1-1-2002

Optimization of image processing algorithms via communication hiding in distributed processing systems

Aaron M. Cordes
Iowa State University

Optimization of image processing algorithms via communication hiding in distributed processing systems

by

Aaron M. Cordes

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Engineering

Program of Study Committee:
Arun Somani, Major Professor
Manimaran Govindrasu
David Fernandez-Baca

Iowa State University

Ames, Iowa

2002

Graduate College
Iowa State University

This is to certify that the master's thesis of

Aaron M. Cordes

has met the thesis requirements of Iowa State University

Signatures have been redacted for privacy

Table of Contents

List of Figures iv

List of Tables v

Abstract vi

1. Introduction 1

2. Problem Overview 2
   2.1. Previous work in image processing and object extraction 2
   2.2. ALOA system details 3
   2.4. Runway detection and Hough transform overview 6
   2.5. Hough transform performance optimizations 11
      2.5.1. Conversion to Duda/Hart algorithm 11
      2.5.2. Conversion to integer arithmetic via arithmetic code 13
      2.5.3. Optimizing the arithmetic code 14
      2.5.4. Varying values for θ_{quanta} 15
      2.5.5. Conclusions 17

3. Parallel Computation and Algorithm Pipelining in ALOA 19
   3.1. Previous Work in Algorithm Pipelining 20
   3.2. Pipelining the ALOA system 22
      3.2.1. Results of pipelining in NIU system 23
      3.2.2. Results of pipelining in xCat cluster 24
      3.2.3. Conclusions 26

4. Communication Hiding in ALOA 27
   4.1. Previous work in communication hiding 27
   4.2. Communication hiding in ALOA and its relevance in high-speed networks 29
      4.2.1. Results of communication hiding in NIU system 32
      4.2.2. Results of communication hiding in xCat cluster 33
      4.2.3. Conclusions 35

5. Summary 36

References 38

Acknowledgements 41

List of Figures

Figure 1. Sample radar image from a landing airplane. 4
Figure 2. Data flow diagram for ALOA system 5
Figure 3. Original sample image of airplane landing. 7
Figure 4. Sample image after noise reduction (left) and edge detection (right) 7
Figure 5. Representation of a straight line in polar coordinate space. 8
Figure 6. Duda and Hart version of Hough transform algorithm. 9
Figure 7. Graphical representation of Hough transform. 9
Figure 8. Output from ALOA system with edges of runway detected. 10
Figure 9. Original slope/intercept form of Hough transform. 12
Figure 10. Resulting images for various values of θ_{quanta} 16
Figure 11. Theoretical algorithm pipelined system 20
Figure 12. Speedup due to two stages of pipelining in the NIU system. 24
Figure 13. Speedups in xCat cluster as pipeline stages increase 25
Figure 14. Efficiency of hardware usage in xCat cluster as pipeline stages increase. 25
Figure 15. Theoretical distributed system with data source node providing data to all processing nodes. 31
Figure 16. Speedup of two-stage NIU pipeline due to communication hiding. 32
Figure 17. Speedup in NIU client/server system due to communication hiding 33
Figure 18. Speedup in pipelined system on xCat cluster due to communication hiding. 34
Figure 19. Speedup in distributed system on xCat cluster due to communication hiding. 34

List of Tables

Table 1. Original runtimes of Hough transform. 11
Table 2. Comparison of runtimes of two versions of Hough transform. 13
Table 3. Speedup due to integer-based Hough transform. 14
Table 4. Speedup due to replacing division with binary shift. 15
Table 5. Runtimes and speedups for various values of θ_{quanta}. 17

Abstract

Real-time image processing is an important topic studied in the realm of computer systems. The task of real-time image processing is found in a wide range of applications, from multimedia systems to automobiles to military systems. Typically these systems require high throughput and low latency to perform at their required specifications. Therefore, hardware, software, and communications optimizations in these systems are very important factors in meeting these specifications.

This thesis analyzes the implementation and optimization of a real-world image processing system destined for an aircraft environment. It discusses the steps of optimizing the software in the system, and then looks at how the system can be distributed over multiple processing nodes via functional pipelining. Next, the thesis discusses the optimization of interprocessor communication via communication hiding. Finally, it analyzes whether communication hiding is even necessary given today's high-speed networking and communication interfaces.

1. Introduction

Real-time image processing is an important problem being studied today. The problem has a wide range of applications, from multimedia, to military, to automotive realms. Real-time image processing usually consists of three steps. First, the original image is received. Second, the image is analyzed, and possibly modified or enhanced. Third, the resulting enhanced image is output for a user to visualize. The requirements of these systems generally entail a relatively high output rate and low latency. Therefore, the problem of optimizing the hardware, software, and communications of these systems to maximize output rate and minimize latency is very important.

This thesis discusses the implementation and optimization of a real-world image processing system. The system is to be installed in an aircraft. The system's function is the analysis and modification of radar images so that the aircraft can land in low visibility. However, many of the topics in this thesis are applicable to general image processing and distributed systems.

There are three main parts to this thesis. The first part discusses optimization in software of image processing algorithms used in a radar analysis system. These optimizations will be analyzed for their effectiveness. The second part of the thesis discusses steps taken to reduce the runtime of the algorithms through parallel processing. In this case, the algorithms are parallelized with functional pipelining, which pipelines the algorithms over multiple processing nodes. The third part of the thesis looks at lowering the runtimes of the algorithms through hiding any necessary communication in the background. Also, the question of whether communication hiding is necessary given the current batch of high-speed networking hardware is discussed.

2. Problem Overview

The focus of the thesis is on the implementation and optimization of the Autonomous Landing and Obstacle Avoidance system (ALOA). ALOA is a system designed in conjunction with Lockheed Martin that is to be used on aircraft. The main goal of the system is to enable a pilot to land aircraft on a runway, even in the case of low visibility due to fog, rain, snow, or other weather-related problems. The ALOA system achieves this goal through analysis of radar images gathered from the aircraft's high-frequency active radar system.

