A Smart Memory Accelerated Computed Tomography Parallel Backprojection

Qiuling Zhu, Larry Pileggi, and Franz Franchetti

Department of Electrical and Computer Engineering
Carnegie Mellon University, Pittsburgh, PA, USA
qiulingz@andrew.cmu.edu

Abstract. As nanoscale lithography challenges mandate greater pattern regularity and commonality for logic and memory circuits, new opportunities are created to affordably synthesize more powerful smart memory blocks for specific applications. Leveraging the ability to embed logic inside the memory block boundary, we demonstrate the synthesis of smart memory architectures that exploits the inherent memory address patterns of the backprojection algorithm to enable efficient parallel image reconstruction at minimum hardware overhead. An end-to-end design framework in sub-20nm CMOS technologies was constructed for the physical synthesis of smart memories and evaluation of the huge design space. Our experimental results show that customizing memory for the computerized tomography (CT) parallel backprojection can achieve more than 30% area and power savings while offering significant performance improvements with marginal sacrifice of image accuracy.

Keywords: Smart Memory, Logic and Memory Synthesis, Computed Tomography, Parallel Backprojection.

1 Introduction

Computationally intensive algorithms in medical image processing (e.g., computerized tomography (CT)) require rapid processing of large amounts of data and often rely on hardware acceleration [1–3]. Inherent parallelism in the algorithms is exploited to achieve the required performance by increasing the number of parallel functional units at a cost of power and area. The overall performance is often defined by the limited bandwidth of the on-chip memory as well as the high cost of memory access.

One approach to address these challenges is to optimize the on-chip memory organization by constructing a customized smart memory module that is optimized for a particular function for higher performance and/or energy efficiency [4, 5]. However, such customization is generally unaffordable for an application-specific IC embedded memory for which cost dictates that it is "compiled" from a set of SRAM hard IP components (e.g., physical implementations of bitcells and peripheral circuits). Such memory compilation limits the possibility of application-specific customization and hinders the system design space exploration.

A. Burg et al. (Eds.): VLSI-SoC 2012, IFIP AICT 418, pp. 21–44, 2013.

Recent studies of sub-20nm CMOS design indicate that memory and logic circuits can be implemented together using a small set of well-characterized pattern constructs [6, 7]. Our early silicon experiments in a sub-20nm commercial SOI CMOS process demonstrate that this construct-based design enables logic and bitcells to be placed in a much closer proximity to each other without yield or hotspots pattern concerns. While such patterning appears to be more restrictive to accommodate the physical realities of sub-20nm CMOS, the ability to make the patterns the only required hard IP allows us to efficiently and affordably customize the SRAM blocks. More importantly, it enables the synthesis (not just compilation) of customized memory blocks with user control of flexible SRAM architectures and facilitate *smart memory compilation.*

To efficiently leverage this new technology, however, algorithms and hardware architectures need to be revised. In this paper we revisit the well-known Shepp and Logan's backprojection algorithm that is widely used in the CT image reconstruction [3]. It is observed that in the parallel implementation of the algorithm, the memory address differences are fairly small for adjacent projection angles and adjacent pixels. We exploit this property via a customized memory structure that could feed in-parallel running image processing engines (IPEs) with a large amount of required projection data in one clock cycle. The implementation is realized by embedding "intelligent" functionality into the traditional interleaved memory organization and allow multiple memory sub-banks to share the memory periphery. Novel periphery-sharing smart memory strategies are explored, and an efficient parallel-pipeline backprojection architecture is proposed. We further construct a smart memory design framework that provides the end user with finer control of the customized SRAM architecture parameters, thus enabling automatic generation of the specified implementation. Physical implementations were carried out in a commercial sub-20nm SOI CMOS process. Our results indicate that there is more than 40% area savings and 30% power savings while providing significant performance improvements. The marginal impact on accuracy is minimized with appropriate constraints on the algorithm.

Related Work. In other related work various fast approaches have been proposed to improve the backprojection implementation [2, 8, 9, 3]. As pointed out in [3], these approaches may be classified into three categories; namely, algorithmic improvement, dedicated hardware, and parallel processing. However, this paper shows that it is possible to combine these three aspects to deliver a more efficient backprojection architecture by taking advantage of the availability of smart memory synthesis. Our approach optimizes the parallel backprojection architecture, especially the on-chip memory architecture, by exploiting the inherent memory address pattern that has not been previously explored.

2 Background

Filtered backprojection is the most commonly used approach for image reconstruction from parallel-beam projection data. Before analyzing the inherent

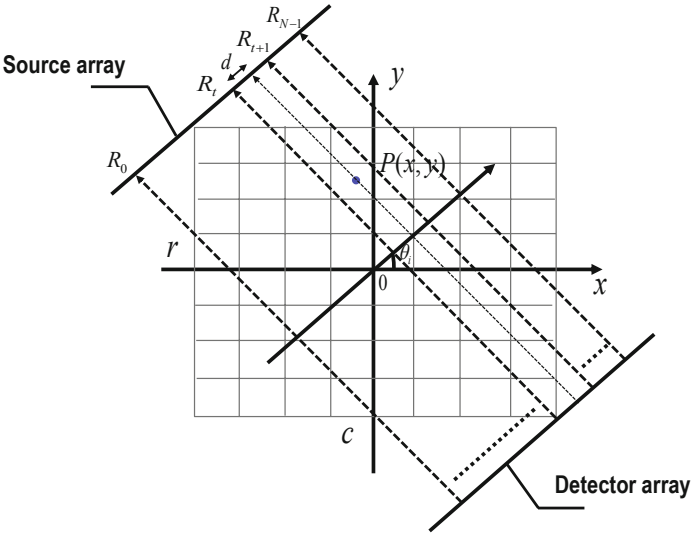


Fig. 1. Illustration of Parallel-Beam Projection: The object to be scanned is placed between the evenly spaced array of an unidirectional X-ray source and the detector. Radiation beams from the X-ray source pass through the object and are measured at the detector, forms the projections of the image.

memory access pattern and building the corresponding customized memory architecture, in this section we will first introduce the parallel-beam CT scanning system and the commonly used backprojection algorithm.

2.1 CT Scanning Method

Tomography is a non-invasive imaging technique allowing for the visualization of the internal structures of an object. Tomography has found widespread applications in many scientific fields, including physics, chemistry, astronomy, geophysics, and medicine. A parallel-beam CT scanning system uses an array of equally spaced unidirectional sources of focused X-ray beams. The object to be scanned is placed between the sources array and the detector. Radiation beams from the source pass through the object and are measured at the detector (see Fig. 1). A complete set of projections is obtained by rotating the arrays and taking measurements for different angles over 180°, forming the Radon transform of the image (i.e., projection data), and it contains information needed for the reconstruction of an image. A set of values given by all detectors in the array comprises a one-dimensional projection data. The inverse of the projection data allows to reconstruct the tomographic images (i.e., backprojection) [10, 1]. The Radon transform and its inverse provide the mathematical basis for reconstructing tomographic images from the measured projection data.

2.2 Shepp and Logan Backprojection Algorithm

The most widely known reconstruction-from-projections test image is the Shepp-Logan phantom. Introduced in 1974 it is still in common use today as a reference image for reconstruction algorithms. The Shepp and Logan backprojection algorithm is the most well-known backprojection algortithm [3, 11]. In the conventional Shepp and Logan backprojection algorithm, for each pixel, P, located at (x, y), and each projection angle θ_i, the first step in backprojection is to locate the pixel in an appropriate beam (ray). If the center of P is not on a ray, the distance (d) to its adjacent rays is calculated and the contribution from the adjacent rays to the pixel (Q_p) is computed according to the linear interpolation equation (1), assuming that pixel is enclosed by the t_{th} and $(t+1)_{th}$th rays,

$$Q_p(x, y, \theta_i) = R_t + (d/L) \cdot (R_{t+1} - R_t), \tag{1}$$

where R_t is the value of t_{th} ray, d is the interpolation distance, and L is the ray interval. Q_p represents the contribution of the projection of angle θ_i to the current pixel value.

