Algorithm for Hyperfast Cone-Beam Spiral Backprojection

Sven Steckmann*[1], Michael Knaup[1], and and Marc Kachelrieß[1]

Institute of Medical Physics (IMP), University of Erlangen–Nürnberg, Henkestr. 91, 91052 Erlangen, Germany.

Abstract. Cone–beam spiral backprojection is computationally highly demanding. At first sight, the backprojection requirements are similar to those of cone–beam backprojection from circular scans such as it is performed in the widely used Feldkamp algorithm. However, there is an additional complication: the illumination of each voxel, i.e. the range of angles the voxel is seen by the x–ray cone is a complex function of the voxel position. The weight function has no analytically closed form and must be numerically determined. Storage of the weights is prohibitive since the amount of memory required equals the number of voxels per spiral rotation times the number of projections a voxel receives contributions and therefore is in the order of 10^9 to 10^{11} floating point values for typical spiral scans. We propose a new algorithm that combines the spiral symmetry with the ability of today's 64 bit CPUs to store large amounts of precomputed weights. Our new backprojection algorithm achieves up to 11 Giga voxel updates per second (GUPS) on a standard four–socked quad core CPU (Intel Xeon 7300 platform, 3 GHz, Intel Corporation). This equals the reconstruction of 225 images per second assuming each slice consists of 512×512 pixels, receiving contributions from 512 projections.

1 Introduction

High performance image reconstruction (HPIR) basically means high performance backprojection since backprojection is the most demanding step in the image reconstruction pipeline. Recently, we published our work on high performance parallel beam backprojection and high performance perspective cone–beam backprojection using cell broadband engine (CBE) based implementations and central processing unit (CPU) based implementations [1].

Spiral backprojection is, however, more complicated since the illumination of the voxels by the cone exhibits a complex voxel location–dependent behavior that must be taken into account during the backprojection step. Basically, this requires to compute a weight function $w(x, y, z, \alpha)$, where (x, y, z) is the voxel position and α is the angle of the ray that is backprojected, and apply this function during the backprojection. Numerous algorithms have been published that already use voxel–specific weighting [2–5]. Others assume some simplifications to circumvent the problem of voxel–specific weighting and thereby compromise either dose usage or image quality [6–12]. Also implementations of quasiexact

* Corresponding author: sven.steckmann@imp.uni–erlangen.de

and exact reconstruction algorithms such as [13–15] may profit from our new approach.

This paper proposes a new backprojection algorithm that is able to perform the correct voxel weighting and that can be used together with any spiral reconstruction algorithm, be it exact, approximate or iterative (in which case a corresponding forward projector must be programmed as well). This paper does not propose a new type of image reconstruction.

2 Backprojection

Let α be the projection angle. The source position as a function of α is given as

$$\boldsymbol{s}(\alpha) = \begin{pmatrix} R_\mathrm{F}\sin\alpha \\ -R_\mathrm{F}\cos\alpha \\ \text{đ}\alpha \end{pmatrix}$$

where $\text{đ} = d/2\pi$ and d is the table increment per full rotation and R_F the radius of the focal trajectory. Backprojection means evaluating

$$f(x,y,z) = \int d\alpha\, w(x,y,z,\alpha)\hat{p}(\alpha,u,v)$$

with $u = u(x,y,z,\alpha)$ and $v = v(x,y,z,\alpha)$ being the detector's lateral and longitudinal coordinates given by the intersection of the ray from $\boldsymbol{s}(\alpha)$ through the voxel (x,y,z) with the detector surface.

Our aim is to precompute and store the weight function $w(x,y,z,\alpha)$ as well as the detector look–up coordinates $u(x,y,z,\alpha)$ and $v(x,y,z,\alpha)$ before image reconstruction. Our aim further is to vectorize the backprojection algorithm.

