

Manipal Dot Net Pvt. Ltd.,
37 Ananth Nagar,
Manipal 576 104 INDIA
http://www.manipal.net

Technical Report # MDN-TR-2013-0601
June 15, 2013

For further details, contact:
Dr. Prashanth B. Bhat, Ph.D.
CTO
prashanth.bhat@manipal.net
http://www.linkedin.com/in/prashanthbhat

GPU-based Image Reconstruction for Real Time X-Ray Fluoroscopy with a Regular Detector Grid

Abstract

This paper describes a strategy to parallelize and implement in real-time the multi-plane tomosynthetic image reconstruction algorithm used in common x-ray fluoroscopy systems such as the Scanning Beam Digital X-Ray (SBDX) system, on high performance computing platforms such as general purpose Graphical Processing Units (GPU). The authors contrast two different parallelizing schemata, namely, the detector centric and the pixel centric parallelization approaches under the specific assumption of a regular detector grid and separability of the *x* and *y* dimensions. In both of these schemes, the use of look-up tables (LUT) helps to reduce run time computations but requires effective memory management strategies. An optimal implementation of these schemes on GPUs also needs to maintain a high level of achieved occupancy by setting an appropriate thread-block configuration. The paper reports results of implementing the two parallelization schemes and associated optimizations on Nvidia GPUs and demonstrates 15 fps performance using a single GeForce GTX690 card. The paper concludes that the pixel centric approach has better arithmetic intensity and superior scalability properties which makes it ideal for use in multi-GPU systems.

1 Introduction

The Scanning Beam Digital X-Ray (SBDX) system represents a significant advance over conventional x-ray fluoroscopy systems in terms of reduction of radiation exposure to patients and staff, and improved SNR due to lesser detection of x-ray scatter [1,2,3]. This is achieved by restricting the x-ray beam to a small field of view and directing it to a small area detector. However, this means that the full field of view fluoroscopic images at multiple focal planes need to be digitally reconstructed from a number of smaller views detected by the detector – essentially a tomosynthetic procedure. The compute intensity of fluoroscopic reconstruction is evident from the typical detector data throughput rates involved (30 Gbps) and the required floating point operations (1.5 TFLOPS).

Like several other medical imaging algorithms, there are important aspects of fluoroscopic reconstruction that enable its parallelization and make it suitable for implementation on any parallel processing device. While current fluoroscopic systems are known to use custom-designed hardware based on field programmable gate arrays (FPGA), systems based on off-the-shelf GPUs can potentially give improved performance while substantially reducing costs [4].

In prior applications of GPUs for various medical imaging applications, researchers have noticed that the same algorithm can have two distinct formulations with very different run time performance namely, the gather and scatter formulations. As explained in [4], '… both gather and scatter formulations produce the same output and have the same theoretical complexity. However, on the GPU, gather operations are more efficient than equivalent scatter operations because memory reads and writes are asymmetric: memory reads can be cached and are therefore faster than memory writes ... Last, by writing data in an orderly fashion, gather operations avoid write hazards. Scatter operations require slower atomic operations to avoid such write hazards.'

Along similar lines, we show that the multi-plane tomosynthetic fluoroscopic image reconstruction also admits two different parallelizing schemes, namely the detector centric and pixel centric schemes

which correspond to scatter and gather formulations respectively. We explore various aspects that lead to an effective implementation of these schemes – the arithmetic intensity of the parallelizing schemes, the use of look-up tables (LUT) to avoid repeated computations, different ways of managing the GPU's memory resources and maintaining a high level of achieved occupancy by choosing different thread-block configurations for apportioning of the computations. Our experimental results reported in this paper compare the scalability of the two parallelization schemes and their performance on some of the recent GPUs from the Nvidia family.

The modeling of detector geometry is an important factor influencing the quality of fluoroscopic images. We have assumed that the detector elements are arranged on a 2D plane in a regular and rectangular grid. Real-world detectors, however, have a slightly irregular 3D structure [2,5]. Our assumptions render the reconstruction problem separable in the *x* and *y* dimensions, which helps reduce memory bandwidth requirements substantially. We believe that the immense computational benefits of separability makes it an effective trade-off vis-à-vis image quality.

This paper does not consider the procedure of multi-plane compositing which follows the stage of multi-plane reconstruction and where the multiple reconstructed planes are combined to produce a best focus image [2]. The focus is on image reconstruction of individual planes. Multi-plane compositing will be considered in a future effort.

The rest of the paper is organized as follows – In the next section, we briefly describe the SBDX system. In section 3, we describe the tomosynthetic image reconstruction algorithm and derive the detector centric and pixel centric formulations. In section 4, we provide details of the different computing platforms, the programming techniques and heuristics that have been applied in our experiments. In section 5, we present the run time results on different GPUs and compare their performance. Finally we present our conclusions and directions for future research.

2 The SBDX System

The SBDX system [1,2] consists of a large area X-ray source that is spatio-temporally sampled using a multi-hole collimator. The collimator holes restrict the emerging x-ray photons to those directed towards the detector. The collimator holes in the SBDX system considered in this paper are arranged in the form of an 100x100 array.

The scanning technique used by the x-ray source can include many different collimator hole patterns and many different frame rates. For the purposes of this paper we shall assume that the scanning pattern involves one complete raster scan of the 100x100 holes per frame, and at a rate of 30 fps.