In the above equation, the address t to the projection data memory and the interpolation distance d are computed as follows (assuming the target image has the dimension size of $r \times c$):

$$t_{x,y,\theta_i} = \left(x - \frac{r}{2}\right) \cdot \cos\theta_i - \left(y - \frac{c}{2}\right) \cdot \sin\theta_i + t_{\text{offset}}. \tag{2}$$

and the interpolation distance d is calculated as follows:

$$d = t(\theta) - \lfloor t(\theta) \rfloor. \tag{3}$$

Existing Algorithm Optimization. The above procedures, locating and interpolation, are to be repeated for every pixel and for every projection angle. However, there exists computational redundancy that can be explored to save the operations in the iterations. To do this, $2D$ Shepp and Logan algorithm exploits the property of constant difference of address t for those pixels on the same row or column. Considering two adjacent pixels located at (x, y) and $(x+1, y)$, and backprojection angle θ, we can calculate the addresses to the projection memory for the two pixels based on (2). And the difference of their addresses, $t_{x+1,y,\theta_i} - t_{x,y,\theta_i}$ is equal to $cos(\theta)$, which is a constant for a given θ. Let δt_x denotes the constant difference along the x direction. Then, in the $2-D$ Shepp and Logan algorithm, instead of evaluating equation (2) for every pixel, it simply adds a constant of $cos(\theta)$ to the previous adjacent address index (t_{x,y,θ_i}) to generate the new address index (t_{x+1,y,θ_i}). The same rule can be applied to the y direction to calculate the address index ($t_{x,y+1,\theta_i}$) by adding the constant difference δt_y of $sin(\theta)$ to (t_{x,y,θ_i}).

3 Memory Address Pattern Analysis

This paper moves one step forward by taking advantage of these constant address differences of δt_x and δt_y that exist in the conventional Shepp and Logan

Backprojection algorithm, to simplify not only the address calculation but also the underlying memory hardware. Furthermore, we will demonstrate that the address differences when the projection angle θ changes are also within a very small and predictable range that could be also exploited to further optimize the hardware memory design.

3.1 Address Difference for Adjacent Projections

For each pixel (x, y) and each projection angle (θ_i), the beam index t_{x,y,θ_i} (i.e., address to the projection memory) is already shown as in (2). To illustrate the inherent address patterns that were hidden in the algorithm, we show the address to the next projection of angle θ_{i+1} in (4):

$$t_{x,y,\theta_{i+1}} = \left(x - \frac{r}{2}\right) \cdot \cos\left(\theta_{i+1}\right) - \left(y - \frac{c}{2}\right) \cdot \sin\left(\theta_{i+1}\right) + t_{\text{offset}}. \tag{4}$$

The address difference (δt_1) between (2) and (4) could be as

$$\delta t_1 = \left(x - \frac{r}{2}\right) \cdot \delta cos_{\theta_i} + \left(\frac{c}{2} - y\right) \cdot \delta sin_{\theta_i}, \tag{5}$$

with $\delta cos_{\theta_i} = \cos(\theta_{i+1}) - \cos(\theta_i)$ and $\delta sin_{\theta_i} = \sin(\theta_{i+1}) - \sin(\theta_i)$.

δcos_{θ_i} can be rewritten as:

$$\delta cos_{\theta_i} = \cos\theta_{i+1} - \cos\theta_i = -2\sin\frac{\theta_{i+1} + \theta_i}{2}\sin\frac{\theta_{i+1} - \theta_i}{2} \tag{6}$$

For $\theta_i = \frac{2\pi i}{N}$, $\frac{\theta_{i+1} - \theta_i}{2}$ is the constant π/N, so δcos_{θ_i} is simplified as

$$\delta cos_{\theta_i} = -2\sin\left(\frac{\pi}{N}\right)\sin\left(\frac{\pi(2i+1)}{N}\right) \tag{7}$$

Similarly, we have:

$$\delta sin_{\theta_i} = 2\sin\left(\frac{\pi}{N}\right)\cos\left(\frac{\pi(2i+1)}{N}\right) \tag{8}$$

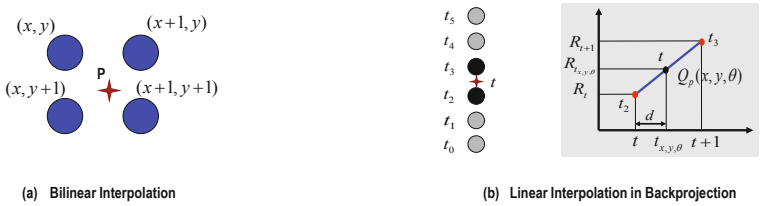
So, (5) can be written as:

$$\delta t_1 = \left(x - \frac{r}{2}\right) \cdot \left(-2\sin(\frac{\pi}{N})\sin\frac{\pi(2i+1)}{N}\right) + \left(\frac{c}{2} - y\right) \cdot \left(2\sin(\frac{\pi}{N})\cos\frac{\pi(2i+1)}{N}\right) \tag{9}$$

Therefore, using trigonometric identities, we can compute the bound on (5) as follows:

$$|\delta t_1| \leq |2 \cdot \sin\left(\frac{\pi}{N}\right) \cdot \frac{r}{2} \cdot \left(\cos\left(\frac{\pi(2i+1)}{N}\right) - \sin\left(\frac{\pi(2i+1)}{N}\right)\right)|. \tag{10}$$

(10) has a maximum bound of $\sqrt{2}\pi \cdot \frac{r}{N}$ for relatively large N.



(a) Bilinear Interpolation (b) Linear Interpolation in Backprojection

Fig. 2. Interpolation in CT Backprojection

Here we assuming $r = c$ is the dimension size of a square image and N is the number of projection angles. It is shown that δt_{θ_1} is restricted in a very limited range when the ratio of r and N is relatively small. For example, δt_{θ_1} must be less than 1 when $\frac{r}{N} \leq \frac{1}{8}$.

This observation can easily extend to the scenario of computing the contribution of consecutive k projection angles to the same pixel (x, y). In this situation, the address differences will be accumulated and the resulting accumulating address difference between the next k projection memory of angle θ_k and the first memory of angle θ_1 for the same pixel $P(x, y)$ will increase proportionally to k:

$$|\delta t_k| = |t_{x,y,\theta_{i+k}} - t_{x,y,\theta_i}| \leq \sqrt{2}\pi \cdot \frac{r}{N} \cdot k \cdot \approx 4.44 \cdot \frac{r}{N} \cdot k. \tag{11}$$

For certain value of k, δt_k will still be within a very small value.

3.2 Address Difference for Adjacent Pixels

In the above section, we have derived the beam index differences for a fixed pixel when projection angles increment. Next, we will show that the address differences when both pixel coordinate and projection angle increment are also bounded by a limited range.

For demonstration purpose, we define the problem as to reconstruct four neighborhood pixels in parallel, that is, (x, y), $(x + 1, y)$, $(x, y + 1)$, $(x + 1, y + 1)$. We will encounter this problem for parallel image reconstruction. For example, Fig. 2 shows the example to compute four neighborhood pixels, (x, y), $(x + 1, y)$, $(x, y + 1)$, $(x + 1, y + 1)$, in parallel. The similar problem could also happen in a higher-level interpolation, that is, the calculation of the non-existing pixel P requires to compute its four neighborhood pixels first and apply a bilinear interpolation afterwards.

We denote the address of the first pixel (x, y) in the first projection memory θ_i as the reference address (t_{x,y,θ_i}). Then, for other three pixels, $(x + 1, y)$, $(x, y+1)$, $(x+1, y+1)$ in the same projection memory of θ_i, their address differences from t_{x,y,θ_i}, can be estimated as shown in (12), (13), (14):

$$|t_{x+1,y,\theta_i} - t_{x,y,\theta_i}| = |\cos(\theta_i)| \leq 1 \tag{12}$$

$$|t_{x,y+1,\theta_i} - t_{x,y,\theta_i}| = |\sin(\theta_i)| \leq 1 \tag{13}$$

$$|t_{x+1,y+1,\theta_i} - t_{x,y,\theta_i}| = |\cos(\theta_i) + \sin(\theta_i)| \leq \sqrt{2} \tag{14}$$

It can be observed that all the shown three address differences are all in a very small range. For the same four pixels, let's now analyze their addresses to the next adjacent projection memory of angle θ_{i+1}. For the first pixel located at (x, y), its address difference from t_{x,y,θ_i} has been calculated in (10) and here we repeated it in (15):

$$|t_{x,y,\theta_{i+1}} - t_{x,y,\theta_i}| = |\delta t_{\theta_1}| \leq \sqrt{2}\pi \cdot \frac{r}{N} \tag{15}$$

