

Implementation of a cone-beam backprojection algorithm on the Cell Broadband Engine processor

Olivier Bockenbach¹, Michael Knaup², and Marc Kachelrieß²

1 Mercury Computer Systems, GmbH, Lepiusstr. 70, 12163 Berlin, Germany (0) 30 700 968 0

2 IMP, University of Erlangen-Nürnberg, Henckestr. 91, 91052 Erlangen, Germany

ABSTRACT

Tomographic image reconstruction is computationally very demanding. In all cases the backprojection represents the performance bottleneck due to the high operational count and due to the high demand put on the memory subsystem. In the past, solving this problem has led to the implementation of specific architectures, connecting Application Specific Integrated Circuits (ASICs) or Field Programmable Gate Arrays (FPGAs) to memory through dedicated high speed busses. More recently, there have also been attempts to use Graphic Processing Units (GPUs) to perform the backprojection step.

Originally aimed at the gaming market, IBM, Toshiba and Sony have introduced the Cell Broadband Engine (CBE) processor, often considered as a multicomputer on a chip. Clocked at 3 GHz, the Cell allows for a theoretical performance of 192 GFlops and a peak data transfer rate over the internal bus of 200 GB/s. This performance indeed makes the Cell a very attractive architecture for implementing tomographic image reconstruction algorithms.

In this study, we investigate the relative performance of a perspective backprojection algorithm when implemented on a standard PC and on the Cell processor. We compare these results to the performance achievable with FPGAs based boards and high end GPUs.

The cone-beam backprojection performance was assessed by backprojecting a full circle scan of 512 projections of 1024x1024 pixels into a volume of size 512x512x512 voxels. It took 3.2 minutes on the PC (single CPU) and is as fast as 13.6 seconds on the Cell.

Keywords: Computed Tomography, Backprojection, Cell Broadband Engine, GPU, FPGA

1 INTRODUCTION

Cone-beam image reconstruction as proposed by Feldkamp¹ is computationally very demanding. In most analytical methods the backprojection represents the performance bottleneck due to the high operational count and to the resulting high demand put on the memory subsystem. In the past, solving this problem has led to the implementation of specific architectures, connecting ASICs or FPGAs⁴ to memory through dedicated high speed busses. More recently, because those devices offer a very high memory bandwidth, there have also been attempts to use graphic processing units (GPUs) to perform the backprojection step⁵.

The aim of this investigation is to implement a 3D cone-beam perspective backprojection algorithm for the Cell processor and to benchmark its performance against other alternatives, such as PC, FPGA or GPU-based implementations.

The paper is organized as follows. Section 2 introduces the perspective backprojection algorithm. Analytical expressions and implementation details are discussed. Specifically, this section contains the description of the implementation alternatives that can influence the performance of the backprojection step. In addition, section 2 also contains the description of the different hardware platforms. Section 3 provides the performance assessment method and the performance values achieved with the different hardware platforms. Section 4 discusses the results obtained with alternative architectures, based on FPGAs and GPUs.

2 METHODS AND MATERIAL

2.1 Hardware

2.1.1 PC platform

Our reference backprojection implementations run on a standard PC with a single Xeon processor clocked at 3.06 GHz and a front bus side at 533 MHz.

2.1.2 CBE board

Several Cell based products are now available on the market, ranging from standalone servers to PCI-Express (PCIe) acceleration cards, in single processor or dual SMP processor configurations. We have selected to use the Cell Accelerator Board (CAB) from Mercury Computer Systems for the purpose of this study (Figure 1). The board can be plugged into a standard PC with adequate cooling and power supply. Running a Yellow Dog Linux, the Cell is accessible over standard communication schemes through the Gigabit Ethernet (GE) link or over PCIe.

The board is build around a powerful South Bridge allowing for simultaneous data transfer transactions to take place between the CBE processor, main memory, GE and PCIe. All transactions can run at the maximum speed allowed by the medium or bus device. The CBE processor has access to 1 GB of RamBus memory (XDR) for random access and Direct Memory Access (DMA). In addition, it can use up to 8 GB of DDR2 memory through DMA only.