To achieve full vectorization of the backprojection algorithm and to reduce the memory required to store the voxel–specific weights we now use the fact that the system has a spiral symmetry. Let

$$\hat{f}(x,y,\alpha) = f(x\cos\alpha - y\sin\alpha, x\sin\alpha + y\cos\alpha, \text{đ}\alpha)$$

be a new representation of the volume f. In the new volume $\hat{f}$ each slice rotates in the same fashion as the spiral trajectory. Hence $\hat{f}$ exhibits the same spiral symmetry as the data acquisition itself. The slice's z–position is determined by the projection angle α. Now, backprojection is

$$\hat{f}(x,y,\alpha) = \int da\, w(x,y,a)\hat{p}(\alpha + a, u, v)$$

with $u = u(x,y,a)$ and $v = v(x,y,a)$. The angle a is counting relative to the slice position.

This new backprojection equation is much more favorable to implementation since neither the integrand's weight function nor the look–up values u and v depend on the absolute angle α, they only depend on the relative angle a. Hence we can loop over a range of z–positions α and add the corresponding projection entries to the volume without reevaluation of the weighting or look–up values. If we represent $\hat{f}$ and $\hat{p}$ with α being the linear, and thus fast, variable in memory we achieve a fully vectorization of the backprojection. The spiral symmetry further helps to reduce the weight table w and the look–up positions u and v from four–dimensional arrays to significantly smaller three–dimensional tables. These can be easily held in memory of modern PCs where 16 to 32 GB of memory are typically available. The (non–optimized) reference source code of listing 1 illustrates the algorithm for backprojection.

Listing 1: Reference backprojection algorithm.

```
void SpiralBP(int const I, // Number of x-pixels x
              int const J, // Number of y-pixels y
              int const K, // Number of slices   z
              int const L, // Number of rows     v
              int const M, // Number of channels u
              int const N, // Number of views    a
                int const * const mLut, // u(x, y, a)
                int const * const lLut, // v(x, y, a)
              float const * const wLut, // w(x, y, a)
              float const * const Raw,  // p(alpha+a, u, v)
              float       * const Vol)  // f(x, y, alpha)
    {
    #define mlut(i, j, n) mLut[((0+i)*J+j)*N+n]
    #define llut(i, j, n) lLut[((0+i)*J+j)*N+n]
    #define wlut(i, j, n) wLut[((0+i)*J+j)*N+n]
    #define V(i, j, k)     Vol[((0+i)*J+j)*K+k]
    #define R(l, m, n, k) Raw[(((0+l)*M+m)*(N+K)+n+k]

    for(int i=0; i<I; i++) // x-loop
    for(int j=0; j<J; j++) // y-loop
    for(int n=0; n<N; n++) // a-loop
        {
          int const m=mlut(i, j, n); // u(x, y, a)
          int const l=llut(i, j, n); // v(x, y, a)
        float const w=wlut(i, j, n); // w(x, y, a)

        for(int k=0; k<K; k++) V(i, j, k)+=w*R(l, m, n, k); // z-loop
        }
    }
```

The listing clearly shows why this algorithm is good for vectorization. The innermost loop, i.e. the loop over `k`, is the fastest index. For treating this loop as a vector, the data belonging to this loop must be arranged linear in memory. This can be seen from the data layout as specified by `#define` directives: the index `k` is a simple offset in the data and thus accesses are linear in memory.

The limitation of this version of the algorithm is that the distance of adjacent slices is fixed to the increment Δz of the scanner between two adjacent projections. This problem can be resolved by a new variable `dN` that counts how many multiples of Δz the slices shall be separated. The extened algorithm is given in listing 2. Note that adding the freedom to specify the longitudinal sampling requires only the simple decomposition of the view number `n` into `n=kk*dN+dn`. This is nothing but a reordering of our projections.

Note that the innermost loop remains to be fast and vectorizeable. The main difference to our listing 1 is that the memory layout of the rawdata has become slightly more complex and is now five–dimensional.

