



High Performance Medical Reconstruction using Stream Programming Paradigms

June 2008

High Performance Computing Group, NeST Software [www.nestsoftware.com]

Abstract— This paper describes the implementation and results of CT reconstruction using Filtered Back Projection on various stream programming paradigms.

Key words— Filtered Back Projection, CUDA, Cell.

I. INTRODUCTION

The complexity of Medical image reconstruction requires tens to hundreds of billions of computations per second. Until few years ago, special purpose processors designed especially for such applications were used. Such processors require significant design effort and are thus difficult to change as new algorithms in reconstructions evolve and have limited parallelism. Hence the demand for flexibility in medical applications motivated the use of stream processors with massively parallel architecture.

Stream processing architectures offers data parallel kind of parallelism. In data parallelism, a set of operations are performed on large amount of data in a parallel fashion. Many of the reconstruction algorithms are high suitable for this data parallel paradigm. The central idea behind stream processing is to organize the application into *streams* - set of data - and *kernels* - a series of operations applied to each element in the stream.

There are several stream processors commercially available. Here in this paper, we consider two emerging processors - Graphical Processing Unit (GPU) [2] and Cell Processors [3] for benchmarking our reconstruction application provide specific instructions to allow application developers to tag kernels and/or streams.

GPUs are designed for high end game and 3D simulation application. They are massively parallel architectures with large amount of arithmetic processing units. Programming a GPU requires the knowledge of graphics pipeline architecture. But off late the advent of General Purpose on GPU (GPGPU)[4] lead to design of tools based on common programming languages. Cell Processors are designed to bridge the gap between conventional desktop processors and more specialized

high-performance processors, such as GPUs. Cell processor can be split into four components: external input and output structures, the main processor called the Power Processing Element (PPE) eight fully-functional co-processors called the Synergistic Processing Elements.

GPU can be programmed using Shader languages of Graphical APIs such as OpenGL [2] and DirectX [5], which requires knowledge in Graphics architecture. Then there are C style programming languages such as CUDA [5], which requires minimum or no background in Graphics. As the Cell processor is general purpose processor, languages currently available are similar to C programming.

In this paper we consider the X-ray beam computed tomography (CT) reconstruction of 2Dimage/3D image from 2D projections as or image reconstruction example. The 2D/3D image reconstruction is a very demanding computational task. The reconstruction from projections is done using Filtered Back Projection [1] with time complexity $-O(N^4)$ where N is the number of detector pixels in one detector row.

In the rest of the paper we give an overview of the programming tools used for GPU and Cell Processor. Also we describe the implementation of Filtered Back Projection using some of these programming paradigms. We conclude with the paper by benchmarking results with conventional processors.

II. FILTERED BACK PROJECTION

Filtered Back Projection is a conventional technique used for CT reconstruction for 2D Fan Beam Data and 3D Cone Beam Data. Filter Back Projection can be divided into (1) Filtering and (2) Back Projection.

Algorithm of Discrete FBP



The following is the pseudo algorithm of the FBP

```
//Filter the projection images along horizontal lines
For Each Projection
  For Each row of the Projection
    For Each col of the Projection
      Perform the convolution. (Filter) row wise
// Perform the back projection along the projection rays
For each Filtered Image
  Get the angle of Projection for this filtered image
  For each output Image of the reconstructed volume.
  Get the angle of Projection of this image.
  Calculate increment factor U using angle for indexing (look up) into the Filtered Image and also weight calculation.
  For each row of the output image
    For each column of the output image
      Lookup the image at calculated index.
      Accumulate the output with the  $1/U^2$ 
```

The computational complexity of the Filter Back Projection (FBP) is biquadratic. To reconstruct a volume of size 400 x 400 x 240 images from 360 projections requires floating point operations of the order of 10^9 than available with a normal processor. The most computation-intensive step of the Filtered Back Projection is back projection. Roughly 97% of the operations involve the back projection when reconstructing the images from the projections.

