Parallel Processing Techniques for the Processing of Synthetic Aperture Radar Data on FPGAs

William Chapman, Sanjay Ranka, Sartaj Sahni, and Mark Schmalz

University of Florida, Department of CISE, Gainesville FL 32611-6120

Uttam K. Majumder

Air Force Research Laboratory Sensors Directorate
2241 Avionics Circle, Wright-Patterson AFB, OH 45433

Abstract

This paper presents a design for the parallel processing of synthetic aperture radar data using one or more Field Programmable Gate Arrays (FPGAs). Our design supports real-time computation of a two-dimensional image from a matrix of echo pulses and their corresponding response values. Components of this design include: (a) central processing pipeline to perform back projection calculations, (b) pre-fetch cache to minimize external memory access latency, (c) memory bridge that serves as the primary on-chip storage for pulse data, and (d) a pixel queue to direct image data in and out of the pipeline. Design parameters may be adjusted to achieve optimum performance, and multiple instances of this design may be replicated on-chip to achieve prespecified performance objectives. We provide a complexity analysis as a function of the input and output parameters. Simulation results based on an implementation of this design show that our design achieves 160 GOPs per instance on a simulated Altera Stratix III EP3SL150 FPGA, and scales well for output image size ranging from 500 x 500 pixels to 5,000 x 5,000 pixels.

Introduction

The use of radar for area surveillance has been of keen interest to scientists and engineers in fields such as medical tomography, meteorology and geology [4]. One such application, radar based terrain mapping, has recently become the subject of

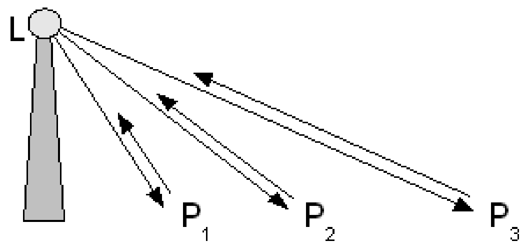


Figure 1: A radar pulse is transmitted, and response intensity is measured as a function of time. Responses from more distant objects traverse a longer path and will return to the sensor at a later time.

increased interest in the field of high- performance parallel computing. Terrain mapping involves the use of radar devices to generate high-resolution images of one or more ground objects (e.g., land or forest), as well as manufactured object(s). As shown in Figure 1, a radar transmitter at known location *L* broadcasts a pulse at time T_0. Responses from objects within the sensor's field of view are measured as a function of time. Reflective objects further from the pulse location will cause echoes that arrive later than echoes for proximal objects. In the example shown, the response for P_1 will appear at an earlier time than P_2, and P_2 will appear earlier than P_3, and so forth. The time these responses arrive can be readily computed by dividing the distance between *L* and P_x by the speed of light *c*:

$$\left(\frac{1}{c}\right)\sqrt{(P_{ix}-L_x)^2+(P_{iy}-L_y)^2+(P_{iz}-L_z)^2}$$

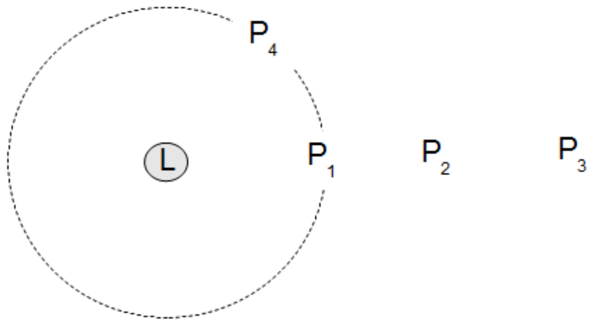


Figure 2. Configuration of Figure 1 shown in top view. An object at P_4 is the same distance from L as P_1, and will have the same response time.

Unfortunately, while prediction of arrival time for a given reflected pulse component is simple, the inverse computation that establishes an object map from multiple pulses is more challenging, as shown below.

For example, in Figure 2, an additional point at P_4 will have the same distance, and response time, as the point at P_1. Given the response data for only one pulse, it is impossible to generate a physically accurate representation of the original image, because reflective objects could only be echo-located to the nearest range band (shown in Figure 2 as concentric circles centered at the pulse emitter).

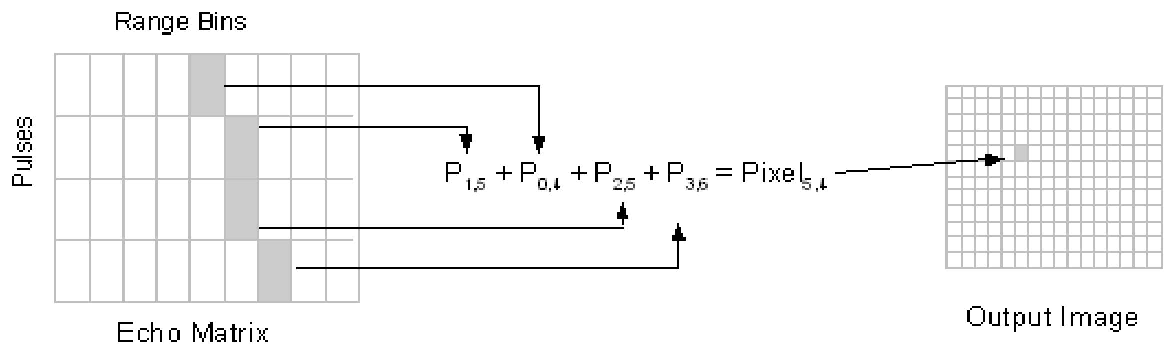
Given multiple views of a ground object, synthetic aperture radar provides a method of reconstructing the object map, by measuring and mathematically combining multiple return pulses at multiple locations over time, to increase effective aperture. In the example of Figures 1 and 2, the sampling process would be repeated with L in several different locations around P_1, P_2, and P_3. This helps resolve spatial ambiguity between P_1 and P_4, allowing reflective objects to be more accurately and precisely mapped to their correct locations. In addition, the effect of sampling error (e.g., due to interference and environmental factors) is reduced by the averaging inherent in repetitive sampling, and by superposition during reconstruction. However, the rendering of output images from pulse response data is computationally more expensive as the number of pulses increases. For example, a 500 x 500 pixel SAR image took over four hours to compute on a consumer-grade Hewlett Packard 2 GHz Centrino processor.