3 Slice Rotation

The improved backprojection performance comes at the price of having the slices rotating together with the spiral. This requires us to add an additional rotation

Listing 2: Algorithm with slice increments.

```
void SpiralBP(int const I, // Number of x-pixels x
              int const J, // Number of y-pixels y
              int const K, // Number of slices   z
              int const L, // Number of rows     v
              int const M, // Number of channels u
              int const N, // Number of views    a
              int const dN, // Slice increment
              int const * const mLut, // u(x, y, a)
              int const * const lLut, // v(x, y, a)
              float const * const wLut, // w(x, y, a)
              float const * const Raw,  // p(alpha+a, u, v)
              float       * const Vol)  // f(x, y, alpha)
  {
  #define mlut(i, j, n) mLut[((0+i)*J+j)*N+n]
  #define llut(i, j, n) lLut[((0+i)*J+j)*N+n]
  #define wlut(i, j, n) wLut[((0+i)*J+j)*N+n]
  #define R(l, m, dn, kk, k) Raw[(((0+dn)*L+l)*M+m)*KTot+kk+k]
  #define V(i, j, k) Vol[((0+i)*J+j)*K+k]

  int const KK=N/dN, KTot=KK+K;

  for(int i=0; i<I; i++) // x-loop
  for(int j=0; j<J; j++) // y-loop
  for(int dn=0; dn<dN; dn++) // a-loop (dn-part)
  for(int kk=0; kk<KK; kk++) // a-loop (kk-part)
      {
      int const n=kk*dN+dn; // compose n using kk and dn

        int const m=mlut(i, j, n); // u(x, y, a)
        int const l=llut(i, j, n); // v(x, y, a)
      float const w=wlut(i, j, n); // w(x, y, a)

      for(int k=0; k<K; k++)  V(i, j, k)+=w*R(l, m, dn, kk, k);  // z-loop
      }
  }
```

step. A more significant disadvantage of the approach is that the field of view (FOV), i.e. the region of the field of measurement (FOM) that shall be reconstructed, must be centered in the isocenter and it will be circular. To obtain rectangular FOVs some clipping must occur after the rotation step. In comparison to the high reconstruction speed, this does not carry any weight if we drop some reconstructed voxels.

Rotation Algorithm

To rotate the slices back we have to implement the transform

$$f(x, y, z) = \hat{f}(x\cos\alpha + y\sin\alpha, y\cos\alpha - x\sin\alpha, \alpha).$$

The image processing literature suggests many possible solutions to rotate images [16–18]. To be fast and accurate we use a simple destination–driven resampling of the rotated images with a two–by–two point interpolation. This interpolation uses a trapezoidal function to prevent loss of spatial resolution and to avoid creating aliasing. Slice rotation requires interpolation only in the lateral variables and no longitudinal interpolation since the rotated slices are generated just at the required z–positions (i.e. the values of α are chosen to match the required slice positions z).

For interpolation we use the trapezoidal function $T_w(\chi)$. It consists of a plateau of size $1-w$ and has a full width of $1+w$. Mathematically,

$$T_w(\chi) = \frac{1}{2w}\begin{cases} 0 & \text{if } |2\chi| > 1+w \\ 1+w-|2\chi| & \text{else if } |2\chi| > 1-w \\ 2w & \text{else} \end{cases}$$

The parameter χ stands for the index domain where interpolation occurs (e.g. for pixel columns i or rows j). Note that $w = 0$ results in nearest neighbor interpolation while $w = 1$ gives a linear interpolation algorithm.

The slice rotation algorithm uses the parameter $w = 0.5$ for interpolation in the x– and y–direction.

Kernel Modifications

To compensate for the smoothing caused by the interpolation during slice rotation we have to modify the reconstruction kernel to return to the original spatial resolution and image noise level. An empirical approach is used to do so. To describe the smoothing of the point spread function during the interpolation we used a convolution with a Gaussian function. In Fourier domain this means:

$$\hat{K}_m = K_m e^{a\,m^2}.$$

K_m, with m being the channel index, represents the standard kernel (in Fourier domain) that would be used for a conventional spiral backprojection algorithm without slice rotation. $\hat{K}_m$ is our kernel modification. The parameter $a > 0$ is empirical chosen to match the spatial resolution and the image noise to the noise obtained with a standard backprojection algorithm.

More Advanced Grids

Since slice rotation requires a final rotation and therefore a final resampling step and since this requires an empirical kernel adaptation one can also think of doing the complete backprojection on a non–Cartesian grid at no additional costs. Resampling to the Cartesian grid can then be done together with the slice rotation at no additional cost. As presented in reference [19] a hexagonal grid has several advantages over the standard Cartesian sampling. In comparison to the Cartesian grid, the hexagonal grid needs only $\sqrt{3}/2 \approx 0.866$ times the number of pixels for the same sampling density.