III. CUDA

Traditionally GPGPU applications were developed using Graphical APIs and their shading language which requires knowledge of Graphics Pipeline. Moreover these APIs exposed very little of the underlying hardware. CUDA, designed from the ground-up is an efficient general purpose computation on GPUs. It gives computationally intensive applications access to the tremendous processing power of the latest NVIDIA GeForce 8 GPUs through a C-like programming interface. The GeForce 8 series GPUs have up to 128 processors running at 1.5 GHz and up to 1.5GB of on-board memory. It uses a C-like programming language and does not require remapping algorithms to graphics concepts. CUDA exposes several hardware features that are not available via the graphics API. The most significant of these is shared memory, which is a small (currently 16KB per multiprocessor) area of on-chip memory which can be accessed in parallel by

blocks of threads. This allows caching of frequently used data and can provide large speedups over using textures to access data. Combined with thread synchronization primitive, this allows cooperative parallel processing of on-chip data, greatly reducing the expensive off-chip bandwidth requirements of many parallel algorithms. This benefits a number of common applications such as linear algebra, Fast Fourier Transforms, and image processing filters. The current generation of GPU from NVIDIA has support for IEEE single precision. Double precision support will be available in the next generation towards the end of the year. There are however several fields in which significant results can be obtained with single precision: image and signal processing and some numerical methods like pseudo spectral approximation are just few examples

CUDA Programming Model

The programmer writes a serial program that calls parallel *kernels*, which may be simple functions or full programs. A kernel executes in parallel across a set of parallel threads. The programmer organizes these threads into a hierarchy of grids of thread blocks. A thread block is a set of concurrent threads that can cooperate among themselves through barrier synchronization and shared access to a memory space private to the block. A *grid* is a set of thread blocks that may each be executed independently and thus may execute in parallel.

When invoking a kernel, the programmer specifies the number of threads per block and the number of blocks making up the grid. Each thread is given a unique thread ID number *threadIdx* within its thread block, numbered 0, 1, 2, ..., *blockDim* - 1, and each thread block is given a unique block ID number *blockIdx* within its grid. CUDA supports thread blocks containing up to 512 threads. For convenience, thread blocks and grids may have one, two, or three dimensions, accessed via *.x*, *.y*, and *.z* index fields.

FBP Implementation

The FBP implementation of CPU consists of reading the image files and its configuration files containing the information such as angle of Projection, geometric parameters for Region of Interest (ROI), computing the Filter mask and the geometric shape of the output image. The compute intensive portions like filtering and back projection, implemented in GPU, are packed in two different kernels. Prior to the kernel execution, the input data is copied to the GPU's device memory. As this data

is required for both filter kernel as well as back projection kernel, it is allocated in the global memory of the device, which helps to avoid copying multiple times. In order to achieve high level of parallelism, size of blocks and threads in the block are selected appropriately. We divide the data into several 1D block and threads.

The input image data is stored and accessed via global memory. 1D convolution mask as well the a row input data, which is used repeatedly used for convolving each of row of image is stored and accessed via shared memory than from the global memory. This will significantly reduce memory bandwidth which affects the performance. Data reuse is high with in a thread block and hence a block of threads are loaded into the shared memory. Data stored in the shared memory are declared as volatile variable in order to disable to compiler optimization for reading and writing to shared memory as long as previous statement is met [6]. This is yet another optimization strategy to reduce the memory bandwidth.

The back projection algorithm contains computations involving several array look ups. Unlike in CPU, array look ups in GPUs are costlier. Hence they are replaced with index based computations. This is done in order to exploit arithmetic efficiency of GPU. Compared to filtering operation, the data reuse is not that high in back projection, which makes the shared memory usage inefficient in this situation. Here texture look up of the filtered image is the found to be better choice than accessing the data stored in the shared memory. This is because texture memory is spatially cached. More over back projection computation performed on large amount of data cannot be stored in the shared memory whose size

