Optimizing Katsevich Image Reconstruction Algorithm on Multicore Processors

Eric Fontaine
School of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332
eric.fontaine@gatech.edu

Hsien-Hsin S. Lee
School of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332
leehs@gatech.edu

Abstract

The Katsevich image reconstruction algorithm is the first theoretically exact cone beam image reconstruction algorithm for a helical scanning path in computed tomography (CT). However, it requires much more computation and memory than other CT algorithms. Fortunately, there are many opportunities for coarse-grained parallelism using multiple threads and fine-grained parallelism using SIMD units that can be exploited by emerging multicore processors. In this paper, we implemented and optimized Katsevich image reconstruction based on the previously proposed π-interval method and cone beam cover method and parallelized them using OpenMP API and SIMD instructions. We also exploited symmetry in the backprojection stage. Our results show that reconstructing a $1024 \times 1024 \times 1024$ image using 5120 512×128 projections on a dual-socket quad-core system took 23,798 seconds on our baseline and 642 seconds on our final version, a more than 37 times speedup. Furthermore, by parallelizing the code with more threads we found that the scalability is eventually hinged by the limited front-side bus bandwidth.

1. INTRODUCTION

Computed tomography is a technique that attempts to reconstruct the 3-D density of a volume by analyzing many 2-D projections of that volume at different angles. A typical projection setup is shown in Figure 1. Here, the volume of interest (VOI) consists of a cylindrical 3-D region of space. It is represented digitally on a computer as a 3-D array of voxels ("volume elements") containing the value of the density at each small cubical region of the volume. Since x-rays are typically used in practice to generate the projections, the voxels actually hold the amount of x-ray attenuation, not the material density. The x-rays originate at the projection source, travel through the VOI, and hit a detector opposite the projection source. Each projection is stored digitally on a computer as a 2-D array of texels ("texture elements") containing the total attenuation of each ray through the VOI. The essential step to reconstruct the original image is backprojection - by taking each projection and smearing its values back over the VOI along the paths of the original x-rays. The projections usually first undergo a filtering step using some type of high-pass filter to correct for the frequency components lost during backprojection. There exist a plethora of so called filtered-backprojection (FBP) algorithms designed for various projection scanning paths and beam types, all of which contain these two important steps, with varying degrees of quality and speed. Examples of beam types are: cone beam rays that spread radially away from a source point in 3-D, fan beam rays that spread radially away but only lie in a single 2-D plane, and parallel beam rays that travel in the same direction. Scanning paths are usually circular or helical.

The Katsevich image reconstruction algorithm, developed in 2002, is the first theoretically exact cone beam image reconstruction algorithm for helical scanning paths [6]. Helical scanning paths are important, because they allow the reconstruction of long objects such as humans with greater resolution than circular scanning paths. A helical scanning path usually has a shorter x-ray acquisition time than does a series of circular scanning paths. The Katsevich algorithm is of the FBP form, because the formula can be expressed as a backprojection of filtered projections. However, Katsevich reconstruction requires more computation and memory than other FBP algorithms such as circular FDK [2]. In addition to the filtering and backprojection stage, the Katsevich reconstruction algorithm also requires differentiation of the projections, remapping to and back from filtering coordinates, calculation of π-intervals, special weighting of the projection borders, as well as more projection data. Like most FBP algorithms, the most time consuming stage is backprojection.

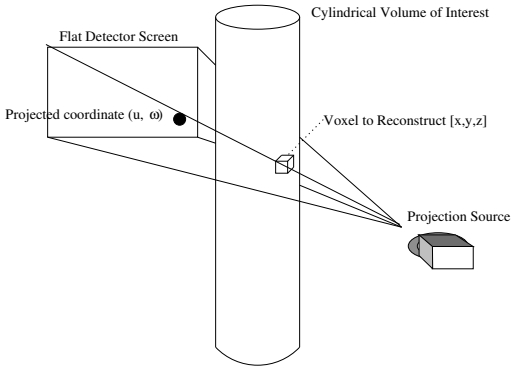


Figure 1: 2-D projection of a 3-D VOI

Fortunately, there exist many opportunities for coarse-grained data parallelism. For instance, the processing and backprojection of one projection is independent from that of other projections. Similarly, the backprojection of one region of the 3-D volume is independent of other regions. Multi-core processors can exploit this coarse-grained data parallelism by assigning different projections and/or different reconstruction regions to different threads running on different cores. In addition, there exist many opportunities for fine-grained data parallelism. The processing of neighboring texels within a single projection are for the most part independent from each other. Backprojection of neighboring voxels are also independent of each other. These examples of fine-grained data parallelism are ideal candidates for SIMDification. Further optimizations to reduce computations can be achieved by exploiting the symmetry of the trigonometric functions used during backprojection. After a little inspection, we determined that the backprojection calculations for every $\frac{\pi}{2}$ projection contain redundancies and therefore can be reused. By packing every $\frac{\pi}{2}$ projection together in a SIMD manner, multiple projections can be backprojected concurrently after calculating the backprojection coordinates only once.

In this paper, we exploited the parallelism in the Katsevich image reconstruction algorithm and performed reconstruction on a real multi-core platform. We first ported a Matlab-based open-source code [9], based on what we refer to as the π-interval method, to C. We implemented basic, well-known C optimizations and used Intel Integrated Performance Primitives (IPP), optimized with SIMD, for projection filtering. We used OpenMP to generate multiple threads to process different projections and different reconstruction regions. We then implemented and optimized the cone beam cover method [10] and parallelized it by processing different projections with different threads. The cone beam cover method is essentially

a loop interchange of the π-interval method. This greatly improved read and write memory locality and removed a significant amount of computation from the inner loop - to the point that four loop iterations could be unrolled and processed simultaneously using SIMD instructions. We also used SIMD for differentiation and used the Intel IPP for remapping to and back from filtering coordinates. Finally, we exploited the symmetry among each $\frac{\pi}{2}$ projection to perform SIMD backprojecting of four packed projections simultaneously.

2. RELATED WORK

The following papers discuss parallelization strategies for Katsevich image reconstruction. Deng [1] parallelized the Katsevich algorithm on a high performance cluster using the Message Passing Interface (MPI). Projections were divided among different processor nodes for differentiation and filtering. Each node broadcasted its processed projections to all other nodes when the filtering is done. Each node is then assigned a different region of the volume to backproject. Yang [10] proposed the cone beam cover method, which removes the burden of using π-intervals to determine the limits of integration and allows each projection to be fully utilized when read into memory. This method was then parallelized on a cluster by dividing up the projections among processors [11].