The x-rays emitted from a single collimator hole pass through the object (a human patient) and are subsequently detected by a small, distant detector. The detector considered in this paper is assumed to consist of an 8x4 array of CdTe tiles. Each tile, known as a detector hybrid, in turn is assumed to consist of an array of 20x20 detector elements giving an overall array of 160x80 detector elements. The detector hybrids are not all in the same plane but have a stepped arrangement. Furthermore, the detector elements in each hybrid are not always arranged in a regular, rectangular fashion. This implies that the actual detector configuration is slightly irregular and has a 3D structure [2,5].

Since both the source-detector distance and the patient-detector distance is large compared to the size of the detector, the 3D structure of the detector is insignificant and therefore it is reasonable to model

the detector as a flat array of detector elements. A further but less sanguine assumption is to model the detector elements as being located on a regular and rectangular grid. In the separable formulation which result from the above assumptions the influence of a detector element in the x direction is decoupled from that in the y direction. This decoupling can be exploited to reduce memory storage requirements substantially – from $O(n^2)$ to $O(n)$. We believe that these assumptions are not overly restrictive and can produce quality fluoroscopic images in real world scenarios.

Each element of the detector array outputs an 8-bit number for each illumination from a collimator hole. The detector data is produced and made available for processing at the same rate at which the collimator array is illuminated by the x-ray source. This works out to a data rate of about 30 Gbps – (160x80 detector elements x 100x100 collimator holes x 30 fps x 8 bits).

3 The Tomosynthetic Image Reconstruction Algorithm

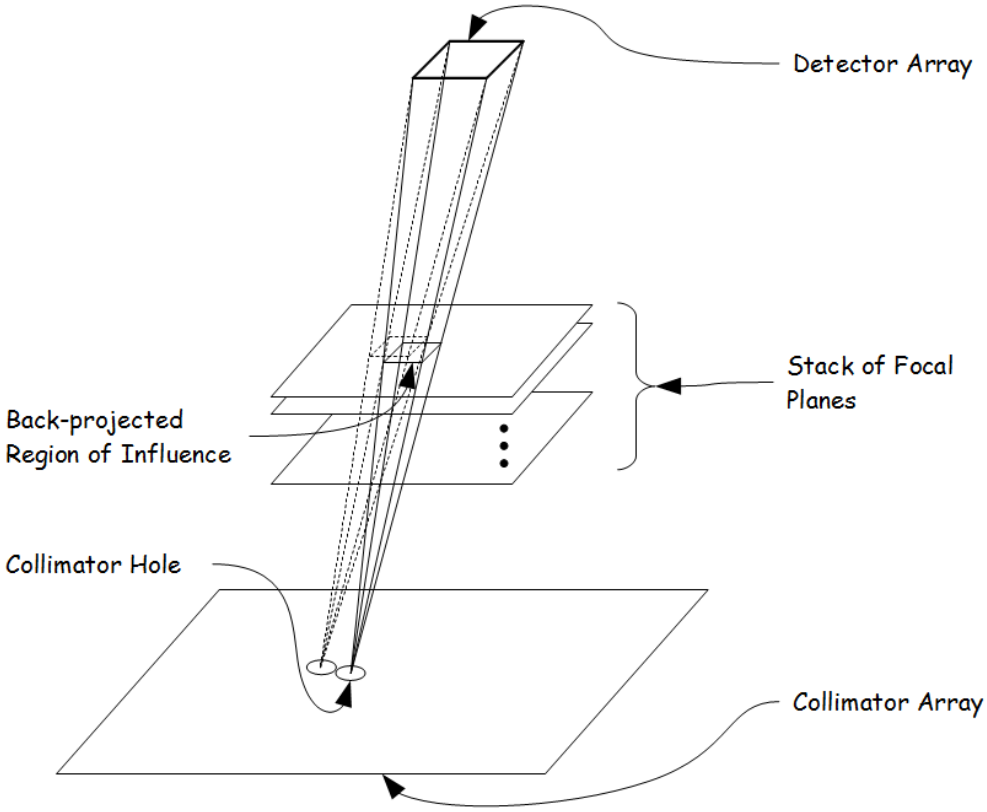

Figure 1. Basic Geometry of Tomosynthetic Image Reconstruction in the SBDX System

The image reconstruction algorithm considered in this paper is a basic *shift-and-add tomosynthesis* described in [2]. The basic geometry of the reconstruction problem is shown in figure 1. The x-rays emitted from a collimator hole pass through the stack of focal planes and are detected by the detector array. In order to reconstruct an image at a focal plane, the detector array is back-projected onto the plane through simple ray tracing which creates a region of influence (ROI) for that particular collimator hole, on that plane. Tomosynthesis works by apportioning the detector data to pixels within the ROI. The final image at the focal plane is obtained by adding the contributions of all the shifted ROIs (from scanning the collimator array) to the pixels of the focal plane. A formal description of the algorithm is as follows:

