Optimized Automatic Target Recognition Algorithm on Scalable Myrinet/Field Programmable Array Nodes

Young H.Cho
Embedded Signal Processing Laboratory
The University of Texas at Austin
Austin, TX 78712 U.S.A.
young@ece.utexas.edu

Abstract

Automatic Target Recognition (ATR) in Synthetic Aperture Radar (SAR) imagery often requires billions of operations per second. This paper describes a compact scalable system developed at Myricom for high-performance implementation of the template-based SAR ATR algorithms developed by Sandia National Laboratories. The Myricom system is mapped on the multiple, concurrent, field programmable array (FPGA) computing nodes connected by Myrinet. These FPGA nodes achieve high efficiency, through the exploitation of the unique characteristics of the ATR algorithm in FPGA. The contributions of this paper are the descriptions of the architectural designs for the ATR system on the scalable FPGA nodes.

1.0 Introduction

As in most real-time image processing algorithms, Sandia National Laboratories' (SNL) real-time ATR in SAR imagery require high bandwidth and large computation resource. Department of defense is concerned with such systems functioning correctly and efficiently in limited space and power. Historically, these systems have been built using custom hardware. However, with very high non-recurring engineering costs for low volume ASICs and subsequent development of new algorithms make the custom special-purpose hardware not favorable. To accommodate the demand and the flexibility, the two-level multicomputer using the FPGA and Myrinet have been developed by Myricom under Defense Advanced Research Project Agency funding[1-5].

2.0 Myricom FPGA nodes

A family of message-passing concurrent computers called multicomputers have been developed to provide a highly scalable and parallel computer system. These systems are ensembles of programmable computer nodes with its own memory connected by a message-passing network[6]. The concept of two-level multicomputer is based on this architecture with each node consisting two levels of processors and a local memory. The first-level processor is mainly responsible for the communication tasks and the second-level processor is used for the application specific tasks. Such design separates the communication layer from the computation layer of the system, thereby allowing a modularity in each node[7].

Historically, the most common use for FPGA has been in product prototyping. Its programmable circuits allow immediate research and development of the hardware products. Once the design is stable, FPGA's high power consumption and high cost per unit makes them less suitable than ASICs in produce in high volume. However, in certain low volume applications, FPGA's reconfigurability can play a key role in producing specialized circuits with efficient execution. The applications with effective speed-up on FPGA usually contains a large amount of parallel computations in streaming data. Examples of these applications include cryptography, neural networks, image and audio processing.[8-11] Even a different approach of the similar algorithm presented in this paper have been programmed into FPGAs to produce effective performance increase[12].

The FPGA nodes developed by Myricom integrates the concepts of reconfigurable computing with the two-level multicomputer to promote flexibility of programmable computational components in a highly scalable network architecture. These nodes are two-level multicomputers whose first level provides the general purpose infrastructure of Myrinet network using the LANai RISC microprocessor. The FPGA function as reconfigurable second level processor responsible for the application specific computing.

3.0 ATR Algorithm

The Sandia SAR ATR system locate and identify the objects of certain class in a image by calculating high probability of detection and low false alarm rate. The ATR algorithm implemented on the Myricom FPGA node is part a hierarchy of algorithms to reduce the processing demands for the image processing. The computation engine mapped on the FPGA of the node is an indexing algorithm called the Second-Level-Detection (SLD). SLD is designed for finding the targets in-the-clear scenarios[1][2].

The ATR system initially collects SAR images from the sensors. These images are then passed on to the Focus of Attention (FOA) subsystem. The FOA determines the areas where targets may be located then extract the corresponding images. Then the SLD driver sends this image and the corresponding templates to the FPGA node to find the targets with the highest match probability in the image.

3.1 SLD Computation

The FOA stage identifies interesting image and composes a list of targets suspected to be in that image. Having access to range and altitude information, the FOA algorithm also determines the elevation for that image, without having to identify the target first. The FOA tasks uses the SLD stage to evaluate the likelihood that the suspected targets are actually in the given image and their position. To do so, the FOA defines tasks for the SLD stage, where each task is composed of an image, a suspected target with its elevation, one or two orientation intervals, and a few parameters.

