Improving the performance of medical CT image reconstruction on multicore processor

Shiva Chaitanya V Chandrachary, Bharat joshi
Department of Electrical and Computer Engineering
University of North Carolina at Charlotte
svishwak@uncc.edu, bsjoshi@uncc.edu

***Abstract*—Image reconstruction is observed to be one of the most common problem because of it's large data movement and non-trivial data dependencies. In the past, these problems were tackled by many high performance hardware such as FPGAs and GPGPUs. This also reflects the investemts to be made in these supercomputers for real time reconstruction of clinically computed tomography (CT) applications. Medical imaging systems are employing high performance computing (HPC) technology to meet their time constraints. This paper presents different optimizations to the volume reconstruction and implement it on a commodity hardware such as x86 based multicore system. This paper chooses to perform its implementaion on Intel Xeon X5365 multicore processor. We perform different levels of parallelization and analyse each of them and report their results with respect to serial implementation. The objective of this paper is to understand the constraints of volume reconstruction in multicore architecture and optimize them while preserving the quality of the reconstructed image.**

Figure 1: C-arm system.[8]

I. Introduction and related work

Computed tomography (CT) is an Imaging procedure that is used to generate a three-dimensional image of the inside of an object, allowing the user to see what is inside it without cutting it open. This is a technology that uses a large series of two-dimensional radiographic images taken around a single axis of rotation to produce tomographic images of specific areas of the scanned object. The usage of CT has gone beyond its classical application in clinical environments and expanded its horizons in industrial CT's such as nondestructive materials testing and imaging of archeological contents like sarcophagi. The typical clinical workflow requires high-speed reconstruction in order to avoid interruption to patient's treatment. From the physician's perspective, it is expected that the computation of reconstructed volume from a large set of acquired two-dimensional X-ray projections terminates roughly at the end of the scanning period ensuring no additional delay in the procedure. Volume data set acquired after post processing is required to be available for analysis by the physician immediately after the scan. This sets a constraint for maximum reconstruction time to allow for real-time processing of the entire algorithm.

The volume reconstruction scheme is a key component of modern CT systems and is compute-intensive. The typical approach to meet real time constraints is to utilize special-hardware such as field programmable gate arrays (FPGAs) [4]. Integrating these types of non-standard hardware into commercial CT systems adds considerable costs in terms of both hardware and software development. Also adding to the system's complexity.

With recent progress in very-large-scale integration (VLSI) design which are driven by Moore's law gives the potential to meet requested CT time constraints. The development in microprocessor consists of many independent compute cores that has the capabilities to execute multiple tasks in parallel. These processors are commonly referred to as multi-core or many-core CPUs.

The volume reconstruction step for recent C-arm systems with flat panel detector can be considered as a prototype for modern clinical CT systems. C-arm CT's, as the one shown in Figure 1, perform the rotational acquisition of 96 high resolution (1248×960 pixels) images. In practice, filtered backprojection (FBP) methods such as Feldkamp algorithm are widely used for performance reasons. The algorithm consists of 2D pre-processing steps, backprojection, and 3D post-processing. The volume reconstruction is performed in backprojection step, making it by far the most time consuming part of the process. The results obtained using multi-core CPU-based implementation in this report still needs several minutes for the reconstruction of volumes with high spatial resolution of 512^3 or more voxels.

This algorithm is characterized by high computational intensity, non-trivial data dependencies, and complex numerical evaluations but also offers embarrassingly parallel structure. Hence this algorithm is particularly suited for GPUs which are designed to handle parallel structures. In the recent past, optimization of Feldkamp algorithm has focused on GPUs (Muller and Yagel 1998) and was reported that a large performance gains were obtained when compared to CPUs (Muller et al. 2007). Studies indicate that large servers are required to meet the performance of GPUs (Hofmann et al. 2011). In this report, we use the *RabbitCT* environment which defines clinically relevant test case.

