FPGA acceleration by asynchronous parallelization for simultaneous image reconstruction and segmentation based on the Mumford-Shah regularization

Wentai Zhang[a], Guojie Luo[a,b,c], Li Shen[a], Thomas Page[d],
Peng Li[e], Ming Jiang[a], Peter Maass[d], Jason Cong[b,e]

[a]Peking University, China; [b]PKU-UCLA Joint Research Institute in Science and Engineering;
[c]Collaborative Innovation Center of High Performance Computing, NUDT, China;
[d]University of Bremen, Germany; [e]University of California, Los Angeles, USA;

ABSTRACT

X-ray computed tomography (CT) is an important technique for noninvasive clinical diagnosis and nondestructive testing. In many applications a number of image processing steps are needed before the image features are available. One of these processing steps is image segmentation, which generates the edge and the structural features of the regions of interest. The conventional flow is to first reconstruct images and then apply image segmentation methods on reconstructed images. In contrast, an emerging technique obtains the tomographic image and segmentation simultaneously, which is especially useful in the case of limited data. An iterative method for simultaneous reconstruction and segmentation (SRS) with Mumford-Shah model has been proposed, which not only regularizes the ill-posed tomographic reconstruction problem, but also produces the image segmentation at the same time. The Mumford-Shah model is both mathematically and computationally chanllenging. In this paper, we propose an asynchronous ray-parallel algorithm of the SRS method and accelerate it using field-programmable gate array (FPGA) devices, which drastically improves the energy efficiency. Experimental results show that the FPGA implementation achieves a 1.2× speedup with an energy efficiency as great as 58×, over the GPU implementation.

Keywords: image reconstruction; image segmentation; Mumford-Shah regularization; gamma-convergence; asynchronous parallelization; FPGA acceleration; hardware pipelining

1. INTRODUCTION

In medical imaging and nondestructive testing, computed tomography is a technique widely used for the determination of the mass density of an object. In X-ray tomography, the sectional image of the object can be reconstructed from the X-ray absorption when the X-ray beams pass through the object along different angles and offsets.

In many applications, further processing steps will be performed after image reconstruction. For example, image segmentation could be used to visually separate healthy tissues from cancerous ones. The conventional approach is to solve each task separately in sequence: 1) image reconstruction; 2) postprocessing; 3) segmentation. An obvious drawback is that the measured data is only used once at the first step, and the possible errors (from noises in the measured data, inappropriate modeling, or inappropriate parameters) are not easily corrected and will propagate into the later steps. As a consequence, methods that combine the reconstruction and a specific processing task have become popular.[1,2] However, the combined approach is time consuming comparing with the runtime of the conventional approach.

In this work we are interested in the FPGA acceleration of the SRS method, which is an iterative method that alternatively optimizes the reconstructed image and the segmentation result. On one hand this approach regularizes the ill-posedness of X-ray tomography, and on the other hand it computes segmentation directly from the measured data.

Send correspondence to Guojie Luo, E-mail: gluo@pku.edu.cn, Telephone: +86 139 2860 3535

A runtime profiling shows that there are two computational kernels in this approach, Radon projection and Radon backprojection. These two kernels are usually executed hundreds of times and consume over 90% of the total runtime. Assume that the tomography task is to reconstruct an $n \times n$ image from k projections with $O(n)$ samples in each projection, both the projection and the backprojection have a time complexity of $O(kn^2)$ with $O(kn^2)$ memory access without data reuse. These kernels are considered both compute-intensive and memory-intensive.

There have been great efforts on accelerating the kernel of Radon projection and backprojection. The earliest efforts belong to the ASIC implementations,[3,4] which mainly rely on pipelining to improve performance when the hardware resources were too limited to exploit parallelism. DSPs[5,6] are optimized for signal processing applications, whose performance depends on the proper use of fixed-point arithmetic and the optimization for cache hits. Heterogeneous multi-core processors (*e.g.*, CBE[7] and Xeon Phi[8]) are also effective, especially when the memory accesses are optimized for the coprocessor architectures (*e.g.*, the scatter/gather pipeline on Xeon Phi). Most attentions go for GPU acceleration, far before the introduction of CUDA and OpenCL. The first GPU-based backprojection was proposed by Cabral *et al.*[9] Along with the increasing functionality and programmability of massive shader processors inside GPUs, backprojection was implemented using CG[10] and DirectX.[11] Recent results include the implementation using CUDA[12] and OpenCL.[13] Finally, the FPGA acceleration[10,14–17] attracts more and more attention, because of the recent breakthroughs in high-level synthesis as well as its energy efficiency compared to CPU and GPU. The first FPGA-based backprojection was implemented by Coric *et al.*,[14] which has a 4-way parallel projection structure with a 6-stage pipeline; an addition tree at the end accumulates the values for the same pixel, and two on-chip memory banks are used to increase the parallel bandwidth. Recently high-level synthesis is applied for productive design and optimization. Xu *et al.*[15] show that the ray-by-ray parallelism is suitable for high-level synthesis and achieves similar performance with the pixel-by-pixel parallelized VHDL implementation. The Cong group[16,17] apply block-based data reuse for 3D images to reduce the memory access contention and achieve an 85× overall speedup.