2.1. Previous work in image processing and object extraction

Analysis of radar images using image processing algorithms is a common problem. A somewhat similar system to ALOA was introduced in [1], but is designed for automobiles instead of aircraft. The paper describes an algorithm for an all-weather driving assistance system. The system employs a high-frequency active radar system – as does the ALOA system – to create a snapshot of the surroundings of the automobile. The system then tries to detect the edges of the road and any possible obstacles. This is done using a parabolic template, which approximates the road curvature as a parabolic curve, then parameterizes this curve in polar coordinate space. The curves are analyzed and a likelihood function picks the best candidate for the edge of the road. Once the edge is found, obstacles are detected by the radar signature reflected back to the radar system.

A low-altitude aircraft-mounted system is discussed in [2] that is used to detect landmines using Forward Looking Infrared (FLIR) images. The system was implemented on a network of processing nodes, with the landmine detection algorithm pipelined over those nodes. The algorithm consists of five steps:

1. Image correction.
2. Target cuing, which picks possible landmine candidates out of an image.
3. Target shape analysis, which decides if the shapes are landmines.

4. Target spatial analysis, which decides where the shapes physically are.
5. Knowledge integration, which combines all results from the previous steps to decide if a minefield is indeed in the image.

It was stated that the probability of detection of individual landmines in a synthetic image was approximately 90%, with a probability of false alarm of 2%. For real FLIR images, the results were approximately 64.6% detection probability and 25.6% false alarm probability.

Synthetic aperture radar (SAR) is a common method used to form ground images taken from high altitudes (either from an airplane or a satellite). These images can then be analyzed to extract and enhance interesting features.

In [3], SAR images are used to analyze land use by humans. A Bayesian network analyzes SAR images in order to classify portions of the image into the specific categories of forest, agricultural, vegetation, and build-up. This system can be used in the realm of community and regional planning to help a community plan for future growth.

In [4], SAR satellite images are used to extract linear features, such as roads and paths, from a high altitude image. By making certain assumptions about roads in a SAR image, such as they are thin, elongated structures and are dark with respect to their surroundings, roads are extracted by using morphological filtering. The authors showed that the method works for cases in which road widths vary widely, and highly textured complex images. However, this method could not be used in a system with high frame-rate needs because the processing of each frame of data lasts on the order of minutes. The system is designed to map remote areas of previously uncharted terrain.

2.2. ALOA system details

As stated previously, the ALOA system is designed for detecting the edges of a runway and any obstacle in the runway as an airplane is landing.

To perform this duty, the system consists of a radar transmitter and receiver, an A/D converter to convert the radar data to digital form, computational hardware to analyze the data, and an output display to show the results to the pilot. The radar transmitter/receiver is mounted in the airplane and sends out high-frequency radar signals (with frequencies in the tens of GHz). At these high frequencies the radar is only effective over relatively short ranges, usually only a mile or two. However, the radar signal that is returned begins to take shape, and individual objects can be seen in the image. An example radar image can be seen in Figure 1, which shows a runway (with the center being the black stripe) with a grove of trees in front of it. Objects that absorb the radar signal, such as the trees, appear dark in the formed image. Meanwhile, objects that reflect the radar signal, such as the snow on the runway, appear as bright spots.

Figure 1. Sample radar image from a landing airplane.

Once the radar signal is received, it is passed to an A/D converter, and then the raw data is converted from the time domain to the frequency domain using a Fast Fourier Transform (which can

be implemented in either hardware or software). The data in the frequency domain can now be used to construct an image of the radar signal, such as the image in Figure 1.

Once the image has been constructed, there are two main tasks to perform. The first is runway extraction, which entails performing analysis on the image to predict where the edge of the runway is most likely to be. The second task is obstacle avoidance. For obvious safety reasons, the plane should not be landed if there are any obstacles in the runway. The obstacle avoidance system attempts to detect any obstacles on the runway and output the obstacles to screen along with the runway extraction data. Once the tasks are performed, the computed data can be overlaid on the original image or on an enhanced version of the image. The flow diagram for the system can be seen in Figure 2.

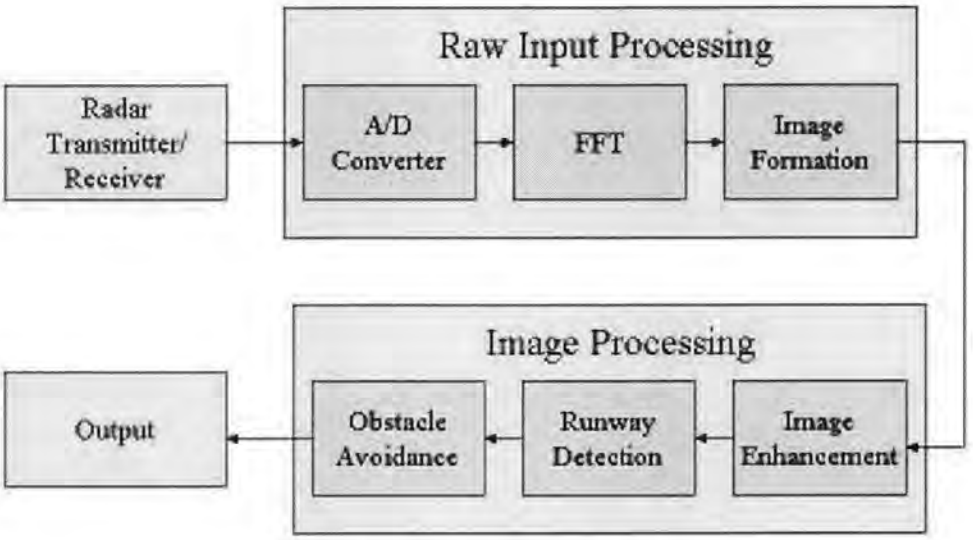


Figure 2. Data flow diagram for ALOA system.