4 Results

For our studies a 3rd generation scanner geometry with $R_\mathrm{F} = 600\,\mathrm{mm}$, $R_\mathrm{D} = 450\,\mathrm{mm}$ and $R_\mathrm{M} = 800\,\mathrm{mm}$ was used. During one rotation $N_{360} = 512$ projections were simulated with a table feed of $d = 32\,\mathrm{mm}$ per rotation. This corresponds to a pitch of one. We are using a detector with $M = 512$ channels of 0.9 mm width and $L = 64$ detector rows with a slice thickness of 0.5 mm (dimensions corresponding to the isocenter).

The patient data were acquired with a Siemens Sensation 64 scanner (Siemens Healthcare, Forchheim, Germany). This scanner has the same geometry as used in our simulations but it acquires $N_{360} = 1160$ projections per rotation with each detector row consisting of $M = 672$ channels.

Images were reconstructed using the EPBP reconstruction algorithm, which is an approximate Feldkamp–type image reconstruction algorithm for spiral (and sequential) CT with a flexible voxel–specific weighting scheme that can allow for either 100% dose usage or for phase–correlated imaging or it can be used to only backproject data from the minimal data window (Pi window) [2]. The reconstructed volume consists of 512^3 voxels.

Image Quality

Since our hyperfast general purpose backprojection approach requires an additional slice rotation step we have to analyze whether this additional step changes the image quality (spatial resolution and noise).

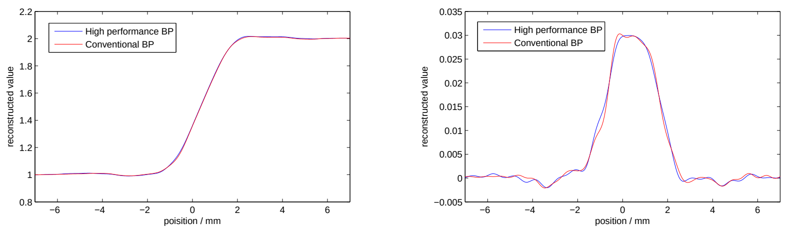
To compare image quality we define the reference gold–standard image to be an unrotated image, i.e. an image before the rotation step that does not need any rotation (apart from multiples of 90° which does not require interpolation). The reference image represents the maximum image quality achievable by the selected reconstruction algorithm.

To measure the noise propagation and compare it to our standard, a simulated cylindrical phantom filled with water was simulated and evaluated. Table 1 shows that the modified reconstruction kernel correctly compensates for differences in image noise due to the rotation step: both the conventional backprojection and our hyperfast approach yield identical image noise.

Algorithm Type	Image Noise	Spatial Resolution
Conventional Backprojection	58.0 HU	0.947 mm
Hyperfast Backprojection	57.0 HU	0.945 mm

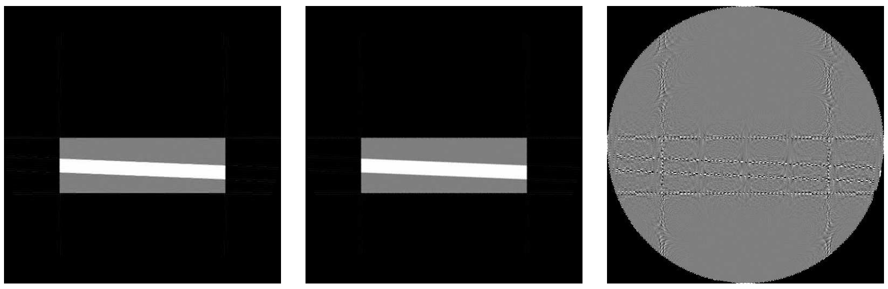
Table 1. The evaluation of image quality shows that both reconstructions are identical. Due to the modified kernel the slice rotation does neither increase or decrease noise or spatial resolution.