Although parallel architectures have greater availability and utility, the cost of developing high performance sequential hardware has remained substantial. Also, since sequential processing solutions for SAR image reconstruction tend to be I/O-bound primarily and compute-bound secondarily, parallel computing is further indicated. We have found that Field Programmable Gate Arrays (FPGAs), which support parallelism through replication of spatial algorithms, are useful for SAR reconstruction. In simulations with Altera Quartus II software, on an Altera Stratix III EP3SL150 FPGA, we have obtained 228 seconds execution time from one instance of our design. Increasing the number of instances operating in parallel resulted in a linear decrease in execution time. We next consider the SAR image reconstruction algorithm employed in this study.

The Filtered Backprojection Algorithm

Several published algorithms are available for reconstructing SAR imagery [1,3]. The filtered back projection algorithm is known for its large computational cost and high quality output images [1] The algorithm, whose details are beyond the scope of this paper, is overviewed as follows.

Firstly, the time dimension of the response data for each pulse is divided into range bins. This typically occurs at collection time, to convert the real time response data stream to a discrete format amenable to storage in memory. Secondly, for each pixel in the output image, for every pulse in the response data: (i) the spatial distance of that pixel from the pulse location is computed; (ii) the range bin that corresponds to this distance is determined; and (iii) the contribution of this range bin is summed with the pixel's current value.



Range Bins
Pulses
Echo Matrix
$P_{1,5} + P_{0,4} + P_{2,5} + P_{3,6} = \text{Pixel}_{5,4}$
Output Image

Figure 3. Data access pattern for the computation of a single pixel. The corresponding range bins for each of the four pulses are summed to produce the value of the output pixel.

We note that that this representation of the filtered back-projection algorithm describes only the data access pattern and the algorithmic complexity of computational and I/O operations. Implementational details are given in [1] and illustrated schematically in Figure 5.

Benefits of FPGA Implementation

As noted previously, the back-projection algorithm is computationally expensive because it requires every pixel to be summed with the contribution of every pulse. In practice, given a large image and a large pulse data matrix, this process can yield high latencies. Fortunately, the process of rendering pixels is inherently data-parallel. As a result, it is possible to achieve a proportional decrease in overall latency by replicating available hardware: low resolution times can be realized via a high degree of parallelism. Unfortunately, modern microprocessors are complex mainly-sequential devices, designed to be applicable in the solution of virtually any mathematical problem. These general-purpose capabilities consume large amounts of space on-chip, which tends to be wasted when a microprocessor is used to perform simple calculations. Since most microprocessors are inherently sequential, they tend to contribute little to the performance of fine-grained parallel implementations, as predicted by Amdahl's Law.

Field Programmable Gate Arrays are fundamentally different from microprocessors, as FPGAs allow designers to define a logical relationship between hardware inputs and hardware outputs in terms of spatial relationships on-chip, rather than in the temporal dimension as sequential instructions. This supports parallel computation of operations that are not serially dependent.

In implementing SAR image reconstruction via a back projection algorithm, the same set of operations are performed for every (*R*,*P*) pair, where *P* is a pixel in the reconstructed image, and *R* is the range bin in a given pulse that corresponds to *P*. These operations can be implemented in hardware as shown in Figure 5.

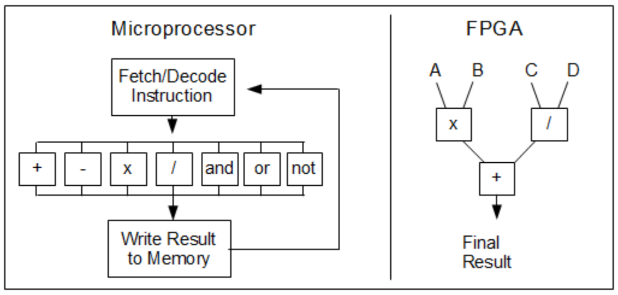


Figure 4. The microprocessor implementation of (A * B + C / D) can only perform one operation at a time, resulting in three iterations through the execute cycle. In addition, the unused subtraction and Boolean operations of the processor are wasted. The FPGA implementation computes multiplication and division operations in parallel, requires only two operation delays, and uses the minimal hardware necessary to implement the algorithm.