RabbitCT is an open competition benchmark for worldwide comparison in backprojection performance and ranking on different architectures using one specific, clinical, high resolution C-arm CT dataset of a rabbit. It also allows implementation alternatives for reconstruction scenarios by applying them to a fixed, well-defined problem. The high computational demand of backprojection algorithm with its embarrassingly parallel structure makes it interesting candidates for its interface with high-performance computing in medical applications.

This paper is organized as follows. In section 2 we introduce Intel Xeon X5365 processor. We also look at bandwith contraints of this processor for this application. Section 3 gives a theoretical background of the backprojection algorithm. In Section 4, our approach to parallelize the algorithm is introduced. Two levels of parallelization is considered in this section. In section 5, the results of tests and validation of its analysis is presented. Finally, we conclude the paper in section 6 by showing the importance of volume reconstruction on a commodity processor.

II. Experimental architecture

Intel x86-based multicore processor, server variant, Xeon X5365 has been chosen to test the performance potential of our parallelization approach. This architecture consists of a dual socket motherboard, thus in essential, we have eight processing cores with each of them running on a single thread (no Hyper-Threading). A quad-core processor with 65 nm feature size, is the successor of the Woodcrest processor and belongs to core microarchitecture. This chip feature a large outer level cache of 8 MB (at level 2), which is shared by two cores with 4 MB per core. This model consists of dual channel memory architecture operating at a bandwidth of 667 MHz per channel between Random Access Memory (RAM) and the memory controller with each channel transferring 64 bits of data per cycle. This makes the processor to have an effective Front Side Bus (FSB) bandwidth of 1333 MHz. Thus, this system has a DDR-2-667 MHz RAM memory with a maximum theoretical transfer rate of 10.67 GB/second. A detailed description of architecture is shown in figure 2. This figure illustrates only one socket of the system but in reality, the system consists of two of these identical memory subsystem.

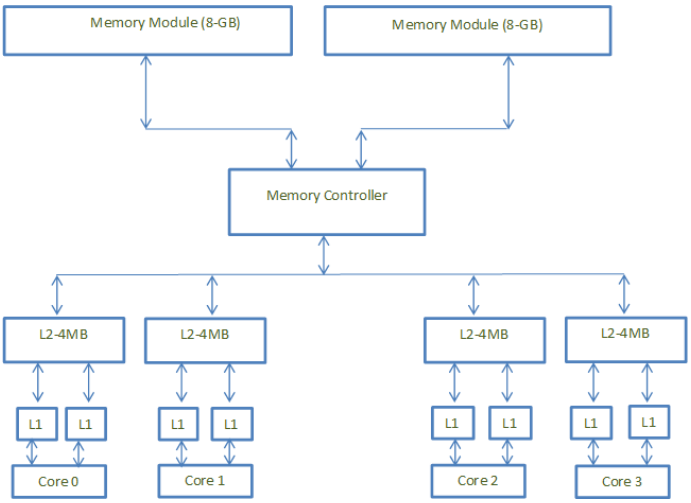

Figure 2: Intel Xeon X5365.

A comprehensive summary of the most important processor features is presented in table 1. Table 1 also contains the bandwidth measurements for a simple read and write benchmark[9] :

A. Cache Read:

This bench mark is designed to provide us with read bandwidth for varying vector lengths.

```
timer start
for iteration count
       for I=0 to vector length
             register += memory[I]

timer stop
```

For the cases where vector length is less than the cache size, the data will be received completely from cache and resulting bandwidth was much higher. This is best illustrated in plot shown in figure 3. Since the architecture under consideration has a last level cache (L2) of 8 MB, the bandwidth reduces drastically after it reaches a point where the data is no more available in the cache to be fetched and request has to be sent to the main memory.