The spatial resolution was assessed using reconstructions of a slanted edge. Figure 2 shows such an image. To measure the spatial resolution the top edge of the inner rhomboid was used.



(a) Edge spread function $p(x)$ (b) Pseudo point spread function $p'(x)$

Fig. 1. The edge spread function and the pseudo point spread function of an example image.

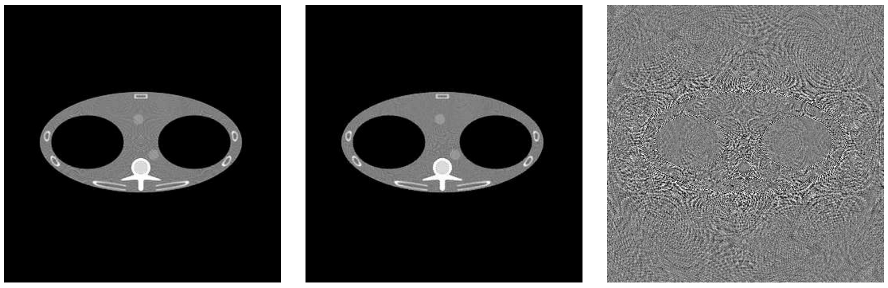


(a) Conventional backprojection reconstruction (b) Hyperfast backprojection reconstruction (c) Difference

Fig. 2. The resolution test phantom (with -1000 HU for air, 0 HU for the phantom body and 1000 HU for the slanted edge part). The images are displayed with C=0 HU and W=1000 HU and the difference image uses C=0 HU and W=250 HU.

Table 1 shows that the hyperfast backprojection with its final rotation step does not introduce any loss in spatial resolution. This result is confirmed regarding figure 1 which shows the edge spread function and the pseudo point spread function for both approaches, a conventional backprojector (without slice rotation) and the hyperfast spiral backprojector with slice rotation. If we have a look at the difference images in figure 2 we recognize some aliasing noise located at the edges of the objects. Since this is a natural behavior and since we cannot appreciate these subtle aliasing artifacts in the original images but only in difference images we consider the influence of the slice rotation to be negligible.

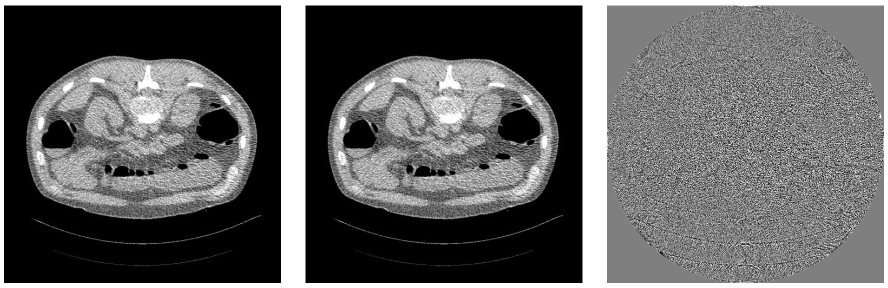
The semianthropomorphic FORBILD thorax phantom (http://www.imp.uni-erlangen.de/phantoms) was simulated and reconstructed to demonstrate the image quality in a more realistic setting. The images and the difference image in figure 2 does not show any conspicuousness or additional artifacts.



(a) Conventional backprojection reconstruction (b) Hyperfast backprojection reconstruction (c) Difference

Fig. 3. FORBILD thorax phantom. Gray level C=0 HU, W=500 HU for the images and C=0 HU, W=100 HU for the difference image.

To demonstrate the quality of the algorithm in the clinical routine, a dataset from a Siemens Sensation 64 scanner was reconstructed with the reference and with the hyperfast algorithm (figure 4). In contrast to the two simulated phantoms, the clinical data contain noise. The difference image shows no anatomical structures inside the patient: hence both algorithms provide identical image quality.



(a) Conventional backprojection reconstruction (b) Hyperfast backprojection reconstruction (c) Difference

Fig. 4. Patient data. The images are scaled with C=0 HU, W=500 HU and with C=0 HU, W=100 HU for the difference image.