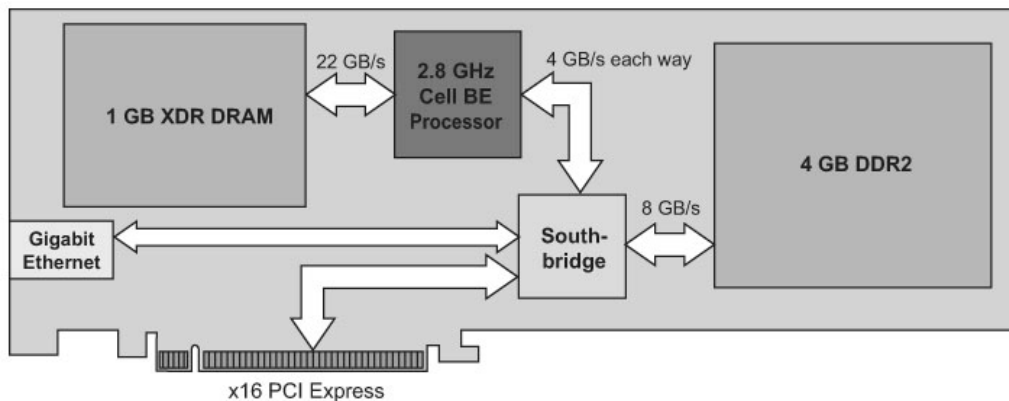


Figure 1: Structure of the Cell Accelerator Board

The CBE³ articulates one PowerPC Element (PPE) and eight Synergistic Processing Elements (SPEs) around a high speed Element Interconnect Bus (EIB) and is therefore considered as a multicomputer on a chip (Figure 2). Clocking at 3 GHz, the CBE allows for a theoretical performance of 192 GFlops and a peak data transfer rate over the internal bus of 200 GB/s.

This performance makes the CBE a very attractive architecture for implementing tomographic image reconstruction algorithms. Each SPE consists of one Synergistic Processing Unit (SPU) with 256 KB of Local Store (LS) and a Memory Flow Controller (MFC) that notably allows for asynchronous DMA operations. The Coherent Interface allows for SMP operations for dual CBE configuration but is beyond the scope of this study since we are running in single processor mode.

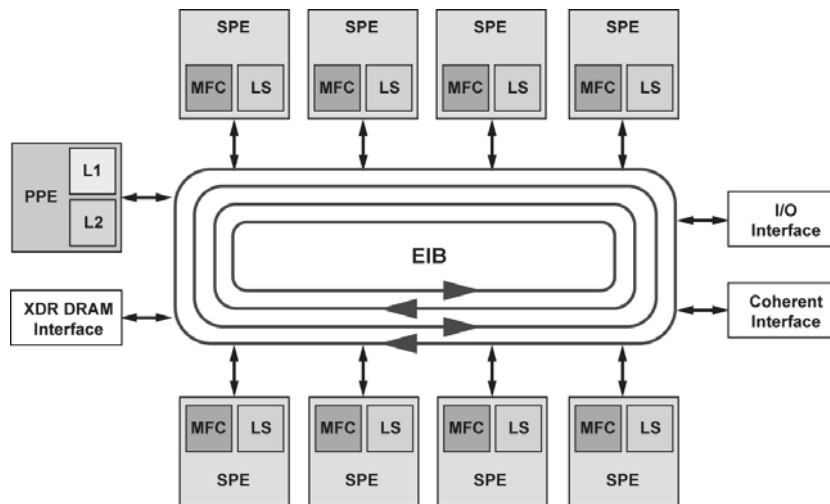


Figure 2: Architecture of the CBE Processor

2.1.3 FPGA based board

Mercury Computer Systems introduced in 2003 an FPGA-based system with one FPGA and two PowerPC (PPC) 7410, packaged as a PCI board. Each PPC has up to 256 MB of SDRAM and a compute node ASIC for assistance in data movements (Figure 3). Several boards could be combined for a scalable system, linked by a RACE++ interconnect.

Typical applications running on this board would use the PPC processors for controlling the processing done on the FPGA, handling exceptional processing cases and dealing with pre and post processing tasks. Data can be efficiently transferred between the different actors due to the presence of powerful DMA engines in the CN ASIC as well as in the FPGA.

The FPGA Compute Node consists of a Xilinx Virtex-II chip directly connected to the RACE++ interconnect fabric. In addition, the FPGA has access to 2 SRAM banks and 4 SDRAM banks clocked at 133 MHz.