The following papers discuss SIMD optimizations for algorithms other than Katsevich image reconstruction. Kachelrie [5] implemented 2-D parallel beam backprojection and 3-D cone beam backprojection for a circular scanning path on the Cell processor. The local store memory management utilized the DMA to hide the memory latency of loading the next needed projection data while backprojecting the current projection. SIMD was used to unroll the inner loop four times and backproject four voxels simultaneously. Hong [3] [4] used SIMD for 3-D PET image reconstruction to exploit multiple symmetries existing in the backprojection stage. Zeng [13] used SIMD and multiple-threads to exploit circular symmetries during backprojection of fan beam filtered backprojection. Zeng's method can handle cone beam algorithms, but only after transforming the cone beam projections into fan beam projections. Our method to exploit symmetry with SIMD was developed independently of the above papers and differs in that our method works for 3-D cone beam reconstruction with helical scanning paths.

3. KATSEVICH ALGORITHM OVERVIEW

Although the original paper by Katsevich [6] provides the mathematical proof, it does not give implementation details. The following summarizes the relevant implementation details for a flat detector as provided by Wunderlich [9] and Noo [7].

3.1 A Flat Discrete Detector Implementation

The Katsevich algorithm uses a helical scanning path parameterized by λ:

$$\mathbf{y}(\lambda) = (Rcos(\lambda), Rsin(\lambda), P\frac{\lambda}{2\pi})$$

where R is the radius of the helix, $D = 2R$, and P is the pitch between two turns of the helix as shown in Figure 2. The cylindrical VOI occupies all points for $x^2+y^2 < r$ and $Zmin < z < Zmax$, where r is the radius of the cylinder, $r < R$, and r/R is the projection field-of-view (FOV). Note that although a 3-D rectangular grid of voxels is used to store the image, only the cylindrical volume inscribed in the 3-D rectangle is actually reconstructed.

There are $K = (nTurns)(nSourcesPerTurn) + 1$ discrete source projections, where $nTurns$ is the number of turns of the helix and $nSourcesPerTurn$ is the number of projection sources per turn. Since there are 2π radians in one turn of the helix, the difference in λ between adjacent projections is $\Delta\lambda = \frac{2\pi}{nSourcesPerTurn}$. Each discrete projection source λ_k (indexed by $k = [0..K]$) on the scanning path produces cone beam rays that traverse radially away

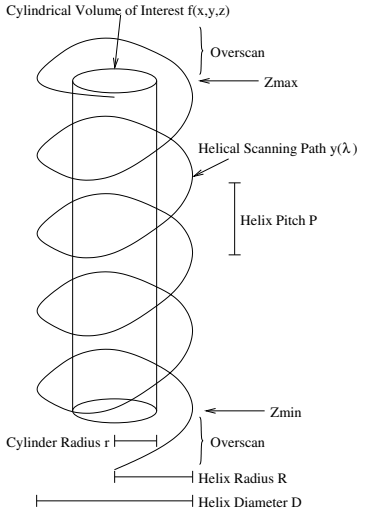


Figure 2: Helical projection scanning path surrounding VOI

from the projection source through the VOI and hit a vertical flat detector screen opposite the projection source. The intensity of a ray is attenuated by the density of the matter it passes through, so the total attenuation of a ray can be expressed as:

$$g(\lambda, \theta) = \int_0^\infty f(\mathbf{y}(\lambda) + t\theta)dt$$

where $\theta(\lambda, u, \omega)$ is a vector representing the 3-D direction of the ray originating at the projection source position $\mathbf{y}(\lambda)$ and striking the detector at (u, ω). The discrete detector array contains an even number of N columns (indexed by $i = [-\frac{N}{2}..\frac{N}{2} - 1]$) and M rows (indexed by $j = [-\frac{M}{2}..\frac{M}{2} - 1]$). Each point of the discrete detector $g0[k, i, j]$ holds the value of the ray $g(\lambda_k, \theta(\lambda_k, u_i, \omega_j))$ originating from the discrete projection source λ_k. The difference in u and ω between adjacent texels is Δu and $\Delta\omega$, respectively. The Katsevich algorithm can reconstruct the 3-D image $f[x, y, z]$ using the following steps:

3.1.1 Differentiation

The projections are first differentiated with respect to u, ω, and λ. The derivative is computed using the chain rule. In discretized coordinates, the partial derivatives are computed by calculating the difference between adjacent texels in the i, j, and k directions shown in Figure 3:

$$\begin{aligned} d\lambda = \quad & (g0[k+1,i,j] - g0[k,i,j] \\ + \quad & g0[k+1,i,j+1] - g0[k,i,j+1] \\ + \quad & g0[k+1,i+1,j] - g0[k,i+1,j] \\ + \quad & g0[k+1,i+1,j+1] - g0[k,i+1,j+1])/(4\Delta\lambda) \end{aligned}$$

$$\begin{aligned} du = \quad & (g0[k,i+1,j] - g0[k,i,j] \\ + \quad & g0[k,i+1,j+1] - g0[k,i,j+1] \\ + \quad & g0[k+1,i+1,j] - g0[k+1,i,j] \\ + \quad & g0[k+1,i+1,j+1] - g0[k+1,i,j+1])/(4\Delta u) \end{aligned}$$

$$\begin{aligned} d\omega = \quad & (g0[k,i,j+1] - g0[k,i,j] \\ + \quad & g0[k,i+1,j+1] - g0[k,i+1,j] \\ + \quad & g0[k+1,i,j+1] - g0[k+1,i,j] \\ + \quad & g0[k+1,i+1,j+1] - g0[k+1,i+1,j])/(4\Delta\omega) \end{aligned}$$