Time Measurements and Implementation Details

For our time measurements we used an Intel Xeon 7300 platform with four X7350 processors running at 2.93 GHz. The results are shown in table 2. For a fair comparison between different algorithms we calculate the total numbers of updates and divide them by 1024^3 to get our giga updates (GU). An update is a summation of the rawdata value to the final volume. After a time measurement we calculate the giga updates we can handle per second (GUPS, giga updates per second). For a spiral trajectory this is not as easily to calculate as for a circle, so let us make an example for the circle trajectory. First: Assume we are using 512 projections per 360°. Our volume contains 512^3 voxels, then we have a total amount of 64 GU. Due to the fact of the more complex spiral, illumination interruption may occur and no analytical formula can be given to calculate the GU-value. In the above described geometry reconstruction of a typical volume having 512^3 voxels we need 47.8 GU using the EPBP algorithm.

The code measured here is an optimized version which uses well known techniques like loop unrolling and subset building to achieve high cache utilization and so best performance (optimization tricks can be found in reference [20]). For multiprocessing (we have up to 16 cores for data processing) the code is parallelized with OpenMP directives.

	BP time	Operation Count	Performance	Frame Rate
EPBP (conv.)	39.5 s	47.8 GU	1.22 GUPS	13.0 s^{-1}
EPBP	4.72 s	48.2 GU	10.7 GUPS	114 s^{-1}
1-Pi	2.29 s	25.3 GU	11.1 GUPS	225 s^{-1}

Table 2. Timing results. The conventional backprojection algorithm is compared to the hyperfast backprojection which improves performance by about a factor of five.

To obtain an overview of the resources required we summarize the size of the weighting and look–up tables and of the data arrays in table 3. The number of entries of the look–up tables was calculated using the following formula:

$$dN \cdot I \cdot J \cdot (KK + K).$$

KK is highly dependent from the algorithm used. KK represents the numbers of projections needed for reconstructing one complete slice. For example we find $KK = 632$ for the EPBP algorithm whereas the 1–Pi window reconstruction yields $KK = 128$. Note that table reftab:tableEntriesMemory was generated using $K = 256$. Consequently it took us two calls to the backprojection function to reconstruct the complete volume with 512 slices.

	Entries	Memory Usage
Weighting tables	$209 \cdot 10^6$	800 MB
Lookup tables	$209 \cdot 10^6$	800 MB
Raw data	$116 \cdot 10^6$	444 MB
Volume data	$67 \cdot 10^6$	256 MB
Sum		2300 MB

Table 3. Table entries and memory usage needed for reconstructing $K = 256$ slices.

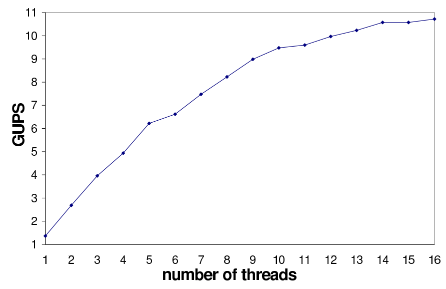


Fig. 5. Scaling on the system using multiple threads.

Hyperfast spiral backprojection requires a large memory bandwidth. To analyze the situation for our architecture we conducted reconstructions with varying number of cores, or threads (figure 5). The graph runs into saturation after about 13 threads which indicates that the memory bandwidth is limited (to about 5 GB/s). In this case the threads have to share the same bus and therefore the bandwidth available is divided. The Cell Broadband Engine (CBE) that has a memory bandwidth of 25 GB/s is more efficient in this case (table 4).

Architecture	Cores	Spiral BP	Perspective BP	Parallel BP
Cell (Mercury CBE)	8	23 GUPS	4.7 GUPS	21 GUPS
Cell (Mercury CBE)	16	46 GUPS	9.4 GUPS	42 GUPS
Intel Xeon 7300	16	11 GUPS	13 GUPS	58 GUPS
NVIDIA GeForce 8800 GTX	128	–	12 GUPS	–

Table 4. Comparison between different reconstruction algorithms and platforms.