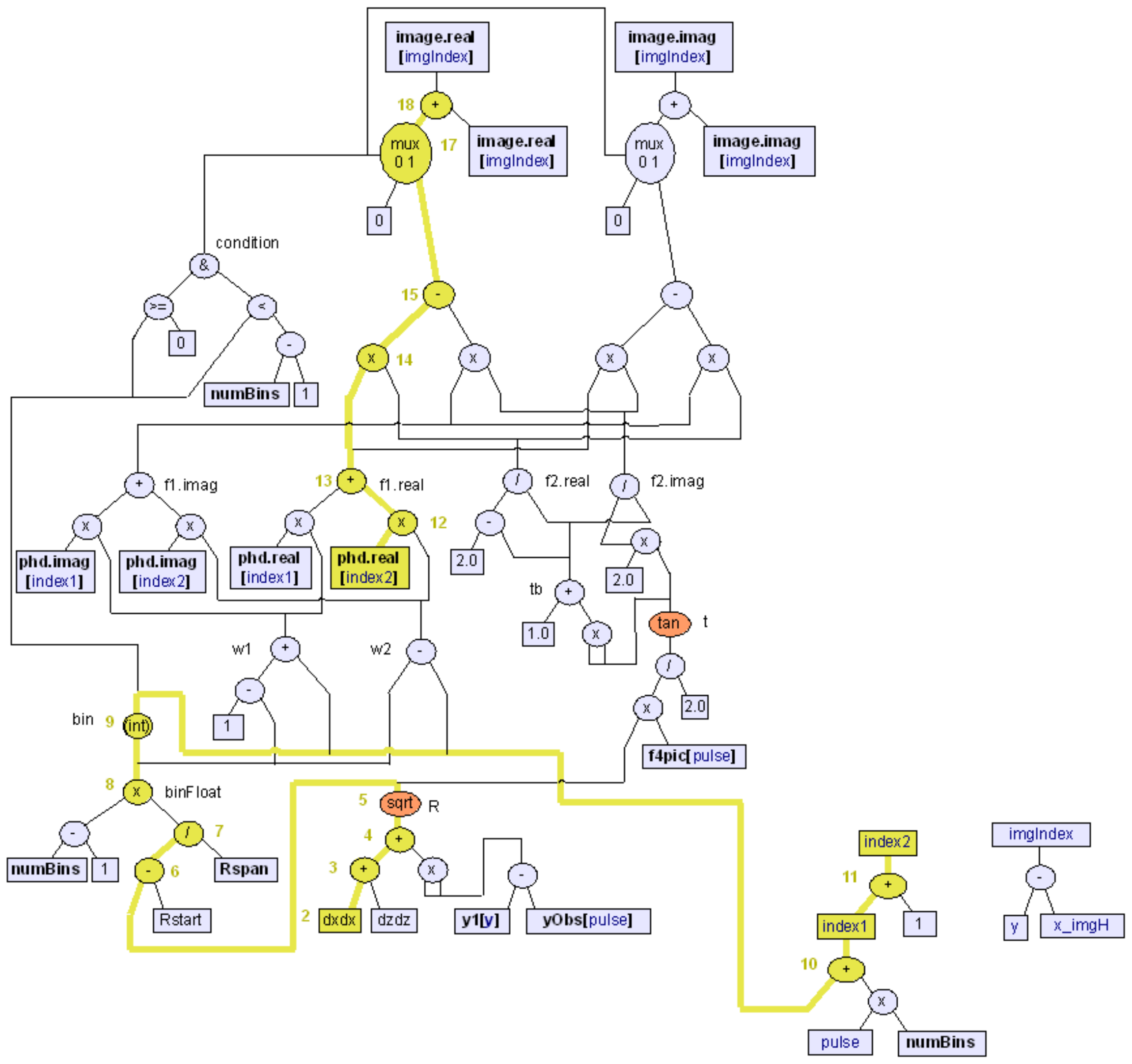


Figure 5. Dataflow diagram of algorithm for computing the contribution of a single pulse for a given reconstructed output pixel, where computations are performed natively in hardware.

Effect of Response Matrix Size

In addition to computational complexity, the back-projection algorithm presents significant data movement challenges. Both the pulse response matrix and the output image can be very large, and most FPGA chips cannot store these structures in their entirety. However, in order to sum the contribution of a given pulse to a pixel, both the correct range bin and pixel must be on-chip at the right time. Fortunately, the data access pattern of the back projection algorithm ensures two properties that are helpful in reducing the cost of data input and output:

Property 1. Let function *f*(*P*, *X*) map pixel *X* to its corresponding range bin in pulse *P*. As *X* is varied from left to right, or right to left, across any row of the output image, one of the following will be true:

1. *f* (*P*,*X*) will be increasing.
2. *f* (*P*,*X*) will be decreasing.
3. *f* (*P*,*X*) will be increasing up to some pixel *Y*; beyond *Y*, *f* (P,*X*) will be decreasing.
4. *f* (*P*,*X*) will be decreasing up to some pixel *Y*; beyond *Y*, *f* (P,*X*) will be increasing.

This property holds as X is varied across any column of the output image from top to bottom, or bottom to top, as shown in Figure 6. Recall that range bin access is directly proportional to the distance of a pixel from its corresponding pulse. In the simple cases (L_1 and L_3), as X is varied along any straight line on the image, the distance between X and the pulse will increase or decrease. In the case where the pulse location is not beyond the endpoint of the line (L_2 is directly above the line), then at some point Y, the distance between Y and the pulse location will be at a minimum. From this property, it follows that if only a subregion of the output image is considered, only a proportionally smaller subregion of the echo matrix will be needed to render the output image subregion.

Property 2. Let the function $N(P, S)$ be the number of range bins for a given pulse required to completely generate a square subregion S of the output image. For any square subregion S and pair of pulses (P_1 , P_2), we have

$$N\left(P_{1,}S\right) \leq \sqrt{(2)}N\left(P_{2,}S\right).$$

This property can be explained by recalling that the range bin needed for a given pixel is linearly related to its distance from the pulse location. For any two points, the factor relating the difference in their physical locations to the difference in their corresponding range bins is constant. In particular, this difference D equals the total distance sampled divided by the number of range bins. Observe that the least number of range bins will be required when the look angle is either parallel or perpendicular to the base of the square subregion. Likewise, the maximum number of range bins is required when the look angle equals 45^O, such that

$$Bins = BD,$$

where B denotes the length of the square base. For the 45^O case,

$$Bins = BxDiagonalLength = B\sqrt{(2)}D$$

The final step is proven using the Pythagorean Theorem.

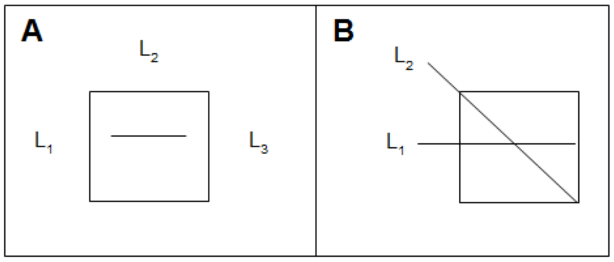


Figure 6. (A) Three pulse locations, L_1, L_2, and L_3 are shown above. As a pixel is varied from left to right along the row indicated by the horizontal line, range bin access: (L_1) increases; (L_3) decreases (L_2) decreases then increases. (B) A square subregion is resolved from two pulses at location L_1 and L_2. The look angles of each pulse are indicated by a solid line. Range bin access is minimized when the look angle is parallel to the base of the subregion (L_1) and maximized when it is 45^O (L_2).

Design

The proposed algorithm design partitions the output image into a set of small tiles. Each tile is square, having width and height K. The process of generating an output image can then be decomposed into the process of generating each of the $(N/K)^2$ tiles independently.