The system is similar in goals to the automobile driving assistance system in [1], with a few differences. The ALOA system must be concerned with the edges of the runway, but it handles the situation differently. The assumption is made that the runway edges will be straight lines, not curves

as in [H]. Since this simplifies the situation, much different algorithms can be used to analyze the radar images.

The ALOA system is eventually intended to output a real-time video sequence that the pilot can follow in low visibility. Consequently, there is a minimum quality of service of 10 frames per second (fps) required for this system. This constraint is the leading motivation of studies performed in this thesis, and two problems were analyzed in this study. First, the problem of optimizing the algorithms in the system in order to meet the output constraints was analyzed. Second, the problem of distributing the algorithms over multiple processing nodes was analyzed.

2.4. Runway detection and Hough transform overview

Runway detection was focused on for this study because it was, at first, the most time consuming algorithm in the system. In the system evaluated, runway detection was performed by using the Hough transform algorithm. The Hough transform, first proposed in [5], can be used to detect shapes (for example, a circle or an ellipse) in an image, even if the image contains relatively large amounts of noise. In its simplest form, the Hough transform can be used to detect straight lines, which is useful in our case for detecting the edges of a runway as a plane is landing. Theoretically, the Hough transform can be used to represent any type of shape, but the required computation time greatly increases as the complexity of the shape increases. For example, a line can be represented by two parameters, slope and intercept. A circle would require three (x offset, y offset, radius), and an ellipse would require four (x offset, y offset, x radius, y radius). Each incremental increase in the number of parameters for the shape adds an additional dimension to both memory requirements and computational requirements, so anything but relatively simple shapes will overwhelm the processing and storage capabilities of today's computers.

Generally, preprocessing is required on the original image in order for the Hough transform to perform optimally [6]. Therefore, noise reduction and edge detection is performed before the Hough transform in order to simplify the work done by the Hough transform. In this system a Gaussian blur was used to reduce noise, and then a Sobel edge detection operator was used to simplify the scene. The resulting image is then fed to the Hough transform. A sample image and the results of each operator can be seen in Figure 3 and Figure 4.

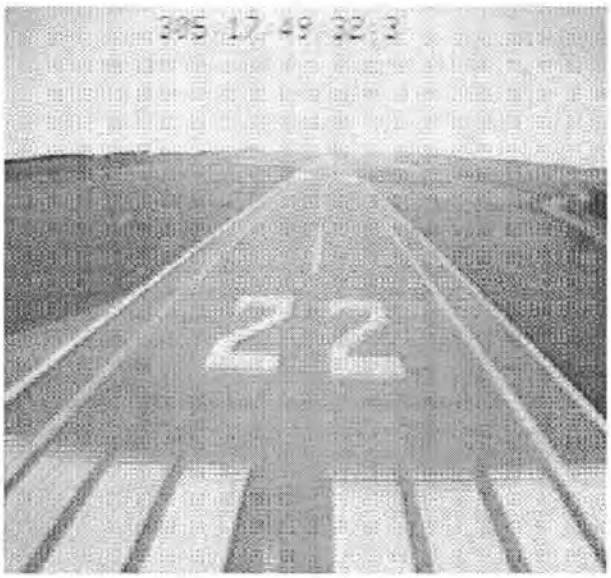


Figure 3. Original sample image of airplane landing.

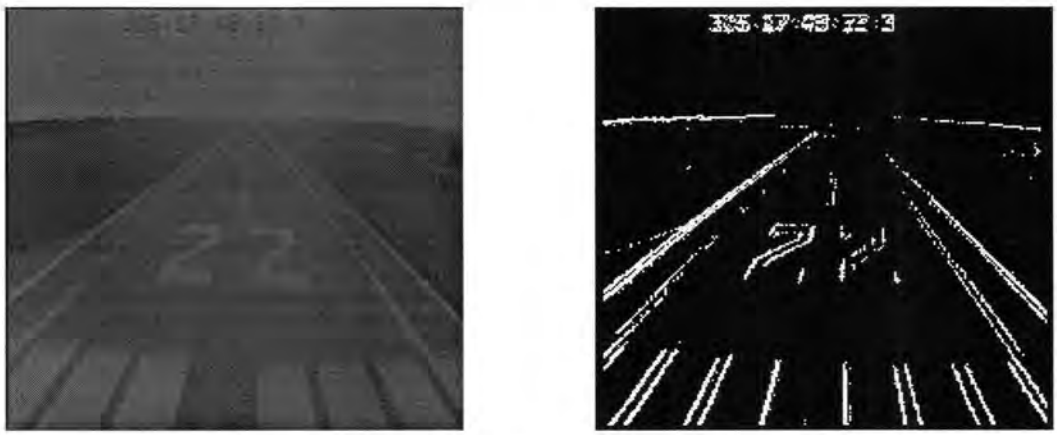


Figure 4. Sample image after noise reduction (left) and edge detection (right).

The Hough transform in general has two separate steps. First, the image is analyzed and votes are taken as to which are the most distinct straight lines in the image. Second, the votes are tallied and the most distinct lines are selected and overlaid on the original image. Duda and Hart [7] modified Hough's original algorithm (which uses slope/intercept form for line representation) in order to simplify computations. In the Duda/Hart algorithm, the lines are parameterized by the polar representation such that a line is represented by a radius and angle (ρ, θ) coordinate in the polar coordinate plane. This means that a line (ρ, θ) is drawn by finding the segment extending ρ units from the origin, at an angle of θ. The actual line being represented is then drawn perpendicular to this segment, as shown in Figure 5.