Comparing this method to high performance implementations of other reconstruction types (RayConStruct, Nürnberg, Germany) we find that the spiral backprojection, although highly optimized, is still slower than the computational intensive and hard to vectorize perspective backprojection (table 4) on the CPU. Tests with a smaller amount of rawdata (a downsampled detector) shows a convergence of the performance from spiral backprojection towards the parallel backprojection.

5 Discussion

Our hyperfast backprojection approach is a general–purpose backprojector since it can be used with any spiral image reconstruction algorithm. It benefits from using the symmetries of the trajectory. The concept itself can be adapted to other trajectories of high symmetry as well. A drawback of the proposed algorithm is the requirement for circular FOVs that are centered about the rotation center. This drawback is somewhat compensated by the high reconstruction speed of the algorithm. The fact that the final slice rotation step is required may be regarded as a disadvantage. However, in combination with more sophisticated reconstruction grids, such as the hexagonal grid for example, it is an ideal setting. Computationally, the final slice rotation is costless.

The performance values shown in this paper are only an example that is valid for this specific scanner geometry and reconstruction algorithm. Our experiments have shown that slight changes of the geometry (e.g. a different cone–angle, or a different numbers of slices) may significantly change those values. The numbers cited here are the most conservative ones we observed. The performance values that we obtain for various geometry range from about 5 GUPS to up to 34 GUPS on the CPU. These significant variations between geometries and algorithms can be understood regarding the fact that memory bandwidth constitutes the major performance bottleneck, today. Different memory access patterns (e.g. by changing the spiral pitch value) may therefore have significant effects.

The most significant impact on the bandwidth bottleneck is the access to the rawdata, which is hardly predictable. The rawdata storage pattern can be improved if one desires to optimize the algorithm for a specific scanner and scan geometry.

6 Other Publications

Recently other implementation for spiral CT were published (Comparisons to parallel-beam or perspective cone-beam backprojection algorithms can be found in reference [1]).

One of them using a standard Siemens geometry of 1160 projections and 672 detector channels [21]. This group uses a HP 6200 Workstation with 2 Intel Xeon 3.2 GHz processors for their backprojection. They backproject one slice with 512×512 pixels with their optimized algorithm within 1.32 s. This reconstruction has an operational count of 0.28 GU and given the speed of 1.32 s per slice we find that they achieve a performance of 0.21 GUPS.

A GPU–based spiral image reconstruction was analyzed in reference [22]. The authors report about reconstructions of simulated data (256 slices) and measured data (216 slices) with a pitch value of one. Each slice has 256×256 pixels and there are 360 projections (simulation) and 720 projection (measurement) per rotation to backproject. With a voxel update number of 5.6 GU for the simulation and 9.5 GU for the measurement this is a rather tiny problem. They are using two different GPUs, the best results are obtained with an NVIDIA GeForce 8800 GTX where they observe up to 1.5 GUPS. Their CPU–based reference code achieves only 0.07 GUPS.

7 Summary and Conclusion

The performance of spiral cone–beam image reconstruction is improved using our hyperfast general–purpose backprojection approach. Due to using the spiral symmetry in an elegant way we achieve a high level of vectorization combined with parallelization. The approach is compatible with today's CPUs but also performs very well on the CBE.

A eight–fold performance improvement was observed when comparing this new approach to our highly optimized conventional spiral cone–beam backprojector. Comparing to other groups our hyperfast approach seems to be faster by order of magnitudes.

The image quality achievable with the new backprojector is identical to conventional reconstructions.

Acknowledgments

This work was supported by Visage Imaging GmbH, Berlin, Germany. Parts of the reconstruction software were provided by RayConStruct GmbH, Nürnberg, Germany. We thank the Intel Corporation, for providing their latest multi–core platforms.