Similarly, for the other three pixels, we show their address differences from t_{x,y,θ_i} in (16), (17), (18) respectively:

$$|t_{x+1,y,\theta_{i+1}} - t_{x,y,\theta_i}| = |\cos(\theta_i) + \delta t_{\theta_1}| \leq 1 + \sqrt{2}\pi \cdot \frac{r}{N} \tag{16}$$

$$|t_{x,y+1,\theta_{i+1}} - t_{x,y,\theta_i}| = |\sin(\theta_i) + \delta t_{\theta_1}| \leq 1 + \sqrt{2}\pi \cdot \frac{r}{N} \tag{17}$$

$$|t_{x+1,y+1,\theta_{i+1}} - t_{x,y,\theta_i}| = |\cos(\theta_i) + \sin(\theta_i) + \delta t_{\theta_1}| \leq \sqrt{2} + \sqrt{2}\pi \cdot \frac{r}{N} \tag{18}$$

It can be observed that all of these memory addresses in adjacent projection angles i and $i + 1$ are all very close to reference address t_{x,y,θ_i}. (18) presents the largest possible address distance among them. This is because the pixel to compute in (18) is located at $(x+1, y+1)$, and it changes from the pixel $p(x, y)$ in both x dimension and y dimension while pixels $p(x+1, y)$ and $p(x, y+1)$ only changes from the pixel $p(x, y)$ in either x dimension or y dimension. Therefore, the address difference between $t_{x+1,y+1,\theta_{i+1}}$ and t_{x,y,θ_i} shown in (18) is relatively larger than the other address differences from (15) to (17).

We could extend the observation to the addresses of these four pixels in the next adjacent k projection memories, that is, for projection angles from θ_i and θ_{i+k}. We can easily prove that the involved addresses are also very close to t_{x,y,θ_i} for the required k, and the maximum possible address difference to t_{x,y,θ_i} is introduced by the last pixel $(x+1, y+1)$ in the last projection memory θ_{i+k},

$$|\delta t_{max}| = |t_{x+1,y+1,\theta_{i+k}} - t_{x,y,\theta_i}| = |\cos\theta_i + \sin\theta_i + k \cdot \delta t_1| \tag{19}$$

(19) has the maximum value of $\sqrt{2} + 4.44 \cdot \frac{r}{N} \cdot k$ and it is limited to small range, e.g., the value must be less than four when $\frac{r}{N} \leq \frac{1}{8}$ and $k = 4$.

The basic idea is, since the address differences for adjacent projections angles and adjacent pixels are small, these addresses will activate the same or adjacent wordlines when such memories are located horizontally in parallel with each other. Such particular memory address pattern leads to opportunities to share the memory decoder among these memories by programming extra "intelligent" logic functionalities into the memory periphery.

4 Backprojection Smart Memory Design

In this section, we describe our approach to optimize the memory organization and backprojection architecture based on the observed memory access patterns.

4.1 Interpolation Memory

As we mentioned, linear interpolation is required if the location of a pixel in a specific view in not on a ray. As shown in Fig. 2 (b), if the beam index in a projection memory, t, is not an integer and located in between $[t_2,t_3]$, then the neighborhood pixels t_2 and t_3 will be accessed and an linear interpolation will be performed to compute the required pixel value t. To improve the processing speed, the neighborhood pixels t_2 and t_3 need to be accessed from the memory in one clock cycle. For the single port memory design, this requires to divide the memory into two different memory banks. Therefore, to run two adjacent backprojections in parallel, it requires to implement two separate projection memories, and each memory is divided into two memory banks. Similarly, to run more adjacent backprojections in parallel, it requires to implement more multi-banking projection memories. However, we will show that it is possible to significantly optimize the hardware implementation of such multi-banking memory system if the discussed memory address patterns are well exploited.

4.2 Consecutive Access Memory

We have discussed that linear interpolation operation requires to access two nearest neighborhood pixels from the projection memory in one clock cycle. We would like to extend this operation to access more than two consecutive pixels from the memory in one clock cycle (i.e., multiple consecutive access memory). We will show later in section 4.4 that such multiple neighborhood pixels access will be required to our smart memory design.

We will first introduce a smart memory structure which can output arbitrary number of adjacent memory entries at arbitrary position in one clock cycle. As we have mentioned, this is traditionally accomplished by distributing data across multiple memory banks so that for any consecutive access all data elements are retrieved from different banks without conflicts. Using multiple SRAM banks incurs high overhead since every memory bank requires its own decoder logic. In our previous work [12], we have proposed a *rectangular-access smart memory* which is able to output an arbitrary rectangular block in a 2D data array. Its $1D$ simplified version, called $1D$ *Consecutive Access Memory*, can be used to output consecutive elements from a $1D$ data array.

We exploit the fact that we always read a constant number of consecutive elements per cycle for each operation. The core observation is that after address decoding, the activated wordlines of all memory banks are always adjacent to each other. Based on that, it's possible to optimize the multi-banking memory system to save the periphery overhead. We employ a customized multi-banking

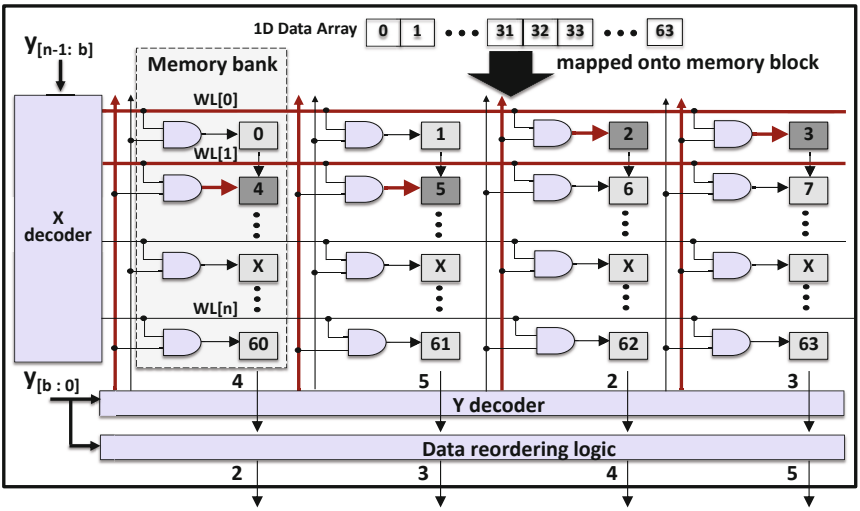


Fig. 3. Consecutive Access Memory. As the basic memory structure in the paper, our customized memory can output consecutive memory entries in one clock cycle and allows parallel memory banks to share the x-decoder.

SRAM design topology [13], which provides around 50% area and power savings compared with the traditional multi-banking memory design. We define the functionality of memory to support one-clock-cycle access of 2^b data points from a 2^n size data array. We build a parameterized memory which is divided into 2^b memory banks and they are located vertically parallel to each other. To control the memory block aspect ratio, we let each word of a memory bank holds 2^c data points. Fig. 3 shows the organization of the memory block when $n = 6$, $b = 2$, $c = 1$. The main idea is to let 2^b memory banks in each memory block share a modified X-decoder by using the same method described in [13]. The X-decoder is specifically designed to activate two adjacent wordlines simultaneously. That is, when one block wordline is asserted, the next block wordline is also asserted by the OR gate operation of every two adjacent wordline signals. Another Y-decoder is used to select one of the two activated wordlines for each memory bank with the AND operations. Each memory bank word holds 2^c data points but each time only one data point of them is required. A column MUX is designed to select one data element for each memory bank and the column MUX is controlled by the lower $(b + c)$ bits of address y $(y_{[b+c-1:0]})$.

