

# Hyperfast Parallel–Beam Backprojection

Marc Kachelrieß, *Member, IEEE*, Michael Knaup, Olivier Bockenbach

**Abstract**—Tomographic image reconstruction, such as the reconstruction of CT projection values, of tomosynthesis data, PET or SPECT events, is computational very demanding. The most time-consuming step is the backprojection which is often limited by the memory bandwidth. Recently, a novel general purpose architecture optimized for distributed computing became available: the Cell Broadband Engine (CBE). Its eight synergistic processing elements (SPEs) currently allow for a theoretical performance of 192 GFlops (3 GHz, 8 units, 4 floats per vector, 2 instructions, multiply and add, per clock).

To maximize image reconstruction speed we modified our parallel–beam backprojection algorithm, that is highly optimized for standard PCs, and optimized the code for the Cell processor. Data mining techniques and double buffering of source data were extensively used to optimally utilize both the memory bandwidth and the available local store of each SPE. The pixel–driven backprojection code uses floating point arithmetic and either linear interpolation (LI) or nearest neighbor (NN) interpolation between neighboring detector channels.

Performance was measured using simulated data with 512 parallel beam projections per half rotation and 1024 detector elements. The data were backprojected into an image of 512 by 512 pixels using our PC–based approach and the new Cell–based algorithm. Both the PC and the CBE were clocked at 3 GHz.

Images obtained were found to be identical with both approaches. A throughput of 11 fps (LI) and 15 fps (NN) was measured on the PC whereas the CBE achieved 126 fps (LI) and 165 fps (NN). Thereby, the Cell greatly outperforms today’s top–notch backprojections based on graphical processing units (GPU). Using both CBEs of our dual Cell–based blade (Mercury Computer Systems) one can backproject 252 images per second with LI and and 330 images per second with NN.

## I. INTRODUCTION

CELL processors are general purpose processors that combine a PowerPC element (PPE) with eight synergistic processor elements (SPE) [1], [2], [3]. The SPEs are the most interesting feature of the Cell processor, as they are the source of its processing power. A single chip contains eight SPEs, each with an synergistic processing unit (SPU), a memory flow controller (MFC), and 256 kB of SRAM that are used as local store (LS) memory. The LS runs in its own address space at the full 3.2 GHz clock frequency.

An SPU uses 128 bit vector operations and can execute up to eight floating point instructions per clock cycle. For our particular focus on backprojecting floating point values (4 bytes each) the data vector consists of four floats. A fast (96 byte per clock) element interconnect bus (EIB) connects the Cell processor’s PPE with the SPEs (figure 1).

Prof. Dr. Marc Kachelrieß: Institute of Medical Physics (IMP), University of Erlangen–Nürnberg, Henkestraße 91, 91052 Erlangen. E–mail: marc.kachelrieß@imp.uni-erlangen.de.

Dr. Michael Knaup: Institute of Medical Physics (IMP), University of Erlangen–Nürnberg, Henkestraße 91, 91052 Erlangen. E–mail: michael.knaup@imp.uni-erlangen.de.

Olivier Bockenbach: Mercury Computer Systems, Lepsiusstr. 70, 12163 Berlin. E–mail:olivier@mc.com

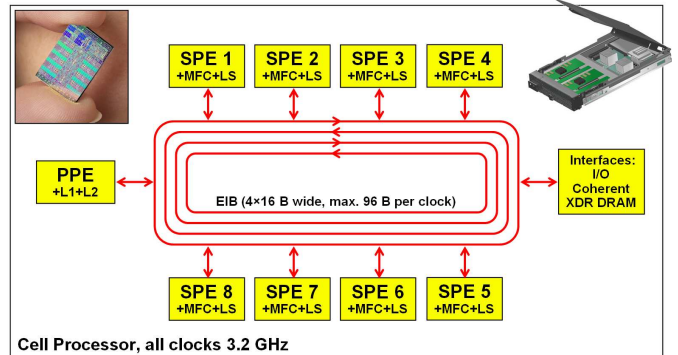


Fig. 1. Block diagram of the Cell with pictures of one CBE and of the Mercury dual Cell–based blade

Up to two instructions per cycle can be issued by each SPU to its seven execution units, organized in two pipelines. To overcome memory latency (the “memory wall”) DMA data transfers from and to the SPU can be scheduled in parallel with core execution.

The PPE can be understood as being the controller or manager that distributes small tasks to the eight SPEs, which are the workers. In our case, communication between the manager and the workers is realized via mailboxes and DMA transfers.

