A Fast and Innovative Approach Towards an Automatic Target Recognition System Implementation on a Reconfigurable Architecture

Luis A. Bathen¹, Nader Bagherzadeh²
MorphoSys Research Group²
¹School of Information and Computer Science, University of California, Irvine, CA 92697 USA
lbathen@uci.edu
²Department of Electrical Engineering and Computer Science, University of California, Irvine, CA 92697 USA
nader@uci.edu

Abstract

Automatic Target Recognition (ATR) is the methodology used by computers to detect, classify and recognize an object without human interaction. ATR is considered to be one of the most computation-intensive applications due to the amount of images that have to be processed in real-time. For this paper, the mapping of the Second Level of Detection (SLD) process of an ATR algorithm has been implemented considering that SLD is the bottleneck of an ATR system. The chosen reconfigurable architecture was the second generation of the MorphoSys reconfigurable architecture (M2). By combining a multi-processing programming scheme and a fast SIMD reconfigurable architecture, this paper proposes a fast and innovative approach at targeting an ATR system.

Keywords: MorphoSys, reconfigurable systems, synthetic aperture radar, automatic target recognition, algorithm mapping, second level of detection.

1 Introduction

Over the past few decades, the need for high performance computing system has increased due to the computational demands for today’s applications. Among some of the most computation-intensive applications are image and signal processing. An example of image processing is automatic target recognition, which is the main focus of this paper. Automatic Target Recognition (ATR) is the machine process of detecting and identifying target objects from real time images. Due to the fact that human lives depend on the speed and reliability of the ATR system and that there are thousands if not millions of images being processed in real-time, to be able to produce a fast and reliable system became to be known as the ATR challenge. General Purpose Processors tend to not be suitable for image processing, which is the main reason why most ATR systems have been implemented by large ASIC boards (i.e. multi-FPGA Systems). The Second Level Of Detection (SLD) is the process of detecting and pinpointing where exactly the target is in the image. SLD is considered to be the bottleneck of an ATR System since it compares an image with a library of thousands of targets templates. This paper proposes a fast ATR system implementation on a reconfigurable architecture and using the SLD processing model as the main algorithm implementation. The reconfigurable architecture chosen for this project is MorphoSys Version 2 (M2). As mentioned before, the main purpose of this paper is to show that M2 is more than capable of handling an ATR system implementation. Furthermore, this paper investigates M2’s multi-processing capabilities. Section 2 introduces ATR, section 3 introduces the MorphoSys reconfigurable architecture, section 4 discusses the different systems that target ATR including MorphoSys Version 1 (M1) [1,2,3], section 5 discusses the mapping of the second level of detection processing model onto M2, section 6 discusses the results, section 7 discusses the conclusions and future work.

2 Automatic Target Recognition

2.1 Introduction

The ATR challenge is considered one of the most computation-intensive applications because of the amount of frames being processed per second, considering that it is being done in real-time. There are many different algorithms and approaches to implement an ATR system. For this paper the chosen processing model was developed
at the Sandia National Laboratory (SNL) [4,5]. This model was designed to detect partially obscured digital targets from Synthetic Aperture Radar (SAR) images generated by the SAR system. Figure 1 shows part of a software system targeting ATR developed by the ATR group at SNL.

![Figure 1: ATR software tool developed by the ATR group at Sandia National Laboratory [6].](image)

The SAR system produces real-time images that contain several million pixels. Images pass through a focus-of-attention stage that identifies regions of interest, each of which contains a potential target. These regions of interest, known as “chips,” must be correlated with a large number of target templates. Target templates are much smaller than chips, and are represented as binary images. The correlation results are sent to a peak detector, which identifies the template and relative offset at which the highest correlation occurs. The correlation of chips with templates is the performance bottleneck for the system. The system described here uses chip sizes of 128-by-128 pixels and template sizes of 8-by-8 pixels. The correlation of a single chip with a single template involves consideration of 14 641 (121X 121) offsets within the chip. This operating scenario involves evaluating many chips against a library of thousands of templates in real-time.

![Figure 2: 8x8 Sample target templates](image)

For each target there are 72 2D templates, each representing a 5 degrees rotation, which makes up a 360 degrees view of the target, for each template there are two different masks. Target templates occur in pairs, one is known as the bright template and identifies locations from which a strong radar return is expected, while the other is known as the surround template and identifies locations where strong radar absorption is expected. Figure 2 shows a pair of masks for an 8x8 template. The ATR system structure is illustrated in Figure 3.