As shown in Fig. 3, both the first wordline $(WL[0])$ and the second wordline $(WL[1])$ are initially activated by X-decoder but Y-decoder further selects the $WL[1]$ for the first two memory banks and $WL[0]$ for the last two memory banks with the additional AND operations. After the column MUX, this memory block outputs data series of '$4-5-2-3$', which are then reordered to be '$2-3-4-5$'. So with some simple logic for data reordering, the smart memory outputs the required 2^b data points in order simultaneously. The distribution of address bits

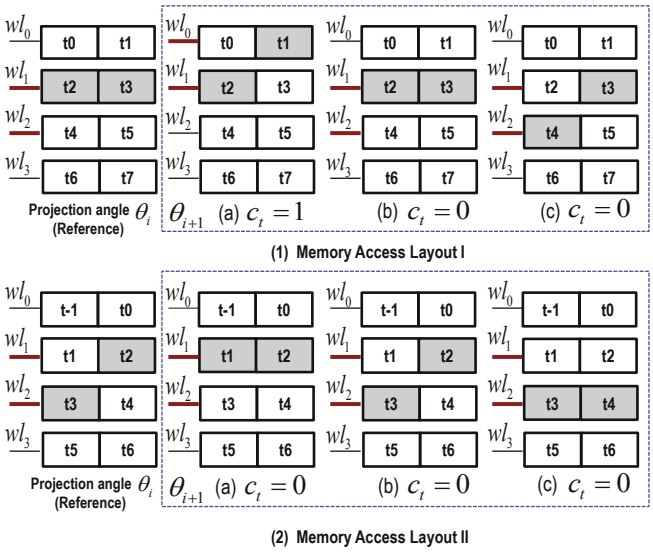


Fig. 4. Data Layout in Adjacent Two Projection Memories. If t_2 and t_3 are required in the first reference memory of the projection θ_i, then beam pixel required in the next memory of projection θ_{i+1} has three possible locations, that is, $[t_1,t_2]$, $[t_2,t_3]$ or $[t_3,t_4]$.

to each memory component is parameterized. By specify these parameters, the resulting memory architecture can be precisely determined. Therefore, we can program the smart memory at the RTL level. Compared with the conventional multi-banking memory design, the amount of memory bank periphery circuits is reduced from 2^b to 1. As is observed in Fig. 3, the resulting memory architecture has the embedded logic gates (e.g. the AND gates) which is tightly integrated with the memory cells, and each logic gate communicates with its local memory cells.

This consecutive access memory serves as the basic memory structure in our method. However, this smart memory structure could be further optimized if provided more knowledge from a particular application. In the rest of paper, we will propose more advanced memory sharing strategies to further optimize the consecutive access memory based on the observed memory access patterns in the backprojection algorithm.

4.3 Decoder-mux and Output-mux

As a simple illustration, in Fig. 4 we show the physical data layout in our consecutive access memory. If the address of projection θ_i is located in between t_2 and t_3 (denoted by $[t_2, t_3]$), then in our previous discussed consecutive access memory design, t_2 and t_3 should either be located in the same wordline or split into two separate wordlines, as shown in the first memory array in Fig. 4 (a) and Fig. 4 (b) respectively. In both situations, two wordlines, wl_1 and wl_2, are activated simultaneously. From the analysis of equation (10), we have derived that

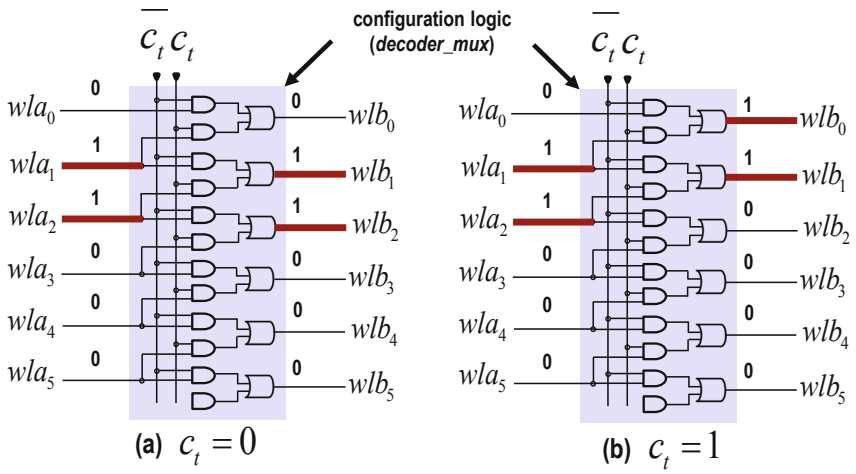


Fig. 5. Decoder-MUX. The wordlines of the first memory (wla_i) are configured to generate the wordlines for the next memory (wlb_i), so that the decoder of the latter memory could be eliminated.

the address difference of the two adjacent memories (δt_{θ_1}) is less than one when $\frac{r}{N} \leq \frac{1}{8}$. This implies that the two adjacent memory addresses after rounding must be either the same or adjacent to each other. Then for the addressed beam index of the next projection memory of angle θ_{i+1}, it will has only three possible locations, that is, $[t_1, t_2]$, $[t_2, t_3]$ or $[t_3, t_4]$, as illustrated in the next three memory layouts of Fig. 4 (1) and Fig. 4 (2). In the illustration we also highlight the corresponding active wordlines if implemented in the consecutive access memory. It's seen that if the active wordlines for the first memory are wl_1 and wl_2, then in the next memory, the active wordlines must be the same in most situations. The only exception is to access t_2 and t_3 from the first projection memory but to access t_1 and t_2 from the second projection memory, as shown in the Fig. 4 (1). In this situation, the active wordlines are shifted upwards by one step. That is, wl_1 and wl_2 are activated in the first projection memory but wl_0 and wl_1 are activated in the second projection memory. We use a control signal c_t to specify the relationship between the two sets of the activated wordlines of the two neighborhood projection memories and c_t can be determined by the input address.

Based on this observation, we propose two "smart" memory approaches which are named *decoder-mux* and *output-mux* respectively

Decoder-mux. In the first approach, called *decoder-mux*, we eliminate the decoder of the second memory and let it share the same decoder with the first memory by adding some configuration logic (which we also call decoder-mux) in between the two sets of memory wordlines. This logic configures the wordlines of the first projection memory (wla_i) to generate the wordlines for the next adjacent projection memory (wlb_i). The relationship between the wordlines of the two adjacent memories can be derived as

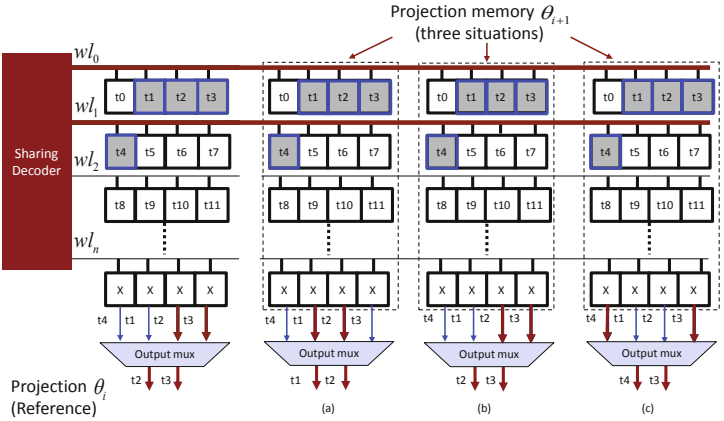


Fig. 6. Output-MUX. The memories are configured to output four pixels simultaneously, and the *output mux* is used to select the required two pixels from the four outputs for the liner interpolation in each backprojection.

$$b_i = (-c_t) \cdot a_i + c_t \cdot a_{i+1}. \tag{20}$$

The configuration can be implemented using only AND and OR logic gates, which ensures the feasibility of the hardware implementation. In Fig. 5, we show an example of the configuration logic involving six wordlines. In this example, wla_1 and wla_2 are activated in the first memory array. After the decoder-mux block, either the same wordlines, wlb_1 and wlb_2, are activated in the second memory when $c_t = 0$ (Fig. 5 (a)), or the neighborhood wordlines, wlb_0 and wlb_1, are activated when $c_t = 1$ (Fig. 5 (b)).

Output-mux. In the alternative approach named *output-mux* the two memories still share the decoder but the configuration logic is located outside of the memory (see Fig. 6). In this approach, memories are designed as the 1×4 consecutive access memories to output more elements than required. In this example, t_2, t_3 along with their nearest neighbors t_1 and t_4 are all read out from the memories. Then the configuration logic (*output-mux*) is used to select the appropriate two elements from the four outputs. In this approach, the active wordlines for the two memories are always the same in all the situations.

