Efficient Backprojection-based Synthetic Aperture Radar Computation with Many-core Processors

Jongsoo Park, Ping Tak Peter Tang, Mikhail Smelyanskiy, Daehyun Kim
Intel Corporation

Thomas Benson
Georgia Tech Research Institute

Abstract—**Tackling computationally challenging problems with high efficiency often requires the combination of algorithmic innovation, advanced architecture, and thorough exploitation of parallelism. We demonstrate this synergy through synthetic aperture radar (SAR) via backprojection, an image reconstruction method that can require hundreds of TFLOPS. Computation cost is significantly reduced by our new algorithm of approximate strength reduction; data movement cost is economized by software locality optimizations facilitated by advanced architecture support; parallelism is fully harnessed in various patterns and granularities. We deliver over 35 billion backprojections per second throughput per compute node on an Intel® Xeon® processor E5-2670-based cluster, equipped with Intel® Xeon Phi™ coprocessors. This corresponds to processing a 3K×3K image within a second using a single node. Our study can be extended to other settings: backprojection is applicable elsewhere including medical imaging, approximate strength reduction is a general code transformation technique, and many-core processors are emerging as a solution to energy-efficient computing.**

I. Introduction

Modern high-performance computing systems are increasingly constrained by their energy consumption [1]. This makes combined optimization efforts spanning multiple layers crucial to solve computationally challenging problems at a reasonable cost. Algorithmic optimizations often play significant roles in improving efficiency, particularly when combined with a clear understanding of the underlying hardware architecture. For example, as the number of cores increases, researchers have devised algorithms with a higher parallelization scalability, thereby often achieving an order of magnitude performance improvement [2]. At an implementation level, parallelism found in applications must be fully exploited. However, careful consideration of the underlying architecture is required because the best efficiency is realized only when parallelization is accompanied with data access patterns optimized for the memory hierarchy [3, 4]. At the same time, the hardware should provide an architecture that enables efficiently harnessing parallelism in various types.

This paper demonstrates such combined optimization efforts using synthetic aperture radar (SAR) [5] image formation via backprojection as an example. SAR backprojection includes key characteristics representative of computing challenges of the future. Although faster SAR image formation algorithms exist, backprojection offers increased versatility with respect to the collection geometry, which is desirable for important scenarios, including wide-area persistent surveillance. However, this versatility is provided with substantially increased computational requirements. In a high-end input scenario presented by Campbell et al. [6], more than 9 trillion backprojections are required per image. SAR backprojection is also often used in real-time applications. For example, the persistent surveillance application presented in [6] imposes a real-time constraint of producing one image each second. Real-time applications can be severely constrained by power consumption, and, therefore, it is critical to fully exploit highly-parallel architectures and efficiently conduct data movement. A large amount of data are streamed through at a rapid rate, a significant portion of which are accessed in an irregular manner. We showcase that these challenges can be addressed by many-core processors and advanced architecture support for irregular memory access.

SAR backprojection also exhibits similarities with reconstruction methods found in other domains, such as medical imaging. For example, the computational pattern is similar to that of X-ray computed tomography [7] and beamforming used in ultrasound imaging [8]. Similarly to SAR backprojection, several reconstruction algorithms depend heavily upon trigonometric function evaluation, interpolation, and irregular memory access [9, 10]. In addition, image reconstruction problems typically have a large degree of parallelism and, thus, are attractive targets for many-core processors.

This paper makes the following contributions:

- We present an algorithmic optimization that reduces the square root, sine, and cosine functions to a few multiplications and additions. Our optimization resembles strength reduction, a well-known compiler optimization [11, 12]. While the standard strength reduction relies on, and is thus constrained to, mathematical equivalence, our optimization exploits the method of approximation. Therefore, we call this approach *approximate strength reduction* (ASR). The application of ASR to the backprojection stage achieves 2–4× speedups, while preserving a similar level of accuracy.
- We exploit hardware gather support of Intel® Xeon Phi™ coprocessors for frequent irregular memory accesses, thus improving the vectorization efficiency. The efficiency of gather access is further improved by exploiting geometric properties of backprojection used in SAR imaging. These optimizations provide an additional 1.4× speedup.

SC12, November 10–16, 2012, Salt Lake City, Utah, USA
978-1-4673-0806-9/12/$31.00 ©2012 IEEE

- We harness parallelism found in SAR backprojection to fully utilize abundant parallel resources available in modern processors. We present parallelization and evaluation of SAR backprojection in the context of a full application—wide-area, persistent surveillance. Each node of our platform comprises Intel® Xeon® E5-2670 (two 8-core processors, Sandy Bridge architecture) and two 60-core Intel® Xeon Phi™ coprocessors*. As a result of exploiting parallelism at multiple levels combined with ASR, each node is capable of processing 35 billion backprojections per second. Due to near-linear scaling, our system with 16 nodes achieves over 0.5 trillion backprojections per second, or equivalently a 13K$\times$13K image within a second. We project the performance to further scale linearly and thus the aforementioned high-end scenario can be handled by approximately 256 nodes.

The rest of this paper is organized as follows: Section II briefly describes SAR backprojection, focusing on its computational characteristics. Section III presents approximate strength reduction transformations for square root, sine, and cosine, which significantly reduces the computational complexity of SAR. Section IV describes how SAR backprojection is parallelized at multiple levels, including SIMD, OpenMP, coprocessor, and MPI, and how it is pipelined to overlap data transfers with computation. Section V evaluates the performance benefits of ASR and efficient mapping to multiple nodes with Xeon processors and Xeon Phi coprocessors. Section VI reviews related work, and Section VII concludes this paper.

II. SAR Imaging via Backprojection

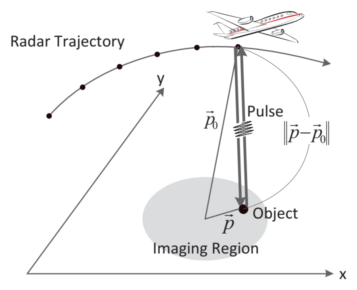


Fig. 1: Synthetic aperture radar in a spotlight mode. A radar mounted on an airplane repeatedly flies around the target imaging area while maintaining an approximate circular orbit.

A large antenna aperture is required to reconstruct a high-resolution image from radar data collected from a single location. Synthetic aperture radar (SAR) [13] relaxes this constraint by collecting data from multiple locations along a platform trajectory using, for example, a radar mounted on an airplane (Fig. 1). Thus, SAR "synthesizes" a large aperture, thereby obtaining high cross-range resolution [5]. Each pulse is modulated with a carrier frequency and sampled at high speed after having been reflected from the imaged scene.

*Evaluation card only and not necessarily reflective of production card specifications.

TABLE I: An example of high-end input parameters [6] and requirements for the real-time constraint (one image per second).

New pulses per image	N	3K
Samples per pulse	S	81K
Image size	$I_x,\ I_y$	57K
Accumulation factor	k	34
Registration control points	N_c	929K
Registration neighborhood size	S_c	31
CCD neighborhood size	N_{cor}	25
CFAR neighborhood size	N_{cfar}	25
	Total	351 TFLOPS
Compute requirement	Backprojection	347 TFLOPS
(after approximate	2D-Correlation	0.7 TFLOPS
strength reduction)	Interpolation	0.2 TFLOPS
	CCD	3 TFLOPS

SAR image formation via backprojection offers several advantages over Fourier-based image formation techniques, such as the polar formatting algorithm (PFA) [14], that are currently more widely applied. PFA has a relatively low computational complexity due to its utilization of the fast Fourier transform, but it imposes assumptions of planarity on both the reconstruction surface and the wavefront within the imaged scene. In addition, PFA assumes an idealized trajectory for the radar platform. To an extent, compensation can be applied for deviations from these assumptions, but image quality degrades as the deviations increase. Backprojection, on the other hand, imposes no assumptions on the planarity of the wavefront or imaged surface and can handle non-ideal collection trajectories. However, backprojection provides this generality with substantially higher computational requirements.

