J Sign Process Syst
DOI 10.1007/s11265-012-0720-4

Local Interpolation-based Polar Format SAR: Algorithm, Hardware Implementation and Design Automation

Qiuling Zhu · Christian R. Berger · Eric L. Turner · Larry Pileggi · Franz Franchetti

Received: 22 June 2012 / Revised: 8 November 2012 / Accepted: 20 November 2012

Abstract In this paper we present a local interpolation-based variant of the well-known polar format algorithm used for synthetic aperture radar (SAR) image formation. We develop the algorithm to match the capabilities of the application-specific logic-in-memory processing paradigm, which off-loads lightweight computation directly into the SRAM and DRAM. Our proposed algorithm performs filtering, an image perspective transformation, and a local 2D interpolation, and supports partial and low-resolution reconstruction. We implement our customized SAR grid interpolation logic-in-memory hardware in advanced 14 nm silicon technology. Our high-level design tools allow to instantiate various optimized design choices to fit image processing and hardware needs of application designers. Our simulation results show that the logic-in-memory approach has the potential to enable substantial improvements in energy efficiency without sacrificing image quality.

Keywords Synthetic aperture radar · Interpolation · Logic in memory · Chip generator

Q. Zhu (✉) · L. Pileggi · F. Franchetti
Department of Electrical and Computer Engineering,
Carnegie Mellon University, Pittsburgh, PA, USA
e-mail: qiulingz@andrew.cmu.edu

C. R. Berger
Wireless System R&D, Marvell Semiconductor,
Santa Clara, CA, USA

E. L. Turner
Department of Electrical Engineering and Computer Science,
University of California Berkeley, Berkeley, CA, USA

1 Introduction

The polar format algorithm (PFA) used for image formation in synthetic aperture radar (SAR) is computationally demanding and data-intensive [1, 2]. Its realtime constraints and low-power requirements make it a promising target for advanced power-saving designs. On the other hand, its overall system performance is often defined by the limited memory bandwidth as well as the high cost of memory access. As a potential solution to address these challenges, the application-specific logic-in-memory (LiM) computing paradigm and its design methodology [3, 4] is proposed to move simple computation directly into the memory, and minimize the data movement from memory to the processors for superior energy efficiency (see Fig. 1).

This idea stems from recent studies of sub-20 nm CMOS design, which indicate that memory and logic circuits can be implemented together using a small set of well-characterized pattern constructs [5, 6]. Our early silicon experiments in a commercial 14 nm SOI CMOS process demonstrate that this construct-based design enables logic and memory 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 14 nm 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 therefore facilitate *smart memory compilation*.

Advances in this chip design methodology gives rise to the application-specific LiM computational paradigm, which moves part of a program's computation directly into the memory but keeps the usual memory interface. It is

Published online: 21 December 2012

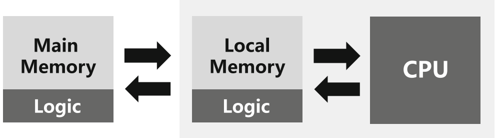


Figure 1 Logic-in-memory computing paradigm: application-specific logic for localized computation is hidden behind a memory abstraction.

easy to program, as all computational operations are hidden behind the memory abstraction. LiM builds on the idea of earlier processing in memory [7], however, puts only simple logic instead of actual processing cores right into the memory structures. Moreover, it requires application-specific logic to reach the desired energy savings. Thus, it is more specialized than the processor-in-memory idea [7, 8]. On the architectural level, the logic-enhanced memories look like normal memories to the CPU, but perform extra (and cheap) operations on the stored data before returning the requested data item to the CPU.

Design automation is required for handling the increased complexity of memory-logic-mixing hardware accelerators and the intricacies of cutting edge and next-generation silicon technology. Physical implementation of our logic and memory-mixing hardware is enabled by the *smart memory compiler* [5, 6]. Further, we build application-specific high-level design tools using the Genesis2 design tool [9, 10]. The combination of these tools enables designers to perform design space exploration at reasonable effort to optimize their designs for energy budgets, image reconstruction quality, and performance.

The major restriction of logic-in-memory is that only localized neighborhood data access can be implemented efficiently, and that algorithms requiring stride-like data access patterns (e.g., the fast Fourier transforms, FFT) are prohibitively expensive to implement. Therefore, algorithms need to be adapted to match the constraints of the logic-in-memory paradigm.

Related Work Synthetic aperture radar is essentially "taking a photo with radar" where a plane's flight path synthesizes a large antenna. A radar mounted on a plane sends repeatedly pulses to the scene patch and records the reflections, rotating the antenna to aim at the same scene center for all pulses. The image is formed by computing the inverse 2D FFT of the recorded data. However, the data is sampled on a polar grid, and the PFA needs to first convert these polar samples into rectangular samples (i.e., polar-to-rectangular re-gridding), so that a standard FFT can be applied for image formation. Without this conversion, a computationally infeasible non-uniform Fourier transform would have to be applied [1]. The polar-to-rectangular conversion is often done separably (first processing all rows and then all columns of the data), for example using FFT-based upsampling followed by picking the nearest neighbor to the actual grid points of interest [2, 11]. The reliance on FFTs makes this approach computationally intensive, moreover, it requires non-local computation due to the well-known FFT data access pattern. An algorithm for logic-in-memory cannot rely on FFTs but requires local computation, thus we need to develop a localized variant of polar-to-rectangular re-gridding.

There are other relevant hardware accelerators for gridding algorithms. For example, [12, 13] present an FPGA accelerator for gridding in Non-uniform FFTs. Their work targets a broader set of applications, regardless of the data acquisition method, i.e., the sampling of source points can be completely arbitrary. In contrast, we focus on the image re-gridding from polar format to rectangular format; specifically with large radian spatial frequency and small coherent integration angular intervals. The prior knowledge from the application allows us to build a dedicated hardware that is particularly optimized to our specific needs. On the other hand, their work demonstrated a complete system solution on an FPGA platform. The purpose of this paper is not to deliver a complete system solution but to implement the kernel part of the re-gridding algorithm to demonstrate the potential of LiM design methodology. Therefore, we narrow our scope to the on-chip data processing and storage.

While FPGAs and GPUs are also good alternatives as hardware accelerators to speed up compute-intensive sections of applications [14, 15], ASIC is still 10 to 100 times more power efficient than FPGA and GPU alternatives [16]. In addition, modern FPGAs contain "hard" blocks such as block memories whose functionality and sizes are fixed. They are hard to customize with fine granularity, which is the essential part of our approach. For example, [12] proposed a multi-port local memory (MPLM) to solve the limited memory bandwidth/port problem for the parallel pixel accessing. Our rectangular access smart memory architecture has the similar functionality as MPLM, however, we move one more step forward and realize the parallel data accessing by embedding "intelligent" functionality into the traditional interleaved multi-bank memory organization and allowing multiple memory subbanks to share one common memory periphery. In other words, we customized the traditional memory architecture in an unusual way to reduce the overhead that exists in the multi-banking memory systems. Our LiM approach provides a novel regular pattern constructs based ASIC solution targeting at sub-22 nm technology nodes, demonstrating the possibilities of re-designing algorithms and re-architecting the hardware to match the advanced technology capabilities and achieving dramatic performance improvements that was not possible with general purpose computing or configurable hardware computing.

Contribution The main contribution of this paper is the derivation of an algorithm for performing SAR polar format re-gridding interpolation in the LiM paradigm, and to provide the necessary design automation tool chain to implement our proposed algorithm in advanced silicon technology. We combine filtering, geometric transformations, and localized 2D interpolation to provide a virtual rectangular 2D memory address space that is overlayed on the polar grid and performs the necessary interpolation on demand. Enabled by this on-demand interpolation our system further provides partial image reconstruction, allowing for reconstructing both low-resolution thumbnails and high-resolution patches at considerably reduced energy cost.

This paper is an extended version of our previous papers that appeared in the proceedings of HPEC [3] and ICASSP [4]. While these previous papers mostly focus on the algorithmic side of our approach, this paper also presents the details of the hardware implementation and the design automation framework. More importantly, we show how to leverage the proposed design framework to co-optimize the algorithm, architecture and circuit design to achieve the maximum performance and energy efficiency.

2 Localized SAR PFA Algorithm

In this section we discuss our localized interpolation-based re-gridding algorithm that underlies our approach.

2.1 Local Interpolation Based Polar Reformatting

The measurements of the radar reflectivity function that are performed by the radar sensor during the plane flight are taken on partial polar annuli, which need to be converted to outputs on a Cartesian grid before FFT-based image formation. Assuming a signal of necessary smoothness, the points in the rectangular grid are similar to their neighboring elements of the Polar Annulus in both the range and cross-range dimensions. Given the high noise in radar data, We use simple local interpolations (e.g., nearest neighbor, bilinear or bi-cubic) to perform re-gridding, as opposed to the usual FFT-based upsampling. In Section 5 we show that this can be indeed done without significant loss of end-to-end accuracy.

To derive the relationship, we take bilinear interpolation for example and begin by superimposing a Cartesian grid on the Polar Annulus. Then, for an output point $P(x)$ in Cartesian space (depicted as red star in Fig. 2a–c), we find the coordinates (pulse number, sample number) of its corresponding neighboring elements in the Polar Annulus (original measurements), shown as four black points in Fig. 2a. We then compute the value of the $P(x)$ through interpolation, taking the weighted sum of its four neighbors, using their euclidian distance as weights. However, the direct computation requires complex nonlinear operations such as square root, arcus tangent, which are not suitable for the LiM paradigm.