$$g1[k,i,j] = d\lambda + \frac{u^2+D^2}{D}du + \frac{u\omega}{D}d\omega$$

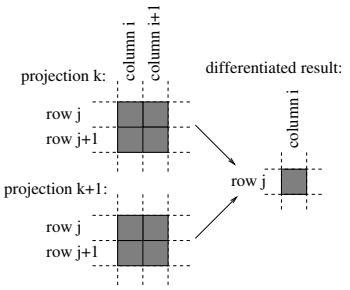


Figure 3: Projection texels required to produce one differentiated texel

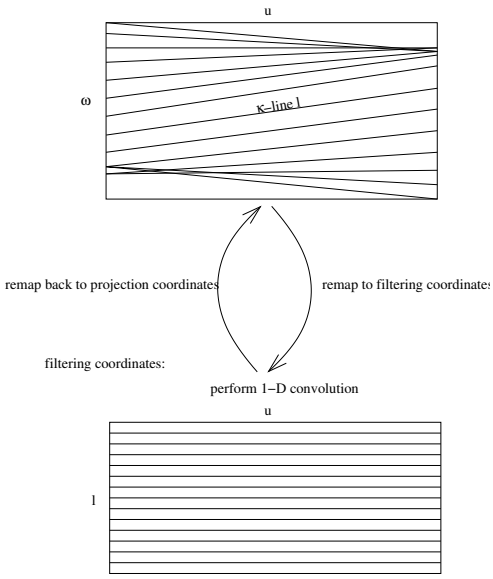


Figure 4: To filter a projection along the κ-lines, a projection is remapped to filtering coordinates where 1-D convolution can be performed and then remapped back to projection coordinates

The differentiated coordinates need to be length corrected since the detector screen is flat, not curved:

$$g2[k,i,j] = g1[k,i,j]\frac{D}{\sqrt{u^2+D^2+\omega^2}}$$

3.1.2 Filtering

The differentiated projections are then filtered along the κ-lines shown in Figure 4. A κ-line is formed by the intersection of a κ-plane with the detector screen, and a κ-plane is any plane that intersects the helix at three equally spaced points $y(\lambda)$, $y(\lambda+\psi)$, and $y(\lambda+2\psi)$ for $\psi \in (-\frac{\pi}{2}, \frac{\pi}{2})$. There are $L+1$ (where L is even) discrete κ-lines $\psi_l = l\Delta\psi$ (indexed by $l = [-L/2..L/2]$) and separated by an amount $\Delta\psi = \frac{\pi+2arcsin(r/R)}{L}$. The projections are first remapped from projection coordinates to filtering coordinates using linear interpolation according to the remapping function $\omega_\kappa(u_i, \psi_l)$:

$$\omega_\kappa(u_i,\psi_l) = \frac{DP}{2\pi R}(\psi_l + \frac{\psi_l}{tan\psi_l}\frac{u}{D})$$

$$g3[k,i,l] = g2[k,i,\frac{\omega_\kappa(u_i,\psi_l)}{\Delta\omega}]$$

This remapping aligns the κ-lines on horizontally rows so that

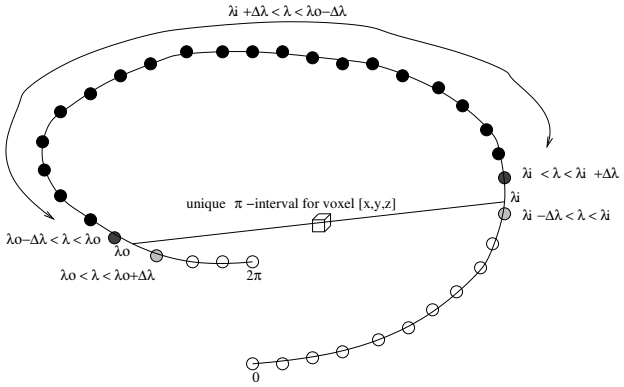


Figure 5: Illustration of a π-interval and the source points on the edges of the π-interval

1-D convolution can be performed over adjacent memory locations with the Hilbert transform kernel $h_H[i]$:

$$g4[k,i,l] = g3[k,i,l] \star h_H[i]$$

The filtered projections are then remapped back to the original coordinates using linear interpolation using $\hat{\psi}(u,\omega)$, the inverse of $\omega_\kappa(u,\psi)$:

$$g5[k,i,j] = g4[k,i,\frac{\hat{\psi}(u_i,\omega_j)}{\Delta\psi}]$$

3.1.3 Backprojection

Finally, these filtered projections are backprojection to reconstruct a particular voxel $[x,y,z]$:

$$f[x,y,z] = -\frac{1}{2\pi}\sum_{k=\lambda_i}^{\lambda_o}\frac{g5[k,\frac{u(\lambda_k,x,y)}{\Delta u},\frac{\omega(\lambda_k,x,y,z)}{\Delta\omega}]}{v(\lambda,x,y)}$$

where λ_o and λ_i represent the extremities of the unique π-interval containing $[x,y,z]$. A π-interval is any segment that intersects the helix at two points within one turn of the helix as shown in Figure 5. The π-intervals are calculated numerically according to [12]. The backprojection coordinates (u,ω) for each voxel $[x,y,z]$ are:

$$v(\lambda,x,y) = R - xcos\lambda - ysin\lambda$$
$$u(\lambda,x,y) = \frac{D}{v(\lambda,x,y)}(-xsin\lambda + ycos\lambda)$$
$$\omega(\lambda,x,y,z) = \frac{D}{v(\lambda,x,y)}(z - \frac{P}{2\pi}\lambda)$$

These coordinates must be converted to integer values to calculate the memory location in the projection data. The fractional component is computed to perform linear interpolation:

$$iu = int(u/\Delta u)$$
$$i\omega = int(\omega/\Delta\omega)$$
$$uFrac = u/\Delta u - float(iu)$$
$$\omega Frac = \omega/\Delta\omega - float(i\omega)$$

The four nearest neighbors to (u,ω) are loaded and weighted according to their fractional parts, shown in Figure 6:

$$\begin{aligned} g5_{interpolated}(\lambda,u,\omega) = \\ & g5[k,iu,i\omega](1-uFrac)(1-\omega Frac) \\ + \quad & g5[k,iu+1,i\omega](uFrac)(1-\omega Frac) \\ + \quad & g5[k,iu,i\omega+1](1-uFrac)(\omega Frac) \\ + \quad & g5[k,iu+1,i\omega+1](uFrac)(\omega Frac) \end{aligned}$$

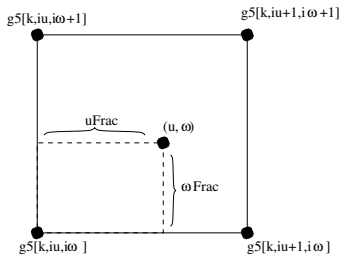


Figure 6: Linear interpolation of 4 neighboring texels

Two different methods for iterating during the backprojection stage are the π-interval method and the cone beam cover method.