Table I shows a high-end input scenario and computational requirements of the persistent surveillance problem from [6], which includes SAR backprojection. Fig. 2 depicts the corresponding data-flow in which stages exhibit different computational and communication characteristics. Backprojection dominates the total compute time; it represents more than 98% of the total FLOP count in the scenario shown in Table I. However, other stages can quickly become bottlenecks due to Amdahl's law [15] if not parallelized, particularly in systems with more than a thousand-fold parallelism such as ours. In addition, special care must be taken to account for relatively small compute-to-bandwidth ratios in other stages.

The backprojection stage computes the distance from the radar platform to each pixel in the imaging region, which

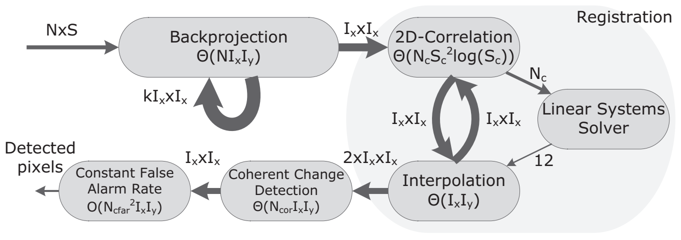


Fig. 2: The data-flow of persistent surveillance problem where SAR images are formed via backprojection. The numbers between stages denote the size of data streamed through per output image. Big O notations inside stages denote compute complexities.

```
1 for $y$ from 0 to $I_y$ - 1

2  for $x$ from 0 to $I_x$ - 1 {
3   $\vec{p}$ = position of pixel ($x$, $y$)
4   $r$ = sqrt(($\vec{p} - \vec{p_0}$) · ($\vec{p} - \vec{p_0}$))
5   bin = $(r - r_0)/dr$
6   arg = (cos($2\pi kr$), sin($2\pi kr$))  $\Rightarrow$

7   sample = interp(In, bin)
      // irregular memory access
8   Out[$x$, $y$] += arg × sample
  }
```

(a) Baseline

```
$\mathbf{0}_1$ for each pixel block {
$\mathbf{0}_2$  pre-compute A, B, C, $\mathbf{\Phi}$, $\mathbf{\Psi}$, and $\mathbf{\Gamma}$
1   for each $y$ in the current block {
$\mathbf{1}_2$    $\gamma$ = (1, 0)
2    for each $x$ in the current block {


5      bin = A[$x$] + B[$y$] + $x$·C[$y$]
$\mathbf{6}_1$      arg = $\mathbf{\Phi}[x] \times \mathbf{\Psi}[y] \times \gamma$ // 8 muls and 4 adds
$\mathbf{6}_2$      $\gamma$ ×= $\mathbf{\Gamma}[y]$ // 4 muls and 4 adds
7      sample = interp(In, bin)

8      Out[$x$, $y$] += arg × sample
     }
   }
 }
```

(b) Strength reduction of sqrt, sin, and cos

Fig. 3: The inner loop of backprojection stage and application of approximate strength reduction to it.
(a) Baseline backprojection inner loop. Lines 4 and 6 involve computationally expensive mathematical functions. In line 7, `interp`(**In**, `bin`) computes (1 - (`bin` - $\lfloor$`bin`$\rfloor$))·**In**[$\lfloor$`bin`$\rfloor$] + (`bin` - $\lfloor$`bin`$\rfloor$)·**In**[$\lfloor$`bin`$\rfloor$ + 1], which requires irregular memory access since `bin` is non-linear with respect to x. Bold faces denote complex numbers, and $\vec{p_0}$, r_0, dr, and k are constants.
(b) After strength reduction of `sqrt`, `sin`, and `cos`. These math functions are reduced to a few multiplications, additions, and array accesses. Section III explains steps involved in this transformation.

involves a square root computation ($\|\vec{p} - \vec{p_0}\|$ in Fig. 1). We then compute the time delay from the platform to the pixel and sample the received signal at the corresponding time. Since signals are discretized in the time domain, we need to interpolate multiple samples to reconstruct the signal at a given time delay. This is repeated per pulse, and we accumulate the corresponding samples. Since the time delay varies in a non-linear fashion as we proceed from pixel to pixel, or from pulse to pulse, memory accesses for sampling are irregular. Furthermore, because the time delay for a pixel-pulse pair is known, an ideal reflector response at that time delay is used as a matched filter for the interpolated samples, which requires a sine and cosine computation for each pixel-pulse pair [5].

Typically, in order to obtain the desired cross-range resolution, we use several seconds worth of collected data to form an image, although we generate an image each second. Thus, we use not only newly incoming pulses but also some previously accumulated ones. Let N be the number of new incoming pulses per second and k be the accumulation factor. Instead of backprojecting $(k+1)N$ pulses per second, we backproject only the new N pulses and combine those with the previous k backprojection results. This approach is valid because the backprojection process is linear. This incremental backprojection is implemented using a circular buffer that stores the prior k and the current backprojection results. The feedback loop of the backprojection stage shown in Fig. 2 references this buffering. This buffering reduces the computational requirement of the backprojection stage by k times at the expense of using more memory, which is beneficial because our system will still be more limited by compute than memory capacity requirements[1].

The stages following backprojection shown in Fig. 2 post-process the reconstructed images. The registration stage corrects for distortions or misalignments in the reconstructed image by aligning it with a reference image. This stage involves (1) finding a transformation that matches the current image closely to the reference image using N_c $S_c \times S_c$ 2D FFTs followed by solving linear systems via normal equations with six unknowns, and (2) applying the transformation using bilinear interpolation for resampling. The coherent change detection (CCD) stage then computes correlations between $N_{cor} \times N_{cor}$-size windows centered at the same position in the current and reference images. Its straightforward implementation requires $\Theta(N_{cor}^2 I_x I_y)$ operations, which can be reduced to $\Theta(N_{cor} I_x I_y)$ by incrementally computing correlation values. The final stage, constant false alarm rate (CFAR) detection, identifies differences between the current and reference images, while maintaining a constant false alarm rate under certain statistical assumptions [5]. Its complexity is $\Theta(N_{cfar} N_d)$, where N_d denotes the number of pixels for which the correlation value produced by CCD falls below a threshold; N_d is typically substantially smaller than $I_x \times I_y$.

III. Approximate Strength Reduction

Fig. 3(a) shows the inner loop of backprojection. As indicated in lines 4 and 6, backprojection spends a significant

[1]In the scenario shown in Table I, the memory capacity requirements will increase from 100 GB to 948 GB, where double buffering for pipelining is taken into account. This requires 119 Xeon Phis, assuming 8GB GDDR each. This is still fewer than the number for the compute requirements—more than 182 are required for 351 TFLOPS to produce one image per second even assuming 100% FLOP efficiency (1,920 GFLOPS per Xeon Phi).

amount of its time in mathematical functions, specifically square root, sine, and cosine. This section describes our new algorithm of approximate strength reduction that converts these functions into a few multiplications and additions.

