A SCALABLE ARCHITECTURE SYSTEM FOR AUTOMATIC TARGET RECOGNITION

Philip David, Philip Emmerman, and Sean Ho

Army Research Laboratory
Adelphi, MD

ABSTRACT

Automatic target recognition (ATR) is a very difficult problem due to the variety of conditions under which an ATR system may be required to operate. Because the number of operations required to execute a particular ATR algorithm can vary greatly from one scenario to another; a fixed hardware and software architecture will usually not be able to execute a given ATR algorithm in all required scenarios within some given real-time constraints. A solution to this problem is to use a scalable architecture. The hardware and software of such an architecture can easily be scaled to meet the processing requirements of a particular scenario. This paper describes a scalable architecture system which we have developed that implements a real-time ATR algorithm.

INTRODUCTION

Automatic target recognition is the process of locating and recognizing targets in data generated by one or more sensors. This is a very difficult problem due to the variety of conditions under which an ATR system may be required to operate: targets may be occluded, they may have low target-to-background contrast, many types of targets may need to be recognized, target appearance can vary greatly with different viewpoints, natural and man-made clutter may be present in the scene, etc. Hence, the amount of computation required to execute a particular ATR algorithm can vary greatly from one scenario to another; a fixed hardware and software architecture will not usually be able to execute a given ATR algorithm in all required scenarios within some given real-time constraints. A solution to this problem is to use a processing architecture that can easily be scaled to meet the processing requirements of a particular scenario. A scalable architecture is a computer architecture that can deliver an increase in performance proportional to an increase in its size. However, efficiently using such an architecture requires a software architecture that scales along with the hardware.

We have implemented the ATR Relational Template Matching (ARTM) algorithm¹. It uses a hierarchy of target

silhouette models to detect and recognize targets in infrared (IR) imagery. The original, sequential algorithm has been developed into a parallel, scalable algorithm that runs on a scalable architecture consisting of Texas Instruments TMS320C40 processors. We describe how the ATR algorithm is decomposed, distributed, and run on the C40's using their parallel, high-speed, interprocessor communication to achieve maximum system performance.

The rest of the paper is organized as follows. We first describe the system's hardware architecture and its operating system. Then a description of the ARTM algorithm and its parallel implementation is presented. This is followed by an analysis of how well the algorithms scale when they are applied to more difficult problems. Finally, we conclude with a discussion of the system's performance.

SCALABLE ARCHITECTURES

Recent hardware and software advances are making the development of low-cost, scalable architectures more practical. At the digital signal processor (DSP) chip level, vendors are adding more powerful I/O and interconnect features that allow designers to combine multiple DSP's more efficiently and with greater flexibility. At the board level, vendors are packaging multiple DSP's on single boards using a broad range of point-to-point and shared memory configurations

The scalable system that runs the ATR code is a heterogeneous system; it consists of a general-purpose single board computer (SBC) and a DSP subsystem. Both are VME bus-based, commercial off-the-shelf (COTS) computer boards (see Fig. 1). The SBC uses a 25-MHz Motorola 68040 processor. This board services requests from the DSP subsystem such as I/O to a remote file system. In the software development and testing stage, binary code and data are forwarded from the SUN to the DSP via this SBC.

1. The ARTM algorithm was developed by Mathematical Technologies, Inc., under sponsorship of the Army Night Vision and Electronic Sensors Directorate and the Army Research Laboratory.

Figure 1: System overview. Three operating systems are involved in running the ATR code on the C40's.

The second board in the system is a quad C40 board; it consists of four 40-MHz TI TMS320C40 DSP CPU's. Each DSP CPU is capable of performing 275 MOPS and has a maximum data throughput of 320 Mbytes/sec, including 20 Mbytes/sec throughput from each of its six interconnected communications ports (see Fig. 2). Three of six communication ports on each C40 can be externally connected to other C40's. In so doing, scalability can be achieved by adding additional C40 boards to the system as the need arises. Another important feature of the C40 is its use of DMA (Direct Memory Access), which permits data to be transferred between memories without the intervention of the DSP's CPU [1,2]. The quad C40 board has 2 Mbytes of global SRAM and 2 Mbytes of SRAM for each of the four processors.

Figure 2: The overall architecture of the Quad C40 DSP board. Dashed lines represent the parallel communication ports.