Term	1	2	3	4	5	6	7	8	9
u	0	0	0	1	1	1	2	2	2
v	0	1	2	0	1	2	0	1	2
$\mathbf{SM_{00}}=$	$B_{00}M_{00}+$	$B_{01}M_{01}+$	$B_{02}M_{02}+$	$B_{10}M_{10}+$	$B_{11}M_{11}+$	$B_{12}M_{12}+$	$B_{20}M_{20}+$	$B_{21}M_{21}+$	$B_{22}M_{22}$
$\mathbf{SM_{01}}=$	$B_{00}M_{01}+$	$B_{01}M_{02}+$	$B_{02}M_{03}+$	$B_{10}M_{11}+$	$B_{11}M_{12}+$	$B_{12}M_{13}+$	$B_{20}M_{21}+$	$B_{21}M_{22}+$	$B_{22}M_{23}$
$\mathbf{SM_{02}}=$	$B_{00}M_{02}+$	$B_{01}M_{03}+$	$B_{02}M_{04}+$	$B_{10}M_{12}+$	$B_{11}M_{13}+$	$B_{12}M_{14}+$	$B_{20}M_{22}+$	$B_{21}M_{23}+$	$B_{22}M_{24}$
$\mathbf{SM_{03}}=$	$B_{00}M_{03}+$	$B_{01}M_{04}+$	$B_{02}M_{05}+$	$B_{10}M_{13}+$	$B_{11}M_{14}+$	$B_{12}M_{15}+$	$B_{20}M_{23}+$	$B_{21}M_{24}+$	$B_{22}M_{25}$

Table 1: Expanded shape sum equation with x=0 and y=0 to 3

The SLD task is to take the extracted image chip, match it against a list of provided target hypotheses, and return the hit information for each image chip consisting of the best two orientation matches, the degree of matching, the corresponding pixel location, and information about which target hypothesis gave rise to these two best matches.

SLD is a binary silhouette matcher that has a bright mask and a surround mask that are mutually exclusive. The bright mask and surround mask are 32 by 32 bit maps, each having only about 100 non-zero pixels.

The system has a database of target models. For each target, there are typically three elevation views of 72 templates defined to correspond all-around views. Each template is composed of several parameters and two masks, a "bright mask" and a "surround mask", where the former defines the image pixels that should be bright for a match, and the latter defines the ones that should not.

Upon receiving the task from the FOA subsystem, the SLD unit matches all the stored templates for this target and elevation and the applicable orientations with the image chip, and computes the level of matching. The two hits with the highest match level are reported to the SLD driver as the most likely candidates to include targets. For each hit, the target type, its orientation, and its elevation, the exact position of the hit in the search area, and the match level are returned. After receiving this information the SLD driver reports this information to the ATR system.

Each target-orientation template consists of two 32 by 32 bit maps, one representing the bright mask and the other representing the surround mask. The image is made of 64 by 64 one byte pixels. The FOA algorithms guarantee that the target, if any, is located in the image such that a six pixel margin around the image is guaranteed not to include the target. Hence, the area of interest in an image is 52 by 52 pixels. In this area, there are 21 by 21 possible places to position the mask. This defines a search-area of 21 search-rows, each 21 position wide. The position (x,y) in the search area corresponds to positioning the lower left corner of the mask over the pixel (x,y) of the image. Hence, 882 matches have to be performed for each orientation that is specified in the matching task.

$$SM_{x,y} = \sum_{u=0}^{31} \sum_{v=31}^{31} B_{uv} M_{x+u,y+v} \qquad \text{(eq. 1)}$$

The purpose of the first step in the SLD algorithm called the "shape sum" is to distinguish the target from its surrounding background. It consists of adaptively estimating the illumination for each position in the search area assuming that the target is at that orientation and location. If the energy is too little or too much then no further processing for that position for that template match is required. Hence, for each mask position in the search area, a specific threshold value is computed as in eq. 1.

$$TH_{x,y} = \frac{SM_{x,y}}{BC} - Bias \qquad \text{(eq. 2)}$$

The next step in the algorithm distinguish the target from the background by thresholding each image pixel with respect to the threshold of the current mask position, as computed before. The same pixel may be above the threshold for some mask positions, but below it for others. This threshold calculation determines the actual bright and surround pixel for each position. The calculation consists of dividing the shape sum by the number of pixels in the bright mask and subtracting a template specific *Bias* constant.