A. Overview

At a high level, our optimization applies the well-known strength reduction compiler optimization to the truncated Taylor series approximation of mathematical functions. The pseudo code below exemplifies a simple strength reduction:

```
                                 0 t = 0
1 for $i$ from 0 to N            1 for $i$ from 0 to N
2   A[i] = c×i          ⇒       $2_1$   A[i] = t
                                 $2_2$   t += c
```

We reduce the "strength" of multiplication operations to additions by exploiting the recurrence relation between `A[i]`s. Similarly, in Fig. 3(b), we reduce square root, sine, and cosine functions to a few multiplications and additions. At a high level, this transformation (1) approximates r as a quadratic function of x and y (i.e., a truncated Taylor series), (2) applies the trigonometric recurrence $sin(a \cdot (i+1)+b) = sin(a \cdot i + b) \cdot cos(a) + cos(a \cdot i + b) \cdot sin(a)$ and a similar recurrence for cosine, and (3) applies the standard strength reduction optimization. However, in contrast to the conventional strength reduction, the transformation shown in Fig. 3(b) is inexact since approximating square roots as quadratic functions introduces errors. Nevertheless, we have a certain level of control over the accuracy by blocking the loop and applying per-block approximations. To distinguish from the conventional strength reduction, we call the transformation presented in Fig. 3(b) *approximate strength reduction* (ASR). We present the details for ASR in the following sections.

B. Strength Reduction through Quadratic Polynomial Approximation

We consider the general setup of a problem as computing a two-variable function $f(x, y)$ in a doubly nested loop:

```
for $m$ from 0 to M - 1
   for $\ell$ from 0 to L - 1
      $z$[$m$, $\ell$] = $f(x_0 + \ell\Delta_x, y_0 + m\Delta_y)$
```

The strength of $f(x_0 + \ell\Delta_x, y_0 + m\Delta_y)$ is to be reduced via a quadratic approximation

$$f(x_0 + \ell\Delta_x, y_0 + m\Delta_y) \\ \simeq f(x_0, y_0) + a_x\ell + a_y m + b_x\ell^2 + b_y m^2 + c_{xy}\ell m, \quad (1)$$

where a_x, a_y, b_x, b_y, and c_{xy} are constants involving Δ_x, Δ_y, and first and second partial derivatives of f.[2]

[2]We approximate around $\ell = 0$ and $m = 0$ for illustrative purposes, instead of using a more accurate approximation around ℓ = L/2 and m = M/2. This more accurate approximation is a straightforward modification from the one presented here and is used in the evaluation section.

```
for $m$ from 0 to M - 1
   for $\ell$ from 0 to L - 1
      $\tilde{z}[m, \ell] = f(x_0, y_0) + a_x\ell + a_y m + b_x\ell^2 + b_y m^2 + c_{xy}\ell m$
```

From here, we can apply standard techniques of (exact) strength reduction as follows:

```
// pre-computation
A[0] = $f(x_0, y_0)$
for $\ell$ from 1 to L - 1
   A[$\ell$] = A[$\ell$ - 1] + ($a_x$ - $b_x$) + $2b_x\ell$
B[0] = 0, C[0] = 0
for $m$ from 1 to M - 1
   B[$m$] = B[$m$ - 1] + ($a_y$ - $b_y$) + $2b_y m$
   C[$m$] = C[$m$ - 1] + $c_{xy}$
// the main loop
for $m$ from 0 to M - 1
   $c$ = 0
   for $\ell$ from 0 to L - 1
      $\tilde{z}$[$m$, $\ell$] = A[$\ell$] + B[$m$] + $c$
      $c$ += C[m]
```

C. Square Root Strength Reduction

Consider the function $f(x, y) = \sqrt{x^2 + y^2 + \alpha^2}$. We approximate using the second-order Taylor series around the point (x_0, y_0):

$$f(x_0 + \ell\Delta_x, y_0 + m\Delta_y) \\ \simeq f(x_0, y_0) + \ell\Delta_x f_x(x_0, y_0) + m\Delta_y f_y(x_0, y_0) + \\ \frac{1}{2!}[\ell^2\Delta_x^2 f_{xx}(x_0, y_0) + 2\ell\Delta_x m\Delta_y f_{xy}(x_0, y_0) + \\ m^2\Delta_y^2 f_{yy}(x_0, y_0)],$$

where f_x, f_y, f_{xx}, f_{xy}, and f_{yy} are partial derivatives of f.

By collecting terms, we can represent $f(x_0 + \ell\Delta_x, y_0 + m\Delta_y)$ in the form shown in Equation 1, where

$$a_x = \frac{\Delta_x x_0}{f(x_0, y_0)}, \quad a_y = \frac{\Delta_y y_0}{f(x_0, y_0)},$$
$$b_x = \frac{\Delta_x^2}{2f(x_0, y_0)} - \frac{\Delta_x^2 x_0^2}{2f(x_0, y_0)^3},$$
$$b_y = \frac{\Delta_y^2}{2f(x_0, y_0)} - \frac{\Delta_y^2 y_0^2}{2f(x_0, y_0)^3},$$
$$c_{xy} = -\frac{\Delta_x\Delta_y}{f(x_0, y_0)^3} x_0 y_0.$$

D. Trigonometric Function Strength Reduction

Consider the function $\mathbf{g}(x, y) = e^{if(x,y)}$ and suppose that $f(x_0 + \ell\Delta_x, y_0 + m\Delta_y)$ can be approximated by a quadratic function as shown in Equation 1.[3] Then,

$$\mathbf{g}(x_0 + \ell\Delta_x, y_0 + m\Delta_y) \\ = \mathbf{g}(x_0, y_0) \cdot e^{ia_x\ell} \cdot e^{ia_y m} \cdot e^{ib_x\ell^2} \cdot e^{ib_y m^2} \cdot e^{c_{xy}\ell m}$$

Therefore, we can transform the loop

[3]Recall that $e^{ix} = cos(x) + isin(x)$

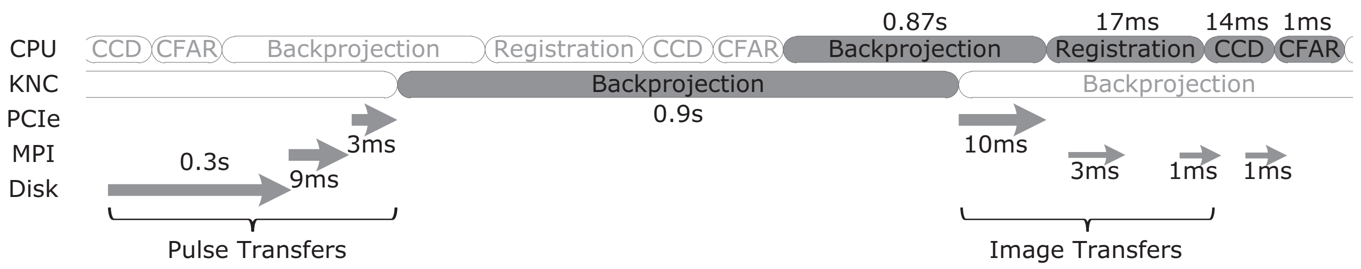


Fig. 4: Pipelining the persistent surveillance problem that uses SAR backprojection. Dark bars and arrows denote one iteration of the pipeline. Numbers denote execution times when 16 nodes are used to output 13K×13K images (the detailed setup is described in Section V).

```
for $m$ from 0 to M - 1
  for $\ell$ from 0 to L - 1
    $\mathbf{z}[m,\ell] = \mathbf{g}(x_0 + \ell\Delta_x, y_0 + m\Delta_y)$
```