OPERATING SYSTEM

Since the host SBC (68040) directs and synchronizes all tasks run on the DSP board and other boards on the VME bus, vxWorks, a real-time multi-tasking operating system is run on the SBC. We would like to minimize the time that is spent in debugging and porting the ATR code from the SUN to the DSP platform. In other words, we would like to preserve as much of the original code as possible. Consequently, a Unix-compatible DSP operating

system (OS) is desired. This OS should provide not only the basic features of a real-time operating system; it should also make the application easily portable to other DSP systems in the future without modifying the already developed ATR code. We selected SPOX, which also provides features such as dynamic memory allocation from multiple memory segments and a C standard I/O library. The C standard I/O server allows the host server to communicate with SPOX tasks running on the DSP board with C standard library functions such as `fopen()`, `printf()`, and `scanf()` [3].

ATR APPLICATION

ATRM Algorithm

The ARTM algorithm is a model-based target recognition algorithm which consists of an off-line algorithm design process and an on-line target recognition process [4,5].

The off-line process uses CAD models to construct a decision tree of templates that are matched to imagery during the on-line process. Each node in the decision tree represents a test for the presence of a target silhouette boundary. The tree implements a coarse-to-fine search of the target-type/target-pose search space: Tests at the higher levels in the tree are very general in that they test for the presence of target silhouette boundaries that could have been generated by any of a number of types of targets in a wide range of poses. Tests at the lower levels in the tree, in contrast, test for the presence of very specific target silhouette boundaries which could only have been generated by single types of targets in very limited poses. Figure 3 illustrates an example decision tree. The tests are "relational" in that, rather than trying to recognize each target's silhouette independent of all other target silhouettes, the tests focus on aspects of the target silhouettes which differentiate the various targets.

Figure 3: A four-class decision tree. To pass a node's test, a target's silhouette boundary must lie in the dark region of the node's template.

The on-line target recognition process applies the tests in the decision tree to each pixel of the image. The test associated with the root node of the decision tree is first

applied to each pixel in the image. If a pixel passes this test, then the tests associated with the node's "children" are applied to that pixel, and so on, until either a terminal test has passed (meaning a target has been recognized at that pixel) or all tests fail (meaning that there is no target at that pixel). Because the center of a target can be located at any pixel in the image, this tree search is carried out at each pixel in the image.

The only difference in the tests that are applied at the different nodes in the decision tree is that the target silhouette boundaries differ from node to node: the lower a node is in the decision tree, the more constrained the target boundary test becomes. A target silhouette boundary is determined to exist at a pixel when the region around that pixel contains a sufficient number of edge points along the boundary of the associated target silhouette template. See [4] for more details.

Parallel ATRM Algorithm

As described above, the sequential ARTM algorithm carries out the same search algorithm at each pixel in the image. The algorithm is therefore inherently parallel. Our parallel implementation is as follows. A copy of the image to be processed is first sent to each processor, and an assignment is made of which pixels in the image each processor should examine. Each processor then applies the ARTM algorithm to its assigned set of pixels independently of the other processors. When all processors have finished, the target regions found by each are merged into a single, consistent set of target regions.

For a system consisting of p processors, one cannot simply divide an image up into p "blocks," one for each processor, and expect a significant speedup of the algorithm. This is because, in a typical IR image, usually only a few regions of the image contain targets or target-like clutter. In the ARTM algorithm, much more computation is required in regions of the image containing targets and target-like clutter than in regions without targets. With this simple block-partitioning of the image, the few processors that receive image blocks containing targets will be busy while the majority of the processors will quickly become idle.

To obtain good load balancing, it is essential that each processor be assigned roughly the same number of target (and target-like clutter) pixels. To this end, we assign every p^{th} column of pixels to the same processor, as illustrated in Figure 4. When the number of processors in a system is significantly greater than the expected number of pixels across a target or target-like clutter, then even this load balancing scheme is not effective, for there will be some processors that will receive no target pixels, and will therefore likely spend much of their time idle. In this case, it is easy to devise other schemes, where the pixels

assigned to each processor are uniformly distributed over the image, and, therefore, the target pixels will be uniformly distributed to the p processors.

Figure 4: Assignment of pixels to processors. In this example, $p = 4$.

We demonstrated above that the ARTM algorithm is highly parallel. To obtain an efficient parallel solution, however, the processors must also have a fast mechanism to share program data (which, in our system, includes images, target silhouette templates, and target regions-of-interest). We have experimented with two mechanisms for sharing data: shared memory and message passing. The performance of the system using each of these mechanisms is described below. With large numbers of processors, however, the use of shared memory is very limiting due to memory contention problems. We therefore concentrated our effort on the message passing architecture.