3.2 π-Interval Method

The π-interval method refers to the method described by Wunderlich [9] which iterates over all filtered projections $g5[k, i, j$ to backproject a single voxel $[x, y, z]$.Each projection belonging to the corresponding π-interval adds its contribution to the reconstructed voxel. This contribution consists of the interpolated value of the projection at the location of the backprojected voxel coordinates, (u, ω). To improve the reconstructed image accuracy, it is necessary to include the adjacent projections just outside of the π-interval and use the trapezoidal rule to smooth the edges of the π-interval according to the following weighting:

$$\begin{array}{ccccccc} \frac{(1+d_{in})^2}{2} & \text{if} & (\lambda_i - \Delta\lambda) & < & \lambda & < & \lambda_i \\ \frac{1}{2} + d_{in} - \frac{d_{in}^2}{2} & \text{if} & \lambda_i & < & \lambda & < & (\lambda_i + \Delta\lambda) \\ 1 & \text{if} & (\lambda_i + \Delta\lambda) & < & \lambda & < & (\lambda_o - \Delta\lambda) \\ \frac{1}{2} + d_{out} - \frac{d_{out}^2}{2} & \text{if} & (\lambda_o - \Delta\lambda) & < & \lambda & < & \lambda_o \\ \frac{(1+d_{in})^2}{2} & \text{if} & \lambda_o & < & \lambda & < & (\lambda_o + \Delta\lambda) \end{array}$$

where $d_{in} = \frac{\lambda - \lambda_i}{\Delta\lambda}$ and $d_{out} = \frac{\lambda_o - \lambda}{\Delta\lambda}$.

3.3 Cone Beam Cover Method

The cone beam cover method avoids the need to calculate the π-intervals. Yang [10] proved that a point lies in the cone beam cover of a projection if and only if that projection belongs to the π-interval of that point. The cone beam cover of a projection is defined to be the set of all points that, when projected onto the detector, lie within the Tam-Danielsson window. The Tam-Danielsson window consists of all the projection points (u, ω) necessary to perform exact reconstruction [8]. The top and bottom of the Tam-Danielsson window are $\omega_{top}(u)$ and $\omega_{bottom}(u)$.

The differentiation and filtering stages are identical to those of the π-interval method. However, immediately after a projection is filtered, it is backprojected to the VOI. It contributes all of its backprojected values to all voxels lying within its cone beam cover. This improves the temporal locality of accessing projection memory, because immediately after a projection is backprojected, it can be discarded from memory, never to be used again. To improve the reconstructed image accuracy, if the bottom or top boundaries of the cone beam cover do not lie within the cone beam cover of the adjacent projection, then the backprojected values need to be weighted by the following amount:

$$0.5 + \frac{\left| z_k^{boundary} - z \right|}{\left| z_k^{boundary} - z_{k\pm1}^{boundary} \right|}$$

The cone beam cover method also simplifies computation in the inner loop, since the backprojected u coordinate of a voxel does not depend on the z coordinate. The outer loop can iterate over all $[x, y]$ in the cylindrical VOI. Then the u backprojection coordinate can be calculated and reused by the inner loop, which iterates over all z that belong to the cone beam cover. Since the backprojection ω coordinate only changes by an amount $\frac{d\omega}{dz}$ when z increases by 1, this difference can be precomputed in the outer loop and used in the inner loop to calculate the next ω coordinate, as shown in Figure 7. To improve read locality, the projection u and ω coordinates are reversed so that projection memory is accessed sequentially when iterating over the z coordinate. Similarly to the π-interval method, linear interpolation is used when looking up projection values. The entire volume is reconstructed once all projections have finished backprojection.

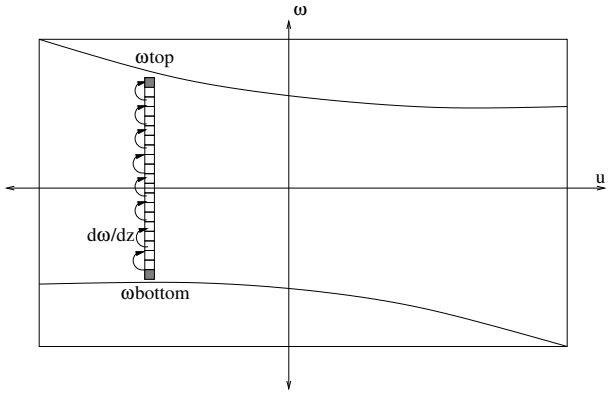


Figure 7: Decrease in backprojection computation afforded by cone beam cover method; simply increment ω by $\frac{d\omega}{dz}$ each z loop iteration

3.4 Basic computation optimizations

Calculating the π-intervals requires solving a non-linear equation iteratively [12]. However, assuming that the helical parameters are constant, the π-intervals only need to be generated once and can be reused every time a new image needs to be reconstructed. In addition, due to the symmetry of a helix, only one horizontal slice at $z = 0$ of the π-intervals needs to be generated. The π-interval for any voxel $[x, y, z]$ can be determined by rotating the $[x, y]$ coordinates by an amount $z\frac{2\pi}{P}$. Linear interpolation is used to look up the rotated π-interval value from the reference horizontal slice $z = 0$.

If a backprojected coordinate lies on outside of the range of the projection coordinates, then it should be discarded. However, in order to avoid expensive *if statements* inside of inner loop, this test can be removed by padding the detector array with zeroes. Noo [7] provides sufficient conditions for the width of the detector array to be just large enough to hold the entire FOV. Thus any backprojected values will lie inside the projection boundaries, so bounds checking is not required.

The backprojection contribution from projections at the edges of the π-interval need to be weighted specially. Instead of using *if statements* in the inner loop to test if a projection is at the edge of the π-interval, the first two and last two iterations of the inner loop are simply unrolled from the remaining iterations of the inner loop and weighted separately. Similarly for the cone beam cover method, the first and last iterations of the inner backprojection loop are unrolled from the rest of the loop.