as follows (compare this with Fig. 3(b)):

```
// pre-computation
$\mathbf{A}$[0] = $\mathbf{g}(x_0, y_0)$
for $\ell$ from 1 to L - 1
  $\mathbf{A}$[$\ell$] = $\mathbf{A}$[$\ell$ - 1]$\times e^{i((a_x - b_x)+2b_x\ell)}$
$\mathbf{B}$[0] = $e^{i0}$, $\mathbf{C}$[0] = $e^{i0}$
for $m$ from 1 to M - 1
  $\mathbf{B}$[$m$] = $\mathbf{B}$[$m$ - 1]$\times e^{i((a_y - b_y)+2b_y m)}$
  $\mathbf{C}$[$m$] = $\mathbf{C}$[$m$ - 1]$\times e^{ic_{xy}}$
// the main loop
for $m$ from 0 to M - 1
  $\mathbf{c}$ = $e^{i0}$
  for $\ell$ from 0 to L - 1
    $\tilde{\mathbf{z}}$[$m$, $\ell$] = $\mathbf{A}[\ell] \times \mathbf{B}[m] \times \mathbf{c}$
    $\mathbf{c}$ $\times$= $\mathbf{C}[m]$
```

E. Accuracy-performance Trade-offs

The accuracy of ASR drops as we move away from the center of approximation. This is because the Taylor approximation shown in Equation 1 incurs larger errors as $\ell\Delta_x$ and $m\Delta_y$ increase: cf. the linear Taylor approximation of $\sqrt{1+x}$ around $x = 0$, $1 + \frac{1}{2}x$, incurs larger errors as $|x|$ increases. We control the accuracy of ASR by blocking the loop and applying ASR to each block as exemplified by the outer loop in Fig. 3(b). Section V shows that when using sufficiently small blocks (64×64), applying ASR to SAR backprojection maintains a similar level of accuracy to an approach with no such approximations.

In addition, we can trade off accuracy for performance by changing the block size: a larger block size increases errors, but reduces the pre-computation time. This trade-off can be helpful in radar applications where the required accuracy ultimately depends upon the final application and can thus vary from task to task. ASR and its application to accuracy-performance trade-offs can be generalized to other mathematical functions in different applications, but such generalization is out of the scope of this paper.

This section presented an algorithmic improvement on backprojection that reduces complex mathematical functions to simpler operations. The application of ASR to SAR backprojection was motivated by its geometric properties: the distance from the radar varies little from pixel-to-pixel and the variation has a regular pattern than can be approximated by a quadratic polynomial. Our algorithm not only reduces mathematical complexity, but also facilitates implementations optimized for the hardware. For example, Section V will show that implementations without ASR require mixed usage of double and single precision operations in the inner-most loop to achieve a desired accuracy. In contrast, ASR can achieve the desired accuracy while performing the inner-most loop computation entirely with single precision. This allows for easier vectorization and increases the effective SIMD width.

IV. Parallelization

The persistent surveillance application imposes a real-time constraint of producing one image per second. The ASR optimization presented in the previous section significantly reduces the computational requirement, but satisfying this constraint is still a challenge.

This section presents the following approaches that address this challenge:

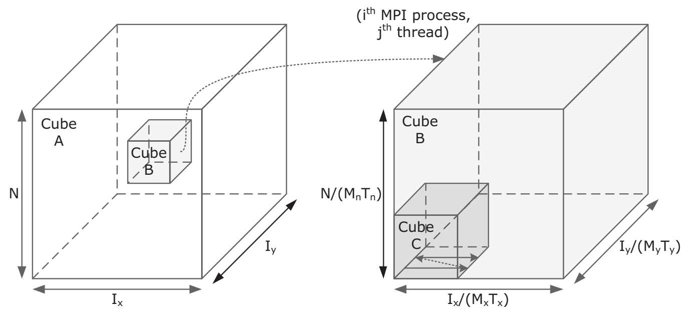
- The application is pipelined to hide the latency of data transfers. The application is less sensitive to the latency than throughput.
- Parallel resources in modern computation platforms are fully exploited. The computation is carefully partitioned between Intel Xeon and Xeon Phi so that the benefits of Xeon Phi's high compute intensity can be maximized.
- Data movement is optimized for locality and vectorization, which is facilitated by architecture support for irregular memory access.

A. Overall parallelization

Fig. 4 illustrates how our application is pipelined. Execution times denoted were measured when 16 compute nodes were processing 13K×13K images. Note that the proportion of backprojection time depicted is significantly smaller than the actual for illustrative purposes. A large portion of backprojection is offloaded to Xeon Phis. Due to its high compute-to-communication ratio, data transfer via PCIe can be easily overlapped with computation. For example, in Fig. 4, transferring input pulses and output images through PCIe take 3 ms

```
for each pulse $n$ from 0 to $N$ - 1 {
 $\mathbf{In}$ = $\mathbf{In}_{\mathbf{global}}$[n]
 for $y$ from 0 to $I_y$ - 1 {
  for $x$ from 0 to $I_x$ - 1 {
   ... // Read $\mathbf{In}$ and update $\mathbf{Out}$[$x$, $y$].
   // See more details in Fig. 3(a)
  }
 }
}
```

(a)



(b)

Fig. 5: (a) Pseudo code showing the structure of baseline implementation of backprojection stage, and (b) partitioning and blocking of it. The 3D iteration space [16] is hierarchically partitioned into cubes in three levels: MPI, OpenMP, and cache blocking levels. Cube B is an OpenMP-level cube and cube C is a cache-blocking-level cube. We have $M_{x/y/n}$ MPI processes along x/y/pulse dimensions and $T_{x/y/n}$ OpenMP threads per MPI process along x/y/pulse dimensions.

and 10 ms, respectively, which is much faster than the 900 ms of backprojection computation time. On the other hand, other stages of the pipeline have smaller compute-to-communication ratios and less parallelism. Therefore, offloading them is not beneficial due to the cost in terms of latency and energy consumption associated with PCIe transfers.

In our implementation, the CPUs spawn two types of threads: compute threads and I/O threads. We use two compute threads per core to exploit 2-way SMT. We have three I/O threads, two of which initiate offloading to Xeon Phis. The remaining I/O thread handles disk I/O, MPI, and PCIe operations to overlap data transfers with computation. These I/O threads do not interfere with the compute threads since they are blocked by I/O events and de-scheduled by the operating system most of time. The I/O threads are synchronized with the compute threads through concurrent bounded queues implemented with Pthread condition variables.

Each stage, except for the linear system solver for registration, is parallelized both via MPI and OpenMP. The linear system solver is not parallelized because its execution time is negligible due to its relatively small system size. A portion of the output image is assigned to each MPI process and then to each OpenMP thread as illustrated by Fig. 5; the following section elaborates on this partitioning. The MPI communication prior to backprojection involves distributing the input pulse data among nodes, but the communication time is still much lower than the overall execution (e.g., 9 ms in Fig. 4), and can be easily overlapped with computation. In other MPI communications, each node sends neighbors small boundary areas of the assigned image portion, all of which take negligible time. For example, before registration, each node sends boundaries with width S_c of the current and reference images, which takes 3 ms in Fig. 4. Before CCD, each node sends boundaries with width N_{corr} of the distortion-corrected image (1 ms). Finally, before CFAR, each node sends boundaries with width N_{cfar} of the correlation values (1 ms).