Coordinate Transformation The main idea underlying our approach is to perform a coordinate transformation that converts the polar grid into a rectangular grid while the original rectangular grid is warped, and then perform interpolation in the transformed space. This allows us to apply standard 2D surface interpolation for polar data to rectangular data reformatting, which has the potential of being efficient in logic-in-memory as no transcendental function needs to be evaluated, neither for the coordinate transformation nor for the interpolation in the transformed space.

The mapping from Fig. 2a to b shows the first step in implementing the interpolation-based polar formatting. We first approximate the partial polar annuli as straight lines, making the full shape quadrilaterally tiled (Fig. 3). We then map the polar annulus (the polar grid on which the SAR data is collected) to a rectangular grid by using a four-corner image geometric mapping, specifically a perspective transformation [17]. The same perspective transformation is used to map the tentative output locations into the same new

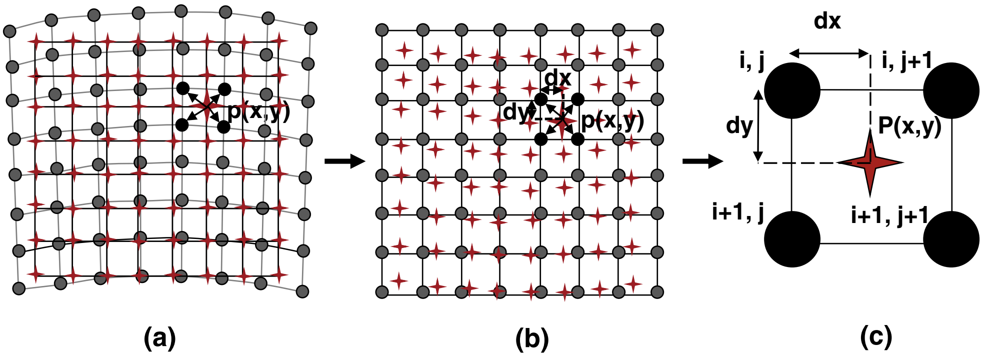


Figure 2 Localized polar-to-rectangular grid interpolation. **a** Interpolation in original coordinates, **b** interpolation in “warped” coordinates, and **c** bilinear interpolation in a square grid.

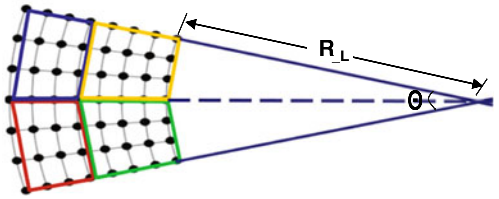


Figure 3 Image tiling for accurate geometric approximation.

coordinate system. After the coordinate transformation, the measurements lie on a rectangular grid, while the tentative outputs lie on a quadrilateral in the new coordinate system, see Fig. 2b. In other words, this mapping distorts the rectangular destination grid in the new coordinate system but preserves its distances to the original data points. The new x and y coordinates of the tentative output locations after mapping indicate the the locations of the corresponding neighborhood measurements and distances d_x and d_y to each of the neighborhood measurements. Then we use standard $2D$ surface interpolations to calculate the values of the tentative outputs from their neighborhood measurements and the interpolation weights. Figure 2c shows the example of the bilinear 2D surface interpolation that requires four neighborhood measurements.

Geometric Approximations Our localized grid interpolation is based on several geometric approximations. Firstly, as we mentioned, we approximate the polar annulus by quadrilateral tiles (Fig. 3) so that a simple quadrilateral-to-quadrilateral four-corner perspective geometry transformation can be used. Secondly, we assume that the measurement grids are evenly distributed on a rectangular grid after the transformation. These approximations could result in distortions in the resulting reconstructed image. As shown in Fig. 3, accurate approximation is achieved if the radian spatial frequency lower bound (R_L) is large enough and the coherent integration angular interval (Θ) is small enough, which is true for most SAR applications. Therefore, an effective solution is to tile the image into small enough parts and perform the geometry approximation on each tile. We tile the output image in the Cartesian grid and find the minimum subset of the polar annulus that contains the corresponding rectangular tile. The resulting distortion is smaller than the intrinsic distortion of perfect SAR image reconstruction.

2.2 SAR Image Partial Reconstruction

When reconstructing large data-set problems for small display devices (e.g., handheld devices) or for more detailed analysis, partial reconstruction would be preferable to prevent energy waste from processing all pixels and then using only a subset. Since our local interpolation-based scheme is reconstructing one pixel at a time in an on-demand fashion, partial reconstruction becomes feasible (see Fig. 4). However, since Polar Annulus is sampled in the Fourier space, this involves a series of digital signal processing operations across both the frequency domain and spatial domain. In the following, we will discuss two partial reconstruction modes that our approach supports: (1) *low resolution full-size image display*, and (2) *high resolution partial-size image display*.

Thumbnail Reconstruction In the first scenario, we get a quick overall view of the whole image without the fine-scale details (a thumb nail). This coarse reconstruction corresponds to multiplying the data in Fourier space (the original data) with a mask which attenuates the high frequency components. Only data elements that correspond to the low frequency components are interpolated and computations for high frequency components are omitted. A much smaller 2D inverse FFT can be used afterwards, saving a substantial amount of operations.

Zoom-in Reconstruction In the second scenario we reconstruct only a small portion of the image (however, at full resolution). This can be seen as multiplication by a mask in the spatial domain zeroing everything but the region of interest, or equivalently, as decimation filtering in the frequency space [18]. Filtering is necessary for image anti-aliasing and the filter decimation factor corresponds to the proportion of the image area to be reconstructed in space. Using Fourier identities we can reconstruct sub-patches of an image at arbitrary position with arbitrary size. In the implementation we rely on the combination of a CIC (cascaded integrator-comb) and short FIR (finite impulse response) filter for decimation. The CIC filter requires no multiplications and its simple hardware implementation can be easily integrated with the logic-in-memory interpolation, however, accuracy requires us to use some FIR filtering.

Computational Cost Savings In standard polar formatting algorithms using FFT-based upsampling for re-gridding, grid interpolation is the the most computationally intensive portion as it involves two FFTs per segment/secant for each range/crossrange line [2, 11]. In our local interpolation approach, all the interpolation related FFT/IFFT operations are avoided. The proposed grid interpolation has economical hardware implementations. Moreover, these operations are computed locally in the memory and therefore consume much less energy compared with in-CPU computing. For partial reconstruction, additional in-memory computation for decimation filters are required. However, the chosen CIC filter only involves eight adders and eight storage registers for any decimation factors. Under partial reconstruction,

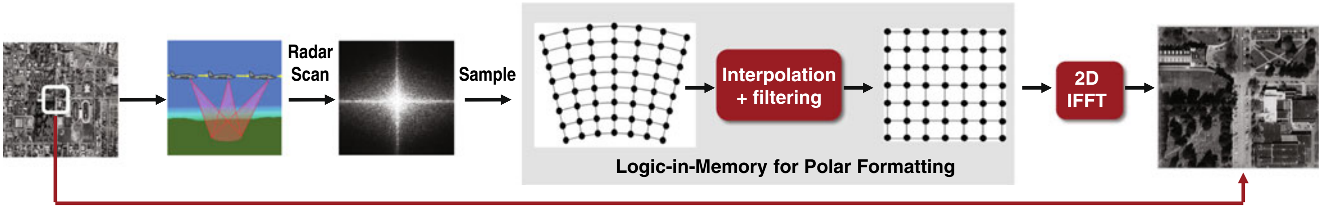


Figure 4 SAR partial image reconstruction.

inverse 2D-FFT size is reduced which saves the unnecessary operations and thus energy. Thus, our approach has a huge potential for operations and energy savings. We will evaluate practically achievable savings in Section 5.

3 Hardware Implementation

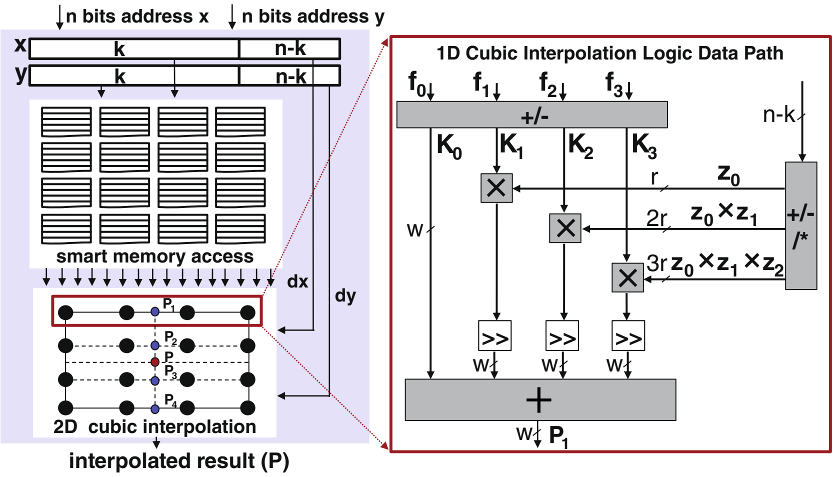
In this section we will describe the hardware implementation details of our proposed LiM-based SAR polar reformatting and partial reconstruction algorithm.