In our message-passing implementation of the ARTM algorithm, processors only need to communicate during initialization, to obtain a copy of the image to be processed, and during the last stage of the algorithm, where the targets detected by each processor are merged into a single, consistent set of target detections. These communications can be achieved most efficiently when the processors are organized in a hierarchy as shown in Figure 5. This organization is a result of the physical organization of the quad-C40 board, as described earlier: each board contains four processors, each with three internal communication ports (to processors on the same board) and three external communication ports (to processors on other boards). The quickest way to broadcast a message with this architecture is as follows. One processor on each board receives messages from other boards. When this processor receives a message, it sends it to the other three on-board processors and to processors on two other different boards. The processors that are initialized via the inter-

nal communication links then send the message out to three processors on three other different boards.

Figure 5. A four-level processor interconnection network. Each circle represents a processor. Solid lines depict connections between internal communication ports. Dotted lines depict connections between external communication ports. Images propagate from the top level to the bottom. Results propagate from the bottom level to the top.

ANALYSIS

We now analyze the scalability of the parallel ARTM algorithm for the average case behavior. There are many performance metrics which can be used to measure the scalability of a parallel system [6]. The system's speedup as a function of problem size and number of processors is the metric that we use here. Our problem size is given by n , where the size of the image to be processed is $n \times n$ pixels. The speedup, S , of a parallel system is defined as the ratio of the time required to run on one processor, T_1 , to the time required to run on p processors, T_p :

$$S = \frac{T_1}{T_p}$$

The analysis which follows assumes that the parallel ARTM algorithm's load balancing scheme is such that all processors finish the decision tree searches at the same time so that the processors experience no idle time between the end of the search and the start of target list merging. For this to occur, each processor must be assigned roughly the same number of target/clutter and non-target/non-clutter pixels. This is possible whenever the number of target/clutter pixels is much larger than the number of processors in the system.

Our analysis of speedup is based on both the run time behavior of an actual system, and on a high-level complexity analysis of the algorithm. We describe the algorithm in terms of a number of high-level, basic operations. The time required to execute each type of operation is determined by measuring the run time of the operation on our four-processor system. Using this timing data, one can generate an equation for the run time complexity of the algorithm for any problem size and number of processors

and, from this, calculate the system's speedup. The basic operations and their measured run times are as follows:

1. *Transfer an image between two processors that have a direct connection in the network.* Pixels are transferred at a rate of $c_1 = 1.6 \times 10^{-7}$ seconds per pixel. (This rate actually varies slightly with the image size, but we assume that it is a constant.)
2. *Perform the decision tree search on a single pixel in the image.* The time to perform this operation can vary greatly from one pixel to the next, but for our average-case complexity analysis, we use the average value of $c_2 = 4.3 \times 10^{-3}$ seconds per pixel.
3. *Cluster pixels into target detections.* If we assume that the target and clutter rate per pixel is constant, then the clustering rate per original image pixel will also be a constant. We have measured this rate to be approximately $c_3 = 6.1 \times 10^{-4}$ seconds per pixel.
4. *Transfer a list of target detections between two processors that have a direct connection in the network.* The size of this list depends on the number of targets and the amount of clutter in the image. Because the ARTM algorithm never detects more than a few targets or false alarms in an image, we can assume that the size of this list is essentially constant, and hence, any list of target detections can be transferred in constant time $c_4 = 1.0 \times 10^{-3}$ seconds.
5. *Merge two lists of target detections into a single, consistent list of target detections.* Because we assume that the length of a list of target detections is a constant, the time to combine two lists, c_5 , will also be constant. We take $c_5 = 3.0 \times 10^{-3}$ seconds.

Using the above basic operations, we can calculate the run times of the sequential and parallel algorithms. The sequential algorithm consists of a tree search (time c_2) for each of the n^2 pixels plus the pixel clustering (time c_3) for n^2 pixels. Thus, the sequential run time is

$$T_1 = (c_2 + c_3) n^2$$

If we let $L(p)$ denote the number of levels in the processor interconnection hierarchy containing p processors, then for the parallel algorithm, we have:

1. *Image propagation time:* Each processor may need to send the image to up to five lower-level processors in the interconnection network. The total image propagation time is thus $5c_1n^2$ seconds for each of the $L(p) - 1$ propagation steps.
2. *Tree search time:* c_2 seconds for each of the n^2/p pixels.
3. *Pixel clustering time:* c_3 seconds for each of the n^2/p pixels.
4. *Results propagation time:* A processor may need to receive target detection lists from up to five lower level processors in the interconnection network. Because a processor can read from only one of its communica-

tions channels at a time, the time for results propagation is $5c_4$ seconds for each of the $L(p) - 1$ propagation steps.