B. Cache Write:

This benchmark is designed to provide us with write bandwidth for varying vector lengths. This benchmark is greatly affected by architectural peculiarities in the memory subsystem. Replacement policy and associativity play important factors in the performance of this benchmark.

```
timer start
```

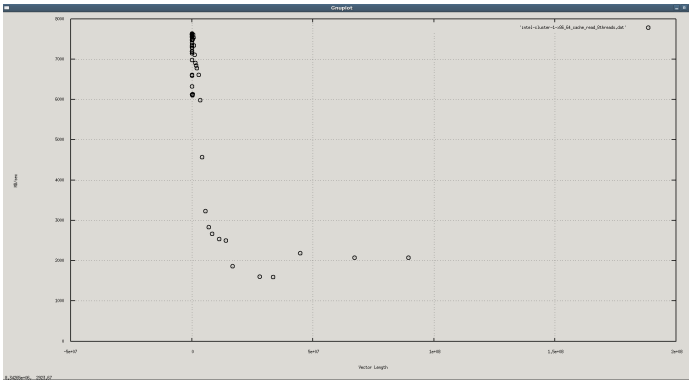

Figure 3: GNU Plot for cache read

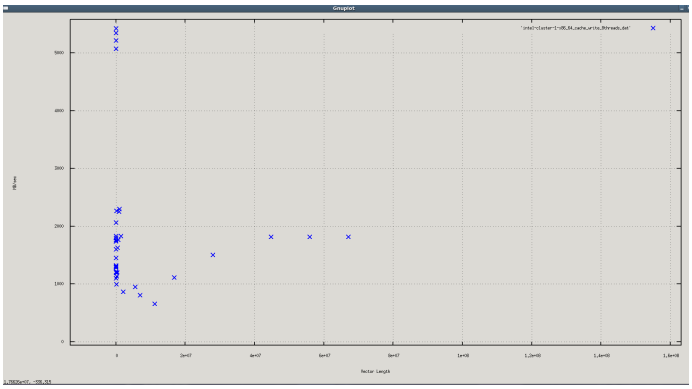

Figure 4: GNU Plot for cache write

```
for iteration count
        for I = 0 to vector length
           memory[I] = register++
timer stop
```

This benchmark is illustrated in a plot shown in figure 4.

This benchmark reflects the performance of memory subsystem of the system. Such a benchmark is helpful in evaluation of the data streaming properties of the problem under consideration. We use GNU GCC compiler version 4.1.2. Since this architecture doesn't support hyperthreading, one thread per core (total of 8 threads) is utilized. The compiled code was profiled on Intel VTune Amplifier XE, version 2013

III. The algorithm

A. Reconstruction task

The dataset required for this experiment was downloaded from RabbitCT website [7]. This dataset contained N=496 high resolution images. The size of the image is $S_x = 1248$ pixels in width and $S_y = 960$ pixels in height. This is a pre-processed and filtered dataset acquired by a C-arm system [1]. Hence, only backprojection step is considered

Microarchitecture	Intel Clovertown
Model	Xeon X5365
Clock (GHz)	3
Node socket	2
Number of cores	8
Number of threads	8
L1 D-cache	8 x 32k
L1 I-cache	8 x 32k
L2 cache	4 x 4 M
FSB	1333 MHz
Pipeline	15 Stage
Bandwidths [GB/s]:	
Read (1 thread)	6.2
Write (1 thread)	5.9

Table I: Microarchitecture details

in this paper. Each projection image is associated to a precalibrated projection matrix $A_n \in R^{3x4}$ that encodes a projection image. The reconstruction was performed on an isocentric cubic volume of $256^3 mm^3$. The side lengths of the cubic reconstruction given by RabbitCT are $L \in \{128, 256, 512, 1024\}$ voxels respectively at an isotropic voxel size of $R_L = (256/L)mm$.

The reconstructed volume is denoted by f(x,y,z) where $x, y, z \in [-128, 128]$ and the origin of the world coordinate system (in mm) is considered to be located at the isocenter of the c-arm system. In the discrete form, the volume is denoted by $f_L(i, j, k)$, where $i, j, k \in [0, \ldots, L-1]$. This is related to world coordinate system as:

$f_L(i, j, k) = f(O_L + iR_L, O_L + jR_L, O_L + kR_L)$
with $O_L = -\frac{1}{2}R_L(L-1)$.