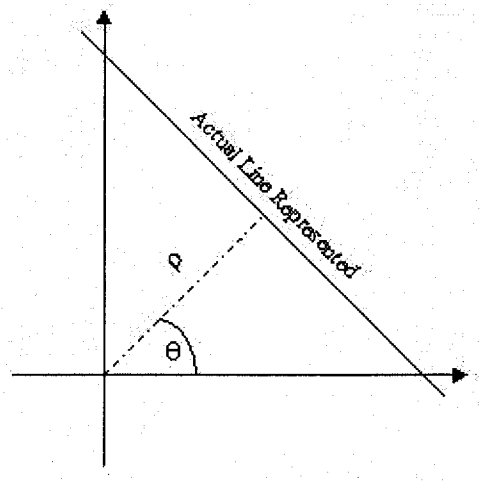


Figure 5. Representation of a straight line in polar coordinate space.

Therefore, the idea behind the Duda and Hart version of the Hough transform is that a given point (x, y) in an image can be a part of a discrete number of straight lines in an image, as long as θ also has a discrete set of values. For each value of θ the equation

$$\rho = x \cos(\theta) + y \sin(\theta) \quad \text{[Eq. 1]}$$

is computed to find the (ρ, θ) parameter for each of the possible lines. For each of these possible (ρ, θ) parameters computed, a bucket that corresponds to the [ρ, θ] coordinate is incremented. Each

bucket represents "votes" for the most distinct lines in the image. The overall Duda and Hart algorithm is shown in Figure 6.

```
for each pixel [x_i, y_j] in the original image {
        if pixel [x_i, y_j] is high (binary 1) {
                for (θ= 0; θ < θ_max; θ += θ_quanta) {
                        ρ = x_i cos(θ) + y_j sin(θ)
                        increment bucket corresponding with [ρ, θ]
                }
        }
}
```

Figure 6. Duda and Hart version of Hough transform algorithm.

Throughout the algorithm, the center of the original image is considered the planar origin so that all possible lines can be parameterized and drawn on the resulting image. When this algorithm is completed, the most distinct line in the image corresponds to the [ρ, θ] bucket with the largest value. These distinct lines appear as peaks or bright spots in a graphical representation of the Hough transform, as can be seen in Figure 7. As many line parameters can be picked out of the array of buckets as are deemed necessary to overlay on the original image.

Figure 7. Graphical representation of Hough transform.

The given problem statement for the ALOA system specified that the runways the system would be used on would be equipped with corner reflectors on the edges of the runway which reflect a very large amount of the radar signal back to the source. These reflectors would appear as bright spots on the radar image. Given that the line of these corner reflectors would appear as a very bright line in the radar image (compared to the rest of the scene), we are working under the assumption given in the problem statement that the two most distinct lines found by the Hough transform would conform to the edges of the runway. This assumption greatly simplifies the picking of the correct peak points in the Hough transform. At this point, the highest peak in the Hough transform with a positive slope, and the highest peak with the negative slope are considered to be the edges of the runway. Future work will entail making the peak extraction more robust.

Once the parameters for the most distinct lines are found, the lines can easily be drawn on the image. Figure 8 shows the results of the Hough transform after lines have been drawn on the image. The figure shows that the system detected the edges of the runway as the most distinct lines in the image, as was intended.

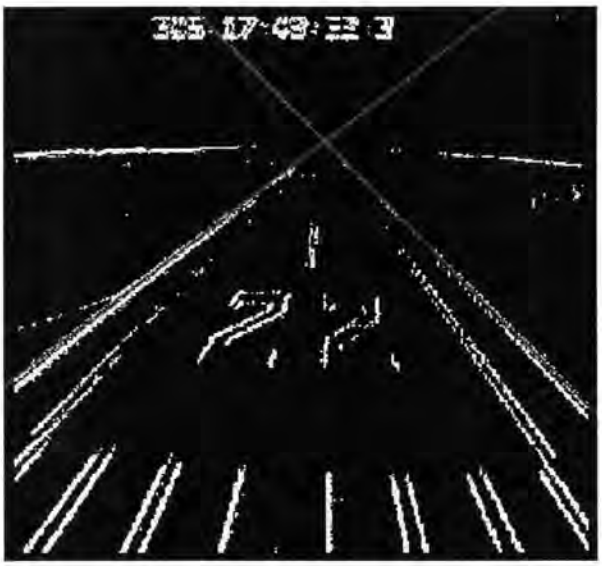


Figure 8. Output from ALOA system with edges of runway detected.

2.5. Hough transform performance optimizations

In this section, we discuss methods of speeding up the original Hough transform algorithm. The original Hough transform algorithm was too slow for the requirements of the system, which was specified as 10 fps. Just the above algorithm by itself would cause the system to be unable to meet the 10 fps requirement. Table 1 shows the time required for the original algorithm. All timings in this section were taken on a 1.5 GHz Pentium 4 computer with 512 MB of RAM.

Table 1. Original runtimes of Hough transform.

Image Size	Hough transform runtime
60x512	165.71 ms
120x512	360.97 ms
240x512	747.20 ms
480x512	1567.43 ms

2.5.1. Conversion to Duda/Hart algorithm

The original algorithm used in the ALOA system followed Hough's original algorithm and did no conversions to the polar coordinate space. Instead it used the simple slope/intercept form to describe a straight line. For each high pixel in an image, all the possible straight lines the pixel lies on is calculated. A bucket is incremented for each of these lines. The original algorithm is shown in Figure 9.

```
for each pixel [x_i, y_j] in the original image {
        if pixel [x_i, y_j] is high (binary 1) {
                for (k= 0; k < rows; k++) {
                        slope = (y_j – k) / x_i
                        θ = tan^-1(slope)
                        increment bucket corresponding with [k, θ]
                }
        }
}
```

Figure 9. Original slope/intercept form of Hough transform.