The back projection algorithm states that each pixel of an output image is computed as the sum of the contributions of each pulse in the echo matrix, where the contribution of a pulse for a given pixel is the value of the range bin corresponding to that pixel. To reconstruct the image, the correct range bin of each pulse must be paired with the corresponding pixel or pixels in the output image.

In the proposed design, shown in Figure 9, the pairing of a single pulse range bin and pixel value occurs in each box labeled Projection Element (PE). PEs are arranged in a vertical pipeline having width W and depth D. Pixels are stored in an external memory device labeled Output Image, and are loaded into the Read Pixel Queue, then processed through the pipeline. At each pipeline stage, a pixel is paired with the pulse that has been loaded into that stage. After leaving the pipe, the processed pixels are stored in a Write Pixel Queue, where they are transferred back to the external memory device.

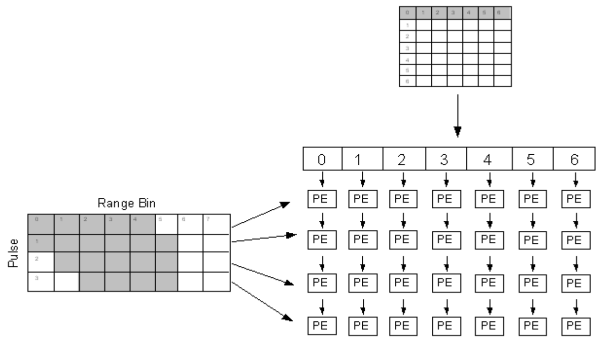


Figure 7. The first design considered consisted of a systolic array of processing elements (PEs). The width of this array is equal to the width of the image tile, and the height is configurable. Rows are passed vertically through the pipe as shown.

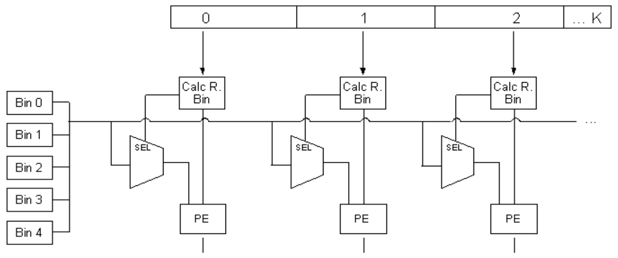


Figure 8. A second considered design is similar to that shown in Figure 7, except that range bins are no longer exchanged in a horizontal network. In this design, the range bins from a given pulse are stored in a shared memory shown at the left of the figure. Every processing element (PE), has a connection to every range bin in its shared memory.

The size of the echo data matrix tends to confound this type of data movement. When a pixel reaches a given stage of the pipeline, the corresponding range bin must be loaded into local memory so its contribution can be summed with the existing pixel value. Due to long latencies in external memory accesses, performance would be greatly improved by loading the echo matrix into an on-chip cache. However, the echo matrix is significantly larger than available on-chip memory, so only a small portion of the echo matrix can be stored on the FPGA. Fortunately, range bin access within each pulse is predictable for a *K* x *K* tile. Within each pulse, the number of range bins needed to compute each tile is proportional to *K*/*N*, where *N* is the size of the output image. In Figure 9, for example, only the shaded region of the Echo Matrix is used to generate the pixels shaded in the Output Image.

Several designs were considered to effectively exploit this access pattern. One such design consisted of a systolic array of processing elements as shown in Figure 7. Each row of pixels from an output image tile would pass through the array vertically, while the necessary range bins would move horizontally. The movement of a row to the next level of the array would cause processing elements to exchange range bins, ensuring the correct bin matched up with the correct pixel. This design effectively exploited the sequential nature of echo matrix data access patterns among a set of images. Range bins could efficiently be shifted from one column to the next to meet the demands of each output image row.

However, this design had a serious shortcoming, namely, it ignored the fact that the number of range bins needed for each pulse does not necessarily have a one-to-one relationship with the width of the tile. In the case where the number of range bins exceeds with the width of the tile, not all bins would be needed to process every row. In the case where the sample look angle is perpendicular to the row, and the sample distance is much larger than the width of the tile, only a single range bin is needed to render the entire output row. By design, there was no mechanism available for storing the unused bins for use by subsequent rows. When the inclusion of such a storage mechanism became obvious, we improved the interconnection network, to increase performance of our design.

The second design (shown in Figure 8) considered addressed this question directly. Here, we replaced