The aim of this investigation is to implement a 2D parallel–beam backprojection algorithm for the Cell processor and to benchmark its performance against PC–based implementations.

## II. METHOD

We consider a backprojection of type

$$f(x, y) = \int d\vartheta p(\vartheta, a(\vartheta)x + b(\vartheta)y + c(\vartheta))$$

with  $f$  being the image and  $p$  being the (convolved) rawdata and where  $a$ ,  $b$ , and  $c$  are arbitrary functions of  $\vartheta$ . For example a scanner with projection angle  $\vartheta$  and ray distance  $\xi$  to the origin would have  $a = \cos \vartheta$ ,  $b = \sin \vartheta$  and  $c = 0$  such that the ray parameterized by the pair  $(\vartheta, \xi)$  is the line  $x \cos \vartheta + y \sin \vartheta = \xi$  and the look–up would occur at  $p(\vartheta, \xi)$ .

### A. Implementation

The backprojection integral is usually realized in a discretized version called pixel–driven backprojection. Our reference code is shown in the listings. Listing 1 shows the nearest neighbor interpolation, listing 2 performs linear interpolation.

Apart from this unoptimized reference code our highly optimized PC–based implementation (pure C++, coded in

---

```

void ParBackProjRefNN(int const N, int const M,
                    int const I, int const J,
                    float const * const a,
                    float const * const b,
                    float const * const c,
                    float const * const Raw,
                    float * const Ima)
{
    for(int n=0; n<N; n++) // projection index
    for(int i=0; i<I; i++) // slow pixel index
    for(int j=0; j<J; j++) // fast pixel index
    {
        float const mreal=a[n]*i+b[n]*j+c[n];
        int const m=int(mreal+0.5);

        Ima[i*J+j]+=Raw[n*M+m];
    }
}

```

---

Listing 1: Reference code for nearest neighbor interpolation. The pixel indices  $i$  and  $j$  correspond to  $x$  and  $y$  and the sinogram indices  $m$  and  $n$  correspond to the detector channel index and the projection index, respectively.

---

```

void ParBackProjRefLI(int const N, int const M,
                    int const I, int const J,
                    float const * const a,
                    float const * const b,
                    float const * const c,
                    float const * const Raw,
                    float * const Ima)
{
    for(int n=0; n<N; n++) // projection index
    for(int i=0; i<I; i++) // slow pixel index
    for(int j=0; j<J; j++) // fast pixel index
    {
        float const mreal=a[n]*i+b[n]*j+c[n];
        int const m=int(mreal);
        float const w=mreal-m;

        Ima[i*J+j]+=(1-w)*Raw[n*M+m]+w*Raw[n*M+m+1];
    }
}

```

---

Listing 2: Reference code for linear interpolation between neighboring detector channels.

early 1999) that is equivalent to the reference code is used to benchmark against the new Cell-based parallel backprojection.

When porting the code to the Cell several constraints had to be followed. The LS is limited to 256 kB and only small portions of the full problem can be handled by each worker. To accommodate demand, the image and rawdata had to be tiled into small sub-images and sub-sinograms. The size of the sub-images and the size of the sub-sinograms was chosen to allow for double buffering of the sinogram data. Two sub-sinograms plus one sub-image plus code stack must fit into the 256 kB local store. Only those portions of a projection that were needed by a worker's particular sub-image were DMAed to the worker. While the worker is busy backprojecting the first sub-sinogram, the DMA of the other sub-sinogram was active. Thereby, the DMA latency is fully hidden behind the backprojection process.

Further, care was taken to make use of the 128 available registers per SPU to fully fill the execution pipelines. Manual loop unrolling and reordering of instructions ensured to achieve a throughput of more than one instruction per clock cycle.

	NN		LI	
	$C/O$	$T$	$C/O$	$T$
PC, reference	93.3	4.1 s	118	5.2 s
PC, optimized	1.54	68 ms	2.12	93 ms
Cell, optimized	0.14	6.1 ms	0.18	7.9 ms

TABLE I  
PERFORMANCE ACHIEVED WITH THE PC-BASED AND WITH THE CELL-BASED IMPLEMENTATIONS.

### B. Performance Assessment

The code was implemented to cope with any number of pixels (also non-square images), projections and channels per projections. We assessed the performance of backprojecting 512 parallel projections into an image of size  $512 \times 512$ . The complexity of the code is  $O = 512^3$  operations. The fact that each projection consisted of 1024 channels is irrelevant to our timing measurement.