$$BS_{x,y} = \sum_{u=0}^{31} \sum_{v=0}^{31} B_{u,v} \left[M_{x+u,y+v} \geq TH_{x,y} \right] \qquad \text{(eq. 3)}$$

$$SS_{x,y} = \sum_{u=0}^{31} \sum_{v=0}^{31} S_{u,v} \left[M_{x+u,y+v} < TH_{x,y} \right] \qquad \text{(eq. 4)}$$

As shown in eq. 3, the pixels under the bright mask that are greater than or equal to the threshold are counted, and if this count exceeds the minimal bright sum (BS_{min}) the processing continues. Now in eq. 4, the pixels under the surround mask that are less than the threshold are counted. If this count exceeds the minimal surround sum (SS_{min}) it is declared a hit.

$$Q_{x,y} = \frac{1}{2}\left(\frac{BS_{x,y}}{BC} + \frac{SS_{x,y}}{SC} \right) \qquad \text{(eq. 5)}$$

Once the position of the hit is determined, the quality of the hit is calculated by taking the average of the percent of bright and surround pixels that were correct as in eq. 5. This quality value is sent back to the driver with the position to determine two best targets.

4.0 FPGA Implementation

Although it is possible to reconfigure the FPGAs during the run time, they are not well suited for real-time reconfiguration. This is because most FPGAs available today can not reconfigure and

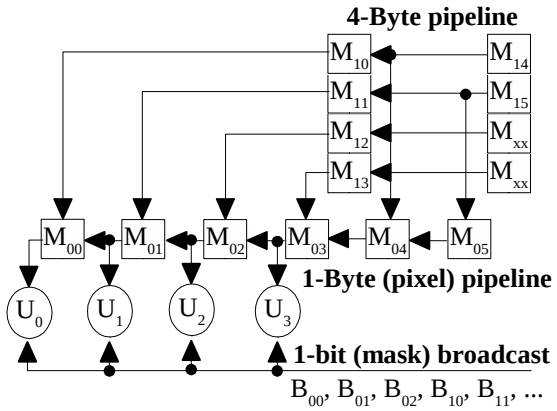

Figure 1: Systolic image array pipeline

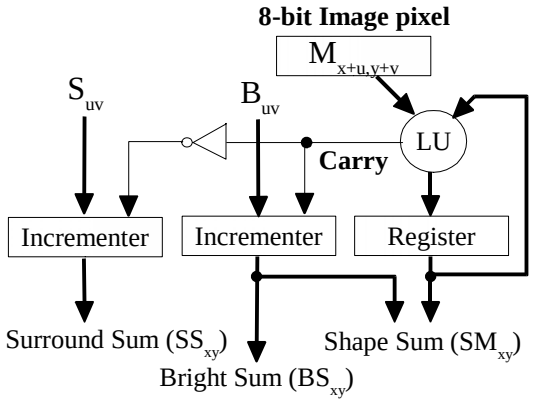

Figure 2: Computation logic for equations 1, 3, and 4

execute the same time. Hence, minimizing reconfiguration time during computation is essential in effective FPGA use. Never the less, when we use the FPGAs as compute engines, the hardware can take on a large range of task parameters through reconfiguration.

Since most of the internal circuitry is available to the programmers as the building blocks, efficient designs can configure the device to perform many operations concurrently. Such parallel execution of the device, limited by the internal logic capacity, enables the FPGA to out-perform the general purpose processor. Thus, to efficiently use the FPGA resources, the understanding of the application is important.

4.1 Parallel Execution

The SLD computing tasks represented with the eq. 1, 3, and 4 need to compute the image correlation of the sliding template masks with the image. In order to determine the design strategies, we examine each equation by applying the algorithm on a smaller data set consisting 6 by 6 image, 3 by 3 mask bit map and 4 by 4 result matrix.

For this data set, the shape sum calculation for a mask requires multiplying all nine mask bits with the corresponding image pixels and summing them to find one of sixteen results. By examining the expanded equation as in table 1, some important characteristic of the equation is observed. First, the same B_{uv} is used to calculate the n^{th} term of all the shape sum results. Thus, when the summation calculations are done in parallel, B_{uv} coefficient can be broadcasted to all the units that calculate each results. Second, the image data that is in the n^{th} term of the SM_{xy} is in the $(n+1)^{th}$ term of SM_{xy-1} except when v returns to 0; the image pixel located in the subsequent row. This fact is useful in implementing the pipeline data path for the image pixels through the parallel summation units.