Since `sin` and `cos` are used repeatedly, and since there are a fixed number of angles used, the values of `sin` and `cos` are found in a precomputed lookup table. Also, the projection remapping coordinates are precomputed and stored in an array. Finally, the top and bottom boundaries of the Tam-Danielsson window are also tabulated based on the u coordinate. This continuous function can be well approximated using linear interpolation, as shown in Figure 8.

Prefetching is used to hide the latency of accessing memory across the memory hierarchy by anticipating future memory accesses. When implementing the cone beam cover method, as one z-strip is backprojected, the projection data and image memory for the next z-strip is prefetched.

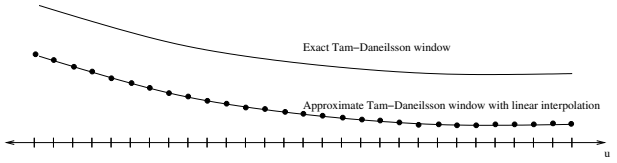


Figure 8: Linear interpolation of Tam-Danielsson window

4. PARALLELIZATION TECHNIQUES

4.1 Coarse-grained using Multi-threads

The parallelize strategy for the π-interval method is shown in Figure 9. We dynamically assign the next adjacent projection to the next available thread. Because the differentiation operation requires data from two adjacent projections, this assignment hopefully improves read locality since the previous adjacent projection is likely to still be present in the shared cache. (Note however that if the last level cache is not shared, then it is more efficient to assign chunks of adjacent projections to each individual thread). After all projections are done with differentiation and filtering, there is an implicit barrier. When performing backprojection, we dynamically assign the next adjacent horizontal slice of the VOI to the next available thread. This improves read locality if the last level cache is shared, since the backprojection of neighboring image slices have similar backprojection coordinates.

When parallelizing the cone beam cover method [10], we dynamically assign the next adjacent projection to the next available thread as shown in Figure 10. Since the cone beam covers of adjacent projections overlap a great deal, this improves locality when accumulating their partial reconstruction to the image memory if the last level cache is shared. However, since different threads now may potentially write to the same image memory location, we must perform some type of write synchronization. This write synchronizations can be performed at a fine-granularity during each accumulation by enclosing the accumulation of a voxel in a critical region. However, this is costly since it occurs in the innermost loop. Instead, different threads use a lock to obtain exclusive access to a specific z-strip of image memory, and then release the lock when done backprojecting that z-strip. Since this synchronization does not occur in an inner loop, it has much lower overhead.

4.2 Fine-grained using SIMD

4.2.1 SIMD Cone Beam Cover Backprojection

The inner loop of the cone beam cover method is simplified enough such that four loop iterations can be unrolled and processed simultaneously using the vertical SIMD model, since four 32-bit floating-point values can fit into one 128-bit SIMD register. A scalar version of the inner loop is run until the first 128-bit aligned image memory address is encountered, because accessing memory along 128-bit aligned boundaries with SIMD reads and writes is much faster than accessing unaligned memory. The backprojection ω integer lookup coordinate $i\omega$ and fractional components $\omega Fracs$ for four consecutive z image coordinates are calculated together in one packed SIMD register from the floating point packed ω coordinates $f\omega$.

Linear interpolation requires accessing the four nearest neighbors. Since neighbors $g[k, iu, i\omega]$ and $g[k, iu, i\omega + 1]$ occupy consecutive memory locations, as well as neighbors $g[k, iu+1, i\omega]$ and $g[k, iu+1, i\omega+1]$, these pairs of neighbors can be loaded together with a 64-bit load. These are shuffled such that the values for each iteration occupy different positions in the SIMD registers. The fractional components of the backprojected coordinates are packed to form SIMD weights that are multiplied with the packed texel values. Thus four interpolated values are computed simultaneously. Finally, these four contributions are accumulated to the aligned image memory.

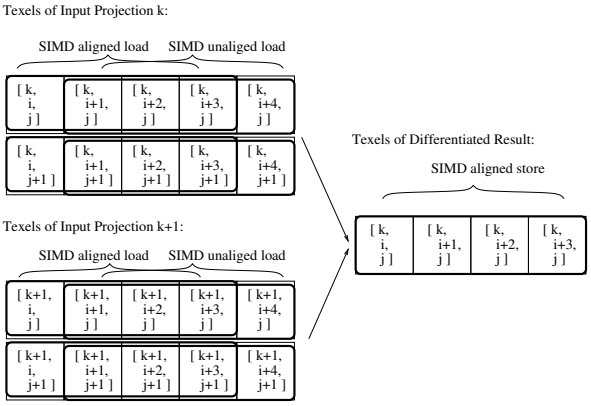


Figure 11: Using SIMD to perform differentiation of 4 texels

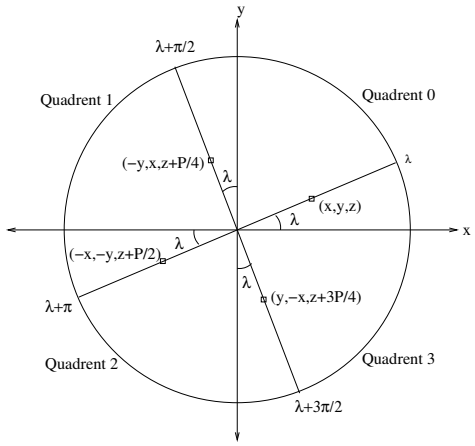


Figure 12: Image coordinates that produce identical backprojection coordinates for each $\frac{\pi}{2}$ rotation

4.2.2 SIMD Differentiation

We also used SIMD for differentiation. Instead of operating on a single texel at a time, four consecutive texels are loaded and subtracted together. Figure 11 shows how the projection memory is accessed when performing SIMD differentiation. Since the difference between adjacent texels needs to be computed, an unaligned load must be performed when reading the location of the neighbors on the right.

4.2.3 SIMD Remapping

In addition, Intel IPP provides a *remapping* function which uses SIMD to remap one 2-D array based on a 2-D coordinate remapping array. Although it requires extra memory to hold the 2-D coordinate remapping array, using the Intel IPP *remapping* function for remapping to filtering coordinates and back to projection coordinates does produce a performance improvement.