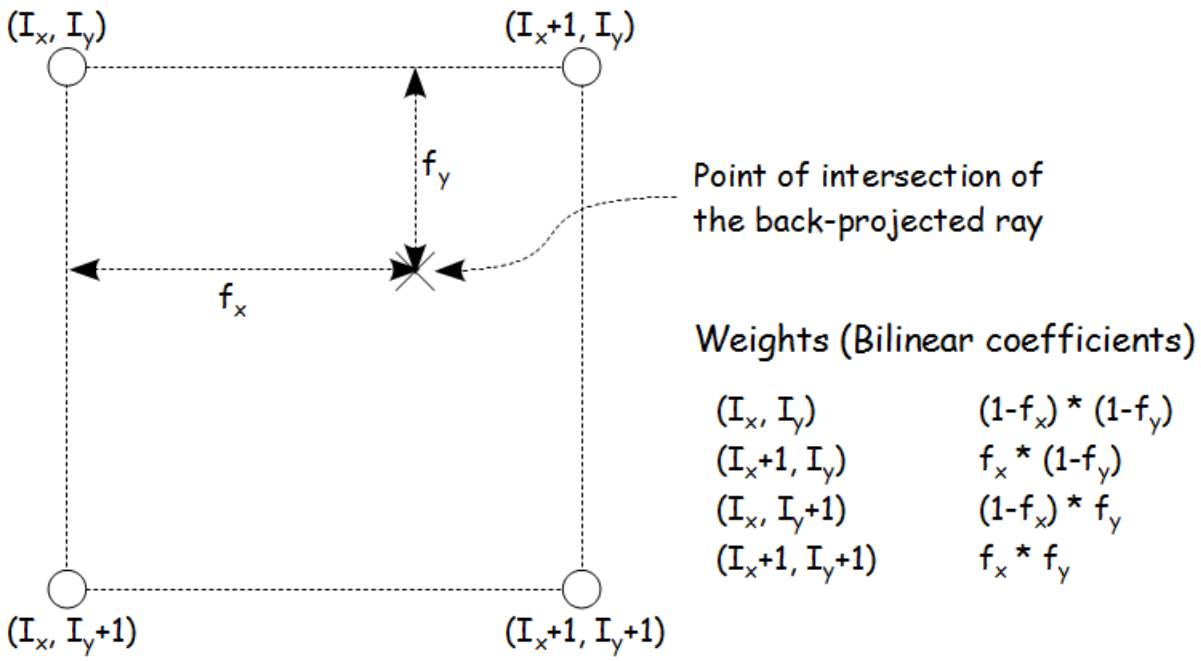

Figure 2. Local view of the reconstruction algorithm – 2x2 pixel grid and bilinear coefficients

The width and height of the collimator and the detector arrays is denoted by (W_c, H_c) and (W_d, H_d) respectively. The collimator and the detector arrays are indexed using indices (c_x, c_y) and (d_x, d_y) respectively. The collimator holes and detector elements are assumed to be located at integer locations corresponding to their indices respectively.

For a given collimator hole (c_x, c_y) and a detector element (d_x, d_y) that detects the x-rays emanating from it, the output of detector element (d_x, d_y) always influences a 2x2 pixel grid in each of the P focal planes through which the x-rays pass. The specific 2x2 pixel grid in each of the planes that is influenced by the combination of collimator hole (c_x, c_y) and detector element (d_x, d_y) is determined by the point of intersection of the line connecting the collimator hole and the detector element with the focal plane.

In particular, the four pixels of the 2x2 pixel grid within which the point of intersection lies, are the only ones that are affected by the combination of collimator hole (c_x, c_y) and detector element (d_x, d_y). Furthermore, the exact magnitude of the influence is given by weighing the output of detector element (d_x, d_y) using bilinear coefficients (f_x, f_y) corresponding to the location of the point of intersection within the 2x2 pixel grid. Figure 2 illustrates this local view of the algorithm.

The image reconstruction algorithm considered in this paper can be shown to be a special case of the one described in [2]. In [2], the region of influence of a single detector element or *spread* is denoted by r, and the width of the pixel or *pitch* is denoted by p. If one assumes, without loss of generality, dimensionless quantities, and sets $r = p = 1.0$, one can easily derive our algorithm.

As defined in [2], as one moves from one collimator hole to an adjacent one, the ROI shifts by an integral number of pixels denoted by the parameter m. This is independent of the focal plane i.e. in any focal plane, shifting from one collimator hole to an adjacent one causes the ROI to shift by m pixels. Thus, without loss of generality, we can claim that a collimator hole (c_x, c_y) has a region of influence

(ROI) centered around ($m*c_x$, $m*c_y$) in any focal plane.