Based on the above understanding, parallel computation unit as in figure 1 can be designed. In order not to waste time while changing the rows of pixels, the pixels pipeline has the capability either to operate as a pipeline or to be directly loaded from another set of registers. At every clock cycle each U_y unit performs one operation, v is incremented modulo 3, and the pixel pipeline shifts by one stage (U_1 to U_0, U_2 to U_1, ...). When v returns to 0, u is incremented modulo 3, and the pixel pipeline is loaded with the entire $(u+x)^{th}$ row of the image. When u returns to 0, the results are offloaded from the U_y, their accumulators are cleared, and x is incremented modulo 4. When x returns to 0, this computing task is completed. The initial loading of the image pixel pipeline is from the image-word pipeline that is word wide, hence 4 times faster than the image-pixel pipeline. This speed advantage guarantees that it will be ready with the next image row when u returns to 0.

4.2 Computation Unit

Developing different FPGA logic for eq. 1, 3, and 4 is an interesting approach to solving this problem. At the end of each stage the FPGA device would be reconfigured with the optimal structure for the next task[12]. As appealing as this approach may sound, it is not very practical since currently available FPGA devices have typical reconfiguration times of hundreds of milliseconds during which the FPGA can not be used for the computations. In addition, each specific set of template configurations has to be designed and compiled before any computation can occur. This process is a time consuming procedure which does not allow dynamic sets of templates to be immediately used in the system. Therefore, we resisted this approach to perform dynamic reconfigurations. Instead, we designed one structure to perform all three stages as shown in figure 2.

As described in the equations, in all three stages there is a need to bring in the image pixels according to the same sequence while broadcasting one bit from a mask. Then it needs to modify the value of U_y as a function of a certain pixel and mask bits. For the shape sum calculation, the 8-bit pixel value M is accumulated into the 16-bit result in U_y only if the value of the corresponding mask bit is 1. Whereas, in bright and surround sum, the 8-bit result Uy is incremented by one based on the value of the mask bit as well as the comparison of M with the corresponding threshold value, TH_y. Since TH_y does not depend on u and v, it is a constant until the next change of x. These observations gave us some hints to revealing a path to common computation steps.

Adding an 8-bit number to a 16-bit number is the same as adding two 8-bit numbers, M, and the lower 8-bits while conditionally incrementing the higher 8-bits of U_y when the addition overflows. This suggests that each stage needs a conditional 8-bit incrementer and an 8-bit adder for either adding the pixel M to U_y or for comparing M with TH_y. This operation is conditionally executed based on the value of the mask bit.

Both bright and surround sum evaluate the same condition of comparing M with TH_y. The difference is that the bright sum is incremented when the overflow is zero while surround sum increment when it is one. This suggested that the unit can

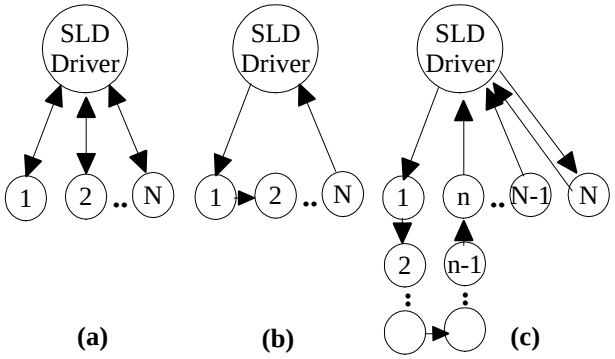

Figure 3: FPGA node topologies

perform both sums at the same time. This design choice not only introduced a concurrency in the computation, but also reduced the required logic in half. They share the distribution of M and TH_y, and the 8-bit adder, but each requires its own broadcast of the mask bits and its own 8-bit incrementer.

5.0 FPGA nodes

As described in the previous section, there are two tasks that the FPGA processes for each image-template pair. First, it calculates a shape sum and then it simultaneously calculates the bright and surround sums. However, there are two additional computations that the FPGA nodes must perform. One is to calculate the thresholds from the shape sums, eq. 2, and the other process is to calculate the two best matches using the hit quality and to report this to the host, eq. 5.

