Parallelization of medical CT image reconstruction on a multicore processor

Shiva Chaitanya V Chandrachary
Department of Electrical and Computer Engineering
University of North Carolina at Charlotte
Charlotte, North Carolina, 28223-0001 USA
Email: svishwak@uncc.edu

***Abstract*—Image reconstruction is observed to be the computational bottleneck in many clinical computed tomography (CT) applications. In the past, these problems were tackled by many high performance hardware such as FPGAs and GPGPUs. This also reflects the investments to be made in these computers for real time reconstruction of CT images [1]. 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 implementation on Intel Xeon X5365 multicore processor. We perform different levels of parallelization and analyze 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 a multicore architecture and optimize them while preserving the quality of the reconstructed image.**

***Index Terms*—Computed tomography, FDK reconstruction, parallel computing, multicore architecture, Pthreads**

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 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 [1]. Also adding to the system's complexity. With recent progress in very-large-scale integration (VLSI) design which is driven by Moore's law gives the potential to meet requested CT time constraints. The development of multi-core or many-core CPUs which consists of several independed compute cores gives the capability to execute multiple application tasks in parallel.

Figure 1: C-arm system. [8]

The recent C-arm systems with flat panel detector comprises a X-ray generator and a flat panel detector. The C-shaped connecting element can be moved horizontally, vertically and around the swivel axis, so that the X-ray images of the patient can be produced from any angle. This type of system 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 496 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 computational intensity and complex numerical evaluations but also offers 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 [5] and was reported that a large performance gains were obtained when compared to CPUs [6]. Studies indicate that large servers are required to meet the performance of GPUs [11]. In this report, we use $RabbitCT$ 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 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 the bandwidth constraints 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. Standard multicore 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. This quad-core processor with feature size of 65 nm, is the successor of the Woodcrest processor and belongs to core microarchitecture. Every two cores share a 4 MB of L2 cache. Each processor consists of 32 KB of L1 instruction cache and 32 KB of L1 data cache. 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 processors.

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] :

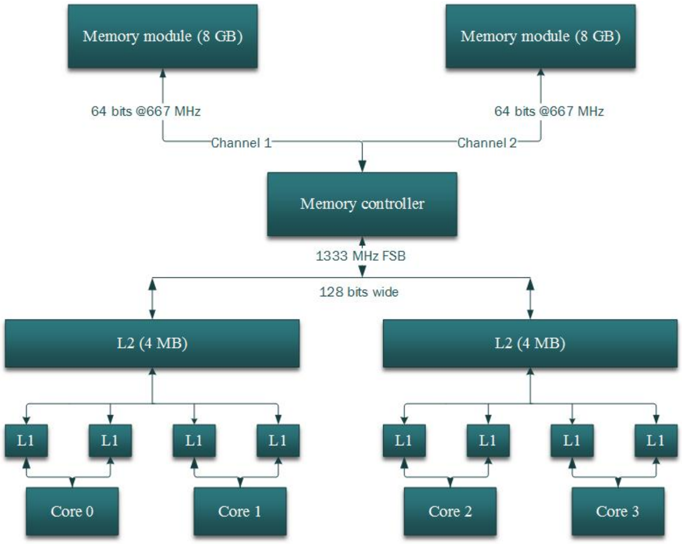

Figure 2: Block diagram representing Intel Xeon X5365.

A. Cache Read:

This benchmark was designed to provide us with read bandwidths for varying vector lengths.

```
timer start

  for I=0 to vector length
    register += memory[I]

timer stop
```

For the cases where vector length was less than the cache size, the data was received completely from the cache and resulting bandwidth was much higher (more than 7 GB/second) for vector lengths less than 8 MB. This is best illustrated in the 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 and request has to be sent to the main memory.