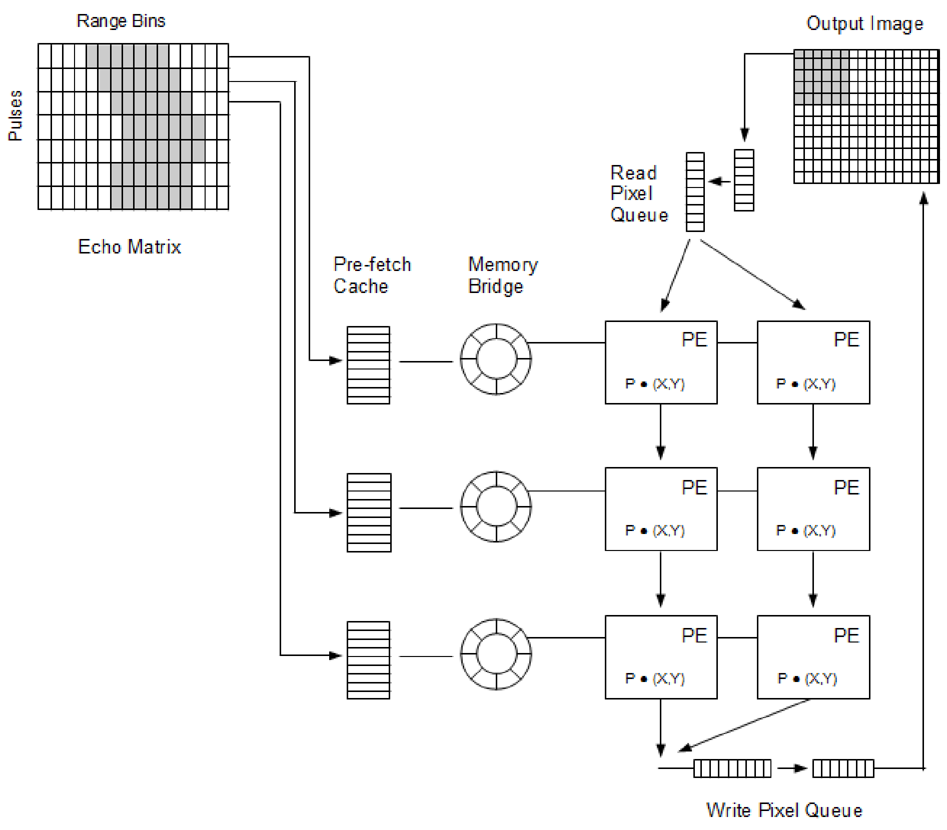


Figure 9. The proposed design consists of one or more vertical pipelines. Pixels pass through a pipeline from top to bottom, meeting a new pulse at each level. Processing elements are connected to one bucket of a circular memory bridge. If the needed range bin is not in that bucket, the processing element can assert a signal requesting the memory bridge to rotate until the needed bin is in the correct bucket. The pipe width can be adjusted to increase pipe throughput.

the horizontal component of the systolic array with a shared memory for each pulse. This memory is connected to each processing element via a multiplexer, allowing the processing element to select which bin is needed for the pixel being computed. This design had the advantage of achieving optimal throughput, as no time would be needed for elements to exchange data with neighbors, but consumed significant on-chip real estate, as FPGAs are not designed to act as crossbar switches. This design also does not exploit the sequential nature of range bin access across neighboring pixels of the output image. In contrast, an optimal design would use this predictability to achieve a more efficient interconnection network.

The current design, shown in Figure 9, replaces the shared memory component with a circular memory bridge (similar to a round-robin queue). Prior to computation of an output (reconstructed) tile, each memory bridge is loaded with the range bins that will be needed to compute the entire tile. Each processing element is then connected to a single storage location, or bucket, on the circular structure. If a processing element is not connected to a bucket holding the correct range bin, it can request a clockwise or counterclockwise rotation. By Properties 1 and 2, it can be shown that if adjacent pixels are passed through the pipeline, then rotation distance will be small relative to the size of the memory bridge. In fact, for a single row of the image, rotations will always occur in the same direction with changes to that direction occurring at most one time. (See the first property above.) Not more than one full rotation is needed to compute any given row. This design exploits both the locality and predictable pattern of range bin access and uses an interconnection network that achieves an efficient trade-off between size and speed.

An additional characteristic of the design shown in Figure 9 is the pre-fetch cache. This feature masks the latency of external memory by permitting the loading of the memory bridge to be executed in parallel with the computation of a tile. If the memory bridge were loaded directly from external memory, then the pipeline would need to be stalled for several cycles until the load was completed. Using the prefetch-cache, it is possible to transfer data from cache to the memory bridge in as little as one clock cycle. As long as the time required to process a tile is greater than the time required to load the cache, the memory access delays are effectively masked from the total latency of the design. An analogous prefetch cache is also used to mask memory latency of the pixel queue.

Design Analysis and Parameters

Complexity of the design is measured in terms of time, space, and IO complexity.

Latency

In order to derive an expression for the latency of rendering an N x N image, it is necessary to consider the two operations that are occurring in parallel during the steady state operation of this design: (1) loading the prefetch cache with a new pulse, and (2) rendering a *K* x *K* image tile with the current pulse. We write the former operation as a function of *K*:

$$L_{MB} = L_{memstart} + CKL_{mem} \qquad (1)$$

where $L_{memstart}$ and L_{mem} refer to the startup time and access speed of the host memory device. The constant *C* is determined by the sampling resolution and size of each range bin, which are held constant for the purpose of this analysis. We then write the latency of rendering a single *K* x *K* image for one pulse, as follows:

$$L_{Render} = L_{fill} + max\left(\frac{K^2}{W}, K^2 L_{mem}\right) \qquad (2)$$

where L_{fill} is the initial hardware time required to start the pipe after reloading the memory bridge, and K^2 is the tile size. The maximum function in this equation reflects the fact that the rate at which pixels is limited by both the processing speed and the memory access speed. Since each pixel must be read from and written to memory, the rate at which pixels can move through the pipe is bounded below by K^2 / L_{mem}. The time to process these pixels is K^2 / *W*, where *W* is the width of the pipe. Alternatively, *W* can denote the number of pixels that are rendered in parallel.

In steady state operation, the larger of L_{MB} and L_{Render} will dominate the latency of rendering a single *K* x *K* tile for a given pulse. Repeating this process for all tiles in the image, and distributing the load across *J* implementations each with a pipe depth of *D*, the total time required to generate the entire *N* x *N* image using *A* pulses is:

$$L = \frac{A}{JD}\left(\frac{N}{K}\right)^2 max\left(L_{MB}, L_{Render}\right) \qquad (3)$$

Space

The design implemented in Figure 9 consists of four central components: a processing element, a memory bridge, a prefetch cache, and a pixel queue. The prefetch cache and memory bridge are of equal size, and the pixel queue is negligibly small compared to the other elements. Under this assumption, we express the space consumed by *J* implementations each with a pipe depth *D*, pipe width of *C*, and memory bridge size *CK*, as follows:

$$S = JD\left(WS_{PE} + 2CK\right) \qquad (4)$$

Memory Access - IO

Firstly, we observe that memory access requirements to render a single *K* x *K* tile on a single pulse, then write that pulse back to memory, is given by:

$$IO_{tile} = 2\mathrm{K}^2 + CK \qquad (5)$$

Using *J* parallel implementations of the design shown in Figure 9, the IO requirements of rendering one tile on *D* pulses then becomes:

$$IO_{par} = J\left(2K^2 + DCK\right) \qquad (6)$$

This equation reflects the fact that increasing the pipe depth *D* results in a linear increase in echo matrix IO requirements for each pass of a tile through the pipe. Increasing *J*, the number of parallel implementations, produces a linear increase in the both echo matrix IO and image IO.