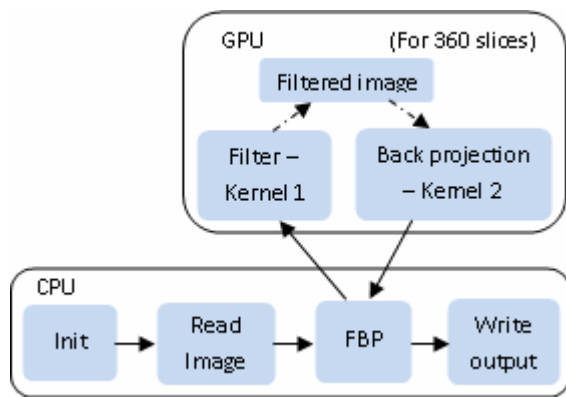


Figure 1: FBP implementation in CUDA

Results

The Filtered Back Projection algorithm was tested on NVIDIA 8800 GTS and Intel Pentium D processor with 3.2 GHz hardware. The table 1 below shows the results for the filtered back projection for an Input Image of size 960 x 768 x 360.

	Filtering & Back Projection Operation (in seconds) of size 960 x 768 x 360	Speed up
Intel Pentium D (3.2 GHZ)	1800	112.5x
NVIDIA GeForce 8800 GTS	16	

Table 1 : Comparison of the time taken for execution of FBP on Intel Pentium D (3.2 GHZ, 2 GB RAM) and the NVIDIA 8800 GTS.

IV. CELL PROGRAMMING

System Overview

The Cell Broadband Engine is a heterogeneous multiprocessor containing 1 PowerPC Processing Unit (PPU), 6 Synergistic Processing Units (SPUs) and a high bandwidth element interconnect bus. There is support for performing data transfers concurrent to instruction execution. The Cell processor achieves efficiency in power consumption because of the distinction between the PPU (that is specialized for control operations and runs the OS) and the SPU (that is specialized for data intensive operations). There is support for vectorization on the PPU and the SPU that can result in up to 4 single precision floating point operations in one cycle. The Cell based machine chosen for implementation of our application is the Play Station 3 (PS3), running the Yellow Dog Linux distribution. Programming is done in C, using C extensions for the Cell platform. GNU C Compiler for the SPU and PPU platforms are used.

Cell Programming Model

The Cell processor supports a stream programming model, which involves the identification of an appropriate kernel function and the execution of the chosen kernel function on all the data elements. The latency of execution is reduced by exploiting the single program multiple data (SPMD) paradigm for executing the same kernel function concurrently on multiple data elements. In this case, the PPU spawns multiple threads and schedules these threads for execution in the 6 SPEs.

The synchronization of threads and transfer of data are controlled from the PPU program. Double buffering is used to allow data transfers from the PPU to SPU(s) concurrent to program execution. In order to minimize the latency of memory access, the software cache in SPUs is utilized. Data transfers from the PPU to SPUs and vice versa are accomplished by DMA and message passing. Vectorization of PPU and SPU code is performed for further increase in data parallel execution. Wherever possible, optimized library functions for the cell processor are invoked for increased execution efficiency.

The code for execution on the PPU and SPUs is developed separately. The executables for the SPUs are embedded within the main PPU executable and can be invoked from within PPU functions as a separate thread, specifying the name of the executable, the SPU on which it has to be executed and other relevant parameters.

Implementation of FBP

The filter and backprojection functions that have to be evaluated corresponding to every input and output voxel respectively, are considered as the kernel functions. Filtering involves the convolution of a one-dimensional filter with each row of input slices. Back projection involves evaluation of weights for each pixel, evaluation of the offset in the filtered image and determining the contribution of the corresponding value towards the back projection of the corresponding pixel. Due to limitations in RAM for the Cell, the input images are processed one at a time, accumulating the weighted contribution of each input image to output voxels.

The issues encountered on the Cell processor include limitations in RAM and SPU local memory as well as issues in restructuring the code for optimal performance on the Cell BE processor. There is an increased programming overhead because the programmer is responsible for management of data transfer and buffering between the PPU and SPUs, synchronization of execution of the PPU and SPUs as well as because of constraints on data alignment, limitation on the size of data that can be transferred by DMA and endian issues.