B. Backprojection Partitioning

Fig. 5(a) presents a high-level structure of the backprojection stage in its unoptimized form (the inner-loop is shown in Fig. 3(a)). The iteration space [16] of the triple-nested loop is depicted as cube A shown in Fig. 5(b). The outer-most loop iterates over N input pulses, and the two inner loops iterate over $I_x \times I_y$ output image pixels. We partition the iteration space in 3D blocks both in MPI and OpenMP as indicated by cube B. For parallelization, we partition output image pixels instead of input pulses when possible. This is because the latter involves privatization followed by reduction of the array $\mathbf{Out}$ with non-trivial cache miss and computation overheads. We resort to partitioning input pulses only when the partition size of output image pixels becomes smaller than the ASR block size. However, this does not occur in practical settings with large enough images, such as those evaluated in Section V.

C. Backprojection Locality Optimization

We improve the locality of accessing arrays $\mathbf{In}$ and $\mathbf{Out}$ using 3D cache blocking within the partition mapped to each OpenMP thread as illustrated by cube C in Fig. 5(b). Within the cube, we iterate along the output image dimension first, since this yields repeated accesses of the same entries of $\mathbf{In}$. For example, when the radar pulse wavefront is as depicted in Fig. 6, iterating along the y axis will yield similar r values, which, in turn, leads to accessing the same entries of $\mathbf{In}$. We thus reorder the x and y loops for each pulse depending on the orientation of the radar pulse wavefront relative to the imaging region. We can analytically compute how many consecutive backprojections access the same entry of $\mathbf{In}$ on average. This value is 5 when reordering optimization is not used and the edge length of the imaging region is 1/10 of the scene-to-

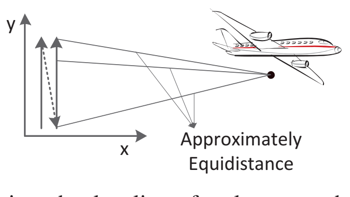


Fig. 6: Optimizing the locality of pulse array by dynamic loop iteration reordering. When the radar is mostly horizontally distanced from the imaging center, iterating along y axis first results in similar r values, thereby improving the locality of accessing input pulses.

radar distance. This value increases to 17 when reordering optimization is applied, hence improving the locality.

To avoid cache conflict misses, each thread writes to a private image buffer so that each 3D block is accessed contiguously without long strides. At the end of the backprojection loop, the content of each temporal buffer is copied into the corresponding portion of the final output buffer. When multiple threads work on the same portion of the image, multiple buffers are reduced while synchronized by barriers.

D. Vectorization

Irregular memory accesses to **In** are a challenge for vectorizing the backprojection loop. On Intel Xeon processors, we layout the input data in an array-of-structure format so that $\mathbf{In}[\lfloor\texttt{bin}\rfloor]\texttt{.real/imag}$ and $\mathbf{In}[\lfloor\texttt{bin}\rfloor + \texttt{1]}\texttt{.real/imag}$ can be loaded together using an 128-bit load instruction. Each iteration loads 8 128-bit values and transposes them into 4 256-bit registers, which requires 30 AVX instructions. In Xeon Phi, we use hardware-supported gather instructions and maintain the input data in a structure-of-array layout. Section V shows that Xeon Phi spends a smaller portion of time in accessing **In** relative to Xeon processors.

While ASR reduces the computation complexity, it introduces loop-carried dependencies in the inner loop. We break the dependency by increasing the recurrence step size to the SIMD width. In Xeon Phis with wider SIMD units, it is important to also vectorize the pre-computation step even though it is located in an outer-loop (i.e., the loop shown in line $\mathbf{0_1}$ of Fig. 3(b)). Thus, we aggregate pre-computations for SIMD-width iterations of the outer-loop and vectorize them.

V. Evaluation

A. Setup

We evaluate the performance of our optimized application on the Endeavor cluster. Each node consists of dual-socket Intel® Xeon® E5-2670 processors (Sandy Bridge architecture) and two Intel® Knights Corner coprocessors, both of which are detailed in Table II.

The real-time constraint of the persistent surveillance application requires producing one image per second. Therefore, in practical settings, one would choose a large enough cluster to satisfy this constraint for a given problem size. Our evaluation setup reflects this practical setting: for each cluster size, we choose the largest input that can be processed while satisfying the real-time constraint. The number of new pulses per image (N) is fixed at 2,809 as shown in Table I. In this case, a single node can process one 3K×3K image per second, but requires more than one second for larger images. Therefore, we use 3K×3K images for single-node experiments. For 16 nodes, the corresponding largest image size is 13K×13K. Other parameters such as S and S_c are scaled proportionally from the values listed in Table I.

The input data are generated by simulating a radar platform circling around clusters of reflectors, which appear and disappear at pre-determined times. The simulation model assumes linear frequency modulated pulses (i.e., chirp) [5]. The input pulses contain baseband complex signals down-converted from the reflected signals. A random perturbation is induced for the radar trajectory to test the robustness of SAR imaging via backprojection. In addition, to test the registration process, shifts in the recorded platform trajectory are induced between images. These random perturbation and induced shifts are designed to mimic inaccuracies in the platform location provided by the inertial navigation system.

B. Single Processor Efficiency Improvements in Backprojection Stage

1) Compute Complexity Reduction through Approximate Strength Reduction: Fig. 7 breaks down the execution times of backprojection before and after ASR optimization. The baseline uses double precision for computing r and for reducing the argument of the sine and cosine functions, which depend upon r, to the range $[-\pi, \pi]$. Using single precision for these calculations incurs errors in the sine and cosine arguments that are large relative to π since r typically has a substantial magnitude (e.g., 20K). Both in Xeon and Xeon Phi, before applying ASR, square roots account for the largest fraction of time because they are computed in double precision. Sine and cosine are computed by approximation polynomials [17] that are vectorized and yield an accuracy equivalent to that of Intel® MKL VML in the Enhanced Performance mode. Argument reduction accounts for 40% of the time spent on the sine and cosine computations on Xeon and Xeon Phi. This implies that we can reduce the sine and cosine time by at most 60% even with hardware sine and cosine units, which typically

TABLE II: System Configuration

	Intel Xeon E5-2670	Knights Corner*
Socket×core×SMT×SIMD	2 × 8 × 2 × 8	1 × 60 × 4 × 16
Clock (GHz)	2.60	1.00
Single precision GFLOP/s	660	1,920
L1/L2/L3 Cache (KB)[4]	32/256/20,480	32/512/-
DRAM	64 GB	8 GB GDDR
PCIe BW	10 GB/s	
Interconnect	QDR Infiniband 4× (peak 4 GB/sec) A two-level 14-ary fat tree	
Software	Intel® Compiler v13.0.0, Intel® MPI v4.0	

*Evaluation card only and not necessarily reflective of production card specifications.
[4]Private L1/L2, shared L3

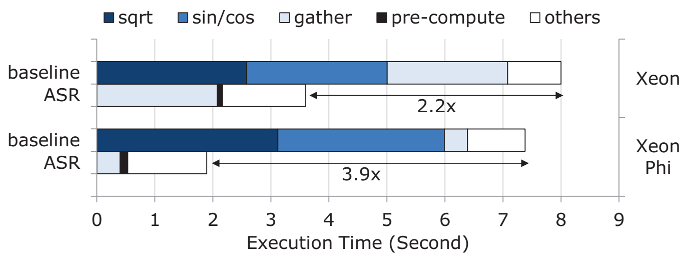


Fig. 7: Performance improvements of backprojection from ASR. Images with 3K×3K size are reconstructed from 2,809 pulses with 4K samples each.