Though there have been extensive studies on accelerating projection and backprojection alone on various platforms, few has considered the application-level acceleration. Obviously it is a nontrivial task, especially on FPGAs with limited hardware resources and memory bandwidth.

In this paper, we make the following contributions.

- We propose an iterative algorithm with simultaneous reconstruction and segmentation using the Mumford-Shah model, which can be applied not only to regularize the ill-posed tomographic reconstruction problem, but also to provide image segmentation.
- We design a pipeline-friendly algorithm, which moves the iterations of the original kernels to the 2nd outermost iteration. This way, the two big kernels (projection and backprojection) are replaced by a large number of tiny kernels with pipelining. Though GPUs are usually suitable for such structures, our experimental results show that FPGA pipelining with duplication outperforms GPUs for this application.
- We propose a new ray-trace algorithm to compute the coefficients in projection and backprojection. Different from previous algorithms which minimize the computation, this algorithm is specifically designed to implement in hardware to achieve high throughput (an initiation interval of one clock cycle) with negligible extra computation.
- We parallelize the pipeline with tiny computational kernels and further improve the design with algorithm-level and inter-module optimizations, achieving a 1.2× speedup over the GPU implementation with 58× higher energy efficiency.

The paper is organized as follows. Section 2 introduces the SRS problem, and Section 3 proposes an asynchronous ray-parallel algorithm. Section 4 presents the design optimization techniques for FPGAs. Section 5 presents the experimental results.

2. BACKGROUND AND PROBLEM FORMULATION

The SRS problem is to find an image f and its meaningful segmentation K, given the projection data g (e.g., the X-ray attenuation rate collected at the sensors in CT equipment).

The projection data $g = \mathcal{R}f^\dagger$ is the Radon transformation of the slice of the object being scanned with true mass density $f^\dagger$. The Radon transform is defined as

$$(\mathcal{R}f)(\sigma,\omega) = \int_{\langle x,w\rangle=0} f(\sigma\omega + x)dx,$$

where the pair (σ,ω) defines a unique measurement line. Parameter ω is a unit vector perpendicular to the line, and parameter σ is the distance of the line to the origin. The goal of reconstruction is to solve for an image f such that $\mathcal{R}f = g$.

The segmentation is to find a meaningful decomposition of the image domain

$$\Omega = S_1 \cup S_2 \cup \cdots \cup S_l \cup K,$$

where $S_i \in \Omega$ are disjoint connected open subsets, K is the union of the boundaries of S_i in Ω. A meaningful decomposition has the properties that the image f varies smoothly and/or slowly within each S_i, and it varies discontinuously and/or rapidly across K.

In this work we focus on obtaining the pair (f, K) by minimizing the following Mumford-Shah type function

$$MS(f,K) = ||\mathcal{R}f - g||^2 + \alpha \int_{\Omega\setminus K} |\nabla f|^2 dx + \beta \,\text{length}(K),$$

where $\alpha, \beta \geq 0$, and $\nabla f = (\partial f/\partial x\ ,\ \partial f/\partial y)$ is the gradient of f. The objective functional $MS(f,K)$ has three terms: 1) a least squares term (Frobenius norm), forcing the reconstruction image f to have a projection that matches the measured data g; 2) a L^2-penalty term for the gradient of f, forcing f to be smooth everywhere in Ω except at the edges K; and 3) a penalty of K's length, forcing the edges K to be "short".

In applying the Mumford-Shah regularization, several issues arise. The primary issue comes from the edge set K because it is discrete and its updates are difficult to trace. Thus, we follow the approach by Ambrosio and Tortorelli,[18] where the edges are approximated by a smooth edge indicator function. We minimize the function

$$AT_\varepsilon(f,v) = ||\mathcal{R}f - g||^2 + \alpha \int_\Omega v^2|\nabla f|^2 dx + \beta \int_\Omega \left(\varepsilon|\nabla v|^2 + \frac{(1-v)^2}{4\varepsilon}\right) dx,$$

for a small constant $\varepsilon > 0$. Here, f is still the image and v is a smooth *edge indicator* function, whose values are approximately 0 for the points on the edge set K and 1 on points away from K. For $\varepsilon \to 0$ a minimizer $AT_\varepsilon(f,v)$ is an approximated minimizer of $MS(f,K)$, which is a solution to the SRS problem.

3. ALGORITHM DESCRIPTION

We present two block-coordinate descent methods to solve the SRS problem inn this section. The first method is an image/edge alternating descent method, which updates a block of image or edge variables at a time; and the other one is an asynchronous ray-parallel descent method, which updates a block of variables related to a ray at a time. The latter method performs computations within a small subset of variables, and thus enables an efficient implementation on FPGAs.