The original algorithm must convert all slopes to angles, otherwise the storage array would not be bounded as slopes increase to infinity in the case of vertical or close-to-vertical lines. The inverse tangent can be calculated fairly quickly using table lookups, but the reverse operation must again be calculated later on when searching the storage array for peak points, which adds to the runtime. Also, because k depends on the number of rows in the image, the work required grows much more quickly than the Duda/Hart algorithm as an image becomes larger. With the Duda/Hart algorithm, the internal loop is bounded by $\theta = 180°$ no matter what the image size, so the workload will not grow as quickly as the number of rows in an image increases.

As was mentioned, the slope/intercept representation of a line causes problems when the drawn line is either vertical or close to vertical. In this case, both the slope and intercept of the line approach infinity and become hard to handle in the system. These become special cases that increase computation time and make the system more complicated. Therefore, the decision was made to move to the Duda/Hart algorithm. At first glance, the Duda/Hart algorithm may not seem like much of an advantage, but a closer look shows why the algorithm is faster in the long run.

The formula $\rho = x_i \cos(\theta) + y_j \sin(\theta)$ consists of two trigonometric calculations, plus two floating point multiplications and a floating point addition. With no change, this is an expensive operation, but many techniques can be performed which make this operation relatively fast. The first method is to change the cosine and sine calculations to a table lookup. The table of cosine and sine values are precalculated and stored in the program. Then the calculations are a simple array reference, depending on the value of θ. If θ is an integer value, no conversions need to be done to reference the array, and the operation takes only as long as a memory reference in the computer system. If we assume that these values are stored in cache after the first reference (a reasonable assumption due to spatial locality), the memory reference is very fast. Table 2 shows that the speedup gained by moving to the Duda/Hart algorithm and implementing table lookup is very significant. The minimum speedup found is 4.51, while the maximum is 4.96.

Table 2. Comparison of runtimes of two versions of Hough transform.

Image Size (pixels)	Hough transform runtime (original Algorithm)	Hough transform runtime (Duda/Hart algorithm	Speedup (times faster)
60x512	165.71 ms	36.69 ms	4.51
120x512	360.97 ms	76.23 ms	4.73
240x512	747.20 ms	154.82 ms	4.82
480x512	1567.43 ms	316.25 ms	4.96

2.5.2. Conversion to integer arithmetic via arithmetic code

At this point, the main calculation of the Hough transform has been reduced to two floating point multiplications and a floating point addition. However, these calculations can all be done with integer arithmetic instead of floating point through use of an arithmetic code. As analyzed in [8], Equation 1 can be changed to $\rho = [x_i \cos_{scaled}(\theta) + y_j \sin_{scaled}(\theta)] / A$, where $\cos_{scaled}(\theta) = \text{floor}[A*\cos(\theta)]$, $\sin_{scaled}(\theta) = \text{floor}[A*\sin(\theta)]$, and A is an integer. This converts all of the arithmetic to integer, greatly speeding up the calculation. The authors of [8] showed that the speedup gained on

various systems with A = 1000 ranged from 2.00 to 3.53, depending on the size of the image tested. In this study, moving from floating point to integer arithmetic did not result in as large of a speedup, as can be seen in Table 3, but the results are still significant. One reason for the difference may be that the floating point hardware used in this study is more efficient than the floating point hardware used in [8], which would reduce the advantage of using the integer-based Hough transform. As in the study in [8], the speedup tends to increase as the image size increases.

Table 3. Speedup due to integer-based Hough transform.

Image Size (pixels)	Speedup by moving from floating point to integer arithmetic (A = 1000)
60x512	1.56
120x512	1.76
240x512	1.81
480x512	1.82

2.5.3. Optimizing the arithmetic code

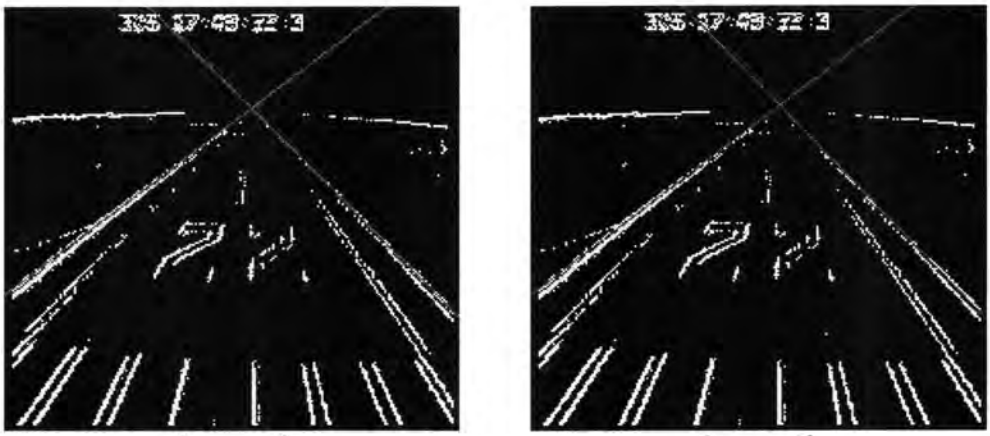
A simple extension of this idea that was not tested in [8] is setting A to a power of 2. In this case, we can replace the integer division by A with a binary shift to the right of $\log_2(A)$ places, which is a much simpler and faster operation in microprocessors. It was found that this simple procedure resulted in a larger speedup than using A = 1000. It is easy to see why we get such a speedup. Originally, the algorithm had two trigonometric calculations, two floating point multiplications, and one floating point addition in each step. Now it only has two integer multiplications, an integer addition, and a binary shift. Table 4 shows the speedup gained by setting A to a power of 2 (in this case, 1024) instead of 1000.