The standard and the optimized code ran on a single 3.06 GHz Xeon processor with 533 MHz front side bus while the cell-based implementation uses a 3.2 GHz Cell processor running on a dual Cell blade (Mercury Computer Systems). The time  $T$  per slice was measured using the system clock. To improve the timing accuracy and to overcome the granularity of the system clock we report the average of a large number of reconstructed images. Care was taken that no other significant CPU workload impaired our measurements. All performance values were scaled to 3.0 GHz, for convenience.

Additionally, we compute the number of CPU clock cycles per operation as  $C/O$  with  $C = FT$  being the number of clock cycles per reconstructed image and  $F$  being the clock frequency that equals 3.06 GHz for our PC, 3.2 GHz for the Cell system and 3.0 GHz for the scaled values, respectively.

## III. RESULTS

The timing results for a nearest neighbor (NN) and a linear interpolation (LI) backprojection are shown in table I. The NN and the LI reference algorithms are the code segments provided in the listings. The reference algorithm is PC-based but not optimized. The PC-based optimal backprojection is a highly optimized backprojection code that is in use by our group since 1999. The Cell-based code is also highly optimized as detailed earlier in this paper. All algorithms are equivalent to the reference code.

Apparently, the CBE achieves a backprojection rate of 165 fps with nearest neighbor interpolation and 126 fps with linear interpolation. Considering that two CBEs are available per blade one may backproject 330 images per second per dual Cell board.

How does our implementation compare to the theoretical peak performance? Theoretically, and this assumes optimal optimization, one may not do better than updating four pixels per step. An update step requires at least two loads, one add and one store for nearest neighbor. The add runs on the even pipeline and can theoretically be completely hidden by the

	without DMA	DMA get	DMA put	total
ParBackProjLI	4047 ms	16 ms	7 ms	4070 ms

TABLE II

DMA LATENCIES FOR THE PARALLEL BACKPROJECTION WITH LINEAR INTERPOLATION. DMA GET: RAWDATA FLOW FROM MANAGER TO WORKER. DMA PUT: VOLUME FLOW FROM WORKER TO MANAGER. THE STATISTICAL ERROR FOR THE PARALLEL BACKPROJECTION IS BELOW 0.1 MS.

three load/stores that execute in parallel on the odd pipeline. Per clock (we have 8 workers and assumed that each step updates 4 pixels) one can theoretically update  $32/3$  pixels, i.e.  $C/O \geq 0.09375$ . Similarly, linear interpolation requires four load/stores and two multiply-adds which means  $32/4$  pixel updates per clock. Hence  $C/O \geq 0.125$  must hold. Regarding the measured values our implementation reaches 69% (NN) and 71% (LI) of the theoretical peak performance. The theoretical peak performance of 192 GFlops cannot be reached by our implementation since the multiply-adds (i.e. the Flops) are hidden between a larger number of load/stores.

	Type	Hardware	Time	Comment
Leeser et al. [4]	LI / i09	CPU	4.66 s	
	LI / i09	FPGA	125 ms	
Schiwietz et al. [5]	LI / f32	CPU	22.6 s	incl. FFT
	LI / ?	GPU	176 ms	incl. FFT
Xue et al. [6]	? / f32	CPU	7.13 s	
	? / i32	FPGA	273 ms	
	? / i32	GPU	295 ms	
	? / i16	GPU	143 ms	
Kachelrieß et al.	NN / f32	CPU	68 ms	
	LI / f32	CPU	93 ms	
	NN / f32	CBE	6.1 ms	
	LI / f32	CBE	7.9 ms	

TABLE III

BACKPROJECTION PERFORMANCE FOR THE PARALLEL BACKPROJECTION. ALL VALUES HAVE BEEN SCALED TO 512 PROJECTIONS AND  $512^2$  PIXELS. ALL VALUES WERE FURTHER SCALED TO A SINGLE PROCESSING UNIT, I.E. TO ONE CPU, ONE FPGA, ONE GPU AND TO ONE CBE, RESPECTIVELY, AND TO 3.0 GHZ IN THE CASE OF CPU- AND CELL-BASED ALGORITHMS. THE TYPE COLUMN SPECIFIES THE INTERPOLATION TYPE, NN OR LI, AND THE TYPE OF ARITHMETIC USED: F+NUMBER OF BITS DENOTES FLOATING POINT ARITHMETICS WHILE I+NUMBER OF BITS STANDS FOR INTEGER (FIXED POINT) ARITHMETICS.