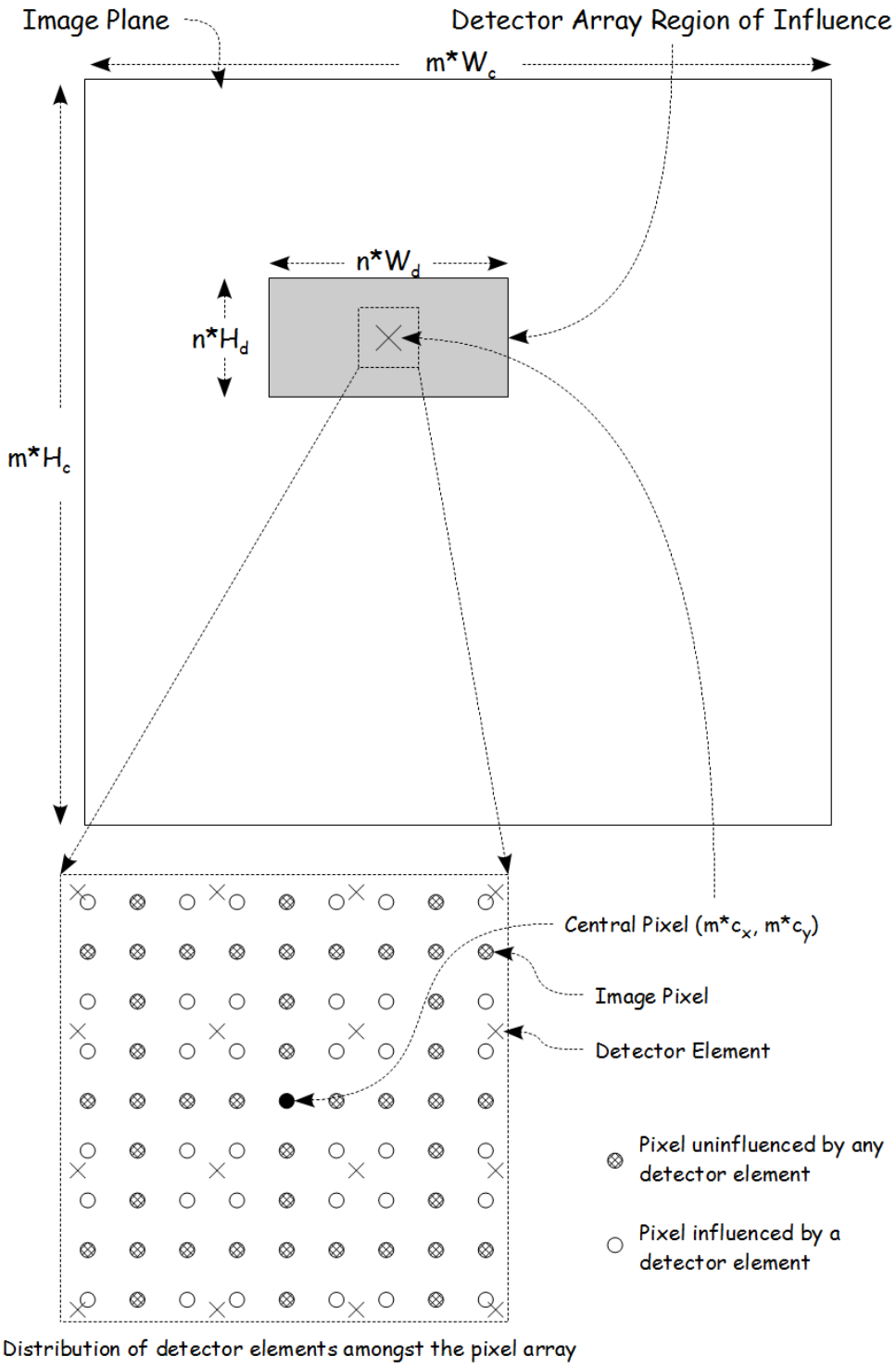

Figure 3. Global view of the image reconstruction algorithm

What changes from one focal plane to another is the extent of the ROI centered on ($m*c_x$, $m*c_y$) for the given collimator hole. In other words, the specific 2x2 pixel grid and the weights corresponding to the four pixels that are affected by a specific collimator hole and detector element combination changes from one focal plane to next. Since back-projection onto a plane is a linear operation, the change

manifests as a linear scaling of the ROI and this scaling factor (also called *reconstruction ratio* in [2]) is denoted by the parameter n. Thus, the size of the ROI for a given focal plane is ($n*W_d$, $n*H_d$) for any collimator hole. Varying n is thus synonymous with changing the focal plane of the image, and the P image planes are therefore reconstructed using P different n values. Figure 3 illustrates the global view of how the collimator and the detector arrays influence a given image plane through the parameters m and n. In our experiments we have held the value of m constant at 10 for each plane. However, the value of n is varied over a range from about 0.6 to 3.8 as one moves from one focal plane to another.

The local view from figure 2 and the global view from figure 3 can be combined to produce the following set of equations that describe the operation of the reconstruction algorithm. For a given combination of collimator hole (c_x, c_y) and detector element (d_x, d_y), the affected pixels of the 2x2 grid are given by

$$(I_x, I_y), (I_x + 1, I_y), (I_x, I_y + 1) \text{ and } (I_x + 1, I_y + 1)$$

where

$$I_x = floor(m * c_x + n * (d_x - (W_d - 1)/2)) \quad (1)$$
$$I_y = floor(m * c_y + n * (d_y - (H_d - 1)/2)) \quad (2)$$

It follows that,

$$f_x = m * c_x + n * (d_x - (W_d - 1)/2) - I_x \quad (3)$$
$$f_y = m * c_y + n * (d_y - (H_d - 1)/2) - I_y \quad (4)$$

From equations (1) to (4), it is clear that the problem is separable i.e. the influence of back-projection in the x dimension via I_x and f_x depends only on the x indices of the collimator holes and detector elements while the influence in the y dimension via I_y and f_y depends only on their y indices.