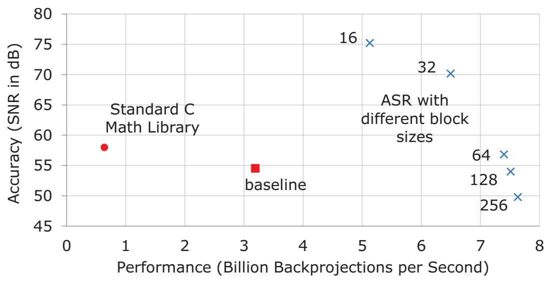


Fig. 8: Accuracy-performance trade-offs using ASR. A dual-socket Xeon is used with the same input data for Fig. 7.

require that operands are within the range $[-\pi, \pi]$ in order to provide the desired accuracy.

ASR removes square root, sine, and cosine computations from the inner-loop with minimal pre-computation overhead, achieving 2.2× and 3.9× speedups in Xeon and Xeon Phi, respectively. ASR requires maintaining a few pre-computed values such as the arrays `A` and `B` shown in Fig. 3(b), but all of them fit comfortably in the L1 cache.

Although ASR approximates square roots as polynomial functions, by performing the accuracy sensitive pre-computation in double precision, we can achieve a comparable accuracy relative to the baseline. With ASR, double precision is easily accommodated during pre-computation for the sine and cosine operations. However, in the baseline, it would be expensive to include double precision in the inner loop for the sine and cosine operations. Fig. 8 presents how accuracy and performance vary with different ASR block sizes relative to the baseline. We measure the accuracy using the signal-to-noise ratio (SNR) computed as $10 \cdot log_{10} \frac{\sum(reference)^2}{\sum(measured\_signal - reference)^2}$. We compute the reference using full double precision calculations. We use the logarithmic decibel (dB) scale for SNR. For example, a 20dB higher SNR indicates an 1-digit higher accuracy. As described above, the baseline computes sine and cosine in an accuracy similar to that of Intel MKL VML, which results in 55dB. Using sine and cosine functions in the standard C math library instead results in a marginally better accuracy of 58dB. Even with the standard C math library, when computing r in single precision, the accuracy drops to 12dB. In addition, the sine and cosine functions in the standard C math library are not vectorized as efficient as the baseline due to its more complex control flow for supporting a wide-range of inputs.

Fig. 8 shows that, with block sizes less than 64×64, ASR achieves an accuracy higher than the baseline. We choose 64×64 in other experiments, including Fig. 7, to evaluate the performance of ASR with a similar level of backprojection accuracy as the baseline.

2) Parallelization and Locality Optimizations: Xeon and Xeon Phi achieve 4.6× and 10× speedups from vectorization with 8 and 16-wide SIMD units, respectively[5]. These non-linear speedups mostly stem from overhead associated with irregular access to the input pulse data. In particular, Xeon spends 53% of the total backprojection time on accessing the pulse data due to the overhead of transposing array-of-structure representations into structure-of-array formats in registers. The dynamic reordering of x and y loops reduces the input pulse data access time by 42% in Xeon Phi by reducing the number of cache lines accessed per gather instruction.

Support for hardware threads on Xeon Phi helps more (Xeon 1.2× vs. Xeon Phi 2.2×) due to its more hardware threads per core and in-order pipeline. OpenMP parallelization realizes near-linear speedups in both: 15.9× using 16 cores and 63× speedups using 60 cores. The super-linear scaling on Xeon Phi is due to the working set per core decreasing and the effective cache capacity thereby increasing.

Overall, Xeon and Xeon Phi achieve 42% and 28% of the ideal peak FLOP performance despite the overhead associated with irregular memory access. The lower efficiency of Xeon Phi is mainly due to the ideal Xeon Phi peak FLOP only being realized when there is a fused multiply-add every cycle, which is not the case for backprojection. Each backprojection iteration contains 38 FLOPs, 22 of which can be fused.

C. *Offloading Backprojection to KNCs*

TABLE III: Single-node backprojection throughput

	Throughput (Billion Backproject./Sec.)	Speedup vs. 2-socket Xeon	FLOP Efficiency
Xeon (2-socket)	7.4	1.0×	42%
1 Xeon Phi	14.0	1.9×	28%
Xeon + 2 Xeon Phi	35.5	4.8×	30%

Table III shows the backprojection throughput when Xeon and Xeon Phi are used individually, and then in combination. A Xeon Phi coprocessor is about 2× faster than a dual-socket Xeon, and we achieve a 5× speedup when two Xeon Phis are used together with Xeon. Since each system can have CPUs and Xeon Phis with different computation capabilities, we adjust the ratio of offloading to Xeon Phis based on the execution time ratio observed with the first few images.

We herein use 3K×3K images, for which the single node Xeon and Xeon Phi combination can satisfy the real-time constraint. With this input, backprojection takes around 0.75 second per image. Around 150 MB of data need to be transferred for offloading and 6 GB/s PCIe throughput is realized, resulting in 0.03 seconds of transfer time. This is overlapped with compute by asynchronous PCIe transfers using the `#pragma offload_transfer` and `#pragma offload_wait` constructs supported in the Intel C compiler.

D. *Scaling the SAR Pipeline to Multi-nodes*

While the preceding sections evaluate the performance of backprojection stage in isolation with a single node, this section evaluates the entire pipeline with multiple nodes. The pipeline outputs a stream of images, one image per second, and we measure the performance of the pipeline after reaching a steady state. We conduct weak scaling experiments that reflect the real-time constraint of processing one image per second:

[5]When measuring the vectorization speedups, ASR is applied to both non-vectorized and vectorized versions of backprojection.

TABLE IV: Multi-node weak scaling. For each number of nodes, we select the largest image size that can be processed while satisfying the real-time constraint.

Nodes	Input				Throughput (Billion	MPI Parallelization
	I_x, I_y	N	k	S	Backprojections/Sec.	Efficiency
1	3K		2	4K	35	1.00
2	4K		3	6K	71	1.01
4	6K	2,809	4	9K	138	0.97
8	9K		6	13K	265	0.94
16	13K		9	19K	530	0.93

TABLE V: The projection of largest inputs that can be handled while satisfying the real-time constraint

Nodes	Input				Throughput (Billion	Parallelization	Time Breakdown (%)				
	I_x, I_y	N	k	S	backprojections/Sec.)	Efficiency	Registration	CCD	PCIe	MPI	Disk
32	18K		12	26K	1,060	0.93	0.11	0.30	1.63	3.71	10.38
64	27K	2,809	17	38K	2,115	0.93	0.18	0.45	1.52	3.45	7.19
128	38K		23	54K	4,213	0.93	0.39	0.63	1.45	3.35	5.05
256	54K		33	77K	8,373	0.92	0.76	0.89	1.40	3.37	3.52

for each cluster size, we choose the largest image size that can be handled while satisfying the real-time constraint. Thus, the throughput measured in images per second is slightly higher than one in each configuration (larger clusters process larger images). Therefore, we measure the throughput in terms of backprojections per second in order to quantify the scaling of compute capabilities. The time spent on other stages is also taken into account when computing the backprojections per second throughput.