4.4 Horizontal and Vertical Parallel Backprojection

The method of *decoder-mux* and *output-mux* can be further extended to let more than two adjacent projection memories share one memory decoder. When more projection memories are involved, the address differences will be accumulated. As explained in the formulae (11), the address difference of the next k projection memory from the first reference memory is increasing proportionally with k. Therefore, we will have to configure the smart memory design in order to

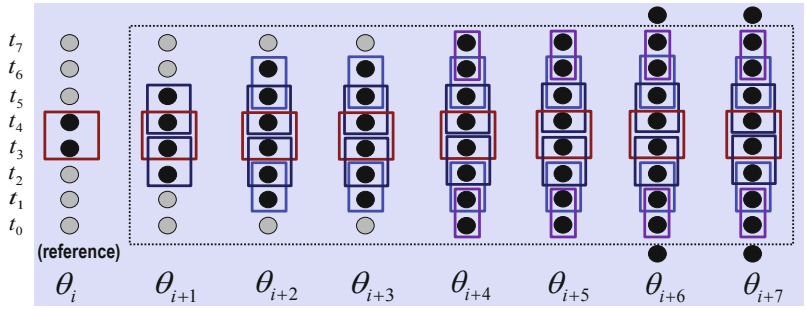


Fig. 7. Parallel Projection Memory Accessing. The highlighted two-pixel groups represent the beam pixels that have chances to be accessed in each projection memory.

accommodate the increased address differences if we want to let more than two adjacent projection memories share one memory decoder.

To exploit the proposed smart memory mechanisms to obtain superior hardware efficiency of the parallel backprojection, we propose two parallel approaches, that is, *horizontal and vertical parallel backprojection.*

Horizontal Parallel Backprojection. The horizontal parallel backprojection can perform more than two backprojections in parallel and all the involved projection memories share the same memory decoder using either *decoder-mux* or *output-mux* approach. Fig. 7 shows the example of accessing in eight adjacent projection memories. Assuming that the pixels addressed by the first memory addresses are t_3 and t_4, we highlight the possible locations of the two pixels accessed in the next seven memories. We observe that they are all clustered locally around t_3 and t_4, and are bounded by t_0 and t_7. For example, the pixels required for projection θ_{i+3} could be any two adjacent pixels within $[t_1, t_6]$. Required pixels spread out further from t_3 and t_4 for memories that are further away from the first memory as explained by formulae (11). Similar to the *output-mux* design shown in Fig. 6, we configure each projection memory as an 1×8 consecutive access memory to output all the shown eight pixels and use another 8-to-2 output-mux to select the appropriate two outputs from the eight outputs for each projection memory. In this way, all the eight memories could share the same decoder and seven memories decoders are saved. However, as the projection memories output more pixels than required, many memory outputs are actually wasted. An approach to use these wasted pixels is applying vertical parallel backprojection, as discussed next.

Vertical Parallel Backprojection. From (12) to (19), we discuss the address differences for performing the backprojections of four neighborhood pixels, that is, (x, y), $(x+1, y)$, $(x, y+1)$, $(x+1, y+1)$, concurrently. Backprojection of each pixel per projection angle requires one linear interpolation and involves memory accessing of two pixels, so totally it requires eight pixels to be accessed from each projection memory. To analyze the address distribution of these pixels, we compute all the involved addresses to the projection memories of projection angle θ_i and projection angle θ_{i+1} respectively, assuming $r/N = 1/4$. We let

t_{x,y,θ_i} be the reference address, and we assume that it is located at t_{13} (see Fig. 8). In the middle column of Fig. 8, we explicitly present the differences of other addresses from the reference address t_{x,y,θ_i}. And in the last column of Fig. 8 we indicate the possible locations of all the accessed pixels. It's seen that the addresses in the first memory array are all localized in between t_{11} and t_{15} , therefore, the access of them will only touch the middle six pixels. In the second projection memory, the accessed pixels are localized in between t_{20} and t_{26}, and therefore any of shown eight pixels in the second memory array could be touched. For a small r/N, it can be expected that the locations of accessing pixels in more adjacent memory arrays will also be localized in between the shown eight pixels. In this way, we support the vertical parallel backprojection which can perform the backprojections of multiple neighborhood pixels in parallel. The memory architecture needs no changes for the vertical parallel backprojection since we just take advantage of the unused memory outputs from the horizontal parallel backprojection. By implementing both horizontal and vertical parallel backprojection concurrently using the modified consecutive access memory, all the memory outputs are utilized and a much higher throughput is achieved.

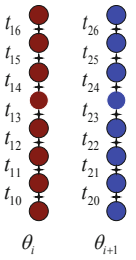
5 Parallel Backprojection Architecture

The CT image reconstruction naturally lends itself to parallel processing since each backprojection can be processed independently. In this section, we will first introduce the conventional pipeline parallel backprojection architecture. Then we will develop a more advanced memory sharing pipeline parallel backprojection architecture based on the smart memory structures that we have introduced.

5.1 Parallel Pipeline BackProjection Architecture

An existing efficient architecture for projectionbased processing is the parallel pipeline backprojection engine (PPPE) [14, 1] due to its simplicity and potential speed. Fig. 9 (a) illustrates the structure of the PPPE based backprojection system, which employs an array of identical IPEs to reconstruct the image recursively, where each IPE performs the same tasks on a different projection. The input image is presented to each IPE on the pipelined image bus, one pixel at a time in a raster-scan format. In raster-scan format the x coordinate of the image is incremented every clock cycle and the y coordinate is incremented every line.

To start the operation, the first IPE in the pipeline is fed a blank image and adds the contribution of the first projection one pixel at a time. After the first IPE adds its contribution, it passes the pixel to the next IPE in the pipelined image bus and each IPE of the pipe adds its projection's contribution to the image. Therefore, each IPE_n in the pipe performs the backprojection for the angle θ_n, and add the resulting value to the input pixel, and then passes the pixel onto IPE_{n+1} as it receives another pixel from IPE_{n-1}. As the image pixel is sent through the pipelined array, the pixel value is reconstructed after accumulating the backprojected values from all the projections. The pipelined calculation and the raster-scan input allow high data throughput of one pixel per clock cycle.



	Beam Index	Index Difference to t_{x,y,θ_i}	Possibly Accessed Data
θ_i	t_{x,y,θ_i}	0	t_{13}
	t_{x+1,y,θ_i}	$\lvert\cos\theta_i\rvert \le 1$	t_{12} t_{13} t_{14}
	$t_{x,y+1,\theta_i}$	$\lvert\sin\theta_i\rvert \le 1$	t_{12} t_{13} t_{14}
	$t_{x+1,y+1,\theta_i}$	$\lvert\sin\theta_i+\cos\theta_i\rvert \le \sqrt{2}$	t_{11} t_{12} t_{13} t_{14} t_{15}
θ_{i+1}	$t_{x,y,\theta_{i+1}}$	$\lvert\delta t_{\theta_i}\rvert \le 1.11$	t_{21} t_{22} t_{23} t_{24} t_{25}
	$t_{x+1,y,\theta_{i+1}}$	$\lvert\cos\theta_i+\delta t_{\theta_i}\rvert \le 2.11$	t_{20} t_{21} t_{22} t_{23} t_{24} t_{25} t_{26}
	$t_{x,y+1,\theta_{i+1}}$	$\lvert\sin\theta_i+\delta t_{\theta_i}\rvert \le 2.11$	t_{20} t_{21} t_{22} t_{23} t_{24} t_{25} t_{26}
	$t_{x+1,y+1,\theta_{i+1}}$	$\lvert\sin\theta_i+\cos\theta_i+\delta t_{\theta_i}\rvert \le 2.52$	t_{20} t_{21} t_{22} t_{23} t_{24} t_{25} t_{26}

Fig. 8. Address Differences Analysis

5.2 Advanced Memory Sharing Parallel Pipeline Backprojection Architecture

If there are fewer IPE in the pipeline than angles (N_θ), then multiple passes through the IPE array are required to reconstruct the image. However, the performance will be decreased proportionally when the number of the IPE decreases. As an effective solution to increase the performance but minimize the hardware cost, we can modify the pipeline backprojection architecture to an more advanced memory-sharing based parallel pipeline backprojection engine (MSPPPE) by taking advantage of the our previous discussed horizonal and vertical backprojection methods. MSPPPE is also composed of a pipeline of identical image processing engines, however, each IPE will perform multiple backprojections to multiple pixels concurrently.

Base on the horizontal parallel backprojection, we let each IPE perform more than one backprojections simultaneously and each IPE needs to hold all of the involved projection data on-chip. So conventionally each projection memory is implemented as a multi-banking memory system in order to supply the data that are required in the parallel CT backprojection. Based on the above horizontal parallel backprojection approach, in each IPE we can combine all the projection data memory into one large memory block by locating them horizontally in parallel with each other so that all of these projection memories could share one memory decoder. In this way, the large overhead that associate with the multiple memory-banking design that were required in the parallel backprojection design can be eliminated. On the other hand, to take advantage of the vertical parallel backprojection, we increase the raster-scan bandwidth by letting more than one pixels pass through the pipeline simultaneously. Although the calculation of the contribution of every projection to every pixel needs to be performed in parallel, only the ALU needs to be duplicated to enable the parallel computing. The memory structure and its associate cost will be the same as above since we will just reuse the redundant output from the horizontal parallel backprojection.