Since m is a constant integer, we can rewrite equations (1) – (4) as

$$I_{xoff} = I_x - m * c_x = floor(n * (d_x - (W_d - 1)/2)) \quad (5)$$
$$I_{yoff} = I_y - m * c_y = floor(n * (d_y - (H_d - 1)/2)) \quad (6)$$
$$f_x = n * (d_x - (W_d - 1)/2) - floor(n * (d_x - (W_d - 1)/2)) \quad (7)$$
$$f_y = n * (d_y - (H_d - 1)/2) - floor(n * (d_y - (H_d - 1)/2)) \quad (8)$$

where (I_{xoff}, I_{yoff}) denotes the offset of pixel (I_x, I_y) from the center of the ROI ($m*c_x$, $m*c_y$). From equations (5) and (6), it follows that the offset (I_{xoff}, I_{yoff}) of a pixel from the center of the ROI of a given collimator hole and which is affected by a given detector element, is independent of the pixel location and the collimator hole, and depends only on the location of the detector element. Similarly, equations (7) and (8) imply that the bilinear coefficients (f_x, f_y) used to weigh the output of a given detector element, also depend only on the location of the detector element and are independent of the location of the pixel that is affected by the detector element as well as the given collimator hole.

3.1 Detector centric Parallelization

These observations form the basis for a detector centric parallelization scheme in which a single parallel processing thread handles a single detector element, apportioning it's measurement to the

neighborhood image pixels. The processing can be accelerated by constructing a look-up table (LUT) to encode for those computations that are repeated in different processing threads at different times. Using equations (5) – (8) the LUT encodes for each detector element the offsets of the pixels affected by it, and the bilinear coefficients that are used to weigh its output. Due to separability, it suffices to have two 1D LUTs, one indexed by d_x and encoding I_{xoff} and f_x and the other indexed by d_y and encoding I_{yoff} and f_y respectively. The total size of this pair of 1D LUTs is proportional to $(W_d + H_d)$ and one such pair is needed for each of the focal planes being reconstructed.

3.2 Pixel centric Parallelization

Alternatively, it is possible to view the parallelization of the reconstruction algorithm from a pixel centric perspective. From the point of view of a pixel, if the pixel lies within the ROI of a given collimator hole, it is likely, but is by no means certain, that it would be affected by the output of a detector element. In other words, while every pixel that is affected by a collimator hole lies within its ROI, not every pixel within the ROI of a collimator hole is necessarily affected. This is because as the focal plane changes and the value of n increases, the ROI gets stretched which may result in some pixels being beyond the spread of any detector element (illustrated in figure 3). Conversely when the value of n decreases, the ROI shrinks and as a result, a pixel within the ROI can be affected by several detector elements. Since our detector elements are located on a regular grid, depending on the value of n, the number of detector elements affecting a particular pixel within the ROI can be either 0 or any power of 2 – 1, 2, 4 ….

It is possible, therefore, to have a single parallel processing thread handle a single pixel and to accumulate the measurements of all those detector elements that affect it. As in the detector centric case it is possible to build a LUT to encode for all those computations that get repeated in different processing threads at different times. The LUT from a pixel centric parallelization perspective has an entry for every possible offset within the ROI of a collimator hole for a given focal plane. Once again, separability allows us to decouple the x and y dimensions. Thus the LUT for x dimension is indexed by I_{xoff} and the entry indicates, using equation (5), whether the pixel that lies at the corresponding offset is affected by a detector element or not. If it is affected by a detector element, then it also stores, using equations (5) and (7), the x indices of the detector elements that affect it and the bilinear coefficients used to weigh their outputs. Similarly a LUT for the y dimension can be constructed by substituting equations (6) and (8) in place of equations (5) and (7) respectively. Unlike the detector centric LUTs, the pixel centric LUTs have a different size for each focal plane of the image, which is $n(W_d + H_d)$.

4 The Computing Platform and Programming

The detector centric and pixel centric parallelization schemes have been implemented on three Nvidia GPUs of the Kepler family, namely the GeForce GTX690, the Tesla K10 and the Tesla K20. The various architectural and compute features of these GPUs are given in table 1. All three GPUs are parallel throughput processors based on a single instruction multiple data (SIMD) architecture.

Comparing the three GPUs becomes complicated due to the fact that the GeForce GTX690 and the Tesla K10 have two GPUs on a single card, but the Tesla K20 has only one. As can be seen from table 1, the Tesla K20 has the lowest GPU clock rate. However, it has the highest number of cores per GPU, the largest memory bus width and the largest L2 cache which is shared among all the streaming multiprocessors in the Kepler architecture. The GeForce GTX690 has the highest memory clock rate. However since both the GeForce GTX690 and the Tesla K10 have two GPUs on the same card, they

have a larger number of cores overall.

Since all our computation is single precision floating point (SPFP), of particular interest to us is the SPFP compute capability of the three GPUs. The Tesla K20 has the highest SPFP compute capability compared to single GPUs of others. However the overall SPFP compute capability of the GeForce GTX690 and the Tesla K10 is higher due to them being dual GPU cards.

GPU	**Tesla K10**	**Tesla K20**	**GeForce GTX690**
Specification	**1 card 2 GPUs**	**1 card 1 GPU**	**1 card 2 GPUs**
Number of cores	2 * 1536	2496	2 * 1536
GPU Clock rate (MHz)	745	704	1020
Memory Clock rate (MHz)	2500	2600	3004
Memory Bus Width (bit)	256	320	256
Maximum Number of threads per Multiprocessor	2048	2048	2048
Maximum Number of threads per block	1024	1024	1024
Global Memory (GB)	2 * 4	5	2 * 2
Peak Single Precision Floating Point Performance (TFLOPS) for card	4.58	3.52	5.62
Peak Double Precision Floating Point Performance (TFLOPS) for card	0.19	1.17	0.19
L2 Cache (bytes)	524288	1310720	524288
Shared Memory per block (bytes)	49152	49152	49152

Table 1. GPU Specifications

The programming has been done using the NVIDIA CUDA parallel programming model. CUDA is an extension of the C language, and enables parallel programming in the SIMD paradigm. CUDA provides APIs for launching functions (called kernels) on the GPU cores, copying data between the host CPU's memory and the GPU memory, and for using different types of memory on the GPU card. The same image reconstruction CUDA program was run on all the three GPUs, no specific optimizations were done for any of the GPU.