B. Cache Write:

This benchmark is designed to provide us with write bandwidths 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
```

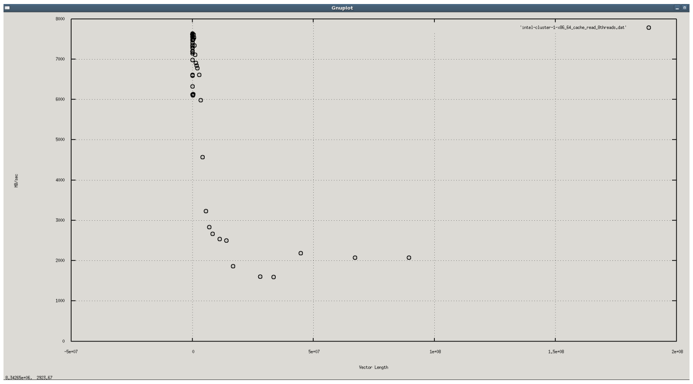

Figure 3: Plot showing cache reads

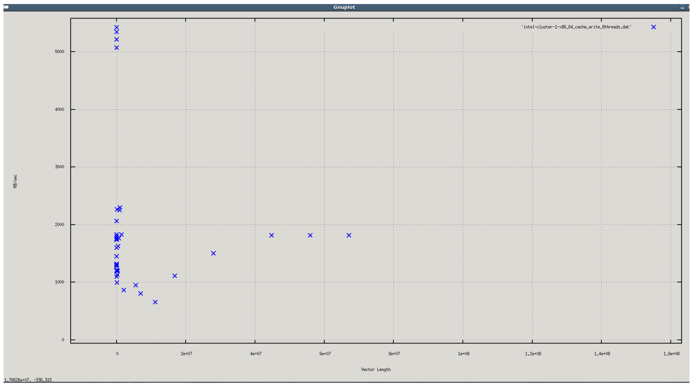

Figure 4: Plot showing cache writes

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

This benchmark is illustrated in a plot shown in Figure 4. For vector lengths less than 8 MB, the memory requests for write operation will be fulfilled by L1 and L2 cache. This was recorded to be more than 7 GB/second as is the case for reads. Since this architecture consists of write-back cache with write allocate policy, every write miss generates a request to its lower memory. The data read is then placed in the cache block and then the write operation is performed. Lesser write bandwidth compared to read bandwidth was recorded in this benchmark. This behavior was attributed towards the capacity of write buffers and the priority given by the processor for read operations.

This benchmark reflects the performance of memory subsystem of the system. We use GNU GCC compiler version 4.1.2. Pthreads were used to bind $core0$ through $core7$ with one Pthread per core. The compiled code was profiled on Intel VTune Amplifier XE, version 2013

Table I: Micro-architecture details

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

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 the C-arm system [1]. Hence, only backprojection step is considered in this paper. Each projection image is associated to a precalibrated projection matrix $A_n \in R^{3\times4}$ 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 $O_L = -\frac{1}{2}R_L(L-1)$.

This paper reports the implementation, testing and performance analysis of the reconstruction process for 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 angle. The redundant rays in the projection data are weighted with Parker weighting [10] to yield an accurate reconstruction of C-arm projection data. Pre-processing of the projection data like cosine weighting, physical correction, and ramp filtering is required. 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

$$A_n = \begin{pmatrix} a_0 & a_3 & a_6 & a_9 \\ a_1 & a_4 & a_7 & a_{10} \\ a_2 & a_5 & a_8 & a_{11} \end{pmatrix}$$

The projection 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 an 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 $i = \lfloor x \rfloor, j = \lfloor y \rfloor, \alpha = x - \lfloor x \rfloor$, and $\beta = y - \lfloor y \rfloor$.
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 :

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

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 8 bytes 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.

IV. Parallel implementaion

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 in general not hit exactly one pixel. Therefore, bilinear interpolation of four nearby pixels are computed. The final updated volume is stored in f_L.

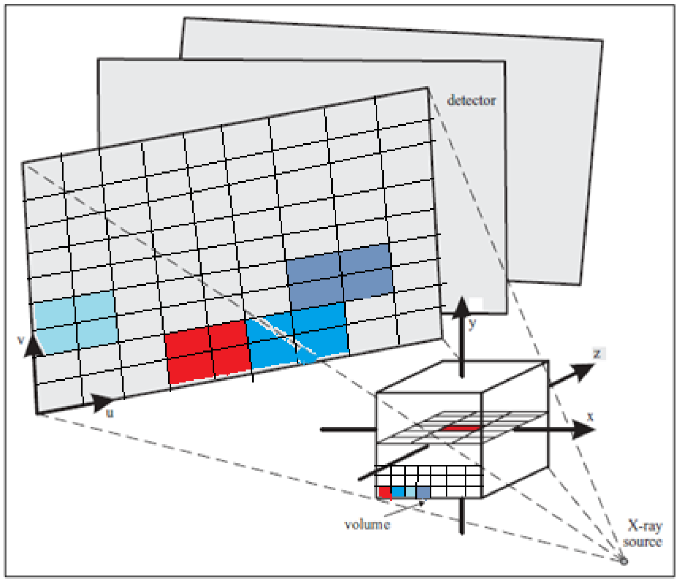

Figure 5: Geometric setup for generating CT projection images. The size of the volume is $256^3 mm^3$. x-y-z plane represents volume while u-v the image plane

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, viz., hardware and software parallelization.

A. *Hardware parallelization:*