3.1 General Idea

Although $AT_\varepsilon(f,v)$ is not jointly convex in (f,v), it is convex in f or v separately. Therefore, we can use the image/edge alternating descent method for the minimization. In the alternating descent method, we compute the gradient of $AT_\varepsilon(f,v)$ with respect to f and v by

$$\nabla_f AT_\varepsilon(f,v) = 2\mathcal{R}^\top(\mathcal{R}f-g) - 2\alpha \operatorname{div}\left(v^2\nabla f\right),$$
$$\nabla_v AT_\varepsilon(f,v) = 2\alpha|\nabla f|^2 v + \frac{\beta}{2\varepsilon}(v-1) - 2\beta\varepsilon\Delta v,$$

where $\operatorname{div} = \partial/\partial x + \partial/\partial y$ and $\Delta = \partial^2/\partial x^2 + \partial^2/\partial y^2$.

It is obvious that the projection $\mathcal{R}$ and the backprojection $\mathcal{R}^\top$ are the computational bottlenecks. Instead of applying $\mathcal{R}$ and $\mathcal{R}^\top$ on the whole image, we decompose the image into overlapping beams according to the paths of line integrals in the $\mathcal{R}$ operator. For example, if a CT scan uses 768 detectors to measure the attenuation rates of parallel X-rays from 180 different angles, the image is decomposed into 768×180 overlapping beams with a width of at most three pixels. The central pixels in the beam are related to a specific ray that is involved in the projection and backprojection, and the two neighboring pixels on both dimensions of the central one are used by the differential operators.

Each beam is associated with a row in the matrix form of $\mathcal{R}$. We use $\mathcal{R}_{(i)}$ to represent the non-zero entries of $\mathcal{R}_i$, and use $f_{(i)}$ to represent the image pixels related to $\mathcal{R}_{(i)}$. We compute the gradient of $AT_\varepsilon(f,v)$ with respect to $f_{(i)}$ and $v_{(i)}$ by

$$\nabla_{f_{(i)}} AT_\varepsilon(f,v) = 2(\mathcal{R}_{(i)}f_{(i)} - g_i)\mathcal{R}_{(i)} - 2\alpha \operatorname{div}\left(v^2\nabla f\right)_{(i)},$$
$$\nabla_{v_{(i)}} AT_\varepsilon(f,v) = 2\alpha|\nabla f_{(i)}|^2 v_{(i)} + \frac{\beta}{2\varepsilon}(v_{(i)}-1) - 2\beta\varepsilon\Delta v_{(i)}.$$

This way, we replace the heavy operations $\mathcal{R}$ and $\mathcal{R}^\top$ in a big descent step by a few light operations $\mathcal{R}_{(i)}$ in multiple small descent steps. This algorithmic transformation enables a more efficient implementation on FPGAs with limited computational and memory resources.

3.2 Image/Edge Alternating Descent

A simplified alternating descent method is demonstrated in Algorithm 1. At the first minimization step in line 4 we keep the edge variable v fixed and minimize $AT_\varepsilon(f,v)$ in f, and then at the second minimization step in line 5 we keep f fixed and minimize $AT_\varepsilon(f,v)$ in v. We repeat this procedure for a number of iterations.

An implementation, similar in computational demand to our simplified version in Algorithm 1, can be found in Page's master thesis.[19]

Most of the computational demand is from $\mathcal{R}$ and $\mathcal{R}^\top$, the discretized form of which can be expressed as

$$g_i = \textstyle\sum_j r_{ij} f_j \quad \text{(the discretized projection } g = \mathcal{R}f\text{), and,}$$
$$f_j = \textstyle\sum_i r_{ij} g_j \quad \text{(the discretized backprojection } f = \mathcal{R}^\top g\text{),}$$

where f_j is the j-th pixel of the linearized image $f_{1\ldots J}$, g_i is the i-th component of the measured attenuation rates $g_{1\ldots I}$, and r_{ij} is the contribution coefficient of the j-th pixel to the attenuation of the i-th measurement line.

Algorithm 1: Alternating Minimization

Input: measured projection data g
Output: image f, edge indicator v

$f^0 = \mathbf{0}$;
$v^0 = \mathbf{1}$;
for $k = 1$ *to #iterations* **do**
  $\mathbf{f}^{k+1}$: move along $-\nabla_f AT_\varepsilon(\mathbf{f}, v^k)$ starting at $\mathbf{f}^k$ for #steps;
  $\mathbf{v}^{k+1}$: move along $-\nabla_v AT_\varepsilon(f^{k+1}, \mathbf{v})$ starting at $\mathbf{v}^k$ for #steps;
end

Algorithm 2: Ray-Parallel Minimization

Input: measured projection data g
Output: image f, edge indicator v

1 $(f^0, v^0) = (\mathbf{0}, \mathbf{1})$;
2 Initialize stepsize λ;
3 **for** $k = 1$ *to #iterations* **do**
4 $(f^k, v^k) = (f^{k-1}, v^{k-1})$;
5 **forall the** *i-th ray related to* $\mathcal{R}_i$ **do**
6 Fetch $(f_{(i)}, v_{(i)})$ from (f^k, v^k);
7 $f_{(i)} = f_{(i)} - \lambda \cdot \nabla_{f_{(i)}} AT_\varepsilon(f_{(i)}, v_{(i)})$;
8 $v_{(i)} = v_{(i)} - \lambda \cdot \nabla_{v_{(i)}} AT_\varepsilon(f_{(i)}, v_{(i)})$;
9 Commit $(f_{(i)}, v_{(i)})$ to (f^k, v^k);
10 **end**
11 Synchronize;
12 Reduce stepsize λ;
13 **end**