The modified architecture is illustrated in Fig. 9 (b), where we show an example that the input image passes through the IPE on the pipelined image bus, four pixels at a time. Each IPE_n in the pipe performs eight adjacent backprojects from θ_i to θ_{i+7} to the current four pixels ($P(x, y)$, $P(x+1, y)$, $P(x, y+1)$, $P(x+1, y+1)$),

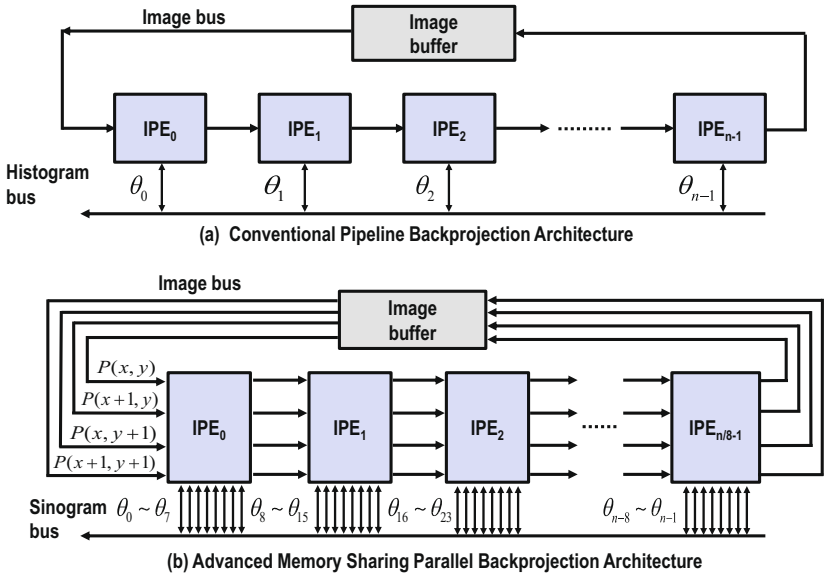


Fig. 9. Parallel Pipeline Backprojection Architecture

and then passes these pixels onto the IPE_{n+1} as it receives another four pixels from IPE_{n-1}. As these pixels are sent through the pipelined array, the pixel values are accumulated from the contributions of all the projections.

6 Design Automation

In this section we analyze the design space and describe our design automation framework for the hardware synthesis of a user-specified backprojection design point.

6.1 Design Tradeoff Space

Designing a CT image reconstruction system is a tradeoff problem involving algorithmic constraints, performance, hardware cost, and image accuracy. The discussion of address patterns in Section 3 shows that the ratio of image dimension size (r) and the projection numbers (N), r/N, is an important algorithm constraint. Smaller r/N indicates smaller adjacent address differences, which allows for more adjacent projection memories sharing the memory decoder, saving more hardware cost and computing latency. However, it also limits the use of the method in applications with larger image size r and/or fewer projection angles N. For larger r/N, the corresponding larger address difference will limit the number of projection memories that can share the decoder. For example, in Fig. 7, the last two projection memories of θ_{i+6} and θ_{i+7} may require to access two pixels at the two ends, which are not accessible along with other eight pixels from the 1×8 consecutive access memory. To solve this problem we could

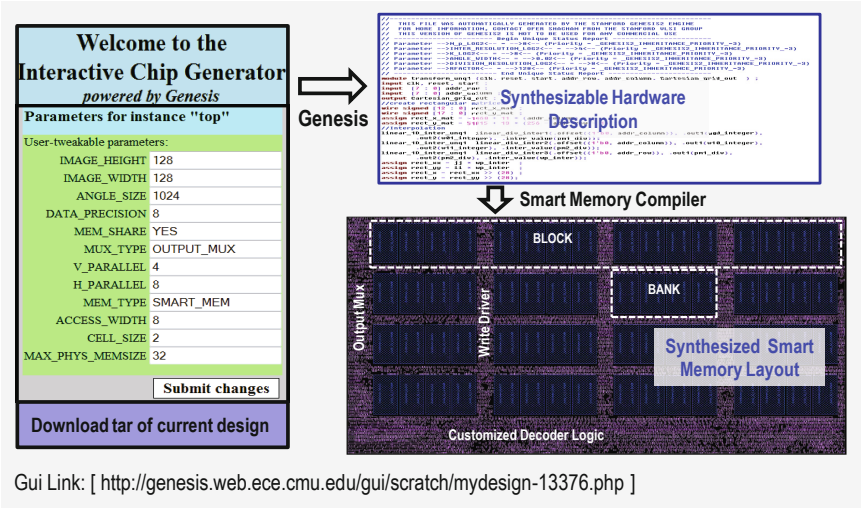


Fig. 10. Smart Memory Design Framework

increase the memory access width and apply more complicated configuration logic. However, this would increase the hardware cost. Alternatively, to lower hardware cost we could assign the nearest neighborhood pixels if the requested pixels are not available, which would result in loss of image accuracy. This shows that different design decisions will result in different tradeoffs. The combination of these design choices constitutes a huge design space. Further, exploring the design tradeoff space requires customized memory designs, which are traditionally prohibitively expensive. Thus, a strong design automation tool is required to make the hardware synthesis feasible.

6.2 Chip Generator and Smart Memory Synthesizer

Application-specific LiM requires to tailor logic and memory design to application or algorithm specifics. Thus, a strong design automation tool is required to make the approach feasible, as hand-designing of LiMs is prohibitively expensive. We have developed a *design generation and design space exploration tool* which will automate the design of proposed customized smart memory blocks.

Our tool provides designers with a graphical user interface to select design parameters, and generate the corresponding hardware for the specified functionality. Un-specified parameters (free parameters) can be optimized by the system. A designer then evaluates the obtained designs and can explore the design space to optimize the design by varying the parameters. We encapsulate all of these design tradeoffs in our automatic design framework and build the backprojection smart memory synthesizer, the user interface is shown in Fig. 10. It enables an application designer to explore the design space to optimize the design by simply varying the parameters and automatically generates the optimized smart memory hardware IP.

Design Exploration and RTL Generation. The tool frontend is built using our chip generator infrastructure "GENESIS" [15, 16] and it's responsible for application interfacing, design optimization and efficient RTL generation. To achieve that, it allows designers to simultaneously code in two interleaved languages: a target language (SystemVerilog) to describe the behavior of hardware and a meta-language (Perl) to decide what hardware to use for given specs. This "dual-language programming" allows to design an entire parameterized family of LiM designs, all at once. Design parameters are set in graphical user interface (GUI) which is defined through XML files. An optimization engine selects optimized values for free parameters. The system supports hierarchical composition of modules and resolving of parameter constraints across modules through all hierarchy levels.

Smart Memory Compiler. The automated design framework discussed so far is capable of mapping LiM application specifications to optimal RTL. Our system also relies on a backend "smart memory" compiler to physically co-synthesize logic and memory. Today's embedded memory is typically synthesized using an SRAM compiler. But the use of commercial SRAM hardware IP is unable to incorporate application-specific customization that are required in the LiM design and also hinders comprehensive design space exploration. LiM physical synthesis is enabled by our *smart memory synthesis framework*, which is developed from the pattern construct based logic and memory co-design methodology [6, 7]. Using this framework, embedded logic in the LiM is synthesized together with the memory cells to a small set of pre-characterized layout pattern constructs. Lithographic compliance between the co-designed logic and memory ensures sub-20nm manufacturability of LiM circuits.

End-to-End LiM Design Framework. In our tool chain we are combining the architectural frontend and physical backend to build an end-to-end LiM design framework. Its input is the design specification and the output is ready to use hardware (RTL, GDS, .lib, .lef). When generating a specified design point, our framework also reports the area, power and latency and send them back to the frontend user interface, from which the designer can evaluate the resulting design and reset the design specs for redesign if necessary. Our LiM framework allows an application designer to generate the optimized "silicon" templates by simply tuning the "knobs".

7 Evaluation and Results

In this section, we evaluate the smart memory architectures with respect to area, power, latency, and accuracy. The design framework is used to generate various design points. Area and power are measured from the physical implementations of the design on a commercial sub-20nm SOI CMOS process at 500MHz and the shown results are all normalized.