Since these two equations require less iterations of more complex computations, the idle cycles of the LANai RISC processors are used to compute them while the FPGAs are working on the summation tasks.

By carefully pipelining each steps of the algorithm, both processors of the FPGA node are used concurrently to complete the SLD task.

5.1 Scalable system

Myricom LANai processor is responsible for the network processes of the FPGA nodes. Therefore, the FPGA nodes can easily become part of any Myrinet network. Myrinet offers high bandwidth interconnection in a highly scalable, switched network. Within such framework, one can build myriad of network topologies consisting of FPGA nodes.

Topology on figure 3a can have each FPGA node work independently from other nodes to find matches. Each node contains a complete set of templates. The host will give each node a match task and the node will find the best two matches and return the answer to the host. This architecture is fault tolerant with respect to node failures; if the host does not hear from a node it can easily resend the match task to another node, and continue to find matches with the remaining nodes. In such topology, tasks can be immediately assigned to each nodes upon the completion of previous task. Thus, each nodes are highly utilized.

When the template set is very large, each node may not be able to store all the templates. Also, if the number of nodes in the system is quite large, the first topology demands more bookkeeping processing on the host. This can cause the host system to become the bottleneck of the system. Figure 3b is an example of a data flow topology where the task can be partitioned by distributing the templates over the nodes which are arranged in a chain. Images and their current best matches are passed down the chain from node to node as each node finishes matching an image against the node's subset of the templates. The two best matches so far are passed along with the image and then returned to the host by the last node. However, unlike the first topology, when one node or link is lost or becomes faulty, the nodes must re-route the match tasks, redistribute the templates of the "lost" node before continuing the tasks. Dynamic load balancing is also not optimal because, even if the same number of templates have to be matched by each node, the amount of work that each node has to do varies, and a node that is momentarily overwhelmed may create idle nodes downstream.

Another topology can be built by combining the two discussed above as shown in figure 3c. Such hybrid approach would include the best of both architectures and allow users to customize their system to fit their needs. Such system are highly scalable as well as fault tolerant.

6.0 Results

The test and measurement setup is as shown in the figure 4. The host is a SparcStation IPX running SunOS 4.1.3 with a Myrinet interface board with 512K memory. The FPGA node, consisting of Lucent technologies ORCA FPGA 40K and Myricom LANai 4.1 running in 3.3V at 40 MHz, communicate with the host through a 8-port Myrinet switch.

Without additional optimization, our implementation of the complete ATR algorithm on one FPGA node processes over 900 templates per second. Each template requires about 450 thousand iterations of 1-bit conditional accumulate for the complete shape sum calculation. Then the threshold calculation require one division followed by subtraction. The bright and surround sum compares all the image pixels against the threshold results. Then 1-bit conditional accumulate is executed for each sums. Then the quality values need to be calculated using two divides, add, and a multiply.

Given 1-bit conditional accumulate, subtract, divide, multiply, and 8-bit compare are one operation each, the total number of 8-bit operations to process one 32 by 32 template over a 64 by 64 image is approximately 3.1 million operations. This indicates that the FPGA node successfully runs over 2.8 billion 8-bit operations per second (Giga operations per second/GOPS).

After the simulations, we found that sparseness of the actual templates reduced the average valid rows of the

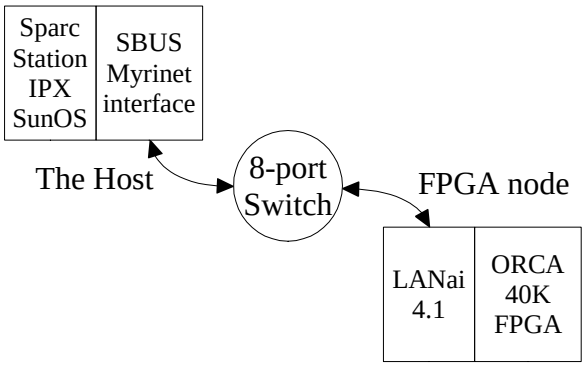

Figure 4: Performance test configuration

templates to be approximately one half of the number of the total template row. When we took advantage of this observation in the FPGA design, the performance increased to approximately 4.0 GOPS. Further simulations revealed more room for improvements, such as dividing the shape sum within the FPGA, transposing narrow template masks, and skipping invalid threshold lines. Although these optimizations were not implemented into the FPGA design, the simulation indicated an average of 7.75 GOPS of performance with all the optimization.