4.2.4 SIMD Symmetry Method

As described in [3], the backprojection of each $\frac{\pi}{2}$ source projection contain redundant computation due to symmetry of the trigonometric functions:

$$sin(\lambda + \frac{\pi}{2}) = cos(\lambda)$$

$$cos(\lambda + \frac{\pi}{2}) = -sin(\lambda)$$

The image coordinates shown in Figure 12 produce identical backprojection coordinates:

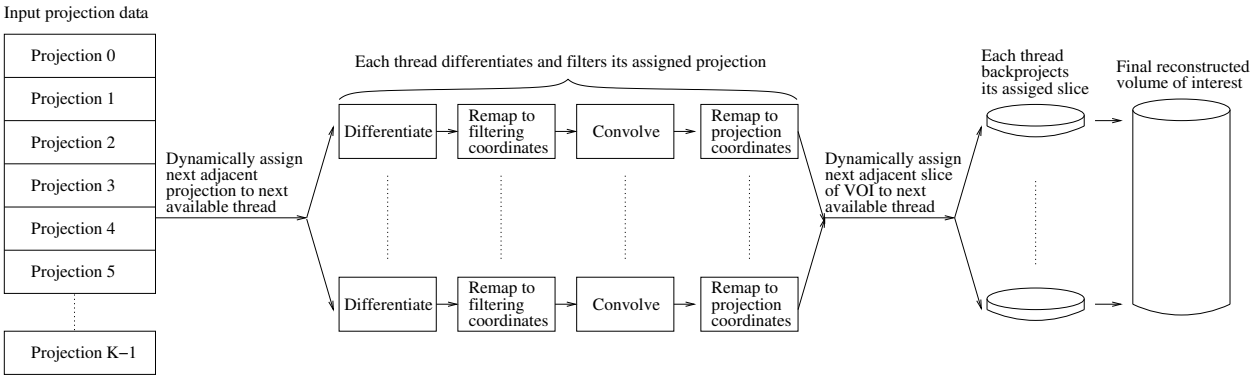


Figure 9: Parallelization strategy for π-interval method

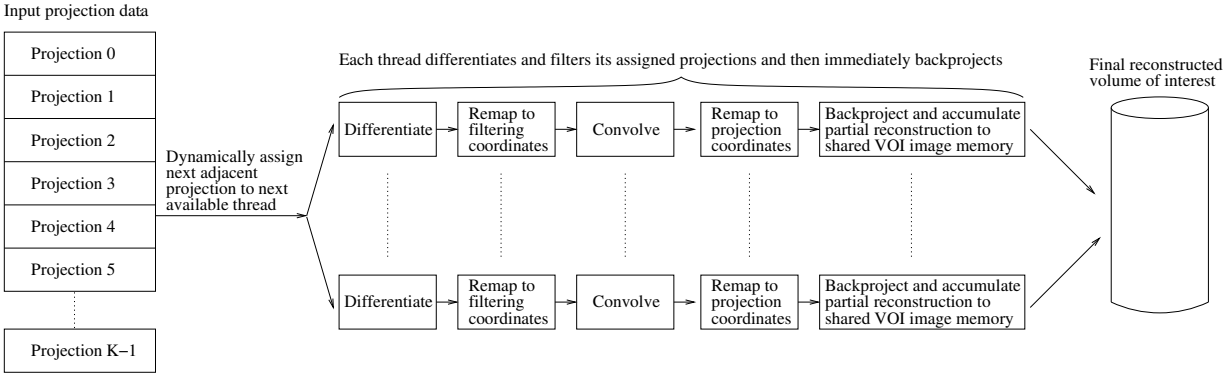


Figure 10: Parallelization strategy for cone beam cover method

$$\omega(\lambda, x, y, z) = \omega(\lambda + \frac{\pi}{2}, -y, x, z + P/4)$$

$$u(\lambda, x, y) = u(\lambda + \frac{\pi}{2}, -y, x)$$

Therefore, instead of iterating over all λ values during backprojection of a z-strip, it is only necessary to iterate over the range of angles comprising a quarter turn of the helix: $[0..\frac{\pi}{2}]$. During each iteration, since the backprojection coordinates of all $\frac{\pi}{2}$ rotations of the coordinates and projections are equivalent, the coordinates are calculated once and reused by all $\frac{\pi}{2}$ projections. The bold voxels in Figure 13 all have identical backprojection coordinates when backprojected to their corresponding source projections. Once the (u, ω) coordinate has been calculated, backprojection for all $\frac{\pi}{2}$ rotations are performed concurrently.

The symmetry described reduces computation of the backprojection coordinates by roughly a factor of $4(nTurns)$. However this method suffers from poor read and write locality because both image data and projection data are accessed over large strides. By repacking the projection and image data, these memory locations are oriented so that they occupy adjacent memory locations. After each projection finishes filtering, it is put into a packed format where columns of each $\frac{\pi}{2}$ projection are interleaved together as illustrated in Figure 14. That way, the values of identical $[i, j]$ coordinates for each $\frac{\pi}{2}$ projection occupy consecutive memory locations. Now backprojection can be performed with SIMD. When accessing projection coordinates during backprojection, four projections are read simultaneously, interpolated simultaneously, weighted simultaneously, and accumulated to image data simultaneously. We refer to this method as the SIMD symmetry method. Since the memory accesses are more sequential and deterministic, the hardware prefetcher can accurately predict the next cache line to bring in.

However, the image data will be written in a packed format shown in the right half of Figure 15. Adjacent locations in packed image memory have $[x, y]$ coordinates that are rotated by $\frac{\pi}{2}$ and z values separated by $\frac{P}{4}$. The packed image data eventually needs to be unpacked into an $[x, y, z]$ format.