3.1 Interpolation Memory Implementation

The core operation in our approach is 2D interpolation (bilinear, biquadratic, bicubic), which is used after the perspectively transformation to calculate the values of the tentative outputs from the neighboring measurements and the interpolation distances in the transformed coordinate system. To implement the interpolation operations efficiently, we design a LiM block called *interpolation memory*. Interpolation memory holds function values at evenly spaced, non-contiguous memory addresses, and the integrated logic performs polynomial interpolation operations on each read reference for locations that do not hold data. Thus, these interpolation memory blocks contain a seed table that stores the known function values, and compute "in-between" values on the fly. It has a larger memory read address space than write address space. Interpolation memory is a very general LiM building block that can benefit many signal and image processing algorithms [17, 19–21].

Memory Access Logic In the left part of Fig. 5, we show the hardware structure of a 2D cubic (bicubic) interpolation memory. Assuming the array of polar format measurements has the size of $2^k \times 2^k$ and the interpolation distance has r-bit resolution. After the perspective transformation, the resulting x-coordinate and y-coordinate of the tentative outputs in the new coordinate system serve as the n-bit input addresses here. Given the input addresses, the *2D interpolation memory* returns the corresponding pixel value at that location, which is actually interpolated from its neighboring measurements in the original polar grid. Internally the input address is split into two parts. The higher k bits are used to address the measurement points in the original polar grid. And the lower $r = n - k$ bits are used to specify the distances between the evaluated output point and its nearest neighborhood measurements. The output pixel values are the weighted approximations of the neighborhood measurements, and the weights are set by interpolation distances. The number of nearest neighborhood memory references to be considered is determined by the interpolation order. Power-of-2 indexing mechanism is applicable for most interesting problems, and it largely simplifies the hardware implementation.

Figure 5 Interpolation Memory Architecture: n bit read address space and k bit write address space (i.e., seed table size); the "in-between" values are approximated from 16 neighboring memory references on the fly.



Interpolation Logic In terms of interpolation operation, 2D interpolation is separable and can be broken into multiple 1D interpolation in both orthogonal axes. For example, the 2D cubic interpolation in Fig. 5 can be separated into four horizontal 1D cubic interpolations and one vertical 1D cubic interpolation (or vice versa). In the right side of Fig. 5, we illustrate the datapath of a 1D cubic interpolation operation. We use Newton's divided differences interpolation polynomial since it is easy to realize in hardware and amenable to be parameterized [22]. The d^{th}-order function value $P_d(x)$ is calculated from its neighborhood pixels values $f(x)$ at points of $x_0, \ldots, x_d$:

$$P_d(x) = k_0 + k_1 \cdot (x - x_0) + \ldots + k_d \cdot (x - x_0) \ldots (x - x_{d-1}).$$

For $i \in [0, d]$, $k_i = f^{(i)}(x)$ is the ith order divided difference of $f(x)$, and the computation of k_i in hardware for integer data types only involves additions and shifts. $z_i = x - x_i$ are so-called the interpolation distances, which are determined by the lower r bits of the input address. The computational complexity, and the overall hardware cost is proportional to the interpolation order. The bit widths of the data path can be precisely specified so as not to implement excessive bits, and not to introduce additional error. This approach can be cheaply implemented in logic-in-memory, both for integer and floating-point output. This insight is a crucial enabling step for our logic-in-memory SAR variant.

3.2 Rectangular-Access Smart Memory

The interpolation operation requires to access multiple consecutive elements in a 2D data array stored in SRAM within a single cycle. Figure 6a and b show the access patterns for the bilinear and bicubic interpolation memories. For example, a 4 × 4 rectangular memory access is needed for the bicubic interpolation. Larger block-size access is required in the implementation of parallel image processing, i.e., to construct multiple pixels in parallel. It is observed during our experiments that the reconstruction of adjacent pixels actually share some of the neighborhood measurements. As shown in Fig. 6c, to compute the adjacent 6 × 6 pixels with bilinear interpolation, all the required neighborhood measurements are clustered within the block of neighborhood 8 × 8 polar grid array. Therefore, a 8 × 8 rectangular access memory is required to output all the 8 × 8 measurements to the processor and then the computation of the samples in the 6 × 6 block can be performed in parallel.

Decoder Sharing for Multiple Memeory Banks Traditionally, these parallel memory accessing is 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. Using logic-in-memory it is possible to build multi-bank memories that share parts of the decoder logic to exploit the known access pattern.

We exploit the fact that we always read a constant number of consecutive elements per cycle for each interpolation. 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 the a customized multi-banking SRAM design topology [23], which provides around 50 % area and power savings compared with the traditional multi-banking memory design. However, the design of such customized memory requires careful circuit design, sizing and layout, which is a significant design cost if it cannot be automated.

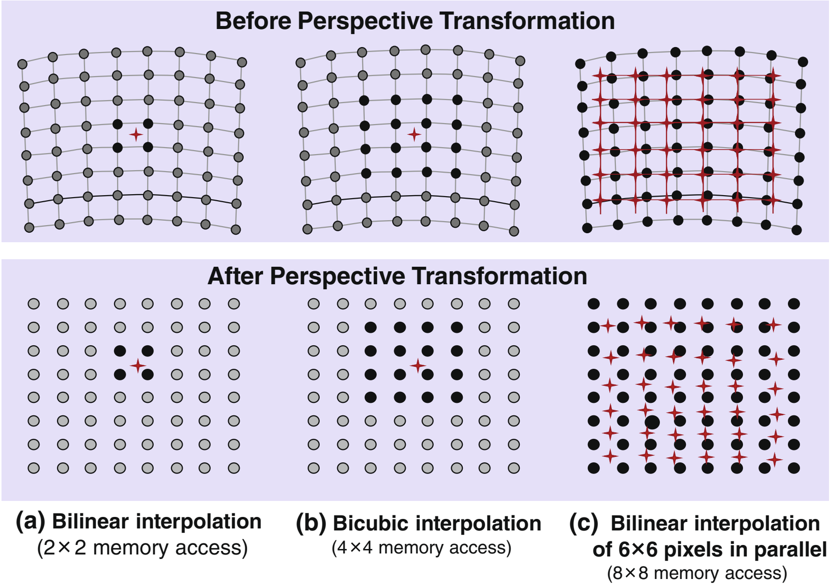

Before Perspective Transformation
After Perspective Transformation
(a) Bilinear interpolation
(2×2 memory access)
(b) Bicubic interpolation
(4×4 memory access)
(c) Bilinear interpolation
of 6×6 pixels in parallel
(8×8 memory access)

Figure 6 Memory Access Pattern: *gray array* represents the stored function values and the black points are the nearest neighbors to be accessed for the interpolation of the non-stored function values (*red stars*).

Single Cycle Rectangular Block Access We define the functionality of memory to support one-clock-cycle rectangular access of $2^a \times 2^b$ data points from a $2^m \times 2^n$ $2D$ data array. The input of the memory system is the top-left coordinate of the accessing rectangular block ($x_{[m-1:0]}$, $y_{[n-1:0]}$) and the outputs are all the data point inside the rectangular block. For bicubic interpolation, we have $a = b = 2$.

To support one-cycle consecutive access of 2^a data points in x dimension and 2^b data points in y dimension, the parameterized memory is divided into 2^a memory blocks; and in each block, there are vertically parallel 2^b memory banks. To control the memory block aspect ratio, we let each word of a memory bank (bank word) holds 2^c data points, therefore a block word contains 2^{b+c} data points. The 2D data array first distributes its 2^m data rows into 2^a memory blocks row by row (e.g., block i holds row[i], row[$2^a + i$], row[$2 \cdot 2^a + i$], etc.). All the 2^a memory blocks have the same structure. Figure 7 shows the organization of block 0 when $m = n = 6$, $a = b = 2$, $c = 2$.

Implementation 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 [23]. 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. 7, 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 bank 0 and $WL[0]$ for the other three banks. After the column MUX, block 0 outputs data series of '8−5−6−7', which are then reordered to be '5 − 6 − 7 − 8'. With some simple logic for data reordering, the smart memory outputs the required $2^a \times 2^b$ data points in order simultaneously. As shown in Fig. 7, the distributions of address bit to each memory component are parameterized. By specify these parameters, the resulting memory architecture can be precisely determined.

Compared with the conventional multi-banking memory design, the amount of memory bank periphery circuits is reduced from 2^{a+b} to 2^a. As is observed in Fig. 7, the resulting memory architecture has the embedded logic gates (e.g. the AND gates) tightly integrated with the memory cells, and each logic gate communicates with its local memory cells. The hardware synthesis of these novel smart memories will be presented in Section 4.