7.0 Conclusion

The correlations are mapped to a linear systolic pipeline. A high degree of parallelism is exploited. In addition to computing an entire row of correlation results in parallel, the FPGA performs the address calculations, data loading, and correlation in parallel. Short inter-register paths allow the design to run at 40MHz, which is limited by the clock rate at which external memory can be fetched.

In this paper we have described a scalable, reconfigurable system under the two-level multicomputer architecture. This system is scalable by using an embedded high-performance networking technology (Myrinet) that has programmable network interfaces. These network interfaces contain microprocessors for local control of the FPGA.

We have demonstrated a scalable FPGA system for an automatic target recognition application. For this application we have measured 4 GOPS in performance. Such measurements were about ten times faster than the system in use at that time as well as about four times faster than the algorithm running on the PowerPC based multicomputer; which was developed during the same period[1].

Further analysis suggested that the performance may be double with additional optimizations, yielding up to 500 GOPS per VME-6U sub-rack with 16 baseboards each with four FPGA nodes. More importantly this performance scales linearly with the number of nodes due to the modularity of each node and the scalability of the message passing network. The only limit to its scalability is the ability of the host to dispatch and handle matching tasks.

The above results were measured using the FPGAs available during 1996. Due to the nature of the algorithm, the performance can scale linearly within the FPGA design by simply adding more computation units to take advantage of the parallelism. With increasing speed and logic areas in FPGA technology today, such improvement is unpredictably large.

8.0 Reference

[1] R. Sivilotti, Y. Cho, D. Cohen, W. Su, and B. Bray, "Scalable Network Based FPGA Accelerators for an Automatic Target Recognition Application," *Proc. IEEE Symp. on FPGAs for Custom Computing Machines*, pp. 282-283.

[2] R. Sivilotti, Y. Cho, W. Su, and D. Cohen, "Scalable, Network-Connected, Reconfigurable, Hardware Accelerators for an Automatic-Target-Recognition Application," *Myricom Technical Report*, May 1998.

[3] R. Sivilotti, Y. Cho, W. Su, and D. Cohen, "Myricom's FPGA-based approach to ATR/SLD," *DARPA ACS PI Meeting Slide Presentation*, November 1997.

[4] R. Sivilotti, Y. Cho, W. Su, and D. Cohen, "Production-quality, LANai-4-based Quad-FPGA-node VME boards," October 1997, *http://www.myri.com/research/darpa/97a-fpga.html.*

[5] C. L. Seitz, "Tactical Network and Multicomputer Technology," March 1997, July 1997, and August 1998, *http://www.myri.com/research/darpa/index.html.*

[6] C. L. Seitz, "Two-Level-Multicomputer Project Summary," July 1996, *http://www.myri.com/research/darpa/ 96summary.html.*

[7] W. C. Athas and L. Seitz, "Multicomputers: message-passing concurrent computers," *IEEE Computer*, vol. 21, pp. 9-24, 1988.

[8] M. Shand and J. Vullemin, "Fast implementations of RSA cryptography," *Proc Symp. Computer Arithmetic*, 1993.

[9] J. G. Eldredge and B. L. Hutchings, "RRANN: The run-time reconfiguration artificial neural network," *Proc. Custom Integrated Circuits Conf.*, San Diego, CA, 1994.

[10] P. M. Athanas and A. L Abbott, "Real-time image processing on a custom computing platform," *IEEE Computer*, vol. 28, pp. 16-24, 1995.

[11] C. Chou, S. Mohanakrishnan, and J. B. Evans, "FPGA implementation of digital filters," *Proc. Signal Processing Applications Technology*, Santa Clara, CA, 1993.

[12] K. Chia, H. Kim, S. Lansing, W. Mangione-Smith, and J. Villasenor, "High-Performance Automatic Target Recognition Through Data-Specific VLSI," *IEEE Trans. on VLSI Systems*, vol 6, no. 3, September 1998.