![Figure 3: High Level Overview of the ATR Process](image)

As described in [4,5,7], ATR has been implemented mostly by Field Programmable Gate Arrays (FPGAs). For this study, ATR will be mapped onto a reconfigurable architecture. The ATR SLD processing model implementations will be discussed in more detail in both section 4 and 5.

### 2.2 Second Level of Detection Algorithm Overview

The SLD algorithm is composed of a series of steps. Assuming we have a 128x128-8bit image pixels, and 8x8-1bit templates, there will be 121x121 different offsets for the process. Starting from the image MS(i,j), the steps are as follows:

- First step is the Shape Sum calculation SM(i,j), the shape sum is used to identify the image surroundings.
  
  $$ SM(i,j) = \sum_{u\in U}^{n-1} \sum_{v\in V}^{n-1} B(u,v) MS(i+u,j+v) $$

- The second step is the threshold value calculation TH(i,j), which is then used by the correlation steps in order to determine between the background and the image.
TH(i,j) = SM(i,j) / BC – Bias

- The third step is the correlation phase which is done by calculating the Bright Sum BS(i,j) and the Surround Sum SS(i,j)

\[
BS(i,j) = \sum_{u=0}^{n-1} \sum_{v=0}^{n-1} B(u,v) [MS(i+u,j+v) \geq TH(i,j)]
\]

\[
SS(i,j) = \sum_{u=0}^{n-1} \sum_{v=0}^{n-1} B(u,v) [MS(i+u,j+v) < TH(i,j)]
\]

- Finally, the last step is to calculate the hit quality HQ(i,j), which is used to get the best hits to pinpoint the best location for the target.

\[
HQ(i,j) = \frac{1}{2} (BS(i,j)/BC + SS(i,j)/SC)
\]

Fields:
- BC is the number of 1's in the bright template, which represent the area where strong radar return is expected.
- SC is the number if 1's in the surround template, which represents the area where strong radar absorption is expected.
- The Bias is an image specific constant.
- \(v_0, u_0\) are the starting offsets of the surround and bright masks, \(i\) and \(j\) represent the \((x,y)\) position of the image.
- \(n\) is the width/height of the image to be processed.

3 Overview of the MorphoSys Reconfigurable Architecture

3.1 Introduction to Reconfigurable Systems

There are different types of computing systems; there are those that target single applications like the Application Specific Integrated Circuits (ASICs) or multiple applications, which is the case of general-purpose processors. Examples of ASICs are battery management for household appliances, control circuit for mobile phones, control and evaluation circuit for motion detectors, interface and signal processing electronics for sensors (light, vibration and magnetic field), etc. Examples of general-purpose processors are our personal computer processors, Intel’s Pentium, Motorola/IBM PowerPC, AMD, etc. Reconfigurable systems on the other hand are between the extremes of ASICs and general-purpose processors. Reconfigurable systems are computing systems that have a reconfigurable hardware-processing unit and sometimes combine it with a software-programmable processor. These systems allow for the reconfiguration or programming of the processing unit to target different applications. A reconfigurable system allows for a wider range of applications than and ASIC based system and by combining the software-programmable processor with the reconfigurable hardware it outperforms general-purpose processors in many applications.

3.2 Significance of Reconfigurable Systems

Let us consider a multimedia application, as described in [1], a multimedia application may include a data parallel task, a bit-level task, irregular computations, high-precision word operations and a real-time component. Fur such complex applications with the wide-ranging subtask, the ASIC approach would lead to large die or a large number of separate chips which is not only expensive in monetary terms but it also risks the performance of the system. Also, if the general-purpose approach is taken, most likely it would not satisfy the performance requirements of the application. On the other hand, if the reconfigurable system (one that combines the reconfigurable hardware with the software-programmable processor) approach is taken, the system may be programmed to target each task, thus meeting the application needs. Such reconfigurable system has the performance of an ASIC approach and the generality of a general-purpose processor.

3.3 Introduction to MorphoSys

MorphoSys, as described in [1,2,3], is a reconfigurable system, an innovative computing system that combines a software-programmable RISC like processor called the TinyRISC and a reconfigurable hardware unit called the Reconfigurable Cell (RC) Array. It targets computation-intensive data-parallel applications; some examples of these applications are video compression, image processing, multimedia, and data security. This paper will focus on the second generation of MorphoSys (M2) [8,9,10]. M2 follows the basic concept of its predecessor, the MorphoSys prototype known as M1. M2’s performance has been enhanced since it has been redesigned in both scalar operand network and memory hierarchy. As described in [8,9], M2 is a SIMD reconfigurable architecture running at 450MHz, its area is 8*8mm² without on-chip memory and 16*16mm² with on-chip memory, with 4W (peak MAC) power usage and 0.13µm technology. The basic structure of an 8*8mm² MorphoSys M2 chip is illustrated in Figure 4.
MorphoSys has three main components, the software-programmable core processor called TinyRISC, the reconfigurable hardware unit which consists of a 64 array of Reconfigurable Cells also know as RC Array which is organized in SIMD fashion. Figure 5 shows the RC Array. Finally, the third main component is the data movement unit called Frame Buffer (FB).