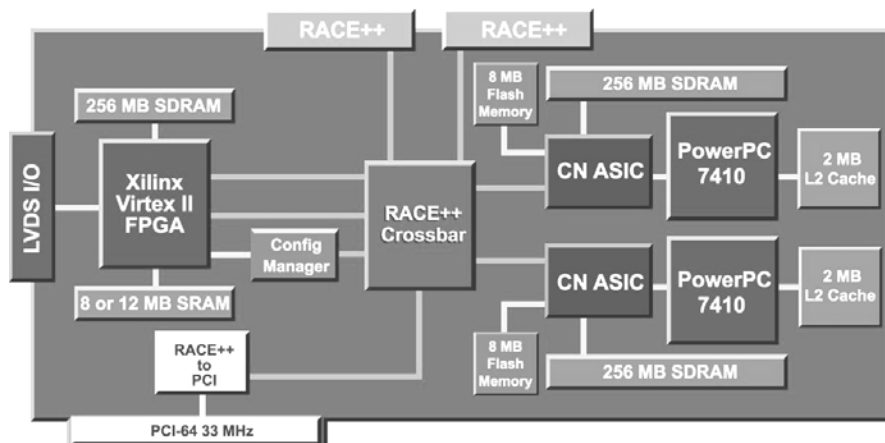


Figure 3 : Architecture of the VantageRT- FCN

The internal structure of a Virtex-II FPGA is depicted on Figure 4. It basically consists of a collection of elementary components tied together by a fast grid interconnect. Those components are of various predefined types : Configurable Logic Blocks (CLB), Block RAMs (BRAM) and fast multipliers. BRAMs can be used as data caches to hold projections or parts of the reconstructed volume. CLBs are intended to provide support for all logical and arithmetic operations.

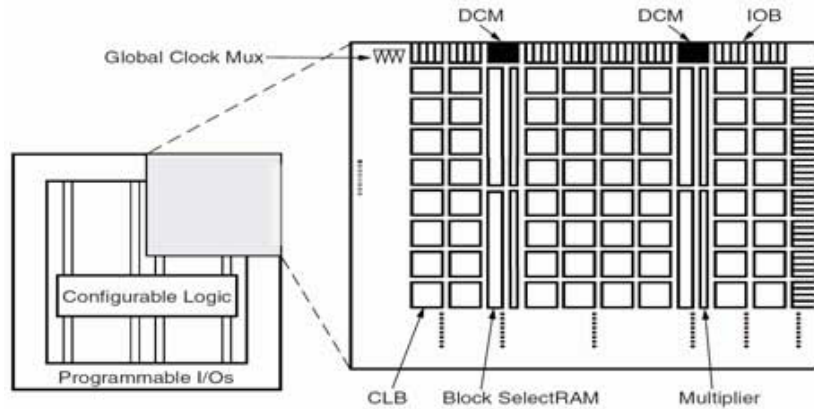


Figure 4: Block Diagram of the Xilinx Virtex-2 FPGA

2.1.4 GPU board

For the purpose of this study we have used an nVidia G70 based board whose block diagram is depicted on Figure 5. The GPU is connected to the host PC processor through a PCIe bus and receives the data and commands strings over this bus. Those command strings are typically originated by the host application as rendering actions, specifying textures and vertex data. They are translated on the fly by the appropriate GPU driver software.

The vertex shaders are located first stage of the pipeline to perform per vertex transformations and are hence connected to texture and vertex caches. The second stage of the pipeline consists of a collection of pixel shaders or fragment processors. They apply the same program to a collection of pixels in parallel.