The block diagram in Figure 3 depicting the implementation of FBP on Cell is shown in Figure 3. The Init block corresponds to a one-time initialization function for the system. Following initialization, the image is loaded onto the main memory of the PPU. The filter kernel is executed on the 6 SPUs. Each SPU operates on a different region of the image. The filter kernel function is executed once for each pixel and is

invoked multiple times to complete the processing of a block corresponding to an SPU. Data transfers accomplished using DMA are initiated as required by the application. Constraints in SPU local memory impose that the filtered image be transferred back to the PPU memory. Since double buffering is used, there is no overhead for data transfers.

After completion of filtering for the input image, backprojection is performed. All necessary output voxels are updated with weighted values of corresponding input pixels from the current image. After completion of processing of the current image, the filtering and backprojection operations are performed for all remaining input images, accumulating the output voxels for every input image. Following completion of FBP for all input images, the output is written onto raw image files.

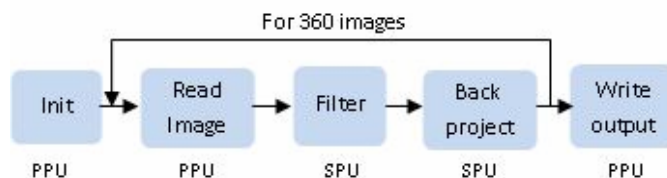


Figure 2: FBP as implemented on the Cell BE

Results

The FBP code is implemented on the Play Station 3 (PS3), which consists of a single Cell processor, 256 MB RAM and 3.2 GHz clock speed. The data used for testing the system is artificial CT data corresponding to the 3 Cylinders dataset. The performance on the Cell processor is compared to that on an Intel Pentium D processor running at the same clock speed and having a RAM of 2 GB. The number of input images is 360 and the dimension of each input image is 768 x 960. Following backprojection 240 output images of dimension 400 x 400 voxels are obtained. Table 2 summarizes the results obtained.

Processor	Time (sec)			Speedup
	Filter	Backproject	Total	
Intel Pentium D	600	1200	1800	6.9x
Cell BE	47.5	212.9	260.4	

Table 2: Comparison of the time taken for execution of FBP on Intel Pentium D (3.2 GHZ, 2 GB RAM) and the Cell BE (3.2GHz, 250MB RAM) processors.



This is the first implementation of FBP on the PS3 machine. Other Cell based implementations have used the Cell Blade, which contains 2 Cell BE processors having 2 PPEs, 16 SPEs and 2GB RAM. The performance obtained with such a system is reported to be 14.16x [7].

REFERENCES

- [1]. F. Nattered, *The Mathematics of Computerized Tomography*, John Wiley & Sons, 2 editions, 1986.
- [2]. [http:// www.opengl.org](http://www.opengl.org)
- [3]. <http://www.microsoft.com/directx>
- [4]. <http://www.gpgpu.org>
- [5]. http://www.nvidia.com/object/cuda_home.html
- [6]. NVIDIA CUDA Programming Guide.
- [7]. Programmer's Guide - Cell BE Programming Tutorial,
[http://www-01.ibm.com/chips/techlib/techlib.nsf/techdocs/FC857AE550F7EB83872571A80061F788/\\$file/CBE_Programming_Tutorial_v3.0.pdf](http://www-01.ibm.com/chips/techlib/techlib.nsf/techdocs/FC857AE550F7EB83872571A80061F788/$file/CBE_Programming_Tutorial_v3.0.pdf)
- [8]. M. Knaup, S. Steckmann, O. Bockenbach and M. Kachelrieß. *Tomographic image reconstruction using the Cell broadband engine (CBE) general purpose hardware*. Proceedings Electronic Imaging, Computational Imaging V, SPIE Vol 6498, 64980P 1-10, January 2007.