In this paper, the Implementation, testing and performance analysis of reconstruction was done with a resolution of $L = 256$ voxels in each direction.

B. Reconstruction algorithm

The FDK algorithm utilized consists of acquisitions that exceeds total orbital extent of 180° + fan anlge. The redundant rays in the projection data are weighted with Parker weighting [10] to yield an accurate reconstruction of short-scan C-arm projection data. Pre-processing of the projection data like cosine weighting, physical correction, and ramp filtering is requred. All these correction steps have already been applied to the available projection image I_n in the dataset provided

by RabbitCT. The discrete form of the FDK algorithm is thus given as[2]:

$$f(x,y,z) = \sum_{n=1}^{N} \frac{1}{W_n(x,y,z)^2} . \hat{p_n}(u_n(x,y,z), v_n(x,y,z)),$$

where

$$w_n(x,y,z) = a_2x + a_5y + a_8z + a_{11},$$

$$u_n(x,y,z) = (a_0x + a_3y + a_6z + a_9).w_n(x,y,z)^{-1} \ ,$$

$$v_n(x,y,z) = (a_1x + a_4y + a_7z + a_{10}).w_n(x,y,z)^{-1},$$

and the projection matrix

$$A_n = \begin{pmatrix} a_0 & a_3 & a_6 & a_9 \\ a_1 & a_4 & a_7 & a_{10} \\ a_2 & a_5 & a_9 & a_{11} \end{pmatrix}$$

The prjection of a voxel will in general not hit exactly one pixel of the 2D CT image. Therefore, the projection value is computed by bilinear interpolation of the four nearby pixels. One volume reconstruction uses 496 CT images of 1248×960 pixels each. The algorithm computes the contribution of each voxel across all projection images and stores the reconstructed volume in array f. Voxel coordinates are denoted by x,y and z while the pixel coordinates are denoted by u and v. Refer figure 5 for the geometric setup. The function $\hat{p_n} : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ peforms bilinear interpolation with zero boundary condition in the projection image I_n [2]. It is given by:

$$\hat{p_n} = (1-\alpha)(1-\beta)p_n(i,j) + \alpha(1-\beta)p_n(i+1,j)$$

$$+(1-\alpha)\beta p_n(i,j+1) + \alpha\beta p_n(i+1,j+1),$$

where $i = |x|, j = |y|, \alpha = x - |x|$, and $\beta = y - |y|$.
The values in the image matrix are accessed by the function

$$p_n : \mathbb{Z} \times \mathbb{Z} \to \mathbb{R},$$

which is given by :

$$f(n) = \begin{cases} I_n & \text{if } i \in \{0, \ldots, S_x - 1\} \wedge j \in \{0, \ldots \ S_y - 1\} \\ 0 & \text{otherwise} \end{cases}$$

One sweep across all the voxels of the volume causes all the pixel values from the projection image to be loaded plus the projection matrix values. The final volume needs to be updated to $f(x,y,z)$ and that causes 8bytes of traffic per voxel and this results in a total of 134 MB (for the problem size of 256^3) or 67 GB for all projections. The cumulative size of all projection images is ≈ 2.4 GB.