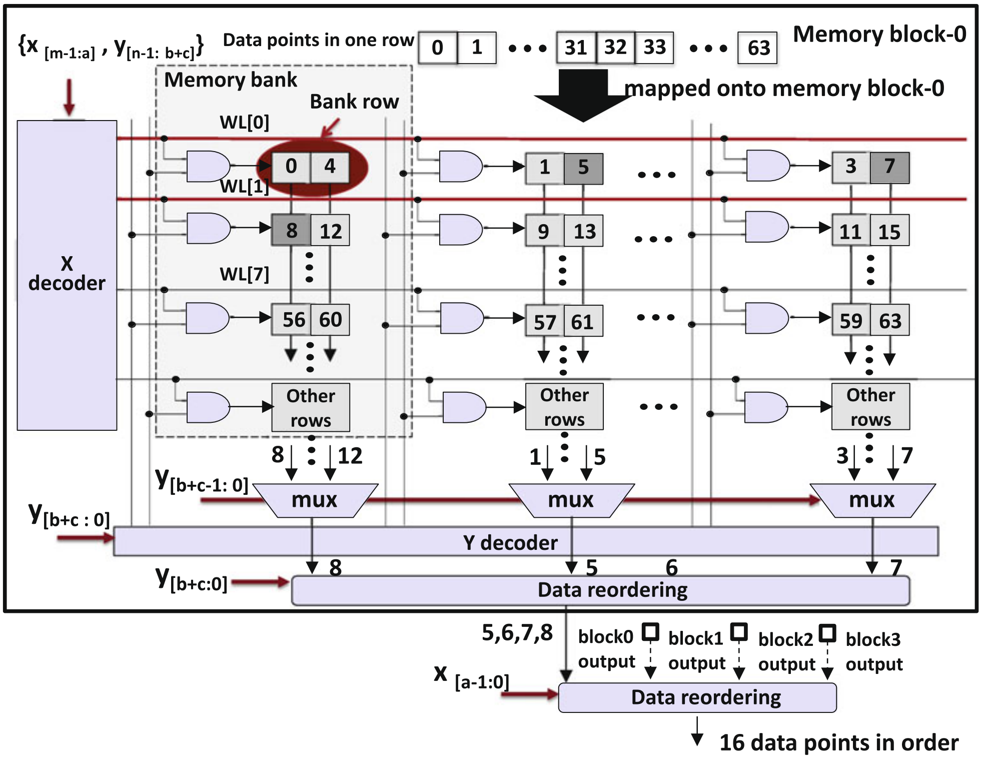


Figure 7 Customized Rectangular Access Memory: customized memory periphery design allows parallel memory banks to share the x-decoder.

3.3 Image Perspective Transformation

Perspective Transformation Another core component of our SAR variant is the perspective transformation. We use it to map both of the original polar measurements and tentative rectangular outputs to the new coordinate system such that the measurements lie on a rectangular grid, while the tentative outputs lie on a quadrilateral. After this mapping, a standard 2D interpolation can be used for the image reformatting. The perspective transformation function is given by

$$[x', y', w'] = [u, v, w] \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}, \tag{1}$$

where $x = x'/w'$, $y = y'/w'$. The coefficients a_{ij} of the transformation function are determined by establishing the correspondences between four corners of the original polar annuli and new coordinate grids [17]. Then the same transformation function is used to map each point (u, v) of the tentative rectangular output to the point (x, y) in the new coordinate system from the following mapping functions,

$$x = \frac{a_{11}u + a_{21}v + a_{31}}{a_{13}u + a_{23}v + a_{33}} \quad \text{and} \quad y = \frac{a_{12}u + a_{22}v + a_{32}}{a_{13}u + a_{23}v + a_{33}}. \tag{2}$$

Division As we can see, the perspective transformation mostly involves simple arithmetic logic like additions and multiplications. Although the division operation is also required, we observed that the denominator is a linear function of the u and v coordinates. Therefore, for the items of $1/(a_{13}u + a_{23}v + a_{33})$ and $1/(a_{13}u + a_{23}v + a_{33})$, we can first evaluate their values at the four corners, that is, $(u = 0, v = 0)$, $(u = 0, v = 1)$, $(u = 1, v = 0)$, $(u = 1, v = 1)$, and then the values at other locations can be computed by a bilinear interpolation from the four corners. This way we convert the division to a bilinear interpolation and a multiplication leading to negligible accuracy loss.

Implementation in LiM The whole geometric transformation logic is embedded into the memory boundary together with the 2D interpolation logic. From the user's point of view, the resulting LiM block is a normal memory that stores the pixel values at rectangular grids and returns the requested pixel value on command. However, internally it actually stores the polar grid measurements in the physical memory and has the application-specific logic computation embedded in the memory boundary. Therefore, LiM block provides a virtual rectangular 2D memory address space that is overlaying the polar grid and performs the necessary logic operation inside the memory abstraction.

3.4 Frequency Filter

The most important arithmetic operation for SAR image partial reconstruction is the filtering, which enables us to implement partial image reconstruction for both low-resolution thumbnails as well as high-resolution scene patches in logic-in-memory. We rely on simple Fourier transform identities to translate phase shifts in frequency space to time-domain displacements [18]. Using this identities we can zoom at any region of interest.

CIC and FIR Filters A wide range of decimation factors is required for different problem size with different display resolution and zoom factors. Straight-forward implementation of finite impulse response (FIR) filters becomes too expensive for long tap lengths that are required to maintain accuracy. In order to include the filters into the logic-in-memory device, it is required that hardware implementation is as simple as possible. A finely-tuned combination of FIR and cascaded integrator-comb (CIC) filters can be implemented very efficiently in logic-in-memory. After evaluating the accuracy-cost decimation filter design space, we use FIR polyphase filter for low decimation factors (for instance, 2 or 4) and use CIC filters for high decimation factor (for instance, 8, 16, 32, 64, or 128). CIC filters are chosen because no multipliers and no intermediate storage are required, and the same filter design can easily be used for a wide range of decimation factors by adding an additional scaling circuit and minimally changing the filter timing. However, a CIC compensation filter, which is usually implemented as FIR inverse sinc filter is usually required to compensate the non-flat passband and wide transition-region of high decimation factor CIC filters. It is performed after the decimation so that there is no much additional cost.

4 Design Automation Framework

We now discuss the design trade-off space and our design automation tools.

4.1 Design Trade-off Analysis

The SAR image formation process requires the choice of a series of problem parameters, and each parameter setting leads to a different hardware implementation. In addition, as the major components of the system, both interpolation and filtering are trade-off problems in terms of performance/accuracy/cost.

Accuracy vs. Interpolation Order Our 2D interpolation memory is based on polynomial interpolation for numerical function evaluation [19, 20]. We only consider up to the 3rd interpolation order, that is, bilinear ($d = 1$), biquadratic ($d = 2$), and bicubic ($d = 3$). The interpolation order (d) together with physical memory size (2^k) determine the interpolation accuracy (binary precision bits, w). Numerical analysis shows that for any function $f(x)$ that has $d + 1$ derivatives, $d+1$ additional precision bits w of the computed $P(x)$ are obtained for each additional physical address bit k for interpolating order d [19]. The tradeoff among these parameters is shown in Eq. 3, in which e is the error bits in the precision bits w that are tolerated by the application.

$$(w - e) \propto (d + 1)k \tag{3}$$

Equation 3 gives rise to a design space involving data precision bits, interpolation accuracy, interpolation order, and interpolation resolution. These choices further lead to different memory/logic area and power costs for a desired accuracy.

Filter Design The parametrization of filter specifications also give rise to a large design space. For example, the transition region of a non-ideal filter will result in added distortion at the image edge. Therefore, the narrower the transition region the better the edge quality, but the higher the filter degree, thus higher the hardware cost. In essence, we can reconstruct a slightly larger image and disregard the boundary region to enable the utilization of a lower-quality (and computationally cheaper) filter. In our implementation, we use an region-of-interest (ROI) parameter to specify the ratio of the used image center area compared to the overall image area, which determines the transition-region of the filter (rolloff factor).

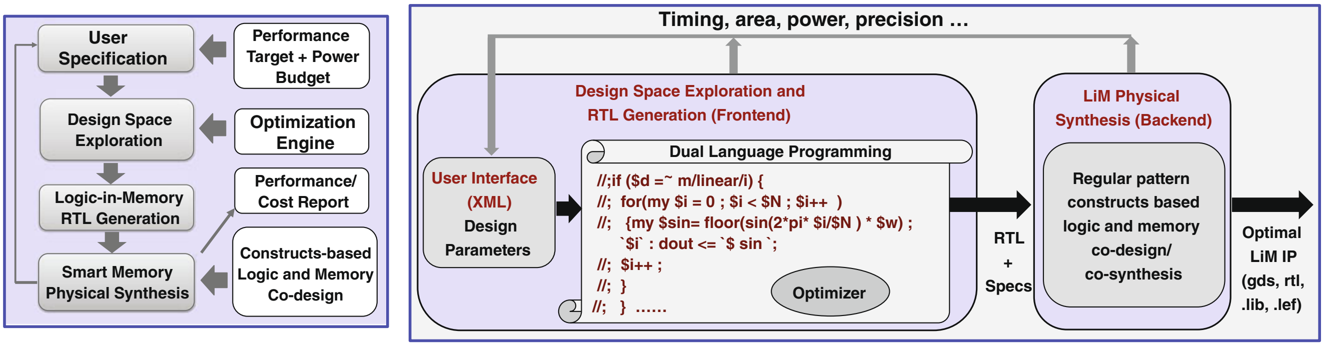
This shows that different design decisions will result in different tradeoffs. The combination of these design choices constitutes a huge design space. Further, exploring the design tradeoff space requires customized memory designs, which are traditionally prohibitively expensive. Thus, a strong design automation tool is required to make the hardware synthesis feasible.

4.2 LiM Design Framework

We have developed a *design generation and design space exploration tool* for the LiM design. The complete design framework structure is shown in Fig. 8.