#### DMA Latency

One of the most prominent feature of the CBE is its fast DMA between the main memory and the worker local store. Since Cell DMA works in parallel to the SPU's command execution pipeline, the DMA latency may be completely hidden for some CPU-limited problems.

To measure the DMA latency for our implementations, we performed dummy reconstructions without DMA transfers and calculated the differences of the total backprojection times to

that of real backprojections. The backprojection times were measured with clock-cycle precision via the so-called worker decremter. The decremter is a counter on each SPU that is decremented on a clock cycle basis. Statistical errors were estimated by repeating all measurements five times.

Table II shows the results. It turned out that the DMA fraction of the total reconstruction time is about 0.57% for the parallel backprojection.

#### IV. OTHER APPROACHES

Other groups have made lots of efforts to speed up CT image reconstruction. Although a fair and quantitative comparison is not always possible, table III lists those performance figures that have been published in this millennium, including those published in this paper. Benchmarks found in older literature are considered obsolete due to the ongoing developments in computer technology.

To allow for some comparison we scale the values found in the literature to the case of backprojecting 512 projections into an image of  $512 \times 512$  pixels. The projection size itself is considered irrelevant. For PC-based implementations the CPU clock rate is scaled to 3.0 GHz. This assumption is quite optimistic since backprojection is usually limited by memory latency and memory speed has not increased that significantly during the last years. Especially for older experiments that have been carried out on slow CPUs this scaling will overestimate the actual performance that could be achieved with the same algorithm on modern CPUs. Note that in most cases comparing the cost-to-performance ratio would be more adequate than just comparing performance. However, there are no reliable cost figures available to us.

We further want to point to the fact that there are significant differences whether the reconstructed FOV is square or circular. A circular FOV contains only  $\pi/4 \approx 79\%$  of the voxels that are contained in the enclosing square. This adds another 21% uncertainty to the values found in the literature if the FOV shape is not disclosed or if pixels outside the FOM are not backprojected.

Divide-and-conquer-type backprojection, such as Fourier-based image reconstruction [7], [8], [9], hierarchical backprojection [10] or the link-method [11], for example, is of completely different type than the standard backprojection algorithms discussed here and therefore not included in our comparison. It should be noted that these methods have the potential to increase reconstruction speed by a factor  $cN/\ln N$  with  $c$  being some (sometimes rather small) constant. Except maybe for Fourier reconstruction there is no highly optimized implementation that can really compete with the standard backprojection performance values listed here. Further, some

of these divide-and-conquer concepts work well in 2D but become difficult or impossible in the cone-beam case. E.g. Fourier reconstruction in 3D only works when the complete Radon data are available [12]. Last but not least they often suffer from a trade-off between reconstruction speed and reconstruction accuracy, except for the Fourier-based algorithms.

Leeser et al. published an FPGA-driven parallel beam backprojection [4]. Using fixed point arithmetic with 9 bit they can backproject 1024 projections into a  $512^2$  image in 0.25 s using “16-way parallel processing”. A great deal of their work has to do with bit reduction which always means a loss of image quality, however. They assume 12 bit input data (which is not sufficient for clinical CT where the data are acquired with at least 20 bit). They compare their results to a 1 GHz CPU-based version that needs 28 s for the 1024 projections.

Schiwietz et al. compare CPU-based with GPU-based MR image reconstruction [5]. Since they start in Fourier domain their code includes the inverse FFT to obtain data ready for backprojection. They use a 3 GHz CPU and an ATI Radeon X1800 XT GPU. The reconstruction of three  $256^2$  images from 504 projections requires 16.7 s on the CPU and 130 ms on the GPU. Normalizing this to one  $512^2$  image and 512 projection yields 22.6 s and 176 ms, respectively.

Xue and coauthors compared CPU with GPU and with FPGA performance for a parallel beam backprojection from 165 projections into a  $256^2$  image [6]. Their PC runs at 3.4 GHz and they compare the ATI X700 Pro and the NVidia GF7800 GPU whereby the Nvidia greatly outperforms the ATI GPU. The CPU code uses floating point arithmetics and an image is backprojected in 507 ms. The FPGA uses fixed point arithmetics with 32 bit precision and does the same job in 22 ms. The GPU (Nvidia) with 32 bit fixed point arithmetics performs in 23.8 ms and with 16 bit it does the backprojection in 11.5 ms. Scaling these values to 512 projections,  $512^2$  image pixels and to 3.0 GHz yields 7.13 s (CPU), 273 ms (FPGA), 295 ms (GPU, 32 bit) and 143 ms (GPU, 16 bit), respectively, for one image.