Table IV reports the throughput with varying node counts. The backprojection throughput is not noticeably degraded from running stages other than backprojection and their associated data transfers. In addition, near-linear speedups are observed by using more nodes, and more than 0.5 trillion backprojections per second throughput is achieved with 16 nodes. This is due to (1) multi-level parallelization of each stage and (2) careful overlapping of data transfers with computation. Each stage can become a bottleneck if not parallelized either via OpenMP or MPI. For example, if we do not parallelize 2D-correlation and interpolation in the registration step using OpenMP, the registration step will take a comparable time as backprojection. By parallelizing all stages, except for the registration linear system solver, we maintain CPU compute time at less than 4% for all stages other than backprojection.

Although we evaluate up to 16 nodes, it is projected that the throughput will scale near-linearly as more nodes are added, and that the corresponding problem size that can be handled will scale accordingly. This projection is based on our analysis tabulated in Table V, which estimates that 256 nodes will handle the input size similar to that of the high-end scenario in Table I. It is shown that backprojection will continue to dominate the compute time and data transfers will continue to be comfortably overlapped with compute. The time breakdowns denote the time spent for each component relative to the total execution time to output one image. It is shown that the proportion of compute times of stages other than backprojection will not grow to be a significant fraction. It is also shown that data transfer times (through PCIe, MPI, and disk I/O) will be kept considerably smaller than the compute time.

The compute time of each component is estimated as (FLOPS required)/((Processors' ideal peak FLOPS)$\times$(FLOP efficiency)). The FLOP efficiency of the 2D-FFTs used in the registration step is assumed to be 10%. Other stages' FLOP efficiencies are assumed to be same as that of backprojection, which is measured to be consistent across 1–16 nodes. The FLOP count of $n \times n$ 2D-FFT is computed as $10n^2 logn$. The FLOP count of bilinear interpolation is computed as $54 I_x I_y$: $I_x I_y$ bilinear interpolations per image and 54 FLOPs per interpolation. The FLOP count of CCD is computed as $20 \cdot 2 N_{corr} I_x I_y$.[6] The data transfer times are estimated assuming each node can realize 6 GB/s PCIe and , 2 GB/s MPI, and 200 MB/s disk I/O bandwidth. We assume that the interconnect has a 3D-torus topology with 2G GB/s channels.

VI. Related Work

In order to overcome the demanding computation requirements of SAR backprojection, researchers have implemented it on a cluster of FPGAs [18]. Their partitioning method is aligned with ours—partition in the output image dimension first, and in the input pulse dimension only when it is necessary. There also have been GPU implementations of SAR backprojection [19–22]. GPUs provide hardware texture lookup support that improves the backprojection performance by up to 15% at the expense of lower accuracy [22]. GPUs also provide hardware trigonometric units but their performance benefits depend on required image formation accuracy, which will be explained later in this section. ASR eliminates trigonometric functions from the inner-loop of backprojection, and it can also be applicable to GPUs.

Researchers have developed methods for reducing the computational complexity of SAR backprojection [23–25]. Typically, these methods hierarchically decimate the phase history

[6]Correlation values are incrementally computed by maintaining $\sum x, \sum y, \sum xy, \sum \|x\|^2$, and $\sum \|y\|^2$, where x and y denote complex pixel values in the current window of the current and reference images, respectively. When we move to the next pixel, the window drops N_{corr} values and obtains N_{corr} values. Updating the maintained sums for each dropped or obtained value requires 20 FLOPs.

data in the pulse dimension for localized regions of the image in a manner that maintains sampling requirements and preserves image quality. Thus, the larger image formation problem is decomposed into several smaller image formation problems each with a corresponding reduced-size data set. In such cases, traditional backprojection is utilized as a base case operation for the reduced-size data sets. Therefore, developments in the optimization of the traditional backprojection algorithm will also directly benefit the reduced-complexity algorithms. In addition, the most computationally intensive component other than backprojection introduced by these hierarchical techniques is typically the fast Fourier transform, for which efficient implementations have been studied extensively.

Strength reduction [11] can be used to reduce exponentiation to multiplications, and division/modulo to subtractions as discussed in [12]. However, the most frequent application is reduction of multiplications to additions. In comparison, our ASR targets more time consuming functions such as square root, sine, and cosine. In addition, while the standard strength reductions maintain equivalent results (modulo errors from limited floating point precision), ASR introduces errors but gives flexibility of trading off accuracy and performance by controlling the block size.

Trigonometric functions are typically computed by polynomials derived from the Chebyshev approximation, whose coefficients are similar to those of Taylor polynomials but provide a near optimal solution (i.e., the maximum error is very close to the smallest possible for any polynomial of the same degree) [17]. ASR also exploits polynomial approximations but is differentiated as follows. First, ASR can achieve a similar accuracy with fewer floating point operations provided that arguments to the trigonometric functions vary little (relative to π) over loop iterations. Second, reducing arguments to a specific range (e.g., [-π, π]) is often the most time-consuming and accuracy-sensitive part of trigonometric function calculation [17], especially when the magnitude of arguments can be large. For example, in SAR backprojection, we need to reduce arguments in double precision in the baseline implementation. Hardware trigonometric functional units implemented in accelerators also typically assume that input arguments are within the range [-π, π] [26]. Consequently, we cannot avoid the complexity of argument reduction even with hardware trigonometric units. In contrast, ASR can achieve a high accuracy mostly using single precision operations for even arguments with large magnitude. CORDIC [27] is another method of computing trigonometric functions, but it is used only in simple hardware without multipliers and floating point units. Similar to Chebyshev-approximation-based approaches, CORDIC also requires arguments to be in a certain range (e.g., [-π/2, π/2]).

Recently, there has been a growing interest on offloading computation to many-core wide-vector coprocessors such as Intel® Xeon Phi™ or GPUs to improve single-node performance [28–30]. Similar to our work, a part of the application with limited parallelism is typically run on CPUs, while the highly data-parallel portion is offloaded to the coprocessor.

VII. Conclusion

This paper demonstrated that the challenging computational requirements of SAR backprojection can be efficiently handled by the combination of algorithmic improvements, data movements optimized for the underlying architecture, and systematic exploitation of parallel resources available in modern compute platforms.

Algorithmic improvements from our approximate strength reduction significantly reduced the computational cost of SAR backprojection. Our method can be applied to other image reconstruction applications that share similarities such as frequent use of trigonometric functions and interpolation. This generalization is not limited to other types of radar applications but also applicable to medical imaging applications with similarly challenging computational requirements. For example, although purposely omitted to focus on SAR, we have applied the ASR method to beamforming used in ultrasound imaging, thereby achieving a 5$\times$ speedup. ASR also exemplified that understanding the mathematical structure of the problem plays a key role in devising domain or application specific algorithmic optimizations. Applying ASR was motivated by geometric properties of SAR imaging via backprojection: the distance from the radar to pixels varies little and the variation has a regular pattern that can be approximated by quadratic polynomials.

Improvements in modern many-core architectures were also demonstrated to be beneficial in addressing computationally challenging applications. The hardware gather support reduced the cost associated with delivering data from irregular locations and helps effectively utilize wide vector units. This paper showcased the synergy of conventional CPUs and emerging many-core coprocessors with a high compute intensity. Many-core coprocessors offloaded the backprojection stage with a high compute-to-communication ratio and large degree of parallelism. Intel® Xeon® processors complemented this by handling portions with relatively low compute-to-communication ratios and orchestrating data transfers to be overlapped with computation.