Our design framework provides designers with a graphical user interface to select application functionality and parameters and then generates synthesizable RTL designs for a specified functionality. Free or un-specified parameters can be optimized by the system. A designer then evaluates the obtained designs and can explore the design space and optimize the design for the application by varying the parameters. The design framework consists of the tool frontend which is built from the architectural chip generation infrastructure *Genesis* [10, 24] and the tool backend that is built from the pattern-construct based smart memory compiler [5, 6].

Genesis Chip Generator The frontend of the design tool chain is a standalone design tool framework named Genesis [10, 24, 25]. It is responsible for application interfacing, design optimization and efficient RTL generation. Genesis is a framework that simplifies the construction of highly parameterized IP blocks. Unlike existing HDLs that calcify any existing flexibility at instantiation, Genesis leaves low level optimization "knobs" free even after aggregation into bigger IP blocks, allowing them to be set and optimized later in the design process. To achieve that, Genesis enables the hardware designers to simultaneously code in two interleaved languages when creating a chip module: a target language (SystemVerilog) to describe the behavior of hardware and a meta-language (Perl) to decide what hardware to use for given specs (see the left part of Fig. 8b).


User Specification
Performance Target + Power Budget
Design Space Exploration
Optimization Engine
Logic-in-Memory RTL Generation
Performance/ Cost Report
Smart Memory Physical Synthesis
Constructs-based Logic and Memory Co-design
Timing, area, power, precision ...
Design Space Exploration and RTL Generation (Frontend)
User Interface (XML) Design Parameters
Dual Language Programming
Optimizer
RTL + Specs
LiM Physical Synthesis (Backend)
Regular pattern constructs based logic and memory co-design/ co-synthesis
Optimal LiM IP (gds, rtl, .lib, .lef)

```
//;if ($d =~ m/linear/i) {
//;  for(my $i = 0 ; $i < $N ; $i++  )
//;   {my $sin= floor(sin(2*pi* $i/$N ) * $w) ;
      `$i` : dout <= `$ sin `;
//;  $i++ ;
//;  }
//;   }  ......
```

(a) LiM Design Flow

(b) LiM Design Framework

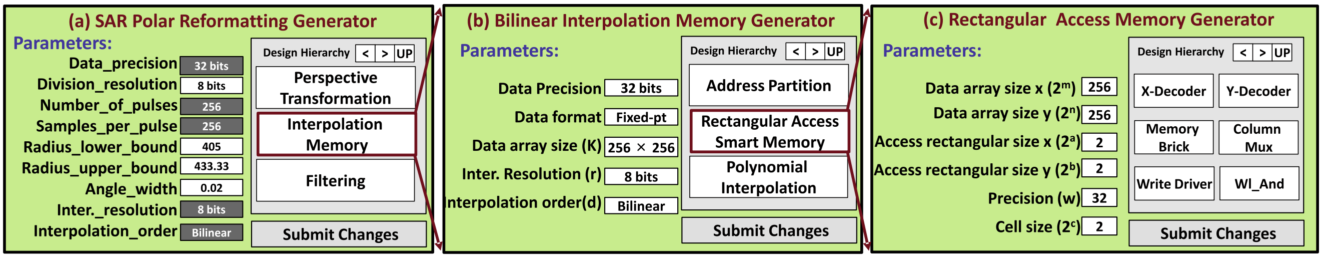
Figure 8 LiM design framework.

The net result is that Genesis enabled us to design an entire family of LiM designs, all at once. After the parameterized design was complete, there is still the matter of controlling all the parameters and they can be made explicitly by the user or automatically by optimization tools. The generator mechanism provides a standardized way, via an XML form, for optimization tools to make design decisions for various given parameters throughout the design hierarchy. Genesis classifies parameters into three groups. First, an inherited or constrained parameter is one that is inherited from, or constrained by decisions made in other modules in the design hierarchy (e.g., interface bit width). The second type of parameter is the free parameter-parameters whose values can be freely assigned by the system and it is best to allow an optimization engine to set the value that maximizes performance under a given power or area constraint. A third type of parameter is the architectural parameter that changes the function or the behavior of the module. These are the parameters that must be set by application designer. An inherit priority rule in Genesis determines the assignment/overwritten policy of parameter values.

Smart Memory Compiler The automated design framework discussed so far is capable of mapping application specifications to optimized RTL. Equally important, a smart backend of the design tool chain is required to efficiently co-synthesize logic and memory (the right part of Fig. 8b). Generic SRAM compilers enable automatic SRAM IP creation based on user specification, but they "compile" memory blocks from a set of pre-determined SRAM hard IP components (e.g., bitcells and peripheral circuits). This compilation strategy not only limits the possibility of application-specific customization but also hinders comprehensive design space exploration, leading to a sub-optimal IP. We have been exploring opportunities for synthesis (not just compilation) of customized logic-in-memory blocks in a commercial sub-20 nm CMOS process and successfully developed a *smart memory* design and synthesis methodology. The smart memory is composed of a group of Memory Arrays, peripheral circuits and application specific random logic implementing a special function. The major step in the design of smart memory is to co-optimize logic, memory and process. In order to predictably print the tight pitches in extreme nodes, the design rules require an extremely regular and gridded design making logic and memory co-design easier, for that we have created a bitcell compliant area-efficient unidirectional logic fabric. This methodology allows to remove any distinction between pushed memory design rules and logic design rules. Therefore, customized memory periphery is synthesized using lithographically compliant unidirectional standard cells which can be mapped together with memory to a small set of pre-characterized layout pattern constructs [5, 6]. Lithographic compliance between the co-designed logic and memory ensures sub-20 nm manufacturability of LiM circuits.

The architectural frontend and physical backend are combined to build an end-to-end LiM design framework [3, 4, 26]. 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".

User Interface Illustration In Fig. 9 we show the user interface of our LiM-based SAR image reformatting design tools. The design parameters are listed in the left panel and module structure is shown in the right panel. Functional parameters (e.g., *Data precision*, *interpolation order*) are set by the application designer. In our example in Fig. 9, the selected operation is the reformatting of a 256 × 256 polar grid array to a rectangular grid array, using bilinear interpolation. The interpolation resolution is set to be 8 bits. To achieve this, a 2D bilinear interpolation memory with a 256 × 256 physical memory size and a 2 × 2 rectangular access size is required, which is a separate LiM design tool we built. It here acts as a sub-module of the image reformatting tool. Constrained by the higher-level image



Parameters:
Data_precision 32 bits
Division_resolution 8 bits
Number_of_pulses 256
Samples_per_pulse 256
Radius_lower_bound 405
Radius_upper_bound 433.33
Angle_width 0.02
Inter._resolution 8 bits
Interpolation_order Bilinear

Parameters:
Data Precision 32 bits
Data format Fixed-pt
Data array size (K) 256 × 256
Inter. Resolution (r) 8 bits
Interpolation order(d) Bilinear

Parameters:
Data array size x (2^m) 256
Data array size y (2^n) 256
Access rectangular size x (2^a) 2
Access rectangular size y (2^b) 2
Precision (w) 32
Cell size (2^c) 2

Figure 9 Design framework user interface.

reformatting tool, its parameters are shown in right part of Fig. 9b. This interpolation memory contains a second-level sub-module: a 2 × 2 rectangular access memory for supplying 2×2 block pixels to its higher level bilinear interpolation memory module. When satisfied with the parameters, the user simply clicks the “Submit Changes” button, and the tool will start to run to generate the dedicated hardware description in Verilog.

As seen in the example, we are building a LiM tool that is hierarchically composed from lower-level LiM design tools. All of these submodules in the designs provide users the hierarchical graphical tools to design instances of the algorithm with the capability of exploring the design space to trade off cost and performance.

5 Experimental Results

In this section we evaluate our logic-in-memory based SAR implementation for accuracy, performance, as well as computational and energy cost. We use our design tool to automatically synthesize the hardware for measurement and build an architectural model to simulate the algorithm.

Consecutive Access Smart Memory Evaluation The smart rectangular access memory is a core component, which we first evaluate in isolation. In Fig. 10a, we compare the hardware cost of rectangular access memory (smart memory) to a traditional multi-banking memory design (dumb memory) in terms of power-delay-product. Both designs have the same functionality to read out 2×2 consecutive memory elements in one clock cycle. We observe that the smart memory achieves around one order of magnitude savings. Figure 10b compares the smart memory area and dumb memory area, for a 128 × 128 size image divided in one tile, four tiles and 16 tiles. The image is loaded into the chip and to be processed one tile after tile. As we can seen, the on-chip local memory size is decreasing proportionally with the increase of the tile number. We also plotted the corresponding computational logic area cost. As can be observed, the logic area is relatively small compared with memory area, especially for the smaller tile numbers (larger tile sizes). This proves that the on-chip system is dominated by the memory area. The dynamic power and leakage power comparison results are shown in Fig. 10c and d. The leakage power savings of the smart memory is not as large as the corresponding dynamic power saving. Smart memory is designed to save the overhead cost of the periphery circuit, but has the same-size memory cell array, and the latter is the major consumer of the leakage power.