3.3 Asynchronous Ray-parallel Descent

Based on the same idea of block-coordinate descent method, we propose a ray-parallel descent method in Algorithm 2. Instead of waiting for the complete computation of projection and backprojection before updating the variables, we shift the updates of partial variables to an earlier step when partial projection and backprojection are available. Specifically, we update the variable related to each ray whenever the partial descent direction related to this ray is available.

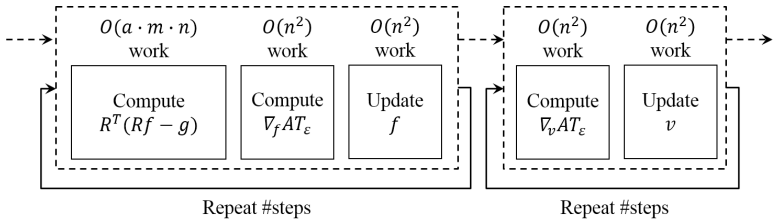
The computational patterns for a single iteration of Algorithm 1 and Algorithm 2 are shown in Figure 1a and Figure 1b, respectively, where a is the number of angles for the X-ray projections, m is the resolution of attenuation rates for a single projection angle, and n is the average length of a measurement line as well as the resolution of the reconstructed image. The ray-parallel minimization reduces the size of computational kernels, by decomposing a big trunk of computation in $O(a \cdot m \cdot n)$ into $O(a \cdot m)$ pieces of tiny computation in $O(n)$ time. In addition, the data locality is increased, such that the partial image and edge variable are updated immediately after fetched for the partial projection and backprojection. Moreover, the computational steps are all in $O(n)$ time, so that these steps are efficient for pipelining. The small computational kernel, the good data locality, and the pipelining-friendly steps are all suitable for FPGA implementations.

Please note that Algorithm 1 and Algorithm 2 are not equivalent. The number of descent steps is also reduced, because each pixel of f and v has been already updated a times after every ray is processed once. The fetch and commit of $(f_{(i)}, v_{(i)})$ in line 6 and 9 of Algorithm 2 are lock-free, and thus the results are nondeterministic when we process p rays in parallel. We can interpret this algorithm as an application of the lock-free parallel stochastic gradient descent approach by Recht *et al.*[20] As pointed out by Bertsekas and Tsitsiklis[21] as well as Liu *et al.*,[22] this method converges under diminishing relaxations and bounded delay between fetch and commit.

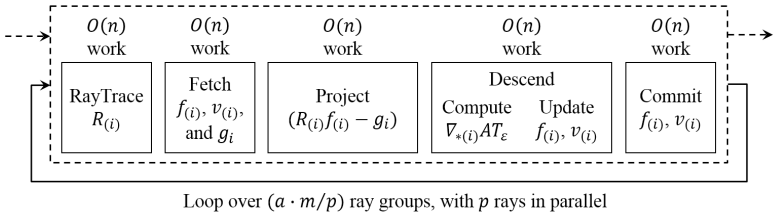
4. FPGA IMPLEMENTATION

FPGA is a reconfigurable integrated circuit. Using commercial off-the-shelf FPGAs has low non-recurring engineering (NRE) cost than manufacturing application-specific integrated circuits. Thus, FPGA is one feasible way for customized computing. It is energy-efficient to obtain desired performance using FPGAs by pipelining, module duplication, data prefetching and reuse, *etc.*

In this Section we present the FPGA implementation of asynchronous ray-parallel algorithm for the SRS problem. First we describe the structure of the data pipeline, and then we elaborate the design of each pipeline stage.



(a) Structure of alternating minimization



(b) Structure of ray-parallel minimization

Figure 1: Structure of the block-coordinate descent methods

For every single ray, if we process one single-precision floating-point pixel per clock cycle, the differential operators read 5×2 pixels from f and v per cycle, and the descent steps write 1×2 pixels back to f and v per cycle. The "bandwidth" requirement is 4.8GB/s per ray for a 100MHz clock frequency. We also present our on-chip data management to enable this data access rate.

The input of the algorithm is the 180×768 pixels of projection data g, and the outputs include a 512×512 image f and a 512×512 edge indicator v.

4.1 Structure of the Data Pipeline

The stages of the data pipeline in Algorithm 2 are balanced, so that it enables an efficient implementation on FPGAs.

The corresponding hardware architecture is shown in Figure 2. The "top" module is responsible for the outer iterations. For example, if it repeatedly issues 180×768 rays 10 times, it initiates 10 outer iterations for the optimization.