Voxels are mapped to its corresponding pixels on the projection image (refer Figure 5) using projection matrix A_n. Bilinear interpolation was done on the four nearby pixels. In this level of parallelization, only one image was processed for every iteration of the inner loop in listing 1. 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 present in this dataset. Here, the entire volume was divided into equal number of chunks (8 in this case since 8 processing cores were available) and each chunk was processed by one core running on a single thread. Pthreads were used to bind $core0$ through $core7$ with one Pthread per core. This was done to ensure that there is no data sharing between the threads which helps in avoiding inter-core communication.

As seen from the processor block diagram in Figure 2, every two cores share 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 were requested for every voxel it processed. Initially there will be capacity misses since that will be the first time images were loaded into cache. According to MESI protocol, when pixels occupy a block in L2 for the first time, they will be in *exclusive* state. When $core2$ starts processing voxels belonging to its chunk of volume, for the same projection image, the request for pixels were generated by $core2$. During that time, $core0$ snoops into the bus and offers to share the data (pixel values) with $core2$. The state is then changed from *exclusive* to *shared*. Since this was just a read operation, it was expected to not have any Read For Ownership (RFO) call and thus all the data values were guaranteed to preserve its *shared* state. The same process was repeated for the rest of all the available cores as each of those cores processes its own chunk of volume. This created local copies of the projection images in each of its L2 making this a ccNUMA friendly data access. Since the working set considered here was huge, each core gets a large chunk of $256 \times 256 \times 32mm^3$ volume to process.

B. *Software parallelization:*

For every voxel loaded by the processing core, it requests for four corresponding pixels from the projection image. The volume was updated in the volume array f_L after bilinear interpolation process. Since this architecture has a write back cache with write allocate 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 memory traffic results to be 66 GB for all projections. This level of parallelization was aimed at reducing this huge amount of data traffic since we have a bandwidth starved processor. In this approach, the outer loop was unrolled by a factor of 2. This approach is conventionally 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 were loaded at a time for every iteration. The volume update operation was done 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 that were loaded in the L1 cache to process the first image was reused to process the second image, thus exploiting temporal locality of the cache. Hence, the complete volume was updated in memory only 248 times instead of 496 times. This approach reduced the bandwidth requirements and the benefits of this optimization was reported in the results section.

Another level of software parallelization was 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. This was basically done to enlarge the block size of the kernel, which gives the compiler more opportunities for rescheduling the instructions and take care of dependencies. Unrolling was done by a factor of 2 since larger unrolling factors showed negative impacts on speedup due to register overflow. The benifits of this optimization was reported in the results section.