Accuracy and Hardware Cost Evaluation We next compare the accuracy of our local interpolation (nearest-neighbor, bilinear and bicubic) SAR algorithm to the conventional FFT upsampling based approach. We simulated a randomized radar scene of point targets and performed re-gridding using each interpolation method, see Fig. 11. As the entire process is linear and any image is a super-position of point-targets, analysis of point-targets is sufficient to emulate a real-world SAR scene. A reference *gold standard* is included that is based on the computationally infeasible non-uniform inverse FFT that has a closed-form solution for point targets. We simulated 1000 randomized scenes and Fig. 12 shows the statistical analysis of the mean square error (MSE) distribution for each method relative to gold standard method. For each trial, the MSE across all pixels is computed for that image compared to the gold standard technique. The x-axis on each figure of Fig. 12 represents the mean square error (MSE). The y-axis is a count of image trials that has this magnitude of MSE. Some trials

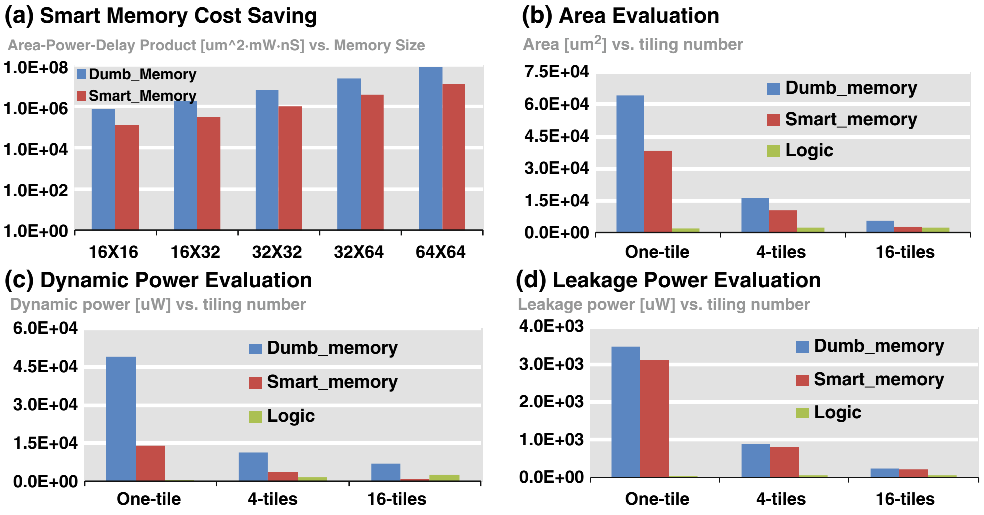
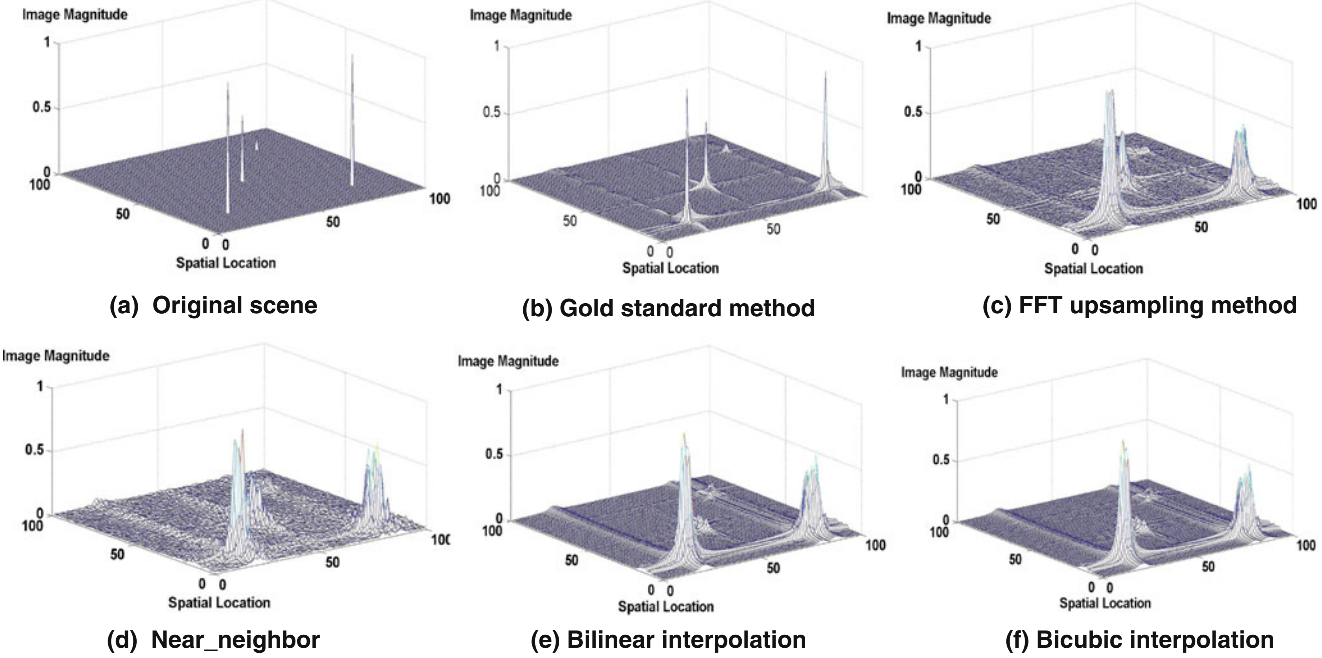


Figure 10 Smart memory cost evaluation.



(a) Original scene

(b) Gold standard method

(c) FFT upsampling method

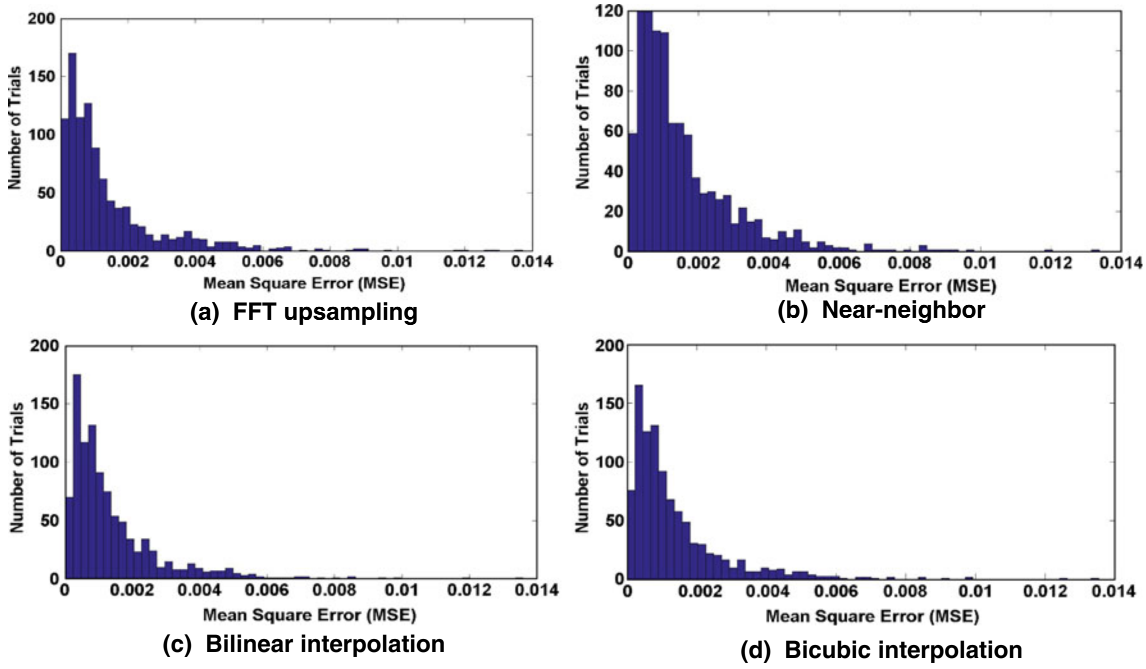
(d) Near_neighbor

(e) Bilinear interpolation

(f) Bicubic interpolation

Figure 11 An original and five reconstructed point target scenes.

have accuracy that are really close to the gold standard, and thus have smaller errors, so they contribute more to the counts on the left-hand side of these histograms. A trial whose image has a higher MSE contributes to the counts on the right-hand side of these histograms. We see that the distortions caused by the traditional FFT upsampling based approach and the local bilinear and bicubic interpolation methods are statistically indistinguishable while the near-neighbor approach is shown to be relatively inferior. Thus, in these SAR settings, using local bilinear or bicubic



(a) FFT upsampling

(b) Near-neighbor

(c) Bilinear interpolation

(d) Bicubic interpolation

Figure 12 Mean square error comparison.