The detector measurements are stored in the GPU's texture memory in order to exploit its 2D locality based cache and independent texture hardware based access. At the start a kernel (the LUT generation kernel) executed on the GPU generates the various LUTs for all the planes. During image reconstruction, the LUTs are stored in the shared memory for faster access. The image reconstruction kernel follows the LUT generation kernel and differs depending on the parallelization scheme.

4.1 Image Reconstruction Kernel: Arithmetic Intensity

In the detector centric approach, a parallel processing thread handles a detector element, apportioning it's measurement to neighboring image pixels. The updating of the image pixels results in scattered writes to global memory leading to write contests. The considerably larger number of writes to global memory in the detector centric scheme is responsible for this kernel having poor arithmetic intensity – the ratio of compute operations to memory operations executed by a thread. Moreover, it is cumbersome to scale on multi-GPUs, as a post-processing step is necessary to merge the partial image pixels reconstructed by individual GPUs.

In the pixel centric approach, each parallel processing thread handles an image pixel, accumulating the contributions of neighboring detector measurements. The partial pixel updates due to detector measurements are accumulated in a register and written to the global memory at the end. This reduces the total number of memory writes which in turn reduces write contests and enhances arithmetic intensity significantly. Also, by eliminating the need for a post-processing step, this approach becomes amenable to multi-GPU scaling.

```
FOR a THREAD reconstructing a pixel at (Ix, Iy)
        COPY LUTs lut_det_loc, lut_ fx and lut_ fy into shared memory
        DETERMINE coll_bnd_box Cxmin, Cxmax, Cymin, Cymax contributing to
        pixel at (Ix, Iy)

        FOR each col_hol within (Cxmin, Cxmax) and (Cymin, Cymax)
                COMPUTE pixel_offset from the center of the ROI of the col_hol
                READ det_loc from LUT lut_det_loc stored at pixel_offset

                IF det_loc has a valid element affecting pixel (Ix, Iy), THEN
                        READ bilinear coefficients fx, fy from LUTs lut_fx, lut_ fy stored at
                        pixel_offset
                        READ the detector element located at det_loc from texture memory
                        INCREMENT image pixels at (Ix, Iy), (Ix+1, Iy), (Ix, Iy+1),
                        (Ix+1, Iy+1) using fx, fy and the detector data
                ENDIF

        ENDFOR
ENDTHREAD
```

Table 2. Pseudo code of the pixel centric parallelization scheme

Table 2 contains a pseudo code of the pixel centric parallelization scheme. From the pseudo code we can see that initially a block of threads populate the shared memory with LUTs containing bilinear coefficients and detector locations. Then, each thread computes a bounding box of collimator holes that affect the image pixel it represents. For each collimator hole in the bounding box, the thread identifies the detector elements influencing it and uses the bilinear coefficients to weigh and accumulate the detector element's measurement. At the end of the loop, the accumulated image pixel is updated in the global memory. A number of heuristics have been used to speed up the processing of the pixel centric kernel. We describe some of them in the following subsections.

4.2 Using Separability to Skip Rows

Due to stretching of the ROI for values of $n > 1.0$, there exist image pixels that are not affected by any detector elements for some collimator holes. In particular, since our detector array is regular and separable, if one collimator hole does not affect a given pixel (i.e. the bilinear weight corresponding to it is zero) in the y direction, then none of the collimator holes in the entire row that contains it, affect the pixel. This happens regularly for $n > 1.0$ and allows us to skip entire rows of collimator holes within the bounding box during execution of a thread. This simple application of separability leads to a significant acceleration of the pixel centric image kernel on the GPU.

4.3 Tuning the Thread-Block Configuration

The thread computations on the GPU is a cascade of arithmetic operations and memory read-writes, among many parallel threads. The GPU distributes arithmetic operations among the on-chip compute resources, while the memory access has to pass through a hierarchy of cache pipelines. The relative balance between compute and memory operations influences *achieved occupancy* which can critically affect overall GPU performance. Programmers employ the heuristic of changing the number of threads and thread blocks in order to arrive at a configuration of optimal achieved occupancy. This is a trial-and-error method and in our case is constrained by the fact that the product of the thread counts and thread block counts must equal the total number of reconstructed pixels of all the planes (for the pixel centric scheme only). The results of this fine-tuning may differ for different GPUs, as individual GPUs work at different clock frequencies and have different amounts of compute resources.

4.4 Two Phase Pipelined Computation

In real world scenarios, the detector data would be generated continuously and will need to be processed at the same rate. In order to simulate this process we have set up a two phase pipelined computation model using CUDA streams and two memory banks each for input and output data which allows overlapping data transfers and GPU computations. The data transfers rely on asynchronous memory copy using the DMA protocol over the PCIe channel and do not interfere with the working of the CPU or the GPU.