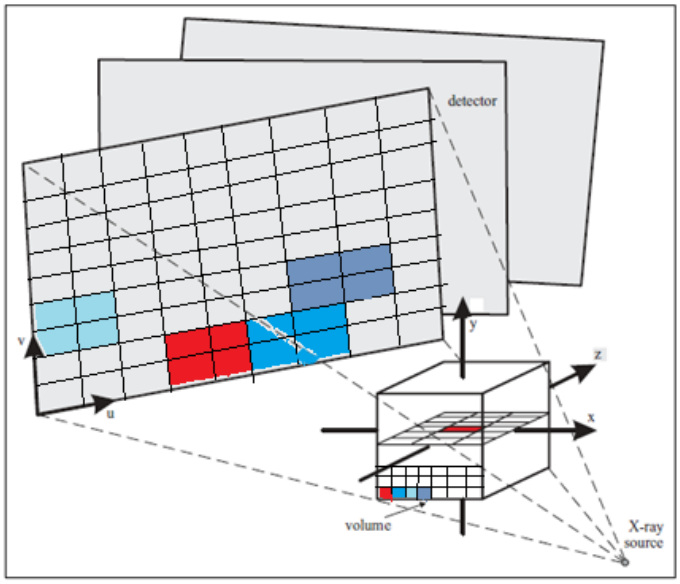

Figure 5: Geometric setup for generating CT projection images. The size of the volume is $256^3 mm^3$. x-y-z plane represents volume while u-v the image plane

IV. Parallel implementaion

The basic backprojection exhibits a streaming access pattern for most of its data traffic. Listing 1 provides a better understanding of the algorithm. The voxel update loop runs for each projection image (496 times in this case). The projection of a voxel will ingeneral not hit one exact pixel, therefore bilinear interpolation of four closest pixels are computed. The final updated volume is stored in f_L.

Listing 1: RabbitCT Algorithm for backprojection of nth projection image

```
input: I_n, A_n, L, O_L, R_L
output: reconstructed volume f_L

for(int k=from; k<to; k++){
 // Calculate the coordinates in the world
 // coordinate system
 double z = O_L + (double)k * R_L;
 for(unsigned int j=0; j<L; j++){
   double y = O_L + (double)j * R_L;
   for(unsigned int i=0; i<L; i++){
    x=O_L + (double)i * R_L;

    //use projection matrix to calculate
    //the index of projection images
    w_n=(A_n[2]*x+A_n[5]*y+A_n[8]*z+A_n[11]);
    u_n=(A_n[0]*x+A_n[3]*y+A_n[6]*z+A_n[9])/w_n;
    v_n=(A_n[1]*x+A_n[4]*y+A_n[7]*z+A_n[10])/w_n;
    //calculate alpha and beta
```

```
    iu=(int)floor(u_n);
    iv=(int)floor(v_n);
    alpha=u_n-iu;
    beta =v_n-iv;

    // load the pixels from nth image
    pixel_bl=I[iv*S_x+iu);
    pixel_br=I[iv*S_x+iu+1);
    pixel_tl=I[(iv+1)*S_x+iu);
    pixel_tr=I[(iv+1)*S_x+iu+1);

    //Do the interpolation
    fx=(1.0-alpha)*(1.0-beta)*pixel_bl+
       (1.0-alpha)*beta*pixel_br+alpha*
       (1.0-beta)*pixel_tl+
                alpha*beta*pixel_tr;

    //Update the final volume
    f_L[k*L*L+j*L+i]+=
               (float)(1.0/(w_n*w_n)*fx);

    }//i
  }//j
}//k
```

The implementation of this algorithm has two levels of parallelization. Hardware and software parallelization.

A. Hardware parallelization:

Imagine a case where there are four voxels that are being processed with only one image for every iteration of the inner loop in listing 1 and for every voxel, the corresponding colored pixels are being accessed for bilinear interpolation (refer Figure 5). Each image consists of 1248 x 960 pixels. With floating point values of each pixel, the total size of each image is 4.8 MB. And 496 such images are considered in this dataset. In this level of parallelization, the whole volume is divided into equal number of slices and they are binded to each available thread (Eight in this case). Each slice is processed by only one pthread. This is done to ensure that there is no data sharing between the threads which helps in avoiding inter-processor communication. Thread binding was done using Pthreads.