Design Analysis - Optimizing Parameters

Minimizing the Value of J

From Equations 3 and 4, we note that pipe depth D and degree of parallelism *J* are both inversely related to latency, and directly related to space. In fact, these values appear as a product *JD* in both equations. This is intuitive, as greater parallelism can be obtained by lengthening the pipe, or operating multiple pipes simultaneously. Regarding only spatial and temporal efficiency, neither option is preferable to the other. However, when Equation 6 is considered, it is clear that increasing D results in lower IO requirements. For that reason, *J* should always be chosen to be the lowest possible value. Figure 11 in the section titled Partitioning the Echo Matrix - Facilitate Parallel IO describes an architecture where *J* = 1 even when the design would be distributed across multiple FPGA boards.

Selecting the Right Value of K

We have observed from Equation 3 that when chip real estate is not bounded above, latency is inversely related to tile size *K*. In an environment where space is unlimited, it is then preferred to let *K* = *N* and process the entire image in a single tile. In practice, however, available space on an FPGA is finite. From Equation 4, it can be seen that, for

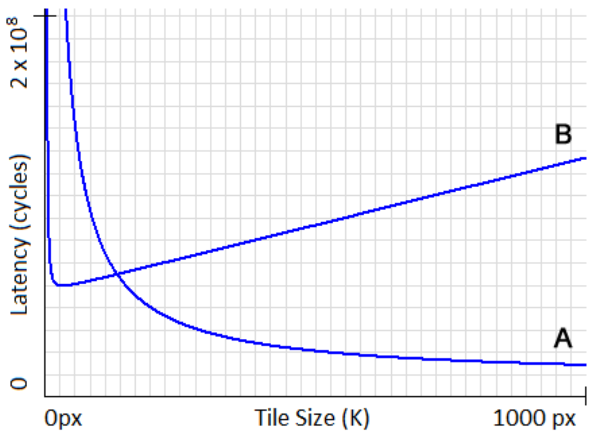


Figure 10. As K is increased toward N, (A) latency limited by memory access decreases. (B) latency limited by processing speed achieves a minimum and then increases linearly.

some fixed *S*, increasing the value of *K* decreases the value of *JD*. This reduces parallelism and increases latency. It is necessary, then, to find an optimal value of *K* such that latency is minimized.

In selecting the optimal value of *K*, we optimize each of the limiting factors described in Equation 3. This separates the analysis into a distinct case for each of the limiting factors: processing speed, image memory access, and echo matrix memory.

The first two cases can be treated together, as neither the processing speed nor access speed of the memory device can be improved by changing *K*. We define the larger of these latencies to be the latency of rendering 1 pixel on 1 pulse and denote it L_{pix}. Echo matrix memory access is ignored, as it is the faster of two parallel operations. Then, treating the FPGA chip space available as a constant and substituting Equation 4 in to Equation 3, the expression for latency is:

$$L = \left(WS_{PE} + 2CK\right)\frac{A}{S}\left(\frac{N}{K}\right)^2\left(L_{fill} + K^2 L_{pix}\right) \qquad (7)$$

When this expression is expanded, its highest and lowest order terms have order 1 and -2 respectively. We observe that the graph of this function decreases quadratically toward some minimum value, and then increases linearly as *K* approaches *N*. The optimal value of *K* occurs at the minimum of this graph.

The third case assumes that echo matrix memory access is the limiting factor in throughput. Echo matrix memory access requirements can be reduced by increasing the value of *K*, which decreases the value of *JD* and the number of prefetch caches that are loading data from memory. By treating chip space as a constant and substituting Equation 4 into Equation 3, we obtain:

$$L = \left(WS_{PE} + 2CK\right)\frac{A}{S}\left(\frac{N}{K}\right)^2\left(L_{memstart} + CKL_{mem}\right) \quad (8)$$

This expression decreases on the interval (0, *N*], reflecting the fact that echo matrix memory access is minimized when the image is processed as a single tile. This is consistent intuitively, because rendering an image in tiles causes some areas of the echo matrix to be fetched multiple times, while rendering a single tile results in only one pass over the echo matrix.

We note that in the third case, Equation 8 is a decreasing function that is greater than Equation 7 as the optimal tile size dictates by cases 1 and 2. Recalling that Equation 7 increases for values of *K* larger than its optimum, the optimal value of *K* for case 3 occurs at the point of intersection.

Selecting the Optimal Value of W

In selecting a pipe width *W*, we first observe that increasing the pipe width allows multiple pixels to be sent down the pipe in parallel. Increasing *W* also has a much lower spatial cost than increasing pipe depth, because it does not require the addition of memory bridges. However, an increase in *W* only reduces the observed latency of the projection elements (PEs), and does not impact memory access. An increase in *W* is then helpful when pipe processing latency (K^2 / *W*) is the limiting factor in device performance,. However, in the case where image or echo matrix memory dominates the expression for latency, increasing *W* does not result in any additional throughput.

In general, an optimal pipe width can be obtained by increasing *W* until echo matrix or image memory access limits device throughput.

Partitioning the Echo Matrix - Facilitate Parallel IO

In the event of large memory accesses, the performance of external memory can be improved by partitioning the echo matrix across multiple independent memory devices. This partitioning, shown in Figure 11 is done on the pulses (rows) of the echo matrix. Each set of pulses is then sent to a smaller memory device which is directly connected to the algorithm instances that act on the specified collection of pulses. Data access within a partition shares its communication bus only with other implementations in the same partition. As a result, the access requirements on each memory device are inversely related to the number of partitions. We have found that there is no theoretical limit to the number of partitions. Due to the fact that only read operations are performed on the echo matrix, it is possible to increase the number of partitions beyond the number of pulses. A single pulse can thus be copied to multiple partitions. By continuing to parallelize the memory access in this manner, it is possible to increase throughput of the external memory architecture by an arbitrary factor. Memory access to the echo matrix is thus scalable, as the requirements on any one memory device does not depend on the number of pulses or implementations.