Table 4. Speedup due to replacing division with binary shift.

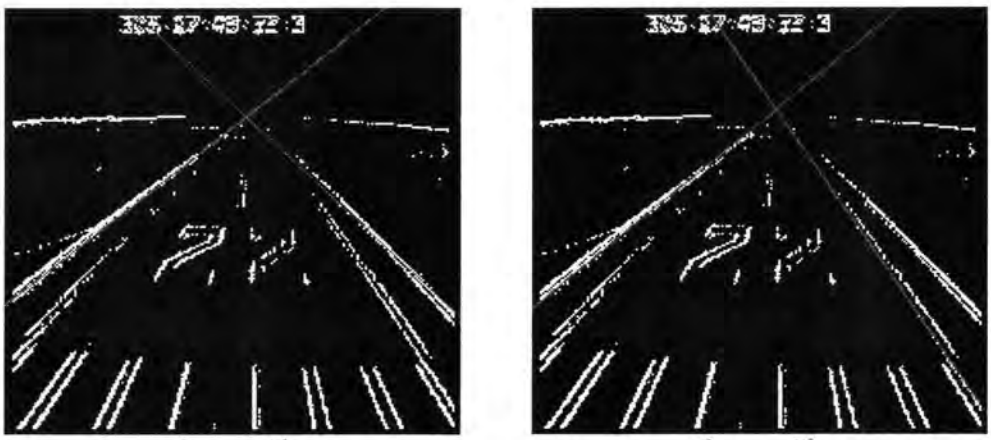
Image Size (pixels)	Additional speedup gained by setting A to a power of 2 instead of A = 1000
60x512	1.48
120x512	1.41
240x512	1.45
480x512	1.41

2.5.4. Varying values for θ_{quanta}

A final way that the Hough transform can be sped up is by increasing the quanta value for θ. It is expected that an increase of θ_{quanta} will result in very close to a linear speedup in computation time for the Hough transform. In general, θ is stepped by the quanta value from 0 to 180°, and the original value used for θ_{quanta} was 1°. θ can be stopped at 180° because $\cos(\theta) = -\cos(\theta + 180°)$ and $\sin(\theta) = -\sin(\theta + 180°)$. Therefore, going past 180° results in redundant calculations. However, the optimal value of the θ_{quanta} is not as easy to determine. If the quanta value is too large, then the accuracy of the final result is lowered. If the quanta value is too small, then the results in the buckets show a "smoothing" effect, because different values that increment a single bucket for larger values of θ_{quanta} now increment many different buckets [6][9]. In this case, there may not be a distinct peak represented in the resulting buckets, and there is a possibility that a less-distinct line will actually be seen as the maximum. Since the goodness of the resulting transform is a largely subjective matter, various values for θ_{quanta} were evaluated for a sample image, and the results are shown in Figure 10.



$\theta_{quanta} = 1^{\circ}$ $\theta_{quanta} = 3^{\circ}$



$\theta_{quanta} = 5^{\circ}$ $\theta_{quanta} = 7^{\circ}$

Figure 10. Resulting images for various values of θ_{quanta}.

Figure 10 shows that the accuracy of the drawn lines is maintained for θ_{quanta} equal to 1°, 3°, and 5°. Close examination shows that there are slight differences in the results, but all three values found the outside edges of the runway to be the most distinct lines. However, when θ_{quanta} is increased to 7°, a less prominent line is found to be the right edge of the runway.

Timings were taken on the sample image to determine the speedup given by increasing the value of θ_{quanta}. θ_{quanta} was tested for values of 1°, 3°, and 5°. The results of the time tests can be seen in Table 5. These timings were taken with the integer Hough transform using the binary shift.

Table 5. Runtimes and speedups for various values of θ_{quanta}.

Image Size (pixels)	Runtime ($\theta_{quanta} = 1°$)	Runtime ($\theta_{quanta} = 3°$)	Runtime ($\theta_{quanta} = 5°$)	Speedup 1° to 3°	Speedup 1° to 5°
60x512	18.53 ms	4.89 ms	3.21 ms	3.79	5.77
120x512	38.74 ms	9.48 ms	5.82 ms	4.09	6.66
240x512	81.40 ms	18.07 ms	11.28 ms	4.50	7.21
480x512	159.22 ms	36.80 ms	21.84 ms	4.32	7.29

An interesting result of these timings is that increasing θ_{quanta} by N results in a speedup larger than N. The reason is that caching is more efficient as N increases because there is a smaller set of accesses into the sine and cosine lookup tables and the array of buckets.

2.5.5. Conclusions

The results in the previous section showed that the various algorithms for runway detection were sped up enough so that the system will be able to achieve the required 10 fps output necessary. It was shown that using the Duda/Hart algorithm instead of the original Hough algorithm results in gains in speed and programming simplicity. Speedups from the Duda/Hart algorithm ranged from 4.51 to 4.96. Converting the algorithm to only integer arithmetic via an arithmetic code resulted in speedups from 1.56 to 1.82. Optimizing the arithmetic code by using a binary shift operator instead of division resulted in additional speedups from 1.41 to 1.48. Finally, scaling the value of θ_{quanta} by N showed additional speedups larger than N, due to more efficient caching.

However, there are still other steps that must be performed which will reduce the output rate of our images. Although not yet implemented, the obstacle avoidance part of the system will probably be computationally complex and push our output under the necessary limit of 10 fps. Also, the runtimes for the Hough transform listed above are for the average case. The worst-case runtime of the Hough transform used for line detection is much slower than the average case used for the timings in the previous section. Timings taken showed that the worst-case runtime for the Hough

transform on a 60x512 image is 76.05 ms. This leaves only 23.95 ms for the rest of the algorithms to be performed, which is too little time considering that image preprocessing and obstacle detection must still be performed. Ultimately, the ALOA system will have to make real-time performance guarantees, meaning the worst-case runtimes must be considered. Because of this, other methods for speeding up computations must be considered, with the most obvious one being parallel processing.