As seen from the processor diagram in Figure 2, each core consists of 4 MB of L2 cache. Therefore we can assume that most part of an image can be loaded into L2. For the sake of analysis, it was assumed that $core0$ starts processing voxels and four pixels be requested for every voxel it processed. Initially there will be capacity misses since this will be the first time the images will be loaded into the memory subsystem. According to MESI protocol, when pixels occupy a block in L2, they will be in *execlusive* state. When $core1$ starts processing it's set of voxels for the same projection image, the request for pixels are generated by that core. At this time $core0$ snoops into the bus and offers to share the pixel values to $core1$. The state is then changed from *exclusive* to *shared*. Since this is just a read operation, it is expected to not have any read for ownership (RFO) call and thus all the pixel values are gauranteed to preserve it's *shared* state. The same process is repeated for the rest of all the available cores as each of the cores processes its own slice of the volume. This creates local copies of the projection images in each of its L2 making this a ccNUMA friendly data access. Since the working set considered here is huge, each core gets huge chuck of $256 \times 256 \times 32mm^3$ volume to process

B. Software parallelization:

For every voxel loaded by the processing core, it requestes for four corresponding pixels. The final volume after bilinear interpolation is stored in the volume array f_L. Since this architecture has write back with write allocate cache policy, the last operation is expensive. It creates 8 bytes of traffic for every update operation. For a problem size of 256^3, the total traffic could be 66 GB for all projections. This level of parallelization is aimed at reducing this huge amount of data traffic since we have a bandwidth starved processor. In this approach the outer loop is unrolled by a factor of 2. This approach is conventially called as *blocking*. Listing 2 shows our approach.

Listing 2: Blocking approach to reduce data traffic

```
for(imageNo=0; imageNo<248; imageNo+=2){
 for(k=0;k<256;k++){
  for(j=0;j<256;j++){
   for(i=0;i<256;i++){

    //Perform bilinear interpolation
    ------------------------
    ------------------------

    //Update the final volume for image 1
   f_L[k*256*256+j*256+i]+=
               (float)(1.0/(w_n*w_n)*image_1);

    //Update the final volume for image 2
   f_L[k*256*256+j*256+i]+=
               (float)(1.0/(w_n*w_n)*image_2);

   }//i
  }//j
 }//k
}//imageNo
```

As seen from the listing 2, two images are loaded at a time for every iteration. The update operation is executed twice, once for each projection. At the problem size considered here, nearly half of each projection image can be stored in

the L2. All the voxel data for one line can be stored in the L1 cache and be reused 1 time thus exploring temporal locality of the cache. Hence the complete volume is updated in memoory only 248 times instead of 496 times. This approach reduces the bandwidth requirements and the benefits of this optimization is reported in our performance results.

Another level of software parallelization is performed by unrolling the outer loop of the kernel (k-loop in listing 2). This approach is called unroll and jam which is another level of unrolling done basically to enlarge the block size of the kernel that might give the compiler enough opportunities for rescheduling the instructions and take care of dependencies. Unrolling was done by a factor of 2 since large unrolling factors started showing negative impacts on speedup due to register overflow. The resuls of this optimization is reported in the next section.

V. RESULTS

As it should be straight forward, the results of hardware parallelization is impressive with a speed up of 7.33 with respect to serial implementation (refer Figure 8) with a response time showing around 101 seconds when compared with 740 seconds of serial implementation. This shows that the implementation performed a very good load balancing. The entire volume was divided equally between 8 threads with no data being shared between any of the threads. This was designed in such a way that there was no thread contention.

Software parallelization was done to reduce the pressure on the memory interface. As seen in Figure 6, the bus traffic reduced by almost 40 percent through this approach. This is attributed to 50 percent reduction in the number of volume update operations which in-turn reduces the number of memory write operations. Unroll and jam is another level of software parallelization which was aimed at giving the compiler enough opportunities to take care of the dependent instructions and reschedule them. As shown in Figure 7, the number of pipeline stalls went down by around 25 percent through these two approaches.

A. Validation of analysis

To validate the performace of the implemented backprojection algorithm, $RabbitCT$ has provided a reference reconstruction f_L^{ref} on their website for each problem size. The validation is performed here in terms of speed and quality. The speed of the reconstruction, t_{avg} is the measure of efficiency of backprojection algorithm by computing the average time to process all the voxels for one projection. This time includes data access time, interpolation time and computation time. This is a most important metric and was already noted above to be around 88 seconds for our approach. This timing result shows how long a physician has to wait until the result of the reconstruction is processed and final image apprears on the computer screen.