References

1. Kachelrieß, M., Knaup, M., Bockenbach, O.: Hyperfast parallel–beam and cone–beam backprojection using the cell general purpose hardware. Medical Physics **34**(4) (April 2007) 1474–1486
2. Kachelrieß, M., Knaup, M., Kalender, W.A.: Extended parallel backprojection for standard 3D and phase–correlated 4D axial and spiral cone–beam CT with arbitrary pitch and 100% dose usage. Medical Physics **31**(6) (June 2004) 1623–1641
3. Kachelrieß, M., Knaup, M., Kalender, W.A.: Multi–threaded cardiac CT. Medical Physics **33**(7) (July 2006) 2435–2447
4. Taguchi, K., Chiang, B.S.S., Silver, M.D.: A new weighting scheme for cone-beam helical CT to reduce the image noise. Phys Med Biol **49**(11) (Jun 2004) 2351–2364
5. Tang, X., Hsieh, J., Nilsen, R.A., Dutta, S., Samsonov, D., Hagiwara, A.: A three-dimensional-weighted cone beam filtered backprojection (CB–FBP) algorithm for image reconstruction in volumetric CT–helical scanning. Phys Med Biol **51** (2006) 855–874
6. Danielsson, P.E., Edholm, P., Eriksson, J., Magnusson-Seger, M., Turbell, H.: Helical cone beam scanning an reconstruction of long objects using 180 degree exposure. Patent Appl. PCT/SE 98/00029 (Jan. 1998)
7. Kachelrieß, M., Schaller, S., Kalender, W.A.: Advanced single-slice rebinning in cone–beam spiral CT. Medical Physics **27**(4) (2000) 754–772
8. Taguchi, K., Aradate, H.: Algorithm for image reconstruction in multi-slice helical CT. Medical Physics **25** (1998) 550–561
9. Larson, G.L., Ruth, C.C., Crawford, C.R.: Nutating slice CT image reconstruction apparatus and method. United States Patent 5,802,134 (1998)
10. Stierstorfer, K., Flohr, T., Bruder, H.: Segmented multiple plane reconstruction: a novel approximate reconstruction scheme for multi-slice spiral CT. Phys Med Biol **47** (2002) 2571–2581

11. Defrise, M., Noo, F., Kudo, H.: A solution to the long-object problem in helical cone–beam tomography. Phys Med Biol **45**(3) (Mar 2000) 623–643
12. Proksa, R., Köhler, T.and Grass, M.T.J.: The n-pi-method for helical cone-beam CT. IEEE Transaction on medical imaging **19** (2000) 848–863
13. Bontus, C., Köhler, T., Proksa, R.: A quasiexact reconstruction algorithm for helical CT using a 3Pi acquisition. Medical Physics **30**(9) (Sep 2003) 2493–2502
14. Katsevich, A.: Theoretically exact FBP–type inversion algorithm for spiral CT. SIAM Journal of Applied Mathematics **62** (2002) 2012–2026
15. Noo, F., Pack, J., Heuscher, D.: Exact helical reconstruction using native cone-beam geometries. Phys Med Biol **48**(23) (Dec 2003) 3787–3818
16. Bella, E.V.R.D., Barclay, A.B., Eisner, R.L., Schafer, R.W.: A comparison of rotation-based methods for iterative reconstructionalgorithms. IEEE transactions on nuclear science **43**(6) (December 1996) 3370–3376
17. Owen, C.B., Makedon, F.: High quality alias free image rotation. In: Proceedings of 30th Asilomar Conference on Signals, Systems, and Computers, Dartmouth College Computer Science (November 1996)
18. Tosoni, L., Lanzavecchia, S., Bellon, P.L.: Image and volume data rotation with 1- and 3–pass algorithms. Comput Appl Biosci **12**(6) (1996) 549–552
19. Knaup, M., Steckmann, S., Bockenbach, O., Kachelrieß, M.: CT image reconstruction using hexagonal grids. IEEE Medical imaging Conference Record **M13–277** (2007) 2074–3076
20. Knaup, M., Kachelrieß, M.: Acceleration techniques for 2D parallel and 3D perspective forward and backprojections. In: 9th International Meeting on Fully Three-Dimensional Image Reconstruction in Radiology and Nuclear Medicine. (2007) 45–48
21. Zeng, K., Bai, E., Wang, G.: A fast CT reconstruction scheme for a general multi-core pc. Journal of Biomedical Imaging **2007**(1) (2007) 1–9
22. Bi, W., Chen, Z., Zhant, L., Xing, Y.: Accelerate helical cone–beam CT with graphics hardware. Proceedings SPIE **6913** (2008) published online