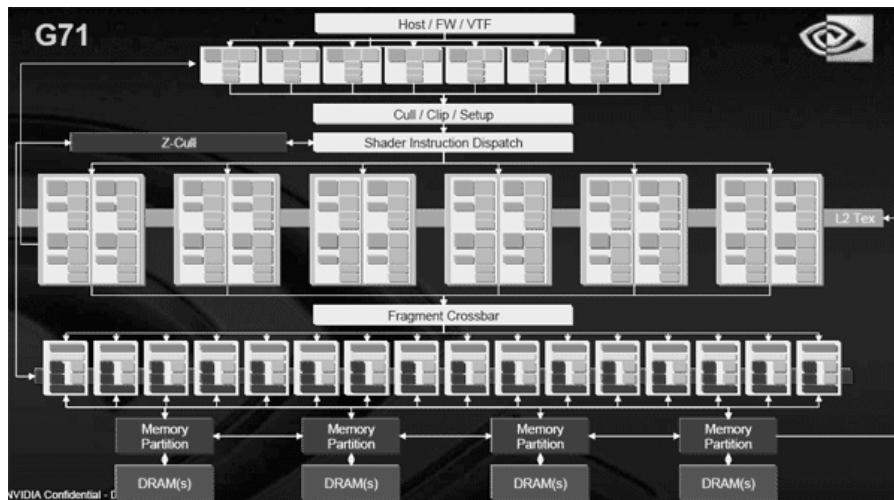


Figure 5 : Block diagram of the G70 nVidia GPU

2.2 Algorithm

We consider a cone-beam backprojection of type

$$f(r) = \int d\alpha \omega^2(\alpha, r) p(\alpha, u(\alpha, r), v(\alpha, r))$$

With

$$u(\alpha, r) = (c_{00}x + c_{01}y + c_{02}z + c_{03})\omega(\alpha, r)$$

$$v(\alpha, r) = (c_{10}x + c_{11}y + c_{12}z + c_{13})\omega(\alpha, r)$$

$$\omega(\alpha, r) = \frac{1}{c_{20}x + c_{21}y + c_{22}z + c_{23}}$$

Where

$$c_{ij} = c_{ij}(\alpha)$$

Here, f is the reconstructed volume, p is the projection data sampled by the detector along the trajectory, $r = (x, y, z)$ denotes the voxel location, α is the trajectory parameter and u and v are the detector coordinates of voxel r at projection angle α .

The coefficients $c_{ij}=c_{ij}(\alpha)$, that define the perspective transform from the detector into the volume, are arbitrary functions of the projection parameter α , in general. And $\omega(\alpha, r)$ defines the distance weight function.

A direct implementation of the Feldkamp algorithm determines the u and v coordinates of the voxel $r = (x, y, z)$ for a given projection geometry, performs a bilinear interpolation of the four neighbor pixels to compute the contribution of this specific projection to the considered voxel r .

3 IMPLEMENTATION

3.1 Implementation principles

The reconstruction of the volume can be implemented in many different ways; for instance, one could select to process the voxels following x , y or z as the primary axis of processing. The z dimension offers most interesting properties in terms of optimization of the processing and was selected as the primary processing axis.

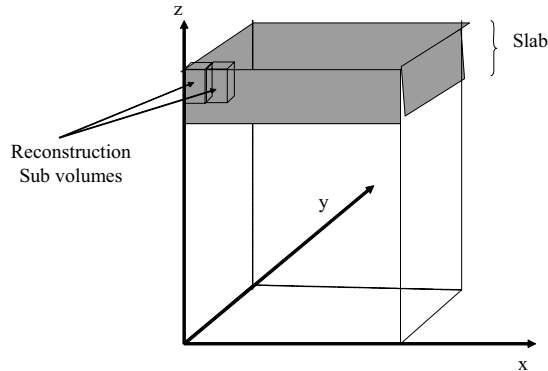


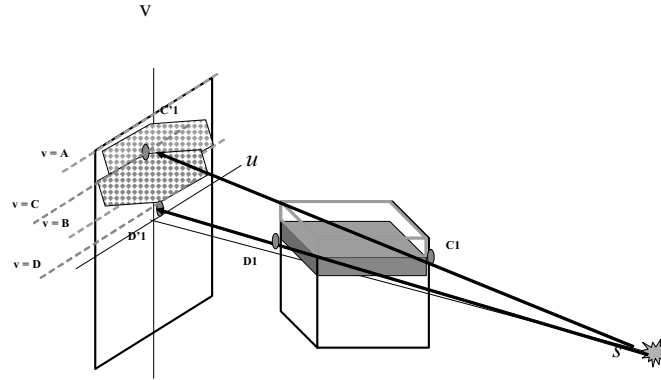
Figure 6: Decomposing the volume in Slabs

Similarly, processing the projections to reconstruct the global volume makes best use of the hardware resources (e.g. registers for processors, BRAM for FPGAs) when it is carried out on sub volumes of regular shape such as the cube.

For the above mentioned reasons, the overall reconstruction method is based on the division of the volume to reconstruct in slabs, as shown Figure 6, each slab being processed as small cubes.

The complete reconstruction of a slab requires all the projections. However, each slab does not require complete full-sized projections. As shown in Figure 7, the surface needed to reconstruct a slab is contained in a rectangular shape. When the side of the reconstruction volume is parallel to the detector plane, the projection of the slab of slices is a rectangle. At other projection angles, the projection of the slab is a hexagon. The height of the hexagon is greatest when

the diagonal of the slices is perpendicular to the detector plane. The height of the hexagon is also largest at the top and



bottom of the reconstruction volume.