5. Target list merge time: A processor can have up to six target lists that need to be merged (counting its own). Since these are merged in pairs, five merges may be required, and thus the merge time is $5c_5$ for each of the $L(p) - 1$ merge steps.

The total parallel run time is then

$$T_p = (L(p) - 1) \times (5c_1n^2 + 5c_4 + 5c_5) + (n^2/p) \times (c_2 + c_3)$$

and the speedup as a function of n and p is

$$S = \frac{(c_2 + c_3)n^2}{(L(p) - 1) \times (5c_1n^2 + 5c_4 + 5c_5) + (n^2/p) \times (c_2 + c_3)}$$

To evaluate the above, we need an expression for $L(p)$, the number of levels in the interconnection network containing p processors. (For our purposes, we can assume that all levels in the interconnection network are full.) Let $r_d(k)$ denote the number of processors of degree d (i.e., which have d children) at level k in the network. Then from Figure 3 we can see that

$$\begin{aligned} r_5(0) &= 1 & r_3(0) &= 0 \\ r_5(k) &= 2r_5(k-1) + 3r_3(k-1) \\ r_3(k) &= 3r_5(k-1) \end{aligned} \quad k \geq 1$$

Solving these linear recurrence relations, we get

$$\begin{aligned} r_5(k) &= \beta_1 \alpha_1^k + \beta_2 \alpha_2^k \\ r_3(k) &= \beta_3 \alpha_1^{k-1} + \beta_4 \alpha_2^{k-1} \end{aligned} \quad k \geq 1$$

where

$$\begin{aligned} \alpha_1 &= 1 + \sqrt{10} & \alpha_2 &= 1 - \sqrt{10} \\ \beta_1 &= \frac{\sqrt{10} + 1}{2\sqrt{10}} & \beta_2 &= \frac{\sqrt{10} - 1}{2\sqrt{10}} \\ \beta_3 &= \frac{3\sqrt{10} + 3}{2\sqrt{10}} & \beta_4 &= \frac{3\sqrt{10} - 3}{2\sqrt{10}} \end{aligned}$$

The number of processors in level k of the network is then $r_3(k) + r_5(k)$, and $L(p)$ can be calculated by the expression

$$L(p) = \min_l \left[\sum_{k=0}^{l-1} (r_3(k) + r_5(k)) \geq p \right]$$

Figure 6 shows the theoretical speedup of the parallel ARTM algorithm as a function of p for a number of values of n . The figure shows that larger and larger speedups can be obtained as the problem size increases. For a given problem size, however, the overhead due to interprocessor communication limits the speedup that is obtainable when the number of processors is increased. Table 1 presents an example of how the system can be "scaled" to maintain an approximately fixed level of performance (i.e., constant run time) as the image size increases. The number of processors required is roughly proportional to the number of pixels in the image.

TABLE 1. Number of processors required to maintain a fixed performance level (< 1 second run time) as image size increases.

Image Size	Run Time (sec)	No. of Processors
8x8	0.275	1
16x16	0.572	2
32x32	0.902	5
64x64	0.975	19
128x128	0.991	79
256x256	0.999	390

PERFORMANCE

Due to the ease of implementation, our first parallel implementation of ARTM used shared memory for inter-processor communication. All shared data structures were stored in a single memory that was shared by all processors. Table 2 lists the elapsed run times and speedups for a number of test images when the algorithm is run on one to four processors. As is apparent from the table, contention for the shared memory becomes a problem when running with three and four processors and severely limits the algorithm's speedup.

Our second implementation of ARTM performs message passing in a distributed memory architecture: after the first processor receives the image data from the host, it broadcasts the data to the other processors in the system via its parallel communication ports. Table 3 lists the elapsed run times and speedups for this system. Here, we obtain a nearly linear speedup of the algorithm. Figure 7

Figure 6: Speedup of the parallel ARTM algorithm as a function of the number of processors.

compares the speedups of the shared and distributed memory architectures.

CONCLUSIONS

We have parallelized the ARTM ATR algorithm and have implemented it on a scalable architecture of C40 processors. The scalability of this system has been demonstrated empirically for a small number of processors and also theoretically for larger numbers of processors. Using these results, one can estimate the number of processors

required to obtain a given level of performance in a particular application.