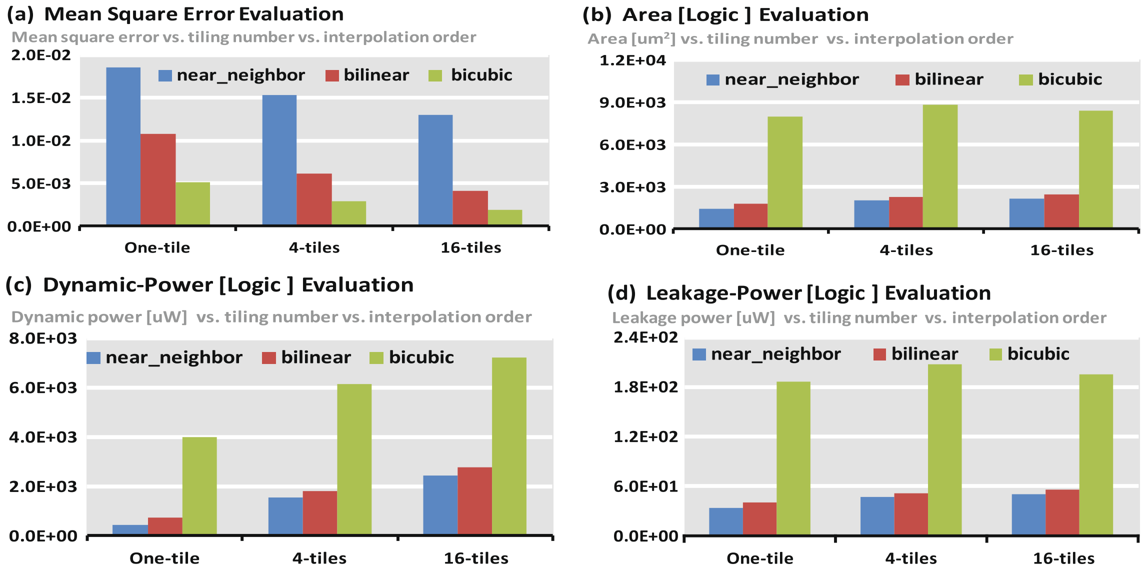


Figure 13 Design tradeoff evaluation.

interpolation for re-gridding does not result in an accuracy loss relative to FFT based upsampling. In Fig. 13a we vary tile numbers and interpolation complexity. As expected, we see the MSE decreasing for larger tile numbers and higher interpolation order. However, as we can seen from Fig. 13b to d, the area and power consumption of the computational logic is also increasing for higher interpolation order. On the other hand, the number of tiles does not have huge impact on the hardware cost of the computational logic. The reason is that when we divide the image into more smaller tiles and process one small tile each time, the bit-precision of data path (e.g., memory address) is decreasing which saves the hardware cost. However, as the processing for different tiles has different geometry related design parameters, it costs extra control logic to configure the hardware at the beginning of the processing based on different tile indices.

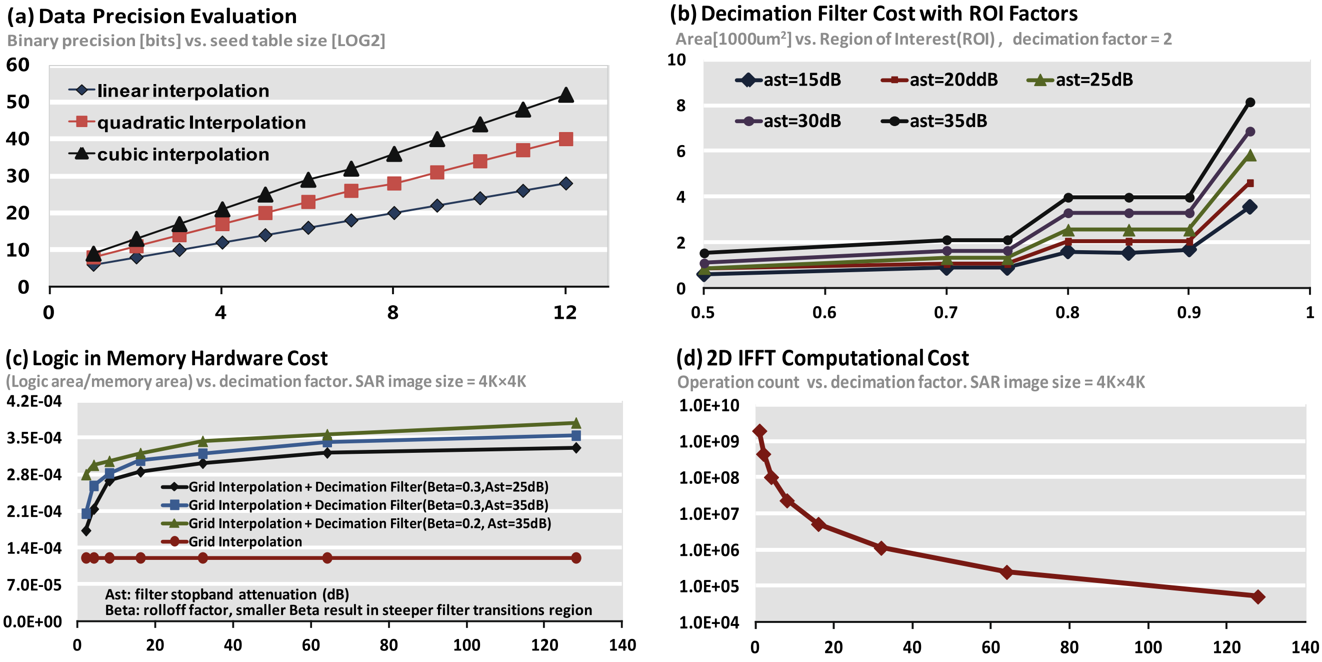


Figure 14 More experimental results.

Figure 14a shows the design trade-off of the interpolation memory in terms of the bit precision, interpolation order and the size of the seed table that stores the measured data points. Quadratic interpolation is also included for the completeness of the discussion, though it is actually not used in our design. As we can see, the accurate data precision bits (y axis) is proportionally increasing with the increasing of the physical memory address bits (x axis) for all the linear/quadratic/cubic interpolation methods. Also as expected, higher order interpolation method achieves better accuracy for the same seed table size. Figure 14b shows the decimation filter area with different region-of-interests (ROIs) and different filter stopband attenuation (ast). The ROI is defined as the ratio of the area of the image centric subset that needs to be accurately reconstructed compared with the overall image area. As expected, either higher ROI or higher ast indicates higher image quality but consumes more hardware cost. Figure 14c demonstrates the overall hardware cost of LiM blocks on a 14 nm commercial CMOS technology; the *y* axis values are the logic area relative to the memory area. The bottom curve shows the grid interpolation area for the full image reconstruction. For partial reconstruction, the top three curves add in the decimation filter area for three filter design specifications. We see that although the area for partial reconstruction increases slightly with the increase of the decimation factor, the *y* axis values are fairly small for all the design points. Thus, the logic area is negligible compared to the memory area for both full and partial reconstruction. In Fig. 14d, we observe that the number of arithmetic operations for the 2D IFFT is decreasing with the increase of the decimation factor in partial reconstruction. Figure 14c and d show that the decrease of operations through smaller IFFTs in partial reconstruction is not increasing the hardware cost substantially.

Energy Efficiency To evaluate the energy efficiency of our logic-in-memory SAR implementation, we simulate the whole SAR polar formatting algorithm in two variants: (1) we run the image reconstruction on a simple processor with a standard SRAM cache, and (2) we replace the cache with our logic-in-memory hardware that performs the interpolation in the memory and run a program reconstructing the image using this memory. We measure the energy consumption using the Wattch simulator, which is an architectural level power simulator based on SimpleScalar [27]. We model the logic-in-memory as direct-mapped on chip memory and scale the memory accessing energy by adding the normalized embedded logic cost from the hardware characterization results. We plot the results for both the conventional and logic-in-memory architecture at different problem sizes from 32 to 512. The results in Fig. 15 show multiple orders of magnitude of energy savings achieved by logic-in-memory especially for large data-size problems.

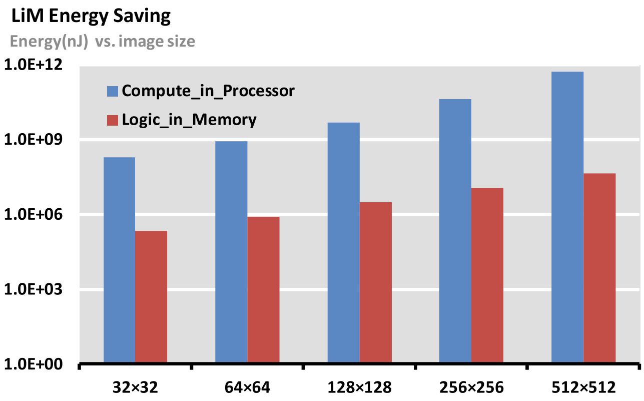


Figure 15 Energy consumption evaluation.

6 Conclusion

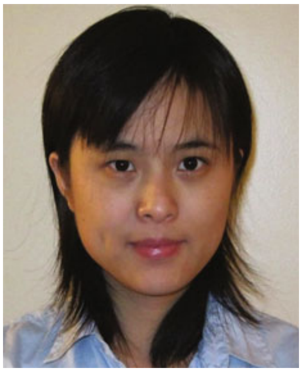
Advances in integrated circuit design enable the energy-saving logic-in-memory paradigm, which moves a part of the computation directly into the memory array. This cutting-edge design methodology requires redesign of well-known algorithms to match its performance characteristics. In this paper we derive a logic-in-memory variant of the polar formatting algorithm used in SAR image formation, and it has the accuracy comparable to the traditional FFT-based polar formatting algorithm but requires much less processing energy. Our algorithm further supports partial image reconstruction. We provide the necessary design automation tool chain to enable users to study the design trade-offs in the energy and performance space. Our experimental results show substantial energy saving at the same accuracy level.