Figure 7: Subset of projection data

The software has been designed to take advantage of these properties to reconstruct the complete volume. Therefore, only the relevant surface of the projections is used for the reconstruction of one slab.

Rectification-based or hybrid methods are in use to speed up the backprojection process². They have demonstrated significant performance gains over direct methods. The trick consists in taking advantage of the resampling and bilinear interpolation of the projection data to realign it to an ideal detector geometry (Figure 8), in order to only use a nearest neighbor approach during the backprojection, saving a significant number of cycles per point.

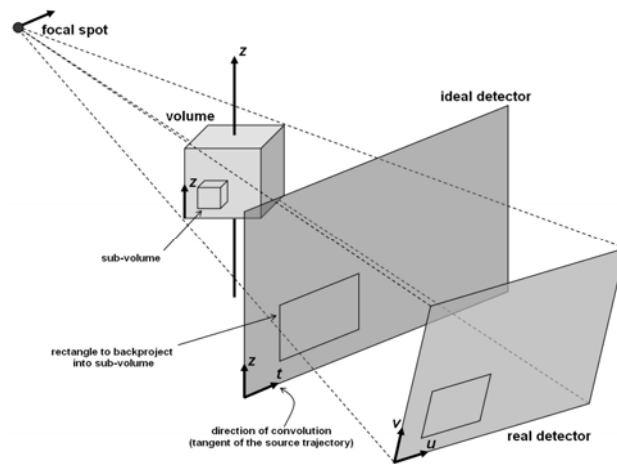


Figure 8: Rebinning from real to ideal detector

3.2 PC Reference

The PC-based code implementation is of the hybrid kind in terms of first performing a detector alignment, based on up-sampling and bilinear interpolation, followed by a voxel-driven backprojection based on nearest neighbor interpolation.

The proposed platform can backproject 512 projections on a 512^3 volume in 3.21 minutes.

3.3 FPGA platform

FPGAs do not hold dedicated hardware for performing the interpolation. However, most of the FPGA chips have significantly more processing power than IO capabilities. Therefore, it is usually more efficient to pay the extra cost of performing the interpolation than to use a hybrid method that inherently loads the IO busses more than direct methods.

Implementing the backprojection on the Mercury platform requires several different steps. Since the total space of the memory accessible by the FPGA cannot hold all the projection data and all of the reconstructed volume, the raw data has to be cut into slabs. The data transfers can be organized in such a way that they overlap with the processing of the previous slab. Once the raw data is present in local memory, projection data is brought into the BRAM of the FPGA; several projections can be handled at the same time and multi buffering can be used for overlapping the data transfers with processing time. FPGA internal BRAMs also hold small cubes from the output volume onto which the raw data is projected on. Once all the cubes from a slab have been processed, the considered slab is sent back for visualization and the FPGA program resumes execution on the next slab, if any.

A complete description of the first implementation can be found in [4]. This platform performs the reconstruction in fixed point math and is capable of backprojecting 512 projections on a 512^3 volume in ~ 25 seconds.

3.4 GPU platform

Graphical image processing requires extraordinary data resampling capabilities. Therefore, most modern GPUs assist the processing elements with specialized circuitry for performing interpolations. Consequently, it makes little sense to try to accelerate the backprojection through the use of hybrid methods.

Implementing the backprojection step on a modern GPU like an nVidia G70 consists in loading the relevant part of each projection as texture data. Newest GPUs now have enough memory to hold both a 512^3 volume and 512 projections. Nevertheless, the reconstruction technique based on slabs allows addressing larger problems and does not influence the performance of the backprojection, so we kept this mechanism in place.

The basic step of the backprojection consists in taking a given slice in the volume, i.e. one xy plane and to consider it as the render target. A given projection is then rendered onto this slice with the pixel shaders, the accumulation taking place through simple blending of the results of successive backprojection steps. Accounting for the $\omega(\alpha, r)$ coefficient is made through the definition of a mask that has the same size as the considered projection data and holds pixel wise the coefficient to apply.

This platform performs the reconstruction in floating point math and can backproject 512 projections on a 512^3 volume in ~ 37 seconds.