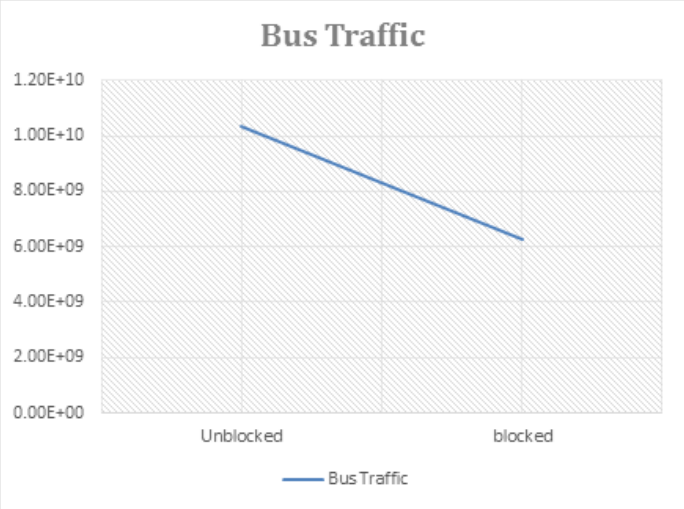

Figure 6: Plot showing reduction in the bus trafic due to software parallelization

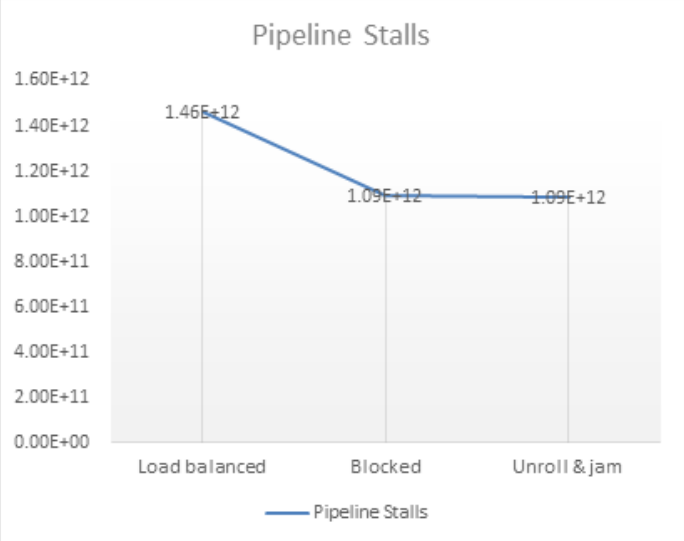

Figure 7: Plot showing reduction in pipeline stalls for different approaches

The quality of the reconstruction is estimated by comparing the results obtaied by our approach and the reference implementation by $RabbitCT$. This reference is used to estimate the numerical accuracy of our reconstructed volume f_L. The mean squared error $q_{mse}(f\_L)$ of reconstruction in Hounsfield units (HU^2) in comparision with reference reconstruction is given by:

$$q_{mse}(f\_L) = \frac{1}{L^3} \sum_{i,j,k} [f\_L(i,j,k) - f^{ref}\_L(i,j,k)]^2$$

The ouput of the reconstruction algorithm is scaled to be in the 12-bit range of $0, \cdots, 4095$ hounsfield units. The peak signal to noise ratio, q_{psnr} is measured in decibels (dB) :