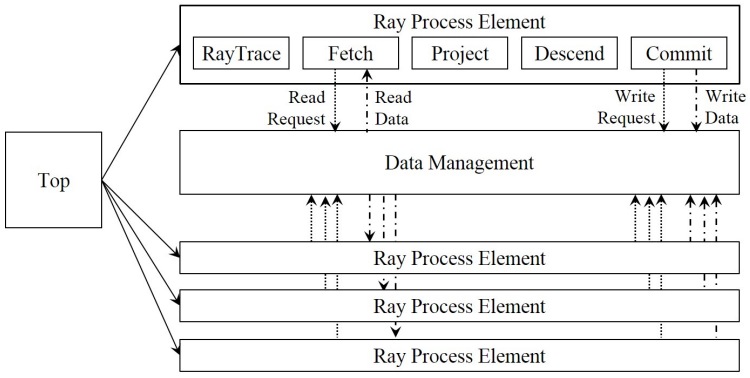


Figure 2: Hardware architecture of the ray-parallel algorithm

Each ray processing element consists of five stages. The ray-trace stage (Section 4.2) generates the coefficients of $\mathcal{R}$ in runtime, because the matrix is too huge to store in memory. The major computation of the projection and descent stages (Section 4.3) is now much more light-weight, where the partial projection becomes a simple

dot product. The fetch and commit stages (Section 4.4) are responsible for exchanging between the centralized data and a local copy, and we will present our on-chip data management strategy in this subsection.

Under proper on-chip data management, our design supports multiple ray processing elements for asynchronous parallelization. Due to the resource limitation, we implement 4× ray processing elements on FPGAs.

4.2 Module of Ray-Trace Stage

The ray-trace is essentially the task to generate the non-zero coefficients $\{r_{i,j}\}$ of the operator $\mathcal{R}$.

The ray-trace algorithm in the classic Bresenham algorithm[23] and SNARK09[24] software package is described in Algorithm 3, where we assume the image occupies the area $[0, n] \times [0, n]$, and the variables are illustrated in Figure 3. The basic idea is to obtain $r_{i,j}$ and/or $r_{i,j+1}$ after the variables y and L (as defined in Figure 3) are incrementally computed.

(a) Case 1: two non-zero intersections (b) Case 2: a single intersection

Figure 3: Illustration of the ray trace algorithm

Algorithm 3: Commonly-used ray trace in SNARK09[24]

Input: the i-th ray intersecting the image at $(0, c_i)$ from angle φ
Output: the intersection lengths $\{r_{i,j}\}$ (and the pixel indices)

$j = 0$;
$y = c_i$;
for $x = 1$ *to* n **do**
  $L = y - \lfloor y \rfloor + \tan\varphi - 1$;
  if $L > 0$ **then**
    $r_{i,j} = \sec\varphi - L\csc\varphi$;
    $r_{i,j+1} = L\csc\varphi$;
    $j = j + 2$;
  end
  else
    $r_{i,j} = \sec\varphi$;
    $j = j + 1$;
  end
  $y = y + \tan\varphi$;
end

Algorithm 4: Pipeline-friendly ray trace with zero padding

Input: the i-th ray intersecting the image at $(0, c_i)$ from angle φ
Output: the intersection lengths $\{r_{i,j}\}$ (and the pixel indices)

$j = 0$;
$y = c_i$;
for $x = 1$ *to* n **do**
  $j = 2(x - 1)$;
  $y = (x - 1)\tan\varphi + c_i$;
  $L = y - \lfloor y \rfloor + \tan\varphi - 1$;
  if $L > 0$ **then**
    $r_{i,j} = \sec\varphi - L\csc\varphi$;
    $r_{i,j+1} = L\csc\varphi$;
  end
  else
    $r_{i,j} = \sec\varphi$;
    $r_{i,j+1} = 0$;
  end
end

This algorithm is efficient for sequential execution that minimizes the total amount of computations, but it causes dependence that are not suited for the pipelining on FPGAs. Specifically, we do not know whether one or two non-zero intersection lengths are generated in each iteration. Such dependence will reduce the throughput of the pipelining execution of the loop. Experiments show that this algorithm can only generate one coefficient every two cycles.

To remove the dependence, we propose a ray trace algorithm with zero padding in Algorithm 4, which always generates two coefficients in each iteration. Although some zero coefficients are generated, different iterations can be executed independently and we can generate one coefficient every cycle. The average number of coefficients per ray is 431 from Algorithm 3, and is 614 from Algorithm 4. We increase the overall efficient of the ray-trace module by about 30%.

We can also add a filter module right after the ray-trace module to eliminate the zero coefficients. This filter will help when multiple ray processing elements are integrated, because the wait time for the data from the fetch module can be used for the elimination.

4.3 Module of Projection and Descent Stages

The descent step is in fact a light-weight version of an inner iteration in Algorithm 1, except that it only performs optimization of a single ray.