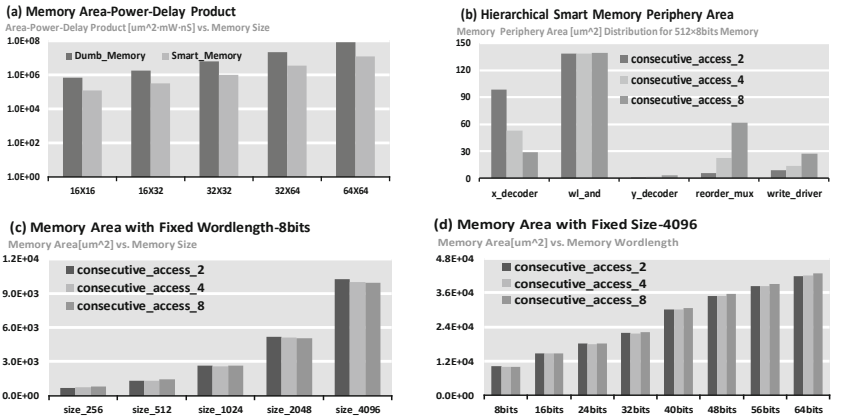


Fig. 11. Consecutive Access Memory Evaluation

7.1 Consecutive Access Memory Evaluation

The smart consecutive access memory is the basic memory structure that we use to implement various backprojection smart memory designs, therefore we evaluate its design efficiency first as shown in Fig. 11. To be consistent with the previous design, we implement the smart consecutive access memory to readout eight consecutive pixels from $1D$ data arrays from size 256 to size 4096. For comparison purpose, we also built the traditional multi-banking memory designs with the same functionalities. In Fig. 11 (a), we demonstrate the power-delay-product of the proposed smart consecutive access memory compared with the traditional multi-banking memory design (i.e., dumb memory), and it shows that the proposed smart memory are one order magnitude more efficient. To better understand the design structure of the smart consecutive access memory, we implement three different consecutive assess memories with different access bandwidths, that is, consecutive assess of two pixels, four pixels and eight pixels. We plot their hierarchical memory periphery area distribution in Fig. 11 (b). We see from the plot that the increase of the access width will decrease the area of the x-decoder while at the same time will increase the area of most other periphery circuit components (e.g., y-decoder, reorder-mux, write-driver and IO registers). This is because when the access width increases, the memory is getting wider and shorter as there will be more memory sub-banks sharing the x-decoder. Cell area is not plotted since it is assumed to be approximately the same for all the designs. For the same reason, the localized wordline AND logic (i.e., *wl_and*) area is the same for all the designs as each memory cell is associated with one AND gate in the customized x-decoder design. In Fig. 11 (c) and (d), we show the overall memory area for smart memory designs with different consecutive access widths at different memory sizes and different memory wordlengths respectively. One important observation is that the increase of the consecutive access width will not increase the overall smart memory area, and sometimes it even decreases the overall memory area for those larger-size memory designs (e.g., memory of

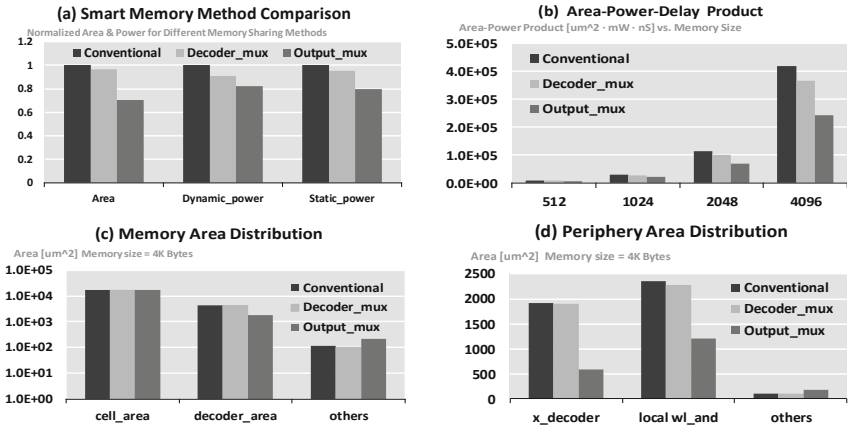


Fig. 12. Backprojection Smart Memory Evaluation

size 4096). This is because larger-size memory is associated with larger memory periphery circuits in the x-dimension (e.g., x-decoder) which can be reduced more in designs with larger access widths. However, the increase of the access width tends to cost more memory area for memories with larger wordlengh since in this situation the periphery circuits in the y-dimension (e.g., y-decoder) is getting larger and more complicated.

7.2 Backprojection Smart Memory Cost Evaluation

Decoder-mux and Output-mux Evaluation. In Fig. 12 (a), we first compare the hardware cost of two smart memory approaches (*decoder-mux* and *output-mux*) to the conventional rectangular access smart memory approach. The memories studied here have the size of 4,096-words and wordlength of 16 bits, and we only consider two memories implemented as 1×8 consecutive access memories sharing the decoder with each other. We observe that the *output-mux* approach is more cost-efficient as saves around 30% area and 20% power while *decoder-mux* only achieves around 5% area saving and 10% power saving. The similar results can be seen in Fig. 12 (b), in which we plot the overall area-power-delay of the three designs at four different memory sizes. As expected, *output-mux* approach saves on average $20\% - 40\%$ in terms of area-power-delay product. On the other hand, *decoder-mux* performs much worse compared with the *output-mux*. The reason is that in *decoder-mux* each wordline is accompanied by a set of configuration logic (two AND gates and one OR gate), and each set of logic communicates with its local wordline. This explains also why *decoder-mux* achieves relatively higher power-efficiency compared to its area-efficiency. In contrast, *output-mux* only requires a single large configuration logic at the memory output while its memories have large access width as they output more pixels than required. Due to the superiority of the *output-mux* method, it will be used for our backprojection system in the following discussions.

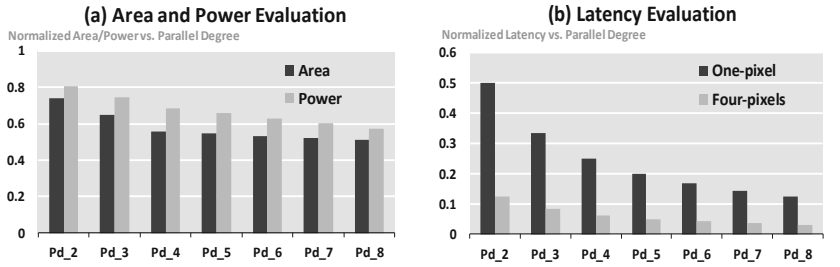


Fig. 13. Memory Sharing Parallel Pipeline Architecture Evaluation

As the main idea of the memory sharing strategy is to reduce the hardware cost by sharing the x-decoder, in order to understand the distribution of the hardware cost of the different components in the memory structure, in Fig. 12 (c) we plot the hierarchical memory area for all the three methods. It is observed that although memory cell array occupies most of the memory area, the periphery area also accounts for a large proportion of overall memory area. As the memory cell area of the three designs are the same, in Fig. 12 (d) we particularly plot the hierarchical memory periphery area for the three methods and we see that the memory periphery is dominated by the x-decoder and the embedded localized wordline AND logic (i.e., *wl_and*) gates. As we discussed in 4.2, the localized wordline AND logic (i.e., *wl_and*) gates are tightly integrated with the memory cell for local wordline activation. As can be seen, both of the decoder area and the local *wl_and* gates area are largely reduced in the output-mux approach as they can be directly shared by all the memory banks.

Parallel Backprojection Architecture Evaluation. In Fig. 13 (a) we evaluate the hardware cost of the MEPPPE memory architecture for reconstructing a 256×256-size image from 1,024 projections. The x-axis is the parallel degree P_d, which is defined as the number of adjacent backprojections that are performed in each IPE concurrently and its value varies from two to eight. In our implementation these P_d projection memories will all share the same memory decoder. The y-axis shows the relative area and power compared to the conventional design where no memory sharing strategies are used. We see that more than 40% area savings and more than 30% power savings can be achieved with the increase of P_d. Fig. 13 (b) shows that the latencies are decreasing proportionally with the increase of P_d as expected. Moreover, we achieve a four times performance improvement by computing four pixels in parallel in each IPE.