3.5 Cell platform

Implementing a Feldkamp backprojection on the CBE consists in distributing the tasks between the processing elements of the CBE processor. Using the PPE as a manager of the reconstruction process while the SPEs are dedicated of performing the real reconstruction task is the approach we have selected.

Since the LS limit of 256 kB per SPU does not allow holding the complete volume, we have used a hierarchical memory layout to tile the volume into small sub-volumes of 32^3 voxels. Such a sub-volume occupies half of the LS. The remaining 128 kB are used to hold the code, the stack, to hold the subset of the projection that is actually needed for the computation of the contribution of that projection angle to the considered subvolume. This subset has been produced during the alignment operation.

The application is designed in such a way that, while the SPE is busy rebinning and backprojecting the first raw data buffer, the DMA of the next raw data patch was active. We thereby achieve to fully hide the data transfer latency behind the perspective backprojection process. This platform is capable of backprojecting 512 projections on a 512^3 volume in ~ 17 seconds.

4 RESULTS AND DISCUSSION

4.1 Image quality

We have run the different implementations against the same data set and obtained clinical quality reconstructed volumes for all of them. Figure 9 gives a typical examples of the kind of quality those implementations allow for. It depicts different views of the reconstruction of a mouse scanned with a micro-CT-scanner TomoScape 30s (VAMP GmbH, Erlangen, Germany).



Figure 9: Reconstruction of a mouse

However, there are slight differences to be observed between the different volumes and they can be taken care of at minimal processing expenses.

The FPGA version performs all of its computation in fixed point. Floating point math cannot be considered, even on modern chips. All of the math has to use a fixed point representation with the inherent limit imposed by the width of the multipliers embedded in the Virtex family chips; 18 bits for the Virtex-2 Pro. There is still the possibility to cascade multipliers to reach better accuracy but the results turn out to be inefficient in terms of performance. The best way to overcome this kind of problems is to carefully design the computation pipeline in order to keep those bits which are relevant to the dynamics of the signal and to recondition the input data so that extreme values do not occur.

The GPU reconstructed volumes show the same kind of issues but at a reduced level. This is mainly due to the fact that, however close to IEEE floating point standards, the floating point processing on an nVidia GPU still has some deviations with respect to the standard. It takes the form of incorrect handling of exceptional cases, such as Not A Number (NaN). One can use the same technique in preconditioning the input data so that these kind of exceptions do not arise. It is rather easier than in the FPGA case, since the conditioning has only to take care of exceptional cases, while the accuracy of the implementation has to be taken into account for the FPGA case.

The Cell implementation suffers to a low degree from inaccuracies related to the computations of estimates instead of real values for operators like divide, square root and exponential. The estimates turn out to be accurate, up to the 6th digit after the floating point. Some special cases at low angles require more accuracy and Raphson-Newton kinds of algorithms can be employed to overcome this issue.

4.2 Performance

At the time of writing, it appears that the Cell processor offers the best performance over all other designed architectures. It should be obvious that we are not comparing apples to apples, i.e. using the best versions in the different technologies. The Cell processor and the GPU we have selected count among the most recent technologies available on the market. The Virtex-II is definitely not among the most recent FPGA packages and the PC reference platform certainly has more powerful successors with the Dual Core and Quad Core processors.

The newest Virtex-4 and Virtex-5 can run at clock speeds around 500MHz, almost five times faster than the version we have investigated. Moreover, Xilinx proposes designs to implement DDR-2 interfaces on the Virtex-4 and Virtex-5

chips, hence giving the same increase of 5x in performance for the memory subsystem. Without considering the increase of real estate of the newest FPGAs, an increase in the performance factor of 5x is the bare minimum to expect.

The real performance increase of DualCore and Quad Core architecture is more difficult to estimate. In the case of the FPGA, we know that all of the devices can run at 5x faster. In the case of multicore processors, the way the IO and memory access resources are shared and distributed is processor designer dependent. Nevertheless, even with a 4x performance improvement, a quadcore system does not come close to the performance of a Cell processor or a GPU, not to mention the newest FPGAs.

We have investigated to implement the same algorithm on different generations of GPUs though. To most extend one can observe a performance increase with newer versions. However, most of the changes in the GPU architectures are kept secret by the manufacturers and hidden in the GPU driver, executed on the host. We believe that the performance of the backprojection can be enhanced to match the performance increase of newer GPUs, at the expense of some investigations and potential re-engineering of the S/W.