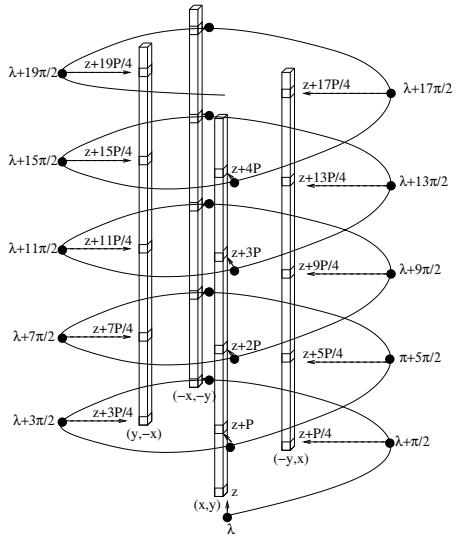


Figure 13: Due to symmetry of each $\frac{\pi}{2}$ rotation, the bolded voxels reuse the same backprojection coordinates when backprojected to their corresponding projection sources

The image unpacking is costly for its poor locality. Instead of waiting for the whole image to be backprojected, only one z-strip and its four rotations are backprojected at a time. Each thread has its own private array to accumulate the partial reconstructions of the

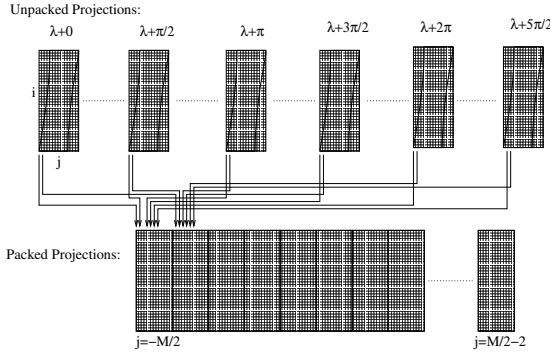


Figure 14: The columns from each $\frac{\pi}{2}$ projection are interleaved so that data from four projections separated by $\frac{\pi}{2}$ can be read simultaneously with a SIMD load

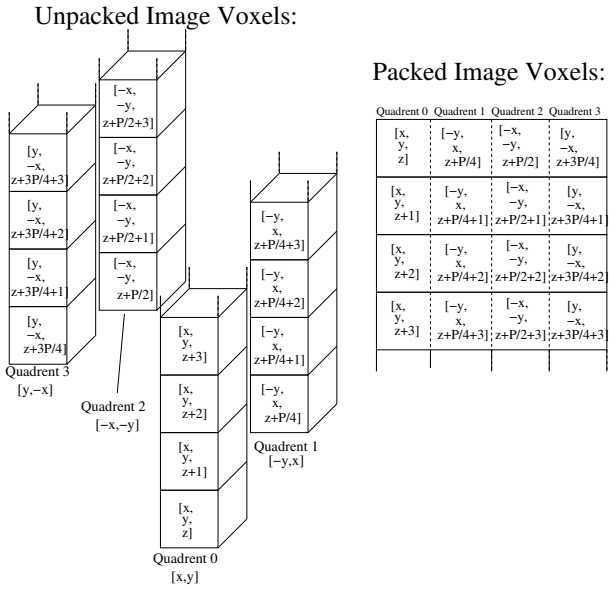


Figure 15: Unpacked image data (left), Packed image data (right)

four z-strips. This partial reconstruction array is small enough that it can hopefully fit into the private L1 cache. After a thread is finished with each z-strip and its four rotations, they are unpacked into the image $[x, y, z]$ format. When this partial reconstruction region has been fully reconstructed, the streaming SIMD store instruction is used to avoid cache pollution since the completed reconstruction region of memory will not be used in the future.

Note that there is a caveat when accessing this packed projection memory. When the image z coordinate increases by 1, the pointer to image memory increases by $4(nTurns + 1)$. However, if the image z coordinate crosses a z value equal to a multiple of P, it has passed an imaginary boundary corresponding to a turn of the helix. Since crossing a turn of the helix is equivalent to increasing the quarter turn offset by four and decreasing the z offset by P, the pointer should be adjusted by the amount $4 - 4P(nTurns + 1)$ if this situation occurs.

5. EXPERIMENTAL METHODOLOGY

We used a system with two 2.33 GHz Intel Clovertown processors with 4 GB of DRAM running Windows Vista. We used the 64-bit Intel C++ Compiler with options /O3 /QxT /Qopenmp and used *Intel vTune* to identify bottlenecks. The time to load the projection data into main memory and write the reconstructed image to disk is not included. Our helical scanning path consists of four turns plus an additional overscan of a half turn below and above the VOI. All results include the basic optimizations in addition to others specifically discussed.

Step	Init-base	Opt-1T	Opt-2T	Opt-4T	Opt-8T
Derivative	20.6	3.8			
Forward remap	4.2	2.1	25.0	18.9	18.8
Filter	18.9	19.5			
Backward remap	31.8	2.5	(3.0x)	(4.0x)	(4.0x)
SIMD pack	0	12.5			
Backprojection (Speedup)	23722.2 (1.0x)	2083.9 (11.4x)	1062.0 (22.3x)	728.8 (32.5x)	623.5 (38.0x)
Overall time (Speedup)	23798.1 (1.0x)	2126.7 (11.2x)	1087.3 (21.9x)	748.2 (31.8x)	642.3 (37.1x)

Table 1: Breakdown of Execution Time and Speedup (1024^3)

6. RESULT ANALYSIS

Figure 16 provides a comparison of the image quality of the final optimized version against the Shepp-Logan phantom at a single horizontal slice. The amplified error image shows the streaking artifacts that result from using a finite number of projections.

Figure 17 shows the reconstruction times of the π-interval method with basic optimizations, the cone beam cover method with SIMD optimizations, and the final SIMD symmetry method. As shown in the figure, the speedup for one single thread with the SIMD symmetry method over the π-interval method and the cone beam cover method are significant for all image sizes. Also shown is the speedup from parallelizing on multiple cores. In general, we achieved significant speedup for the π-interval method; however, speedup for the cone beam cover method and the SIMD symmetry method levels off after about four threads. The π-interval method easily achieves its speedup, because it is not memory-bound. However, for the SIMD symmetry method, performance counter measurements indicate that the front-side bus (FSB) has reached almost 100% utilization in the time-critical inner loop even with just one thread.[1] Since most of the backprojection computation is optimized out, there is little computation remaining to hide the memory access latency. Thus the application has become bandwidth-limited.

To further gain insight from the speedup number, we show the breakdown of the execution time using the SIMD symmetry method compared to the π-interval method in Table 1. Obviously, the majority of speedup is obtained from optimization and parallelization of the backprojection stage. For the 8-thread SIMD symmetry reconstruction, we are able to obtain 37.1x speedup over the single-thread π-interval reconstruction.