7.3 Backprojection Accuracy Evaluation

As we gain in both of hardware cost and performance, the impact on accuracy needs to be evaluated. In Fig. 14 (a), we show the distribution of the locations of the accessed data in eight adjacent projection memories for a real application. We first observe that the locations of the accessed data in eight adjacent projection

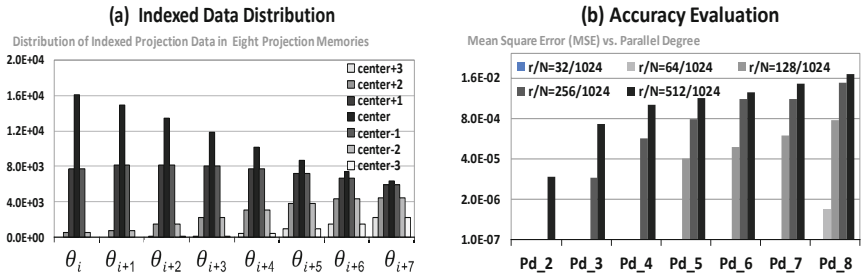


Fig. 14. Image Accuracy Evaluation

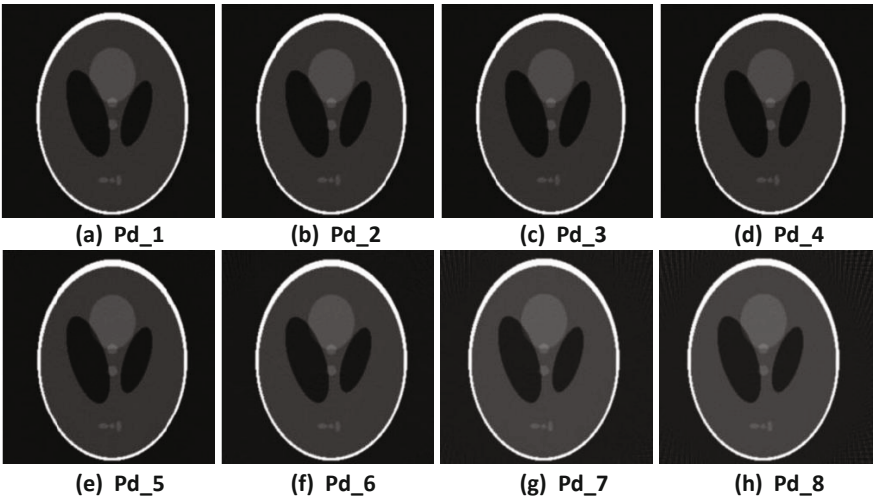


Fig. 15. Display of Reconstructed Image

memories are all localized to the location of *center*. Therefore we could design all the eight projection memories to output the pixels within the range between $center-3$ and $center+3$ so that they could share one memory decoder based on our *output-mux* design. However, it can also be seen that the range of possible locations of the accessed data are increasing when we go from projection memory of angle i to the projection memory of angle $i+7$. For example, starting from projection angle θ_{i+4}, all the shown seven locations will be intensively touched. It can be expected that if we let more adjacent projection memories share the decoder, they could require pixels that are beyond the smart memory outputs. We could approximately assign the nearest pixels if the required pixels are not available but it will then sacrifice the resulting image quality in such situations.

We measure the mean square error (MSE) of the reconstructed image compared to the reference image and plot the results in Fig. 14 (b) for parallel degrees (P_d) from one to eight. As expected, the error increases when either P_d or algorithm parameter (r/N) increases. This is because that we let P_d

projection memories share the same memory decoder, and it will introduce error if the address differences of these P_d projection memories are not small enough which could happen when P_d and (r/N) are large. In our implementation, we carefully manipulate the data precision so that the numerical errors can be ignored in the accuracy comparison. In Fig. 15 we display the reconstructed head phantom images from hardware simulation, which indicates fairly high image quality for all the studied parallel degrees. We also observe the gradual deterioration of the image quality for higher parallel degree, which allows us to tradeoff image accuracy with hardware cost in applications where minor distortion is acceptable.

8 Conclusion

The emergence of construct-based design facilitates the robust synthesis of cost-effective smart memory blocks that are customized for specific applications. This cutting-edge design methodology creates opportunities to re-design algorithms and re-architect the hardware structure to match the advanced technology capabilities. In this paper we propose smart memory architectures and the end-to-end design framework to implement them for the CT image reconstruction problems. The results in a sub-20nm CMOS process demonstrate significant improvements in area, power and performance. Moreover, we present the opportunities to tradeoff hardware cost with acceptable image accuracy based on appropriate algorithm tuning. This paper demonstrates that the embedded memories in data-intensive computing can exploit the smart memory design methodology and the inherent address pattern of the algorithm to achieve superior power and performance efficiency.

Acknowledgement. The authors acknowledge the support of the C2S2 Focus Center, one of six research centers funded under the Focus Center Research Program (FCRP), a Semiconductor Research Corporation entity.

References

1. Agi, I., Hurst, P.J., Current, K.W.: An Image Processing IC for Backprojection and Spatial Histogramming in a Pipelined Array. IEEE Journal of Solid-State Circuits 28(3), 210–221 (1993)
2. Srdjan, C., Miriam, L., Miller, E., Trepanier, M.: Parallel-Beam Backprojection: An FPGA Implementation Optimized for Medical Imaging. FPGA (2002)
3. Chen, C., Cho, Z., Wang, C.: A Fast Implementation of the Incremental Backprojection Algorithms for Parallel Beam Geometries. IEEE Transactions on Nuclear Science 43(6), 3328–3334 (1996)
4. Zhu, Q., Turnerz, E.L., Bergery, C.R., Pileggi, L., Franchetti, F.: Application-Specific Logic-in-Memory for Polar Format Synthetic Aperture Radar. In: IEEE Conference on High Performance Extreme Computing, HPEC (2011)

5. Zhu, Q., Bergery, C.R., Turnerz, E.L., Pileggi, L., Franchetti, F.: Polar Format Synthetic Aperture Radar in Energy Efficient Application-Specific Logic-in-Memory. In: IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 1557–1560 (2012)
6. Morris, D., Rovner, V., Pileggi, L., Strojwas, A., Vaidyanathan, K.: Enabling Application-Specific Integrated Circuits on Limited Pattern Constructs. In: Symp. VLSI Technology (2010)
7. Morris, D., Vaidyanathan, K., Lafferty, N., Lai, K., Liebmann, L., Pileggi, L.: Design of embedded memory and logic based on pattern constructs. In: Symp. VLSI Technology (2011)
8. Luiz, M.C.B., Felipe, M.G.F., Vladimir, C.A., Claudio, L.A.: Reconfigurable Hardware for Tomographic Processing. In: Proceedings of the XI Brazilian Symposium on Integrated Circuit Design, pp. 19–24 (1998)
9. Jang, B., Kaeli, D., Do, S., Pien, H.: Multi GPU Implementation of Iterative Tomographic Reconstruction Algorithm. In: International Symposium on Biomedical Imaging (ISBI), pp. 185–188 (2009)
10. Yu, H.Q.: Memory Architecture for Data Intensive Image Processing Algorithms in Reconfigurable Hardware. Master Thesis (2003)
11. Cho, Z.H., Chen, C.M., Lee, S.Y.: Incremental Algorithm - A New Fast Backprojection Scheme for Parallel Beam Geometries. IEEE Transactions on Medical Image 9(2), 207–217 (1990)
12. Zhu, Q.L., Vaidyanathan, K., Shachamy, O., Horowitz, M., Pileggi, L., Franchetti, F.: Design Automation Framework for Application-Specific Logic-in-Memory Blocks. In: Application-Specific Systems, Architectures and Processors (ASAP), pp. 125–132 (2012)
13. Murachi, Y., Kamino, T., Miyakoshi, J., Kawaguchi, H., Yoshimoto, M.: A Power-Efficient SRAM Core Architecture with Segmentation-Free and Rectangular Accessibility for Super-Parallel Video Processing. In: IEEE International Symposium on VLSI Design, Automation and Test (VLSI-DAT), pp. 63–66 (2008)
14. Hinkle, E.B., Sanz, J.L.C., Jain, A.K., Petkovic, D.: P3E: New life for projection-based image processing. Journal of Parallel and Distributed Computing 4(1), 45–78 (1987)
15. Shacham, O.: Chip multiprocessor generator: automatic generation of custom and heterogeneous compute platforms. PhD Thesis, Stanford (2011)
16. Stanford genesis2 web site, `http://genesis2.stanford.edu/mediawiki/index.php`