3. Parallel Computation and Algorithm Pipelining in ALOA

There are two basic methods of parallel computation. A computation can be described as data-parallel or function-parallel [10]. A computation that is function-parallel breaks a program down into multiple functional units that can be executed simultaneously. A data-parallel computation breaks up data and divides it as equally as possible among multiple processing nodes. Generally in a data-parallel computation the processing nodes perform the same task, but on different sets of data. When all the processors have completed their task, the data is merged again.

According to [11], it is possible to combine these two parallel computation methods to form four separate styles of parallel computation.

1. **Concurrent Function-Parallel Computation** – Processors perform different tasks at the same time. It is stated that this is the most common method used in parallel and distributed systems.
2. **Concurrent Data-Parallel Computation** – Processors perform the same task, but on different sets of data. At the lowest level, a processor with vectoring capabilities performs concurrent data-parallel computations.
3. **Pipelined Function-Parallel Computation** – Processors are organized in stages, and each stage performs a different task. Data that flows through the pipeline is modified at each stage, then sent to the next stage.
4. **Pipelined Data-Parallel Computation** – Processors are again organized in stages, but each stage performs the same task. Data is broken up into smaller pieces and flows through each stage of the pipeline. Communication about the data set at each stage of the pipeline may have to be exchanged between stages.

In the ALOA study, it was observed that the ALOA system is completely linear, with multiple well-defined steps. As a result, it was decided that the type of parallelism to be implemented

in ALOA is pipelined function-parallel computation. Using this model, a group of processing nodes was set up as a pipeline, with each stage of the pipeline being assigned a specific set of tasks applicable to the ALOA system. Each node waits for a set of data (a radar frame, in the case of ALOA), performs its assigned tasks, and passes the resulting data to the next stage in the pipeline. The goal of this system is to increase the throughput without greatly increasing the complexity of the system. Figure 11 depicts an example pipelined system.

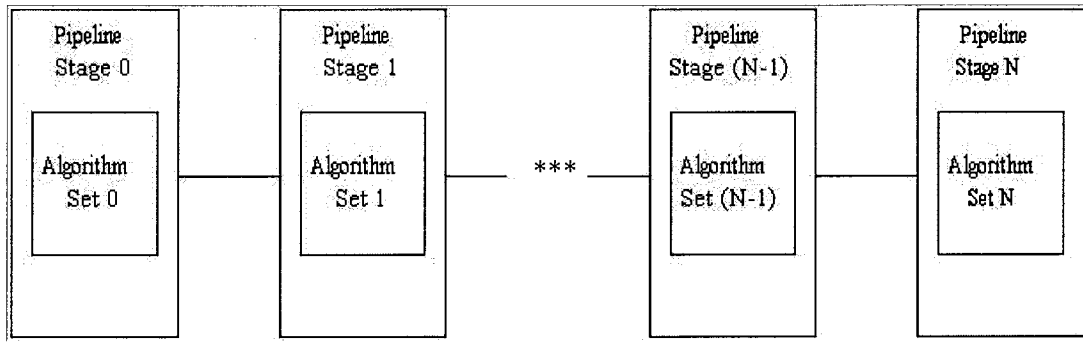


Figure 11. Theoretical algorithm pipelined system.

3.1. Previous Work in Algorithm Pipelining

A study was done in [12] with the goal of making parallel computations of a Scene Adaptive Transform Coding algorithm more efficient. It was stated that the algorithm had been parallelized on a cluster of eight computers and found significant speedups. However, as the number of processing nodes in the cluster increased, the communication overhead began to overwhelm the increase in processing power. To alleviate this problem, the authors implemented functional concurrency in the form of a Pipeline-Tree Architecture (PTA). The goal of the PTA is not to reduce the computation time of the coding algorithm (although it would be desirable), but to lower communication overhead so that the system will be more efficient as the number of processing nodes scales.

To utilize the pipeline efficiently, the input data was partitioned into equal-sized non-overlapping blocks. Each of these blocks is processed independently at each processing node. The

authors note that the independent processing limits the amount of synchronization and communication between processors. The PTA used in this system had 15 processing nodes, with six parallel pipelines each working on a block of data. Equations were developed which describe the situations where the PTA had less communication overhead than the original Tree Architecture (TA). In the actual implementation, it was found that only a 2.2% increase in throughput was gained by moving from the TA to the PTA, but it was noted that is due to the fact that two of the algorithms in the pipeline had a much larger execution time than the rest of the stages in the pipeline, which limited any increases in throughput. The authors did find greater scalability, which was the original intent.

The authors of [13] discuss the importance of scheduling in a distributed pipelined system. They note that many real-time applications that can utilize a pipelined system are implemented on heterogeneous systems with complex timings. This, combined with the strict timing requirements of real-time systems, causes scheduling to be a task of high importance. The authors of [13] assume that there can be multiple application streams that can be pipelined in the system. As a result, the authors developed partitioning rules designed to eliminate unnecessary buffering and latency in the system, ensure the correct function of the pipelining strategy, and allow the problem to be decomposed to take advantage of multiple processing nodes. Using these rules, the authors finally developed equations that analyzed the end-to-end latency and schedulability of the system.

The landmine detection system discussed previously [2] was implemented as a pipeline over multiple processing nodes. It was stated that the low-level algorithms in the system were implemented on a network of processing nodes, where each processing node is an array of i860 vector processors. The communication links between the pipeline stages consisted of a serial point-to-point link for simplicity. It was found that using this rudimentary hardware (by today's standards), the system was still able to perform well and output results in real-time.