Since the SIMD symmetry method demonstrates the best performance in all cases, we study its scalability in Figure 18. The speedup levels off after about four threads for all image sizes for two reasons. First, two threads now share the same L2 cache space, thereby reducing the amount of space each thread has. Secondly, as mentioned earlier, the front-side bus bandwidth eventually becomes the bottleneck, limiting any additional speedup.

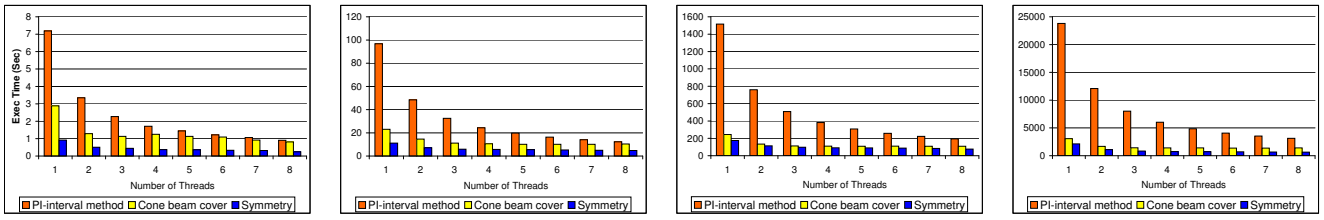
7. CONCLUSION

In this paper, we discussed and demonstrated several optimization and parallelization schemes for Katsevich image reconstruction algorithm used in computed tomography. On our system with two-quad-core Intel processors (eight cores in total), we were able to achieve 37.1x speedup with our final optimized version running on eight threads over the baseline version running on a single thread. In addition, we identified the front-side bus bandwidth to be a major scalability roadblock for this medical imaging application. Without increasing the front-side bus bandwidth, parallelizing this application with even more cores will render little performance benefit.

[1]The peak FSB bandwidth is 10.5GB/sec.

(a) Original (b) Reconstruction (c) Amplified Error

Figure 16: Image Quality of one Horizontal Slice of Shepp-Logan Phantom



(a) 128^3 from 640 128x32 (b) 256^3 from 1280 256x64 (c) 512^3 from 2560 512x128 (d) 1024^3 from 5120 512x128

Figure 17: Reconstruction Time of Different Algorithms for Different Image Sizes

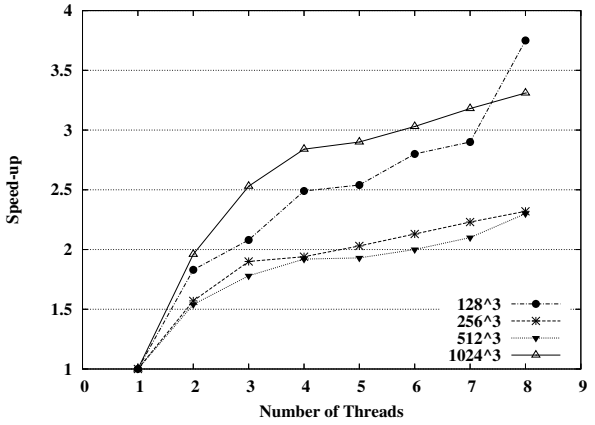


Figure 18: Scalability of the SIMD Symmetry Method

8. ACKNOWLEDGMENTS

This research is supported by Intel's Medical and HealthCare Infrastructure Processor Division and an NSF CAREER Award (CNS-0644096). We would like to thank David Hillyard, Edwin Verplanke, and Dale Easter of Intel Corp. for their support and comments. We would also like to thank Ada Gavrilovska and Karsten Schwan for their encouragement of this work.

9. REFERENCES

[1] J. Deng, H. Yu, J. Ni, T. He, S. Zhao, L. Wang, and G. Wang. A Parallel Implementation of the Katsevich Algorithm for 3-D CT Image Reconstruction. *The Journal of Supercomputing*, 38(1):35–47, 2006.

[2] L. Feldkamp, L. Davis, and J. Kress. Practical cone-beam algorithm. *J. Opt. Soc. Am*, 1(6):612–619, 1984.

[3] I. Hong, S. Chung, H. Kim, Y. Kim, Y. Son, and Z. Cho. Fast forward projection and backward projection algorithm using simd. In *Conference Record of the 2006 IEEE Nuclear Science Symposium*, pages 3361–3368, Oct. 2006.

[4] I. Hong, S. Chung, H. Kim, Y. Kim, Y. Son, and Z. Cho. Ultra Fast Symmetry and SIMD-Based Projection-Backprojection (SSP) Algorithm for 3-D PET Image Reconstruction. *IEEE Transactions on Medical Imaging*, 26(6):789–803, 2007.

[5] M. Kachelrieß, M. Knaup, and O. Bockenbach. Hyperfast parallel-beam and cone-beam backprojection using the cell general purpose hardware. *Medical Physics*, 34:1474, 2007.

[6] A. Katsevich. Theoretically exact filtered backprojection-type inversion algorithm for spiral ct. *SIAM Journal on Applied Mathematics*, 62(6):2012–2026, 2002.

[7] F. Noo, J. Pack, and D. Heuscher. Exact helical reconstruction using native cone-beam geometries. *Physics in Medicine and Biology*, 48(23):3787–3818, 2003.

[8] K. C. Tam, S. Samarasekera, and F. Sauer. Exact cone beam CT with a spiral scan. *Physics in Medicine and Biology*, 43:1015–1024, Apr. 1998.

[9] A. Wunderlich. The Katsevich Inversion Formula for Cone-Beam Computed Tomography. Master's thesis, Department of Mathematics, Oregon State University, Sept. 2006.

[10] J. Yang. Cone beam cover method: An approach to performing backprojection in Katsevich's exact algorithm for spiral cone beam CT. *Journal of X-Ray Science and Technology*, 12(4):199–214, 2004.

[11] J. Yang, X. Guo, Q. Kong, T. Zhou, and M. Jiang. Parallel Implementation of Katsevich's FBP Algorithm. *International Journal of Biomedical Imaging*, 2006.

[12] H. Yu. Studies on implementation of the Katsevich algorithm for spiral cone-beam CT. *Journal of X-Ray Science and Technology*, 12(2):97–116, 2004.

[13] K. Zeng, E. Bai, and G. Wang. A Fast CT Reconstruction Scheme for a General multi-core PC. *International Journal of Biomedical Imaging*, 2007.