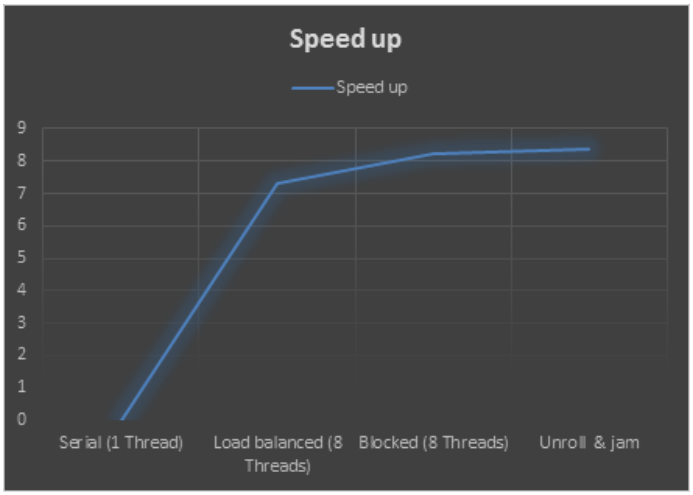

Figure 8: Speed-ups in comparision with the serial implementation

$$q_{psnr}(f\_L) = 10 \log_{10} \frac{4095^2}{q_{mse}(f\_L)}.$$

Our approach resulted in zero errors and a peak signal to noise ratio (q_{psnr}) of infinity.

VI. Conclusions and Outlook

Two major levels of optimization techniques were implemented to the reconstruction algorithm. It was observed that backprojection was the most time consuming part of the algorithm and several optimization techniques involved were focused on reducing the time taken for the entire reconstruction while preserving the quality of the reconstructed image. It was reported that hardware parallelization was the most effective since it produced perfect load balancing between 8 threads. Pthreads were used to bind $core0$ through $core7$ with one Pthread per core. A great speed up of 7.33 through this approach was achieved. Software parallelization was aimed at reducing the pressure on the memory interface and filling pipeline bubble with useful work. Blocking approach and loop unroll were performed in this level of parallelization and a collective speed up of 8.4 was recorded. The reconstruction was completed with a response time of around 88 seconds after software parallelization was performed.

This algorithm can achieve a good performance on a multicore architecture. At higher resolutions, which are used in industrial applications, multicore systems are frequently a choice when cost is a primary concern. The future work includes a thorough analysis of the data movement pattern and vectorization. Vectorization with AVX2 gather operations might yield a better performance due to their wider SIMD width.

References

[1] Treibig J, Hager G, Hofmann, H G,Hornegger J, Wellein G (2012) Pushing the limits for medical image reconstruction onrecent standard multicore processors,URL http://hpc.sagepub.com/content/27/2/162.refs.html

[2] Rohkohl C, keck B, Hofmann H G and Hornegger J (2009), Technical Note: rabbitCT-an open platform for benchmarking 3D cone-beam reconstruction algorithms.

[3] Scherla H, Kowarschikb M, Hofmannc H G, Keckc B, Horneggerc J, Evaluation of State-of-the-Art Hardware Architectures for Fast Cone-Beam CT Reconstruction.

[4] heigl B and Kowarschik M (2007) High-speed reconstruction for C-arm computed tomography. In 9th International Meeting on Fully Three-dimensional Image reconstruction in Radiology and Nuclear Medicine, Linda, pp. 25-28, URL http://www.fully3d.org.

[5] Mueller K and Yagel R (1998) Rappid 3D cone-beam reconstruction with the Algebraic Reconstruction Technique (ART) by utilizing texture mapping graphics hardware. nuclear science symposium, 1998. Conference record, vol. 3, pp. 1552-1559.

[6] Mueller K, Xu F, and Neophytou N (2007), Why do commodity graphics hardware boards (GPUs) work so well for acceleration of computed tomography? In SPIE Electronic Imaging Conference, vol. 6498, 64980N.1-64980N.12

[7] URL www.rabbitct.com

[8] https://software.intel.com/sites/default/files/m/d/4/1/d/8/c-arm.jpg

[9] Mucci, Philip J, Kevin London, and John Thurman (1998). "The CacheBench Report." : 19. Print.

[10] J. R. Parker (1996), "Algorithms for image processing and computer vision." (Wiley, New York).