The computation $\mathcal{R}_i^\top(\mathcal{R}_i f_{(i)} - g_i)$ is still the bottleneck. Reduction operation involves computing the dot product $\mathcal{R}_i f_{(i)}$. Based on the fact that the floating-point multiply-add operation on FPGA consume a constant number of cycles, we can pipeline the reduction operation by introducing extra registers to hold partial dot products. A toy example assuming a multiply-add operation, which costs three clock cycles, is illustrated in Figure 4, and three registers are introduced. In this way, we have enough time to consume two elements, and complete one multiply-add operation every cycle. We only need to sum up all three partial dot products at the end. Thus, the module of projection stage is able to process one pixel per cycle.

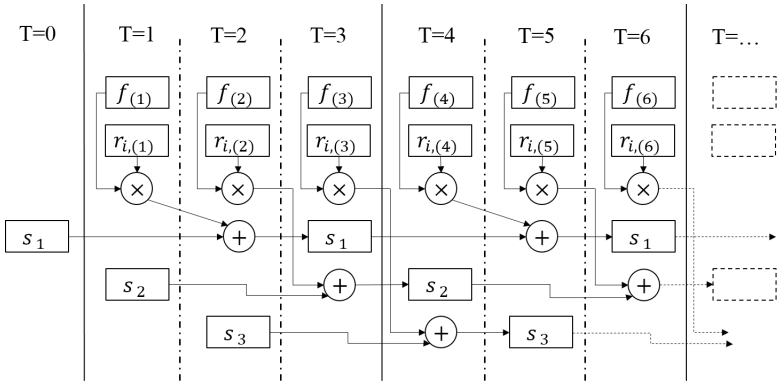


Figure 4: A toy example of pipelined dot product assuming a multiply-add operator consumes three cycles

The remaining pixel-wise computation related to gradient computation and descent movement is relatively easy to pipeline, and thus the descent module can also process one image pixel in f and one edge indicator pixel in v per cycle.

4.4 Modules of Fetch and Commit

There are three major memory consumptions in the algorithm: the projection data g (180×768, 0.5MB), the image f (512×512, 1MB) and the edge indicator v (512×512, 1MB). The projection is read-only, and other two will be frequently updated. In addition, the projection data has a low contention of access, where only one projection point is needed for each ray. Thus, we put the projection g in the off-chip DRAM, and organize the image f and the edge indicator v in the on-chip BRAM.

Because a block of on-chip BRAM has only two ports for reading and writing, we partition the centralized on-chip image f and edge indicator v into 4×4 mega-blocks (each with a size of 128×128 pixels) to enable simultaneous access by several ray processing elements. We implement one access queue for each mega-block, such that the ray processing element with the highest priority will be guaranteed with enough throughput to optimize one pixel per cycle. The other ray processing elements may have to wait until the mega-block that it is going to read or write is idle. The 4×4 mega-blocks enable 2.8 effective ray processing elements to fetch and commit data simultaneously, when the rays are accessed sequentially (*i.e.*, the parallel rays with the same angle

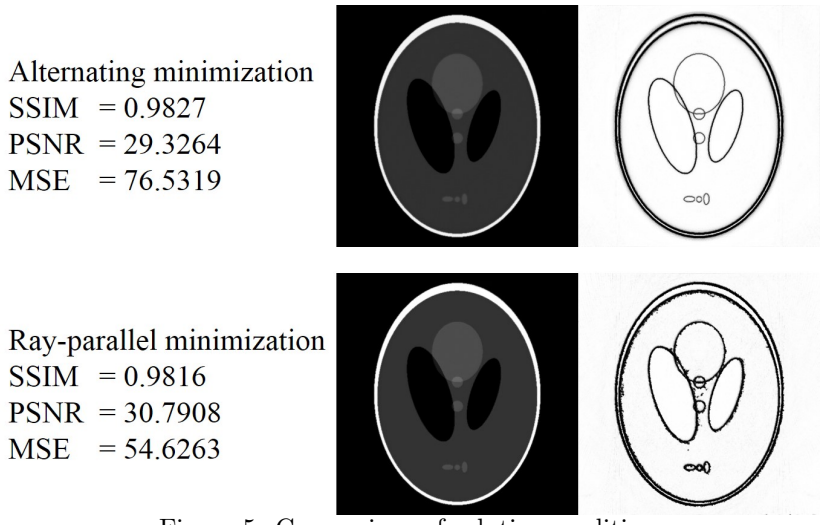


Figure 5: Comparison of solution qualities

are accessed in the same order as they are measured from the leftmost detector to the rightmost one, and then the parallel rays with the next angle are accessed in the same way).

Because the differential operators requires a stencil access of five pixels (*e.g.*, $f_{i,j}, f_{i+1,j}, f_{i,j+1}, f_{i-1,j}, f_{i,j-1}$) to update one pixel, we perform a cyclic partition for every mega-block in both dimensions with a period of 3.

5. EXPERIMENTAL RESULTS

In this section, we first present the solution quality of the asynchronous ray-parallel algorithm for the SRS problem. After that, we evaluate the performance, power and energy consumption of the FPGA implementation of the ray-parallel algorithm, and we also compare it against its CPU and GPU implementations.