4 Related Work

As specified in [1,2,3], a basic approach taken to target ATR has been through the use of Field Programmable Gate Arrays (FPGAs). FPGAs are a good approach because they can be configured at the gate level. This is ideal for bit-correlation, and fine grain procedures. Although, MorphoSys is not a fine grain system, it allows for bit correlating at the word level. Note that unlike other architectures, M2 will actually process raw data, not preprocessed data. In other words, M2 will serve as a multiprocessor for the ATR system. The two systems in [4] and [5] were used for comparison with M2’s predecessor M1. Previous work showed MorphoSys’ computational power and superiority to the Mojave Board that consists of two Xilinx XC4013 FPGAs [4] and the 16 Xilinx 4010 FPGAs based Splash 2 system used in [5]. For 16 pairs of target templates, MorphoSys running at 100 MHz took around 30ms of processing time. When comparing this to the 195 ms and 210 ms of processing time for Splash 2 [5] and Mojave [4] respectively, it is assumable that MorphoSys is more than capable of handling ATR. Also, in order to satisfy the system specifications of an ATR system we would need nine M1 (64 RCs) chips compared to 90 sets of the Mojave system in [4] and 84 sets of the Splash 2 system specified in [5]. Figure 6 shows the ATR performance of the three systems compared.

Figure 5: RC Array

5 Mapping SLD onto MorphoSys

This paper focuses on the Second Level of Detection (SLD) of the ATR processing model developed by the SNL and specified in [7].

5.1 Loading Images onto RC Array

For this study, 8x8x8bit image pixels will be processed and matched against 8x8x1bit masks. The image pixel will be loaded as follows:

Figure 6: ATR Performance Comparison

Figure 4: MorphoSys M2 Architecture Diagram [8,9,10].
For column \(i\) in RC Array, load row \(i\) of image into each RC of the current RC column as 8-bit values. Where \(i\) is 0,1,\ldots,7.

For each pixel \(k\) of the image store it into register \(k\) of each RC. Where \(k\) is 0,1,\ldots,7.

The next step is to load the 8x8x1bit bright and surround templates (masks), and they will be loaded as follows:

For column \(i\) in RC Array, load row \(i\) in mask into each RC of the current load as an 8bit value. Where \(i\) is 0,1,\ldots,7.

Store the 8bit value bright and surround mask in registers 8 and 9 respectively.

The main feature of SIM_ATR is the RC_ARRAY class that has the ability to simulate the MorphoSys RC Array composed of 64 RCs, where each RC runs its own processes. After each RC is done with its process, SIM_ATR uses the RC_ARRAY class to perform column-wise operations as well as row-wise operations just as in the actual MorphoSys processor. Each column in RC_ARRAY runs a different process, thus, RC_ARRAY runs 8 different processes in parallel. SIM_ATR has a built in driver program that simulates a peak detector that gets the offsets with the highest correlation results.

The purpose of SIM_ATR was to get familiar with the SLD algorithm before doing the actual MorphoSys mapping and MuLate testing (MuLate is a cycle accurate tool used to do the MorphoSys algorithm mapping). Note that SIM_ATR does not follow M2’s programming scheme. SIM_ATR simulation results will be discussed in section 6.

### 5.3 MuLate Implementation

A full MuLate implementation of the ATR SLD algorithm has been implemented. For the MuLate implementation there are three main steps that must be done in order to implement an algorithm in MorphoSys assembly. For more information on the programming scheme followed by MorphoSys please refer to [10]. The process is as follows:

- The first step is to write RC Context. Context is the terminology given to the programming scheme used for MorphoSys’ Reconfigurable hardware called RC Array. Due to MorphoSys’ SIMD fashion each context plane (8 instructions going from 0 to 7 for each row/column) is processed in one cycle.

- The second step is to write TinyRISC Assembly code. TinyRISC is the software programmable unit of MorphoSys, it is used to control data movement between Main Memory, Frame Buffer and RC Array as well as to execute row/column context.