The I/O requirements of the image data may also be improved by arranging algorithm instances in a pipeline rather than connecting them to a single shared memory. In this arrangement, the I/O requirement of each implementation does not change. Each image tile must be read from external memory, processed, and then written back. However, instead of reading and writing from a single memory device, read operations are performed from the preceding instance in the pipe

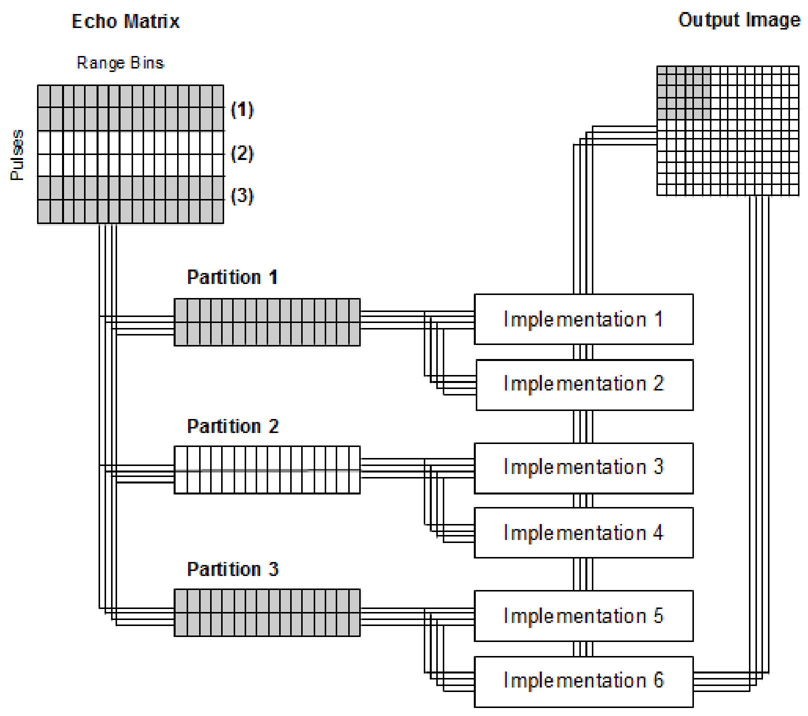


Figure 11. The echo matrix may be distributed across multiple memory devices to reduce IO costs when many instances of the algorithm operate in parallel. In addition, each implementation may receive its pixels from a neighboring implementation, rather than a single shared memory. Each box marked Implementation is an independent instance of the design illustrated in Figure 9.

and writes are performed to the next instance. Only the first and last instances have connections to the shared memory. As a result, access requirements on this device do not increase as the number of implementations in increases.

The external memory architecture described above ensures that, from the perspective of a single memory device, the I/O requirements of the proposed design scale to an arbitrary number of parallel instances and pulses.

Design Considerations for High Performance

The analysis provided in the previous three sections has established the following two important facts:

(1) An increase in the amount of chip space available results in a linear decrease in the latency of processing an image provided that memory access speed is not the limiting factor in throughput.

(2) Using the external memory architecture described in Figure 11 with an appropriate number of partitions, memory will not be the limiting factor in throughput.

It should also be noted that parallel implementations of the proposed design operate independent of one another. There is no requirement that each implementation resides on the same FPGA as its neighbors. As long as each implementation is connected to the memory buses shown in Figure 11, the design will operate correctly.

As a result, it is possible to achieve significantly reduced latencies by increasing the number of FPGA devices. This permits the possibility of real time SAR image processing using a network of FPGAs. The size of these FPGAs and the speed of external memory is not important. Using the proposed design, shortcomings in hardware performance can be overcome by increasing the number of parallel instances and supporting memory.

The efficiency of an FPGA design may be analyzed by considering the amount of time that processing elements are computing output data, as compared to the amount of time they are stalled. In this design, the pipe can stall for three reasons:

(1) The memory bridge is being loaded.

(2) The pixel queue is being loaded.

(3) The memory bridge is rotating.

The pre-fetch cache mechanisms attached to the memory bridge and pixel queue effectively mask the external memory latency, as long as the time required to load the pre-fetch cache for an image tile is less than or equal to the time required to generate the tile. If this observation does not hold, then the device can stall. In practice, the memory access requirements of the pixel queue will be much less than that of the memory bridge. If the requirements of the pixel queue or memory bridge surpass the speed of the memory device, then the throughput of the design will be limited by the speed of the memory. This was examined previously in *Design Analysis and Parameters*.

A stall can also occur any time the memory bridge rotates to align a requested range bin with its corresponding processing bucket. The frequency of stall signals for this reason is proportional to the sampling resolution divided by the image resolution. This relationship is shown notionally in Figures 12 and 13. Simply put, an increase in the output image resolution will reduce the frequency in stall signals, as will a decrease in the sampling resolution.

From Figures 12 and 13, it is seen that oversampling an area can cause the memory bridge to rotate more frequently. If the area is greatly oversampled, it is possible that the several rotations could be required for transition from one pixel to the next between each pixel. In this situation, a circular memory bridge that is only capable of rotating one bucket at a time would perform poorly. If one desires to render low resolution images from high resolution sample data, then a memory bridge

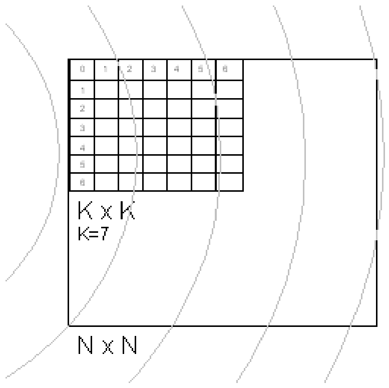


Figure 12. When the sampling resolution is low, and the image resolution is high, pixels often map to the same range bin, or the adjacent range bin, as their neighbors.

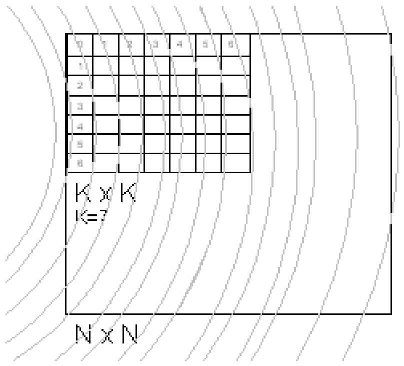


Figure 13. When the sampling resolution is high, and the image resolution is low, pixels map to range bins that are further from the range bins used by their neighbors.

capable of rotating several buckets in a single clock cycle would be preferred.