The CBE processor has a road map. It is unclear how the next generation of Cell processor will look like and how much the instruction set will have changed, not to mention how the individual instructions are pipelined and dispatched. On the other hand, those types of changes only affect the most optimized code, for which modern compilers take away most of the burden.

4.3 Complexity

There are different levels of complexity to be considered when one thinks of developing a reconstruction algorithm on a given platform. The first relates of course to the complexity of developing the algorithm and get the adequate image quality. The second relates to the design of the system hosting the reconstruction processing part. Some implementations call for multi chassis platforms with complex interconnects while others need adequate cooling and power supplies. The third and final aspect relates to the engineering effort required to keep up with the latest technology in order to replace failing parts in the field, once the failing part has been end of lifed by the manufacturer.

From the software implementation section of this paper, one can take that the most obvious and stable implementation has been done on a PC. The Cell offers a multicomputer platform which is very comparable to multi-computers such as those developed by Mercury Computer Systems with RACEWay, RACE++ and RapidIO. Even though all GPU board support OpenGL and DirectX, the level of efficiency for different GPU boards vary a lot, even when they come from the same manufacturer. The internal features, such as the structure of the 2D texture caches, are hidden to the developer for many reasonable reasons. However, the result is that, performances are not predictable across GPU board generations. The implementation section also shows that the coding of reconstruction algorithms are a lot more difficult on FPGAs, mainly because they don't offer floating point operators and that operators such as multiply, divide, sine and cosine. Those functions have to be coded as application dependent Look Up Tables (LUT).

The GPUs are intended to be the graphical processor companion in every PC. Therefore, most of the modern PCs can accommodate the presence of a modern GPU, for power supply and cooling. Consequently, all reconstruction hardware that fits in the same {cooling, power supply} envelope can be hosted in the same host PC. The CAB has been designed to fit into that envelope and can be hosted in any modern PC. FPGA based boards are subject to the inspiration of the designer. FPGAs traditionally draw less power than high clocked devices as a GPU or the CBE processor and are indeed easier to cool. However, FPGA board require special attention to ensure their power and cooling requirements are within the specifications of the host.

However much the host processors have been evolving over the last years, they remain all compatible with each other, most of the time, even binary compatible. That means that a given executable, produced with a given version of tools for a given version of processors is likely to work on many subsequent processor versions without intervention. Therefore, PC based implementation also offer the best solution for field repairs and upgrade when considering the maintenance costs. FPGAs and the Cell processor are designed to stay active for many years and indeed propose a valuable alternative for accelerating medical image reconstruction. GPUs represent the device family that has the most variability in terms of architecture, structure, Application Programming Interfaces and drivers.

5 CONCLUSION

We have investigated the implementation of a Cone-Beam reconstruction algorithm on various platforms. All basic building blocks, GPU, FPGA and Cell are available and can deliver the appropriate image quality, at varying degrees of effort though. Depending on the evaluation criteria, the winner between FPGAs, GPUs, multi-core based PC and the Cell processor may differ. However, the Cell Broadband Engine proposes a fully programmable architecture, accessible from high level programming languages such as C. Its involvement in the gaming industry predicts a long term availability of the parts for realistic field deployment and maintenance in hospitals.

ACKNOWLEDGMENT

The authors gratefully acknowledge the help from Sebastian Schuberth, Scott Thieret, Dr. Iain Goddard, Shepard Siegel and Frank Lauginiger from Mercury Computer Systems for providing extensive explanations and their support in implementing the algorithms on the different platforms.

REFERENCES

- [1] – L. Feldkamp, L. Davis, J. Kress, *Journal of the Optical Society of America* 1 no. 6 (1984) -12-619
- [2] – Riddell, Cyril and Trouset, Yves, Rectification for Cone-Beam Projection and Backprojection, *IEEE Transactions on Medical Imaging* 25(7): 950-962, July 2006
- [3] – H.P. Hofstee. Power efficient processor architecture and the Cell processor. *Proceedings of the 11th International Symposium on High-performance Computer Architecture*, Feb. 2005
- [4] – I. Goddard, M. Trepanier. High Speed cone-beam reconstruction : an embedded system approach. *SPIE Medical Imaging Proc.*, 4681:483-491,2002.
- [5] – K. Müller, F. Xu. Accelerating popular tomographic reconstruction algorithms on commodity PC graphics hardware. *IEEE Transactions on Nuclear Science*, (3):654-663, 2005.