5.1 Analysis of the Quality of SRS Solution

In order to evaluate the quality of the reconstructed image, three metrics have been used: the structural similarity (SSIM) index, the peak signal-to-noice ratio (PSNR), and the mean square error (MSE). SSIM[25] is a measure of structural similarity that compares local patterns of pixel intensities that have been normalized for luminance and contrast. PSNR is often used to measure the quality between the original image and a compressed one. MSE, as the average of squared error, has clear physical meanings but it has been proved to be inconsistent with human eye perception.

We use the Shepp-Logan head phantom to test the quality of reconstruction and segmentation. The visual difference between the alternating minimization and the ray-parallel minimization are illustrated in Figure 5. The upper two figures are the reconstruction and the segmentation from the alternating minimization method, and the lower two figures are from the ray-parallel minimization method. The quantitative comparisons are shown on the left-hand side, where the metrics of SSIM, PSNR, and MSE, compared with the true original image, are listed. The two methods only have negligible difference in SSIM, and the ray-parallel method is better in terms of PSNR and MSE.

5.2 Design Platform and Evaluations

Xilinx Virtex-7 board VC707 is selected to be the target hardware platform in our experiment. Xilinx Vivado Design Suite 2015.1 is invoked to generate the modules in our design.

In our experiment, the Shepp-Logan phantom[26] is used as the test input with an image size of 512×512. The outer iteration number is set to 10.

5.3 Detailed Performance Analysis

Table 1 shows the execution time (T), power (P) and energy consumption (E) of the alternating minimization (alt) and the ray-parallel minimization (ray). The performance is defined by 1/T, the energy efficiency is defined by 1/E, and both are normalized against the smallest value (CPU ×1 (alt) in this case).

The CPU version is compiled by gcc-4.7.2 with options "-O3 -funroll-all-loops -fprefetch-loop-arrays" and runs on an x86 server with Intel Xeon E5-2430 CPU at 2.20GHz and 32GB memory. The label ×1 indicates a single-thread implementation, and the label ×8 indicates a 8-thread implementation. The GPU version runs on an AMD Radeon R7 200 Series GPGPU.

The FPGA (ray) duplicates the ray processing element by 4 times on FPGA, running at a frequency of 100MHz. From the table, we can see that the optimized FPGA (ray) achieves a 9× speedup over CPU ×1 (ray). GPU has more cores and consumes much more energy than FPGA implementations. The FPGA (ray) achieves a 1.2× speedup with an energy efficiency as great as 58×, over the GPU (alt).

Table 1: Experimental Results of Different Implementations

Implementation	T (s)	P (W)	E (J)	1/T	1/E
CPU ×1 (alt)	220.0	95	20900	1	1
CPU ×1 (ray)	33.0	95	3135	7	7
CPU ×8 (ray)	6.9	95	656	32	32
GPU (alt)	6.0	236	1416	37	15
GPU (ray)	4.1	236	968	54	22
FPGA (ray)	3.5	4.8	17	63	1244

Table 2 shows the consumption of hardware resources and the utilization rates of the FPGA (ray) implementation. The utilization rates of LUT and BRAM resources are considerably higher when we use 4 ray processing elements.

Table 2: Resource utilization of FPGA (ray)

Implementation	BRAM_18K	DSP48E	FF	LUT
available	2060	2800	607200	303600
ray processing element (×1)	512	172	20128	25607
data management (1 × 1 mega-block)	1024	0	2058	5708
total (1-ray)	1536 (75%)	172 (6%)	22186 (4%)	31315 (10%)
ray processing element (×4)	512	688	81566	103505
data management (4 × 4 mega-blocks)	1024	0	46282	121614
total (4-ray)	1536 (75%)	688 (25%)	127848 (21%)	225119 (74%)

6. CONCLUSION

In this paper, we proposed an asynchronous ray-parallel algorithm for the SRS problem with the Mumford-Shah model and accelerated it on FPGA devices using a high-level-synthesis-based design methodology. The ray-parallel algorithm is designed for the computational paradigm on FPGAs, and enables an efficient pipelining implementation. Experimental results show that the FPGA implementation achieves a 1.2× speedup with an energy efficiency as great as 58×, over the GPU implementation.

ACKNOWLEDGMENTS

This work is partly supported by National Natural Science Foundation of China (NSFC) Grant 61202073, Research Fund for the Doctoral Program of Higher Education of China (MoE/RFDP) Grant 20120001120124, and Beijing Natural Science Foundation (BJNSF) Grant 4142022.

REFERENCES

[1] Ramlau, R., Klann, E., and Ring, W., "Simultaneous reconstruction and segmentation for tomography data," *PAMM* **7**(1), 1050303–1050305 (2007).

[2] Zhang, Q., Plemmons, R., Kittle, D., Brady, D., and Prasad, S., "Joint segmentation and reconstruction of hyperspectral data with compressed measurements," *Appl. Opt.* **50**, 4417–4435 (Aug 2011).

[3] Hinkle, E. B., Sanz, J. L., Jain, A. K., and Petkovic, D., "P3e: New life for projection-based image processing," *Journal of Parallel and Distributed Computing* **4**(1), 45 – 78 (1987). Special Issue on Parallel Image Processing and Pattern Recognition.