- Finally, all MuLate input data needs to be converted into hexadecimal words (8 hex values, hex is a base 16 number). Context is converted into hex words through a Perl script tool called mload, which was developed by the MorphoSys Research Group. TinyRISC code is converted into
hex words through a C++ tool called *trasm2002*. Finally, all image data needs to be converted into hex as well, this is done through a binary/decimal JAVA converting tool called *Hex_Converter*.

In order to simulate the double summation required by the SLD algorithm each RC will process the inner summation, and the outer summation will be processed performing row-wise operations. Due to RC array’s SIMD nature, one instruction will be processed by all RCs in the same column or row depending on how the context specifies it.

**Figure 9: Row-wise operations for SLD algorithm**

Figure 9 shows the row-wise operations needed to simulate the outer summation of the SLD algorithm as well as the row context for this operation. All three steps, the shape sum SM(i,j), the bright sum BS(i,j) and the surround sum SS(i,j) require similar row-wise operations. The only difference is that the shape sum is a series of 8bit summations and both surround and bright sum are 1bit summations. Unlike other architectures, the actual SLD algorithm was fully implemented by the RC Array, there is no preprocessed data. There are two assumptions being made, the first is that the FOA algorithm has given us an image containing a possible target and the second is that the 128x128x8bit pixel image will be divided into 8x8x8bit pixel images for M2 to process them. Note that we can develop a system consisting of multiple M2 chips that will meet ATR’s specs.

6 Results and Conclusions

The SLD algorithm implementation on SIM_ATR was able to detect both M47 tanks in a 128x128x8bit pixel real image, taking an average time of 0.17ms for each 8x8x8bit pixel image matching with the 8 pairs of templates mapped into the simulated RC_ARRAY.

SIM_ATR simulation was done on a PowerPC G4 running at 800MHz. SIM_ATR simulates the actual loading of the image pixels into RC array, loading of the templates into RC array, and the SLD algorithm implementation. Figure 10 shows the best hit offsets (red) for the 8x8 templates obtained from the built in peak detection feature in SIM_ATR. If we were to assume that 32x32x1bit templates were matched against 32x32x8bit images we can see that the SLD algorithm would be able to detect the actual targets in a 32x32 area (yellow). From the JAVA simulation we can show how we can exploit M2’s parallelism when targeting the SLD algorithm.

**Figure 10: SIM_ATR Simulation Results**

This paper’s results are taken from a full SLD algorithmic implementation, meaning that M2 processed raw data. All inner/outer summations, comparisons, multiplications and divisions are done by the RC Array and controlled by TinyRISC. Implementing the SLD algorithm based on the benchmarks in [1,4,5] is work in progress. Since this is the first time an approach that combines a multi-processing programming scheme with a SIMD reconfigurable architecture has been taken to target an ATR system implementation, comparisons are difficult to make.

<table>
<thead>
<tr>
<th>Phase</th>
<th># Cycles</th>
<th>Time µs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Context Load</td>
<td>1722</td>
<td>3.83</td>
</tr>
<tr>
<td>Data Load</td>
<td>384</td>
<td>3.53e-1</td>
</tr>
<tr>
<td>Shape Sum</td>
<td>36</td>
<td>8.00e-2</td>
</tr>
<tr>
<td>Surround Sum</td>
<td>66</td>
<td>1.46e-1</td>
</tr>
<tr>
<td>Bright Sum</td>
<td>69</td>
<td>1.53e-1</td>
</tr>
<tr>
<td>Threshold</td>
<td>4</td>
<td>8.89e-3</td>
</tr>
<tr>
<td>Hit Quality</td>
<td>6</td>
<td>1.33e-2</td>
</tr>
<tr>
<td>Total Loading</td>
<td>2106</td>
<td>4.68</td>
</tr>
<tr>
<td>Total Processing</td>
<td>181</td>
<td>4.00e-1</td>
</tr>
<tr>
<td>SLD process</td>
<td>2287</td>
<td>5.08</td>
</tr>
</tbody>
</table>

**Table 1: MuLate Results for M2 running at 450MHz**
Table 1 shows the time in µs and number of cycles for the SLD algorithm implementation on M2. Considering that usual chip images produced by the FOA algorithm are 128x128x8bit pixels, if we use 8x8x1bit templates, we would be looking at 121x121 correlations for 8 pairs of templates which means that 14,641 8x8 chunks of the image would have to be processed. Considering that the entire processing of 8 8x8 templates is 181 cycles (4.00e-1µs), we can say that for a 128x128 image we would expect to take 5.89ms.

References