Simulation Results

To estimate the throughput of our design, the elements shown in Figure 9 were implemented using Altera DSP Builder and Matlab Simulink software. They were then imported into Altera's Quartus II simulation software, then analyzed to determine the theoretical optimum throughput. The sample radar data used to generate these results contained 42,120 pulses, 424 range bins per pulse, and sampled a circular area with a simulated radius of 7 km. For all computations, it was assumed that the tile size was one-tenth the size of

Altera Device Information			Latency (S): Square Image of Width N			
FPGA Family	**Device**	**LEs**	***N* = 500 px**	**5,000 px**	**10,000 px**	**20,000 px**
Stratix III	EP3SL50	47,500	26.8	830	2909	10815
Stratix III	EP3SL150	142,000	8.98	277	973	3617
Stratix IV	EP4S40G2	228,000	5.59	173	606	2253
Stratix IV	EP4S100G4	353,600	3.61	111	391	1455
Cyclone III	EP3C55	55,856	22.8	706	2474	9197
Cyclone III LS	EP3CLS100	100,448	12.7	393	1376	5114
Cyclone III	EP3C120	119,088	10.7	331	1160	4313

Table 1. Theoretical optimal latencies of seven representative FPGA devices, with four common output image sizes.

the output image (i.e., 100 tiles per output image). It is also assumed that a sufficiently fast external memory device is available to ensure that memory access does not limit design throughput.

The theoretical optimal latencies of seven representative FPGA devices are shown in Table 1 using four common output image sizes. The output image is assumed to be square, with a width equal to that shown in the table. All latencies are given in seconds.

It is important to note that the Table 1 lists simulated latencies of our design acting on a single FPGA device with differing degrees of parallelism *J*. However, since parallel instances of the design do not communicate with each other, there is no reason why they cannot exist across multiple FPGAs operating in parallel. In practice, as long as each implementation can fit on a single FPGA, the number of independent FPGAs is unlimited in principle. For a Stratix III EP3SL150, the number of FPGA devices required to generate a 500 x 500 pixel output image in 0.5 seconds is 18.

A common metric for evaluating the performance of computational devices is the equivalent number of operations per second. Considering the case of a 5,000 x 5,000 pixel output image, the total number of operations when executed sequentially is 4.42 x 10^{13}. Given a single Stratix III EP3SL150 operating at 1.60 x 10^{11} operations per second, 6,000 such FPGAs operating in parallel could achieve a performance near 1 petaflop.

Related Work

The algorithm and design presented herein are tailored to the rapid parallel processing of synthetic aperture radar data, as well as the reconstruction of a corresponding two-dimensional output image. However, these techniques could also be applied to any domain where echo response data is projected back to form an image of a surface, object, or area. One such application is tomography, or the construction of three-dimensional models from a set of cross-sectional views. Here, each atomic volumetric unit or *voxel* demonstrates the same properties observed by the pixel. Namely, for a given cross-sectional view, each voxel will be associated with a response value that is spatially near the response of its neighboring voxel. Similarly, a given volume of output data will require a predictable quantity of response data for each cross section. The preservation of these properties ensures that a circular memory bridge could also be used as an effective storage mechanism for the on-chip caching of sectional data.

Previous study in the field of tomography has shown FPGAs to be reliable platforms for deployment of back projection algorithms in the rendering of three- dimensional images. Gac (et al) [2] showed that is possible to obtain response times comparable to that of a Graphics Processing Unit (GPU) using a single System on Programmable Chip (SoPC). The techniques described in this document could supplement these developments, allowing designs to be mapped to a large number of independent FPGAs, as needed.

Conclusions and Future Work

We have presented a design for an efficient mechanism for reconstructing images given synthetic aperture radar data. Our design consists of a pre-fetch cache for masking the latency of external memory, a circular memory bridge for providing rapid access to range bins data, a pixel queue for the direction of image data through a processing pipe, and a core processing element for pairing of pixels with their corresponding range bin. This design partitions the output image into small tiles, relying on the fact that the amount of response data required for processing each tile is predictable and proportional to the size of the tile divided by the size of the image. The selection of an optimal tile size is dependent of the size and speed of the hardware, as well as the size of the problem to be solved (per Equations 7 and 8).

External memory access time may be a limiting factor in design performance, but can be mitigated via reduced pipeline depth, at the expense of increased spatial overhead per parallel implementation. The optimal depth is the largest value that allows the pre-fetch cache to effectively mask memory latency. Oversampling the observation area can also decrease design performance, but small alterations to the hardware of the memory bridge can minimize this effect. Simulations show that high throughput can be obtained by increasing the amount of hardware used, either by choosing large FPGA devices, or by connecting multiple devices to the same shared memory architecture.

Acknowledgements

The authors gratefully acknowledge support of this work under Air Force contract #FA8650-09-M-1563.

References

[1] Mita D. Desai and W. Kenneth Jenkins, "Convolution Backprojection Image Reconstruction for Spotlight Mode Synthetic Aperture Radar" *IEEE Transactions on Image Processing, Vol. 1 No. 4,* pp. 505-517, 1992.

[2] Nicolas Gac and Stephane Mancini and Michel Desvignes and Dominique Houzet, "High Speed 3D Tomography on CPU, GPU and FPGA"

[3] Lars M. H. Ulander, Hans Hellsten, Gunnar Stenstrom, "Synthetic-Aperture Radar Processing Using Fast Factorized Back-Projection" *IEEE Transactions on Aerospace and Electronic Systems, Vol. 39 No. 3,* pp. 760-776, 2003.

[4] B. Bizzarri, "*Applicability of SAR data to meteorology and climatology*" *IEEE* Symposium on Applications of Multifrequency/Multipolarization SAR in View of X-EOS (X-SAR for EOS), pp 139-142, 1990.