V. RESULTS

The hardware parallelization approach resulted in a speed up of 7.33 with respect to serial implementation (refer Figure 8). This gave a response time of 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 (1 thread per core) with no data being shared between any of these threads. This parallelization strategy 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 shown in the Figure 6, the bus traffic reduced by almost 40 percent through this approach. This was attributed to 50 percent reduction in the number of volume update operations (248 times instead of 496). As it was discussed before in section 2, memory write operation is expensive and this level of parallelization was aimed at reducing the number of memory write operations. Since in this approach, two update operations take place together, it is guaranteed that the cache block loaded for the previous operation stays in the L1 cache for the next update operation aswel, thus exploiting temporal locality of cache. This greatly reduces the stalls in the memory Re-Order Buffer (ROB).

Unroll and jam is another level of software parallelization which was aimed at giving the compiler enough opportunities to reschedule the dependent instructions. The plot in Figure 7 shows decrease in the number of pipeline stalls by around 25 percent through these two approaches.

A. Validation of analysis

To validate the performance 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. This time includes data access time, interpolation time and time taken for volume update operation.

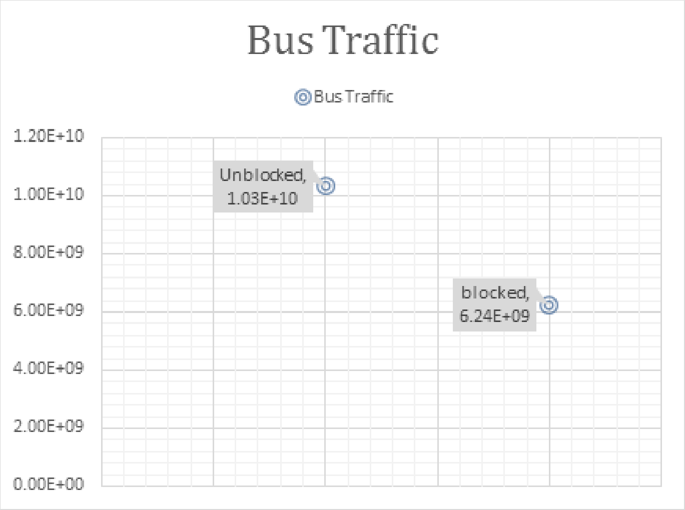

Figure 6: Plot showing reduction in the bus trafic due to software parallelization

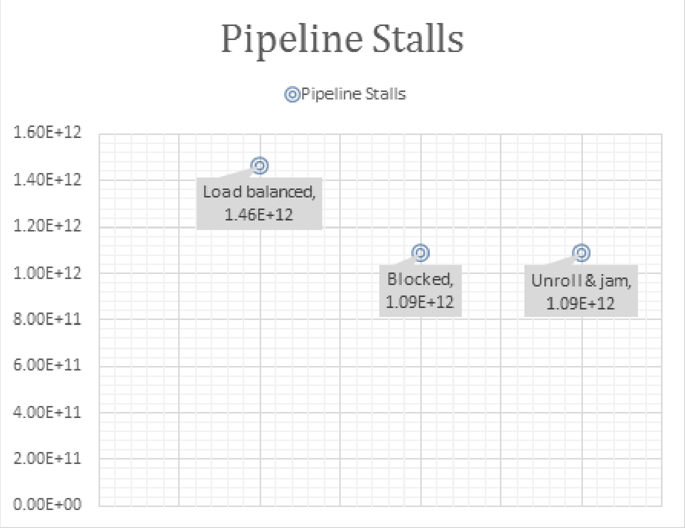

Figure 7: Plot showing reduction in pipeline stalls for different approaches

This is a most important metric and was already noted above to be less than 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 appears on the computer screen.

The quality of the reconstruction is estimated by comparing the results obtained by our approach with 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 comparison with reference reconstruction is given by:

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

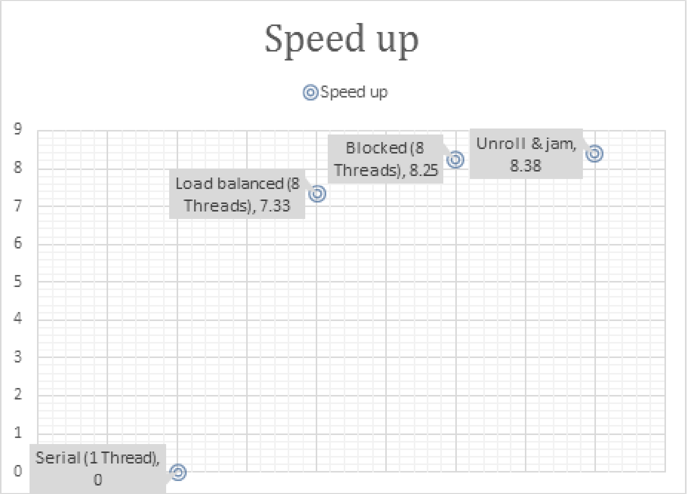

Figure 8: Comparison of speed-ups with respect to serial implementation

The output 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):

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

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

VI. Conclusions and Future Work

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 good 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 less than 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 Advanced Vector Extensions 2 (AVX2) gather operations might yield a better performance due to their wider Single Instructions Multiple Data (SIMD) width.

References

[1] Treibig J, Hager G, Hofmann, H G,Hornegger J, Wellein G (2012) Pushing the limits for medical image reconstruction on recent 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] Scherl H, Kowarschik M, Hofmann H G, Keck B, Hornegger J (2012), 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).

[11] Hoffmann HG, Keck B, Rohkohl C and Hornegger J (2011) Comparing performance of many-core CPUs and GPUs for static and motion compensated reconstruction of C-arm CT data. *Medical physics 38:3940-3944.*