The commercial, off-the-shelf hardware that is used in this system and the open architecture nature of the hardware and software made the system affordable and easy to work with. As such, it is appealing to look at other applications that might benefit from the use of scalable architectures. We are currently considering the feasibility of implementing a multiple-hypothesis tracker [7] on this system. There are a number of broad battlefield applica-

TABLE 2. Elapsed time (in seconds) and speedup for the shared memory algorithm. (Shown as elapsed time/speedup.)

# Proc. Image	1	2	3	4
metd0900	78.2/1	42.3/1.8	34.7/2.3	32.7/2.4
metd0909	107.5/1	56.6/1.9	46.4/2.3	42.7/2.5
metd1268	133.3/1	70.7/1.9	59.1/2.3	54.5/2.4
metd2523	121.0/1	63.9/1.9	52.2/2.3	48.8/2.5
metd2656	116.2/1	61.6/1.9	51.7/2.2	47.5/2.4
metd3241	83.8/1	44.8/1.9	37.7/2.2	34.6/2.4
metd3321	84.7/1	45.3/1.9	37.1/2.3	34.5/2.5
metd3785	97.1/1	51.7/1.9	44.0/2.2	40.3/2.4
metd4446	124.0/1	65.7/1.9	55.7/2.2	50.9/2.4
metd5045	79.7/1	42.5/1.9	35.3/2.3	33.1/2.4
metd5264	88.6/1	48.3/1.8	40.1/2.2	38.0/2.3
average speedup	1.0	1.9	2.2	2.4

TABLE 3. Elapsed time (in secs.) and speedup for the distributed memory algorithm.

# Proc. Image	1	2	3	4
metd0900	59.4/1	29.9/2.0	20.3/2.9	15.3/3.9
metd0909	75.6/1	37.8/2.0	25.7/2.9	19.3/3.9
metd1268	91.1/1	45.6/2.0	30.6/3.0	23.0/4.0
metd2523	84.9/1	42.3/2.0	28.3/3.0	21.4/4.0
metd2656	79.6/1	39.7/2.0	26.8/3.0	20.3/3.9
metd3241	58.1/1	29.3/2.0	19.7/2.9	14.9/3.9
metd3321	59.5/1	30.0/2.0	20.2/2.9	15.3/3.9
metd3785	66.4/1	33.4/2.0	22.6/2.9	16.8/4.0
metd4446	84.3/1	42.0/2.0	28.4/3.0	21.3/4.0
metd5045	55.4/1	27.8/2.0	18.9/2.9	14.4/3.8
metd5264	66.3/1	33.4/2.0	22.7/2.9	17.2/3.9
average speedup	1.0	2.0	2.9	3.9

Figure 7: Speedup of the distributed and shared memory algorithms.

tions for which a scalable architecture approach might also be feasible:

1. Terrestrial, atmospheric, and space-borne sensor images are merged with digital map and entity (friendly and enemy) data to provide realistic fly-through simulations directly to the battlefield prior to tactical engagement.
2. The deployment of coordinated, autonomous, air and ground robotics providing real-time reconnaissance, surveillance, and target acquisition, decoy, mine detection and clearance, electronic warfare, etc.
3. The fusion of all available sensor information enabling real-time situation assessment and awareness combined with advanced human-computer interfaces (visualization, natural language, intelligent database access) to enable the rapid assimilation of this critical information.
4. A mobile, distributed, command and control network that supports real-time, world-wide, teleoperation. A simple example of this is the telemedical application where conus medical experts are interactively supporting medics in field operations.
5. The adaptive, hybrid, terrestrial satellite, communications networks that enable the above.

REFERENCES

1. Kai Hwang and Faye A. Brigg, *Computer Architecture and Parallel Processing*, McGraw-Hill Inc., 1984, pp. 121-123.
2. *Considerations in Choosing a High-Performance Floating-Point DSP*, a technical white paper by Texas Instruments, Inc., 1993, pp. 7.

3. *SPOX-LINK Programming Guide*, Spectrum Microsystems, Inc., 1993, pp. 11-13.

4. Andrew Kramer, David Perschbacher, Randy Johnston, and Teresa Kipp, *Relational Template Matching for FLIR Automatic Target Recognition*, Proc. SPIE, Vol. 1957, 1993.

5. Steven Savitt et al., *ATR Relational Template Matching*, Final technical report prepared for the Army Research Laboratory, September 1993.

6. Ananth Grama, Anshul Gupta, and Vipin Kumar, *Isoefficiency: Measuring the Scalability of Parallel Algorithms and Architectures*, IEEE Parallel & Distributed Technology, August 1993.

7. D. B. Reid, *IEEE Transactions on Automatic Control*, December 1979, pp. 843-854.