[4] Agi, I., Hurst, P., and Current, K., "An image processing IC for backprojection and spatial histogramming in a pipelined array," *IEEE Journal of Solid-State Circuits* **28**(3), 210–221 (1993).

[5] Current, W., Hurst, P., Shieh, E., and Agi, I., "An evaluation of radon transform computations using DSP chips," *Machine Vision and Applications* **3**(2), 63–74 (1990).

[6] Neri-Caldern, R. A., Alcaraz-Corona, S., and Rodrguez-Dagnino, R. M., "Cache-optimized implementation of the filtered backprojection algorithm on a digital signal processor," *Journal of Electronic Imaging* **16**(4), 043010–043010–13 (2007).

[7] Kachelriess, M., Knaup, M., and Bockenbach, O., "Hyperfast parallel–beam backprojection," *Proceedings of the Nuclear Science Symposium Conference Record* **5**, 3111–3114 (2006).

[8] Baer, M. and Kachelriess, M., "High performance parallel beam and perspective cone-beam backprojection on Intel Xeon Phi," *Proceedings of Fully 3D* (2013).

[9] Cabral, B., Cam, N., and Foran, J., "Accelerated volume rendering and tomographic reconstruction using texture mapping hardware," *Proceedings of the 1994 Symposium on Volume Visualization*, 91–98, ACM, New York, NY, USA (1994).

[10] Xue, X., Cheryauka, A., and Tubbs, D., "Acceleration of fluoro-CT reconstruction for a mobile C-Arm on GPU and FPGA hardware: a simulation study," *Proc. SPIE 6142*, 61424L (2006).

[11] Mendl, C., "Real-time radon transform via the GPU graphics pipeline," tech. rep. (2010).

[12] Knaup, M., Steckmann, S., and Kachelriess, M., "GPU-based parallel-beam and cone-beam forward- and backprojection using cuda," *Proceedings of the IEEE Nuclear Science Symposium Conference Record*, 5153–5157 (2008).

[13] Zhang, W., Zhang, L., Sun, S., Xing, Y., Wang, Y., and Zheng, J., "A preliminary study of opencl for accelerating ct reconstruction and image recognition," *Proceedings of the IEEE Nuclear Science Symposium Conference Record*, 4059–4063 (Oct 2009).

[14] Coric, S., Leeser, M., Miller, E., and Trepanier, M., "Parallel-beam backprojection: An FPGA implementation optimized for medical imaging," *Proceedings of the 2002 ACM/SIGDA Tenth International Symposium on Field-programmable Gate Arrays*, 217–226, ACM, New York, NY, USA (2002).

[15] Xu, J., Subramanian, N., Alessio, A., and Hauck, S., "Impulse C vs. VHDL for accelerating tomographic reconstruction," *Proceedings of the 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines*, 171–174 (2010).

[16] Chen, J., Cong, J., Yan, M., and Zou, Y., "FPGA-accelerated 3D reconstruction using compressive sensing," *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, 163–166, ACM, New York, NY, USA (2012).

[17] Choi, Y. K., Cong, J., and Wu, D., "FPGA Implementation of EM Algorithm for 3D CT Reconstruction," *Proceedings of the 22nd IEEE International Symposium on Field-Programmable Custom Computing Machines*, 157–160, IEEE Computer Society, Washington, DC, USA (2014).

[18] Ambrosio, L. and Tortorelli, V. M., "On the approximation of free discontinuity problems," *Boll. Un. Mat. Ital. B (7)* **6**(1), 105–123 (1992).

[19] Page, T., *Simultaneous Reconstruction and Segmentation with the Mumford-Shah Functional for X-Ray Tomography*, Master's thesis, University of Bremen, Germany (2011).

[20] Recht, B., Re, C., Wright, S., and Niu, F., "Hogwild: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent," in [*Advances in Neural Information Processing Systems 24*], Shawe-Taylor, J., Zemel, R. S., Bartlett, P. L., Pereira, F., and Weinberger, K. Q., eds., 693–701, Curran Associates, Inc. (2011).

[21] Bertsekas, D. P. and Tsitsiklis, J. N., [*Parallel and Distributed Computation: Numerical Methods*], Athena Scientific (1997).

[22] Liu, J., Wright, S. J., and Sridhar, S., "An asynchronous parallel randomized Kaczmarz algorithm," *CoRR* **abs/1401.4780** (2014).

[23] Bresenham, J., "Algorithm for computer control of a digital plotter," *IBM Systems Journal* **4**(1), 25 – 30 (1965).

[24] Klukowska, J., Davidi, R., and Herman, G. T., "SNARK09 - a software package for reconstruction of 2D images from 1D projections," *Computer Methods and Programs in Biomedicine* **110**(3), 424 – 440 (2013).

[25] Wang, Z., Bovik, A., Sheikh, H., and Simoncelli, E., "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing* **13**, 600–612 (April 2004).

[26] Kak, A. C. and Slaney, M., [*Principles of Computerized Tomographic Imaging*], IEEE Press (1998). available online at http://www.slaney.org/pct/pct-toc.html.