Acknowledgments 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

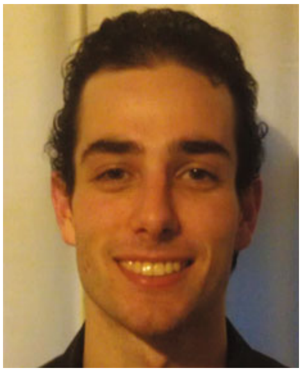
1. Carrara, W., Goodman, R., Majewski, R. (1995). *Spotlight synthetic aperture radar: Signal processing algorithms.* Artech House.
2. McFarlin, D., Franchetti, F., Püschel, M., Moura, J. (2009). High performance synthetic aperture radar image formation on commodity multicore architectures. In *SPIE.*
3. Zhu, Q., Turnerz, E.L., Bergery, C.R., Pileggi, L., Franchetti, F. (2011). Application-specific logic-in-memory for polar format synthetic aperture radar. In *HPEC.*

4. Zhu, Q., Bergery, C.R., Turnerz, E.L., Pileggi, L., Franchetti, F. (2012). Polar format synthetic aperture radar in energy efficient application-specific logic-in-memory. In *ICASSP*.
5. Morris, D., Rovner, V., Pileggi, L., Strojwas, A., Vaidyanathan, K. (2010). Enabling application-specific integrated circuits on limited pattern constructs. In *Symp. VLSI technology*.
6. Morris, D., Vaidyanathan, K., Lafferty, N., Lai, K., Liebmann, L., Pileggi, L. (2011). Design of embedded memory and logic based on pattern constructs. In *Symp. VLSI technology*.
7. Kogge, P.M., Sunaga, T., Miyataka, H., Kitamura, K., Retter, E. (1995). Combined DRAM and logic chip for massively parallel systems. In *Conf. advanced research in VLSI*.
8. Brockman, J.B., & Kogge, P.M. (1997). The case for processing-in-memory. *IEEE Computer*.
9. Shacham, O., Azizi, O., et al. (2010). Rethinking digital design: Why design must change. *IEEE Micro*, *30*(6), 9–24.
10. Shacham, O. (2011). *Chip multiprocessor generator: automatic generation of custom and heterogeneous compute platforms.* PhD thesis, Stanford.
11. Rudin, J. (2007). Implementation of Polar Format SAR Image Formation on the IBM Cell Broadband Engine. In *Proc. HPEC*.
12. Kestur, S., Park, S., Irick, K., Maashri, A., Narayanan, V. (2010). Accelerating the nonuniform fast fourier transform using FPGAs. In *FCCM*.
13. Kestur, S., Irick, K., Park, S., Maashri, A., Narayanan, V., Chakrabari, C. (2011). An Algorithm-Architecture Co-design Framework for Gridding Reconstruction using FPGAs. In *DAC*.
14. Sorensen, T., Schaeffter, T., Noe, K., Hansen, M. (2008). Accelerating the nonequispaced fast fourier transform on commodity graphics hardware. In *IEEE tran. on medical imaging*.
15. Che, S., Li, J., Sheaffer, J.W., Skadron, K., Lach, J. (2008). Accelerating compute-intensive applications with GPUs and FPGA. In *SASP*.
16. Kuon, I., & Rose, J. (2007). Measuring the Gap between FPGAs and ASICs. In *IEEE transactions on computer-aided design of integrated circuits and systems*.
17. Wolberg, G. (1990) Digital image warping (systems). IEEE Computer Society Press.
18. Lyons, R. (2004). *Understanding digital signal processing.* Prentice Hall.
19. Noetzel, A.S. (1989). An interpolating memory unit for function evaluation: analysis and design. *IEEE Transactions on Computers*, *38*(3), 377–384.
20. Meijering, E. (2002). A chronology of interpolation: from ancient astronomy to modern signal and image processing. In *Proceedings of the IEEE* (pp. 319–342).
21. Williams, L. (1983). Pyramidal parametrics. *Computer Graphics*, *17*(3).
22. Atkinson, K.A. (1988). *An introduction to numerical analysis.* Wiley.
23. Murachi, Y., Kamino, T., Miyakoshi, J., Kawaguchi, H., Yoshimoto, M. (2007). *A power-efficient SRAM core architecture with segmentation-free and rectangular accessibility for super-parallel video processing.* (Vol. 107 pp. 47–52): IEICE Tech. Rep.
24. Shacham, O., et al. (2012). Genesis2 chip generator interactive GUI: http://genesis2.stanford.edu/mediawiki/index.php/Main_Page.
25. Solomatnikov, A., Firoozshahian, A., Qadeer, W., Shacham, O., Kelley, K., Asgar, Z., Wachs, M., Hameed, R., Horowitz, M. (2007). *Chip multi-processor generator*.
26. Zhu, Q.L., Vaidyanathan, K., Shachamy, O., Horowitz, M., Pileggi, L., Franchetti, F. (2012). *Design automation framework for application-specific logic-in-memory blocks*.
27. Brooks, D., & Tiwari, V. (2000). *Wattch: a framework for architectural-level power analysis and optimizations*.



Qiuling Zhu received her B.S. degree in Department of Electronic Science and Technology from Huazhong University of Science and Technology, Wuhan, China and her M.S. degree in the Institute of Microelectronics of Tsinghua University, Beijing, China in 2007 and 2009 respectively. She is currently pursuing a Ph.D. degree in Department Electrical and Computer Engineering at Carnegie Mellon University, Pittsburgh, PA, USA., with research interests in sub-22 nm VLSI design and design automation methodology, application-specific computer architecture focusing on memory architecture, and hardware accelerators for signal processing, image processing and computer vision applications.

Christian R. Berger was born in Heidelberg, Germany in 1979. He received the Dipl.-Ing. degree from the Universitaet Karlsruhe (TH), now Karlsruhe Institute of Technology (KIT), in Karlsruhe, Germany in 2005; the Ph.D. degree from the University of Connecticut, Storrs, in 2009, both in electrical engineering. From 2009–2011 he was a Post-Doctoral researcher at Carnegie Mellon University, Pittsburgh, PA. He is now a Staff Systems Engineer in the Wireless R&D Group at Marvell Semiconductor, Santa Clara, CA. His research interests are in the area of signal processing for wireless communication, specifically implementation of multicarrier systems such as OFDM, with focus on synchronization, channel estimation, and implementation complexity. Dr. Berger has served as reviewer for various technical journals and conferences, as well as on the technical program committee of the Fusion conference and the PIMRC symposium.



Eric L. Turner received his B.S. degree in Electrical and Computer Engineering from Carnegie Mellon University in 2011. He worked for MIT Lincoln Laboratory, focusing in Synthetic Aperture Radar Coherent Change Detection. He is currently a Ph.D. Candidate at U.C. Berkeley in Electrical Engineering and Computer Science. His research interests include 3D Modeling, Surface Reconstruction, and Digital Signal Processing.

Larry Pileggi is the Tanoto Professor of Electrical and Computer Engineering at Carnegie Mellon University and the director of the FCRP Center for Circuit and System Solutions (C2S2). He previously held positions at Westinghouse Research and Development and the University of Texas at Austin. He received his Ph.D. in Electrical and Computer Engineering from Carnegie Mellon University in 1989. His research interests include various aspects of digital and analog design and design methodologies. He has consulted for various semiconductor and EDA companies, and was a co-founder of Fabbrix (acquired by PDF Solutions in 2007) and Extreme DA (acquired by Synopsys in 2011).

He has received various awards, including Westinghouse corporation's highest engineering achievement award, the best CAD Transactions paper awards for 1991 and 1999, a Presidential Young Investigator award from the National Science Foundation, Semiconductor Research Corporation (SRC) Technical Excellence Awards in 1991 and 1999, the inaugural Richard A. Newton GSRC Industrial Impact Award, the SRC Aristotle award in 2008, and the IEEE Circuits and Systems Society Mac Van Vlakenburg Award in 2010. He is a co-author of "Electronic Circuit and System Simulation Methods," McGraw-Hill, 1995 and "IC Interconnect Analysis," Springer, 2002. He has published over 250 refereed conference and journal papers and holds 30 U.S. patents. He is a fellow of IEEE.

Franz Franchetti is an Associate Research Professor with the Department of Electrical and Computer Engineering at Carnegie Mellon University. He received the Dipl.-Ing. (M.Sc.) degree in Technical Mathematics and the Dr. techn. (Ph.D.) degree in Computational Mathematics from the Vienna University of Technology in 2000 and 2003, respectively. In 2006 he was member of the team winning the Gordon Bell Prize (Peak Performance Award) and in 2010 he was member of the team winning the HPC Challenge Class II Award (most productive system).

Dr. Franchetti's research focuses on automatic performance tuning and program generation for emerging parallel platforms, including multicore CPUs, clusters and high-performance systems (HPC), graphics processors (GPUs), field programmable gate arrays (FPGAs), and FPGA-acceleration for CPUs. As member of the Spiral research team (www.spiral.net), his research goal is to enable automatic generation of highly optimized software libraries for important kernel functionality. In other collaborative research threads Dr. Franchetti is investigating the applicability of domain-specific transformations within standard compilers, and hardware and software co-design based on high-level hardware and algorithm descriptions, as well as the possibility of application-specific logic within memory. Dr. Franchetti is Thrust Leader of the Security Thrust in Carnegie Mellon's SRC Smart Grid Research Center and Faculty Senator for the ECE Department at Carnegie Mellon. He is CTO and co-founder of SpiralGen, a Pittsburgh, PA company commercializing the technology developed in the Spiral project.