The cycle of events in two phase computation is as follows: Initially a frame of detector data is copied from the host CPU memory to the GPU memory. The GPU then begins processing the detector data, and simultaneously the next frame of detector data is copied from the host to the GPU into a separate memory location. After completion of the data processing, the image pixels are copied to the host from the GPU, while the processing of the next frame of detector data begins on the GPU. As long as the cumulative time for transfer of detector and image pixel data is less than the time taken for processing of a frame of detector data there is no latency in the image reconstruction pipeline and the throughput equals the GPU compute time.

5 Experimental Results

The image reconstruction implementations were run using simulated detector data and number of planes $P = 32$. Each reconstructed plane was composed of 1000x1000 pixels, each pixel being a 32-bit floating point number. The detector measurements were simulated as 4-bit random data (range of 0 to 15) and stored as an 8-bit unsigned character. The compute intensity can be calculated from the detector centric formulation as follows – (8 multiplications + 4 additions) x 160x80 detector elements x 100x100 collimator holes x 32 planes x 30 fps = 1.5 TFLOPS.

The GPU results were validated by comparing the pixels of the 32 image planes with those obtained from a sequential implementation on the CPU. The maximum error was found to be 0.012% and can be attributed to the differing sequence of multiplications and additions in the sequential and the parallel implementations, and consequently the errors that arise as a result of truncations during floating point computations.

In tables 3 and 4, we present the run times for the detector centric and the pixel centric parallelization

schemes respectively. Each row in the tables corresponds to a range of *n* used. The ranges that we have selected help illuminate interesting features of the algorithm. The first range of 0.6 to 2.25 is wide and most diverse because it involves both $n < 1.0$ as well as $n > 1.0$, and therefore the LUTs have to deal with situations where more than one detector element may contribute to a 2x2 grid of image pixels. The other ranges result in a maximum of one detector element contribution to a 2x2 grid of image pixels, but as the upper limit of *n* keeps increasing, the LUTs used in the pixel centric scheme become larger yet increasingly sparser.

Range of *n*	**Run time on GeForce GTX690 (millisecond)**	**Run Time on Intel Quad core Q8400 (sec)**	**Speed Up**
0.60 – 2.25	1029.64	394.85	383.5
1.00 – 1.50	1039.86	366.90	352.8
1.00 – 2.01	1003.45	387.36	386
1.00 – 3.82	966.57	443.21	458.5

Table 3. Reconstruction Run Times for Detector Centric Scheme

Range of *n*	**Run Time on GPU (millisecond)**						**Best case Speed Up**
	Tesla K10 (2GPUs/card)		**Tesla K20 (1 GPU/Card)**		**GeForce GTX690 (2 GPUs/card)**		
	Single	**Dual**	**Single**	**Dual**	**Single**	**Dual**	
0.60 – 2.25	155.16	77.11	152.26	74.89	126.55	62.98	6269.5
1.00 – 1.50	147.10	73.18	145.33	71.60	111.02	54.32	6754.4
1.00 – 2.01	165.70	82.32	162.46	79.90	125.50	59.76	6481.9
1.00 – 3.82	227.84	95.66	223.60	109.23	174.94	80.94	5475.5

Table 4. Reconstruction Run Times for Pixel Centric Scheme

Kernel Configuration			**Single GPU Run Time (millisecond)**		
Threads	**Blocks**	**Shared Memory (bytes)**	**Tesla K10**	**Tesla K20**	**GeForce GTX690**
999	(999, 32)	9990	155.18	152.72	117.21
333	(999 * 3, 32)	4995	166.35	125.64	135.98
320	(999 * 4, 32)	4800	162.79	132.66	131.37
256	(999 * 4, 32)	6400	160.71	115.43	131.43
224	(999 * 5, 32)	5600	166.88	127.42	131.43
205	(999 * 5, 32)	5125	162.88	130.84	133.18

Table 5. Reconstruction Run Times for Different Thread and Thread Block Configuration, *n* range: 0.6 – 2.25

In these experiments the number of threads and thread blocks are kept constant for all ranges of *n* and for all the three GPUs considered. The detector centric scheme has been implemented only on a single GPU of the GeForce GTX690 since it is clear from the numbers that it is unlikely to be better than the pixel centric scheme. The pixel centric image reconstruction CUDA code on the other hand was run on

all the three GPUs in single and dual GPU modes. In the case of the GeForce GTX690 and the Tesla K10, both the GPUs on their cards were used, and in the case of the Tesla K20 two cards were employed. In the dual GPU experiments, each plane was divided into two halves of size 500x1000 and reconstructed separately on one of the two GPUs. The detector arrays necessary for individual reconstructions were copied into the respective GPU's global memories. The two phase pipelined computation was implemented with dual GPUs and found to work with no additional latency.

In table 4, we report the run times of both single and dual GPU experiments and we can see that the run times in the case of dual GPU is half of that of the single GPU demonstrating that the implementation is scalable. Also the performance of the image centric scheme has been contrasted with that of a general purpose CPU and we can see a speed-up factor ranging from approximately 5400 to 6700 depending on the range of *n*. For the diverse range of 0.6 to 2.25, a single GeForce GTX690 card with two GPUs achieves 15 fps.