## V. CONCLUSION

The Cell Broadband Engine enables hyperfast backprojection on a general purpose hardware. Our implementation greatly outperforms other existing hard- or software by at least one order of magnitude. Per second up to 330 images can be backprojected with a dual Cell. Similar performance can be achieved for cone-beam backprojection and for forward projection algorithms [13]. These enormous capabilities may leverage new applications such as real-time image reconstruction or statistical image reconstruction that are not available in clinical routine, yet [14].

## REFERENCES

[1] H. P. Hofstee, “Power efficient processor architecture and the Cell processor,” *Proceedings of the 11th International Symposium on High-Performance Computer Architecture*, Feb. 2005.

- [2] D. Pham, S. Asano, M. Bolliger, M. N. Day, H. P. Hofstee, C. Johns, J. Kahle, A. Kameyama, J. Keaty, Y. Masubuchi, M. Riley, D. Shippy, D. Stasiak, M. Suzuoki, M. Wang, J. Warnock, S. Weitzel, D. Wendel, T. Yamazaki, and K. Yazawa, “The design and implementation of a first-generation Cell processor,” *IEEE International Solid-State Circuits Conference*, pp. 184–185, Feb. 2005.
- [3] B. Flachs, S. Asano, S. H. Dhong, H. P. Hofstee, G. Gervais, R. Kim, T. Le, P. Liu, J. Leenstra, J. Liberty, B. Michael, H. Oh, S. M. Mueller, O. Takahashi, A. Hatakeyama, Y. Watanabe, and N. Yano, “A streaming processing unit for a Cell processor,” *IEEE International Solid-State Circuits Conference*, pp. 134–135, Feb. 2005.
- [4] M. Leeser, S. Coric, E. Miller, H. Yu, and M. Trepanier, “Parallel-beam backprojection: An FPGA implementation optimized for medical imaging,” *Proc. of the Tenth Int. Symposium on FPGA, Monterey, CA*, pp. 217–226, Feb. 2002.
- [5] T. Schiwietz, T.-C. Chang, P. Speier, and R. Westermann, “MR image reconstruction using the GPU,” *SPIE Medical Imaging Proc.*, vol. 6142, pp. 1279–1290, Feb. 2006.
- [6] X. Xue, A. Cheryauka, and D. Tubbs, “Acceleration of fluoro-CT reconstruction for a mobile C-arm on GPU and FPGA hardware: A simulation study,” *SPIE Medical Imaging Proc.*, vol. 6142, pp. 1494–1501, Feb. 2006.
- [7] H. Stark, J. W. Woods, I. Paul, and R. Hingorani, “An investigation of computerized tomography by direct Fourier inversion and optimum interpolation,” *IEEE Transactions on Biomedical Engineering*, vol. BME–28, no. 7, pp. 496–505, July 1981.
- [8] H. Schomberg and J. Timmer, “The gridding method for image reconstruction by Fourier transformation,” *IEEE Transactions on Medical Imaging*, vol. 14, no. 3, pp. 596–607, Sept. 1995.
- [9] S. Schaller, T. Flohr, and P. Steffen, “An efficient Fourier method in 3D reconstruction from cone-beam data,” *IEEE Transactions on Medical Imaging*, vol. 17, pp. 244–250, Feb. 1998.
- [10] S. Basu and Y. Bresler, “An  $O(N^2 \log N)$  filtered backprojection reconstruction algorithm for tomography,” *IEEE Transactions on Medical Imaging*, vol. 9, no. 10, pp. 1760–1773, Oct. 2000.
- [11] P.-E. Danielsson and M. Ingerhed, “Backprojection in  $O(N^2 \log N)$  time,” *IEEE Nuclear Science Symposium Record*, vol. 2, pp. 1279–1283, 1998.
- [12] C. Axelsson and P.-E. Danielsson, “Three-dimensional reconstruction from cone-beam data in  $O(N^3 \log N)$  time,” *Phys. Med. Biol.*, vol. 39, no. 3, pp. 447–491, 1994.
- [13] M. Kachelrieß, M. Knaup, and O. Bockenbach, “Hyperfast perspective cone-beam backprojection,” *IEEE Medical Imaging Conference Record*, pp. M01–007, 2006.
- [14] M. Knaup, W. Kalender, and M. Kachelrieß, “Statistical cone-beam CT image reconstruction using the Cell broadband engine,” *IEEE Medical Imaging Conference Record*, pp. M11–422, 2006.