As a result of our combined optimization efforts, over 0.5 trillion backprojections per second throughput was realized with 16 nodes. When combined with hierarchical backprojection techniques, we believe our optimizations will render computationally challenging SAR imaging via backprojection considerably more affordable.

Acknowledgements

The authors would like to thank Daniel Campbell and the Georgia Tech Research Institute team for the original implementation of backprojection-based SAR and help on understanding radar algorithms. We thank Nicolas Vasilache, Richard Lethin, and Roger Golliver for discussion on the SAR implementation in its initial stage. We also thank Pradeep Dubey, Robert Hanek, Ravi Narayanaswamy, Rakesh Krishnaiyer, and the Intel Endeavor team. This research was, in part, funded by the U.S. Government under contract number HR0011-10-3-0007. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government.

References

[1] P. Kogge, K. Bergman, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau, P. Franzon, W. Harrod, K. Hill, J. Hiller, S. Karp, S. Keckler, D. Klein, R. Lucas, M. Richards, A. Scarpelli, S. Scott, A. Snavely, T. Sterling, R. S. Williams, and K. Yelick, "ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems," 2008, `www.cse.nd.edu/Reports/2008/TR-2008-13.pdf`.

[2] F. Niu, B. Recht, C. Re, and S. J. Wright, "HOGWILD!: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent," in *Advances in Neural Information Processing Systems (NIPS)*, 2011.

[3] M. Wolfe, *High Performance Compilers for Parallel Computing*. Addison-Wesley, 1996.

[4] A. Nguyen, N. Satish, J. Chhugani, C. Kim, and P. Dubey, "3.5-D Blocking Optimization for Stencil Computations on Modern CPUs and GPUs," in *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2010, pp. 1–13.

[5] M. A. Richards, *Fundamentals of Radar Signal Processing*. McGraw-Hill, 2005.

[6] D. P. Campbell, D. A. Cook, and B. P. Mulvaney, "A Streaming Sensor Challenge Problem for Ubiquitous High Performance Computing," http://www.ll.mit.edu/HPEC/agendas/proc11/Day1/Session_1/1025_Campbell.pdf, 2011.

[7] D. C. Munson, Jr., J. D. O'Brien, and W. K. Jenkins, "A Tomographic Formulation of Spotlight-Mode Synthetic Aperture Radar," *Proceedings of the IEEE*, vol. 71, no. 8, pp. 917–925, 1983.

[8] C. V. Jakowatz, Jr., D. E. Wahl, and D. A. Yocky, "Beamforming as a Foundtation for Spotlight-mode SAR Image Formation by Backprojection," in *SPIE 6970*, 2008.

[9] A. Isola, A. Ziegler, T. Koehler, W. J. Niessen, and M. Grass, "Motion-compensated iterative cone-beam CT image reconstruction with adapted blobs as basis functions," *Physics in Medicine and Biology*, vol. 53, p. 6777, 2008.

[10] E. Hansis, J. Bredno, D. Sowards-Emmerd, and L. Shao, "Iterative Reconstruction for Circular Cone-Beam CT with an Offset Flat-Panel Detector," in *IEEE Nuclear Science Symposium Conference Record (NSS/MIC)*, 2010, pp. 2228–2231.

[11] J. Cocke and K. Kennedy, "An Algorithm for Reduction of Operator Strength," *Communications of the ACM*, vol. 20, no. 11, pp. 850–856, 1977.

[12] F. E. Allen, J. Cocke, and K. Kennedy, "Reduction of Operator Strength," *Program Flow Analysis*, pp. 79–101, 1981.

[13] M. D. Desai and W. K. Jenkins, "Convolution Backprojection Image Reconstruction for Spotlight Mode Synthetic Aperture Radar," *Image Processing, IEEE Transactions on Image Processing*, vol. 1, no. 4, pp. 505–517, 1992.

[14] C. V. Jakowatz, Jr., D. E. Wahl, D. C. G. Paul H. Eichel, and P. A. Thompson, *Spotlight-Mode Synthetic Aperture Radar: A Signal Processing Approach*. Kluwer Academic Publishers, 1996.

[15] G. M. Amdahl, "Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities," in *AFIPS spring joint computer conference*, 1967, pp. 483–485.

[16] A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman, *Compilers: Principles, Techniques, and Tools*. Addison-Wesley, 2007.

[17] J.-M. Muller, *Elementary Functions: Algorithms and Implementation*, Birkhauser.

[18] B. Cordes and M. Leeser, "Parallel Backprojection: A Case Study in High-Performance Reconfigurable Computing," *EURASIP Journal on Embedded Systems*, vol. 2009, p. 1, 2009.

[19] T. D. R. Hartley, A. R. Fasih, C. A. Berdanier, F. Özgüner, and U. V. Catalyurek, "Investigating the Use of GPU-Accelerated Nodes for SAR Image Formation," in *IEEE International Conference on Cluster Computing (CLUSTER)*, 2009, pp. 1–8.

[20] A. R. Fasih and T. D. R. Hartley, "GPU-accelerated synthetic aperture radar backprojection in CUDA," in *IEEE Radar Conference*, 2010, pp. 1408–1413.

[21] R. Portillo, S. Arunagiri, P. J. Teller, S. J. Park, L. H. Nguyen, J. C. Deroba, and D. Shires, "Power versus Performance Tradeoffs of GPU-accelerated Backprojection-based Synthetic Aperture Radar Image Formation," *Proceedings of SPIE, Modeling and Simulation for Defense Systems and Applications VI*, vol. 8060, 2011.

[22] T. M. Benson, D. P. Campbell, and D. A. Cook, "Gigapixel Spotlight Synthetic Aperture Radar Backprojection Using Clusters of GPUs and CUDA," in *IEEE Radar Conference*, 2012, pp. 853–858.

[23] M. Soumekh, *Synthetic Aperture Radar Signal Processing with MATLAB Algortihms*. Wiley Interscience, 1999.

[24] S. Xiao, D. C. Munson, Jr., S. Basu, and Y. Bresler, "An N^2logN Back-Projection Algorithm for SAR Image Formation," in *Asilomar Conference on Signals, Systems and Computers*, 2000, pp. 3–7.

[25] D. E. Wahl, D. A. Yocky, , and C. V. Jakowatz, Jr., "An implementation of a fast backprojection image formation algorithm for spotlight-mode SAR," *Algorithms for Synthetic Aperture Radar Imagery XV*, vol. 6970, p. 69700H, 2008.

[26] NVIDIA, "OpenCL Programming Guide for the CUDA Architecture," www.nvidia.com/content/cudazone/download/OpenCL/NVIDIA_OpenCL_ProgrammingGuide.pdf, 2009.

[27] J. E. Volder, "The CORDIC Trigonometric Computing Technique," *IRE Transactions on Electronic Computers*, pp. 330–334, 1959.

[28] J. R. Humphrey, D. K.Price, K. E.Spagnoli, A. L. Paolini, and E. J. Kelmelis, "CULA: hybrid GPU accelerated linear algebra routines," in *SPIE Conference Series*, vol. 7705, Apr. 2010.

[29] S. McIntosh-Smith and J. Irwin, "The best of both worlds: Delivering Aggregated performance for high-performance math libraries in accelerated system," in *ISC*, 2007, pp. 331–340.

[30] M. Deisher, M. Smelyanskiy, B. Nickerson, V. W. Lee, M. Chuvelev, and P. Dubey, "Designing and dynamically load balancing hybrid lu for multi/many-core," *Computer Science-Research and Development*, vol. 26, no. 3-4, pp. 211–220, 2011.