From table 4, we observe that the GeForce GTX690 outperforms both the Tesla K10 and the Tesla K20. One factor that explains this difference is the higher GPU clock rate of the GeForce GTX690 compared to the Tesla K10 and the Tesla K20. Another possible factor could be the error correcting code (ECC) option available in the Tesla K10 and the Tesla K20 to safeguard the memory against corruption. Nvidia documents mention the slight hit in DRAM bandwidth when ECC option is used. We conducted the image reconstruction experiments by both enabling and disabling ECC, but found no change in the run-time performance.

From table 4, it also appears that the Tesla K20 despite having a higher single GPU SPFP compute capability compared to the other two GPUs, performs no better than a single GPU of the Tesla K10 and is worse than a single GPU of the GeForce GTX690. However, by tuning the number of threads and thread blocks, it is possible to improve the Tesla K20's performance. Table 5 presents the run times for a single GPU, by changing the thread-block configuration for the *n* value range of 0.6 to 2.25. The first two columns in table 5 show the thread count per block and the number of blocks in 2D format respectively. As can be seen, the product of the threads and blocks approximate to the total number of reconstructed pixels i.e. 32 million. As the number of threads in a block changes it also changes the number of LUT entries that need to be copied into the shared memory. This is reflected in the third column as the number of bytes copied into the shared memory from the LUTs by all the threads in a block.

The fastest run time for the GeForce GTX690 and the Tesla K10 are 117.21 ms and 155.18 ms respectively and are obtained for the thread-block configuration shown in row 1. The compute architectures of the GeForce GTX690 and the Tesla K10 are the same albeit running at different clock frequencies, thus yielding different run times with the same thread-block configuration. The best run time for the Tesla K20 is 115.43 ms with the configuration shown in row 4. From this table it is clear that an appropriate thread-block configuration can raise the Tesla K20's performance to the same level as a single GPU on the GeForce GTX690 (the GeForce GTX690 still has an advantage over the Tesla K20 since it has two GPUs on the same card). However, we do not see a similar improvement in the case of the Tesla K10.

6 Conclusions

The multi-plane tomosynthetic image reconstruction algorithm used in the SBDX fluoroscopy system was parallelized along detector centric and pixel centric formulations corresponding to the scatter and

gather formulations of other medical imaging problems. They were implemented on three variants of Nvidia Kepler GPU – the GeForce GTX690, the Tesla K10 and the Tesla K20. The implementation employed various heuristics and optimizations – LUTs were used to store detector locations and bilinear coefficients and helped avoid repeated computations; shared and texture memory were used for effective memory management by storing different kinds of data; detector regularity and separability was leveraged to reduce the size of the LUTs as well as reduce the number of computations; and achieved occupancy was improved by tuning the thread-block configuration.

The pixel centric parallelization scheme has better arithmetic intensity and was found to be superior in run time to the detector centric one by a large margin. Furthermore, it was found to be readily scalable onto dual GPU systems with no drop in performance and without posing any logistical difficulty. The performance was seen to scale linearly with the number of GPUs. The fastest implementation was seen in the case of the GeForce GTX690 giving a real time performance of more than 15 fps for the reconstruction of 32 planes each comprising of 1 million image pixels. We can conclude therefore, that partitioning the image into two halves and reconstructing them simultaneously on two GeForce GTX690 boards would double the frame rate to 30 fps.

An important assumption made at the beginning of this work is the regularity of the detector array. The regularity assumption translates into separability of the computations along the x and y dimensions which in turn allows for much smaller LUTs than would be needed. However the actual detector array topology can deviate from this assumption and therefore future developments must figure out a way to extend the parallelization schemes (in particular the pixel centric scheme) to the case of an irregular detector topology.

Some of the other directions for taking this work further include the multi-plane compositing procedure that reduces the multiple planes to a single image viewable by a medical professional. Also, the ability to change depth and location of the reconstructed planes by changing the range of n will be an important addition for real world applications of fluoroscopy.

Acknowledgments

The authors thank Nvidia for providing access to their server nodes with Tesla K10 and Tesla K20 boards for running the reconstruction experiments.

References

[1] E.G. Solomon, B.P. Wilfley, M.S. Van Lysel, A.W. Joseph and J.A. Heanue, 'Scanning Beam Digital X-ray (SBDX) System for Cardiac Angiography', Proceedings of SPIE Physics of Medical Imaging Conference, February 1999.

[2] M.A. Speidel, B.P. Wilfley, J.M. Star-Lack, J.A. Heanue and M.S. Van Lysel, 'Scanning Beam Digital X-ray (SBDX) Technology for Interventional and Diagnostic Cardiac Angiography', Medical Physics, Vol. 33, No. 8, August 2006.

[3] M.A. Speidel, B.P. Wilfley, J.M. Star-Lack, J.A. Heanue, T.D. Betts and M.S. Van Lysel, 'Comparison of Entrance Exposure and Signal-to-Noise Ration between an SBDX Prototype and a Wide-Beam Cardiac Angiographic System', Medical Physics, Vol. 33, No. 8, August 2006.

[4] G. Pratx and L. Xing, 'GPU Computing in Medical Physics: A Review', Medical Physics, Vol. 38, No. 5, May 2011.

[5] J.A. Heanue, D.A. Pearson and R.E. Melen, 'CdZnTe detector array for a scanning-beam digital x-ray system', Proceedings of SPIE Physics of Medical Imaging Conference, February 1999.