3.2. Pipelining the ALOA system

Pipelined function-parallel computation sets up very well for the ALOA system. First, all the algorithms in ALOA are sequential, so the system is inherently linear. This makes it very easy to break up the given tasks over multiple processors. Second, the computation times for each frame of data are relatively small and the system requires high throughput. Parallel programming interfaces such as MPI [14] are not geared towards small individual tasks, because the overhead of communication required to parallelize an algorithm is large relative to the problem size [15]. The only communication required in a pipelined version of ALOA is the passing of frames between processing nodes, meaning simpler communication, such as in a pipeline, is adequate. Third, programming algorithms to run on the pipeline is very simple. Once the pipeline is implemented, individual tasks can be plugged into or removed from any processing node without any additional work. On the other hand, data level parallelism requires the modification of each individual algorithm to take advantage of the parallel libraries.

There are a couple of caveats to the pipelined system. First, maximum performance requires that the runtimes of the tasks on each node are as close to equal as possible. The throughput of the system is limited by the slowest set of tasks at a node. Therefore, analysis of each algorithm and its runtimes is very important before assigning tasks to a processing node. Second, increasing the number of stages in the pipeline can increase the throughput, but can also increase the latency of results. The ALOA system requires both high throughput and low latency so that frames being output to the pilot are still relevant to the scene outside the airplane.

To study the advantages gained by moving from a single processor to a pipelined system, the study utilized a piece of hardware from Lockheed Martin Corporation called the Network Interface Unit (NIU). The NIU is essentially a higher-level messaging protocol wrapped around a fibre-channel interface, which provides 1 Gb/s transfer speeds. The first test system consisted of two hosts, each containing a 1.5 GHz Pentium 4 processor and 512 MB of RAM, and equipped with a NIU

interface card. The second test system was a 32 node xCat cluster [16], with each node containing two 1133 MHz Pentium III processors connected by a high-speed Myrinet network.

The main interest of the study was to see how much of a gain in speed (if any) is given by distributing the ALOA algorithms over multiple processing nodes and implementing the algorithmic pipeline discussed above. Two cases were tested in order to observe the negative affect on throughput due to differing workloads on each processing node. The first case consisted of distributing slightly different workloads across the pipeline stages. To do this, the algorithms used in the previous section to perform runway detection were analyzed and split up close to evenly over the two processing nodes in the NIU system, but the workloads still differed by 10-15%. The first processing node was assigned the tasks of receiving the original image, performing the Gaussian noise reduction and Sobel edge detection, then passing this data to the second node which performed the Hough transform and line drawing. The second case consisted of distributing perfectly equal workloads across the processing nodes.

In both cases the nodes were synchronized by the second stage sending a computation complete message to the first stage when the image had been output. At this time, the first stage was allowed to send its preliminary results to the second stage. To perform this function on the NIU system, an NIU programming library was interfaced to set up the transmission of messages. The pipeline was limited to two stages in the NIU because only two processing nodes were available.

3.2.1. Results of pipelining in NIU system

The NIU system saw a significant speedup due to pipelining over two stages, and the results can be seen in Figure 12. As was expected, the case with even workloads had better throughput than the case with slightly differing workloads. The speedup for the differing workloads varied between approximately 1.5 and 1.6, while the speedup for identical workloads varied between approximately 1.6 and 1.8. In both cases, the lowest speedup occurred when the smallest image was used. This is

expected because communication setup times will be larger relative to computation times for smaller images.

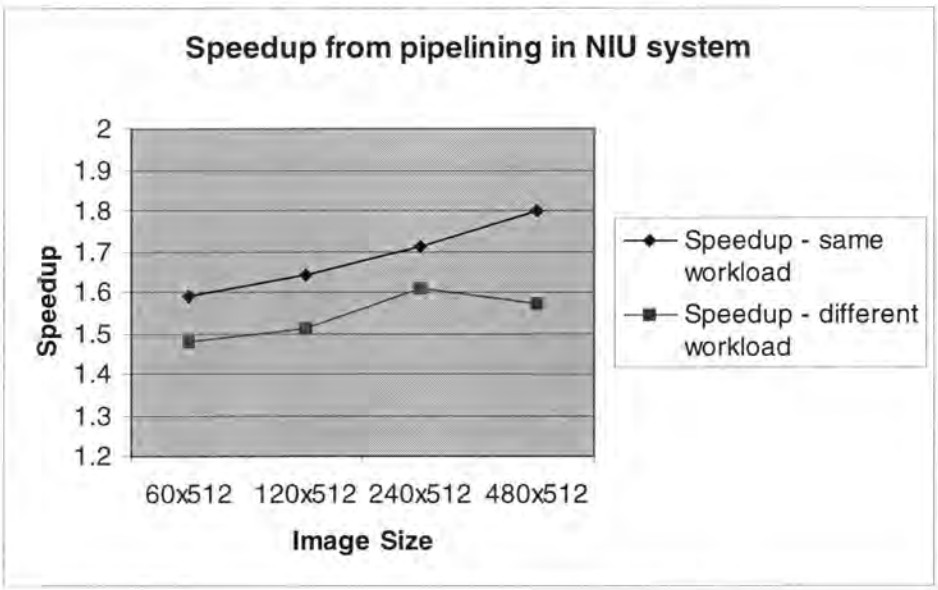


Figure 12. Speedup due to two stages of pipelining in the NIU system.

3.2.2. Results of pipelining in xCat cluster

The xCat cluster utilizes the MPI programming interface for its inter-processor communication. The large number of processing nodes in the cluster allowed us to extend the pipeline to more stages and observe whether significant speedups were realized for these longer pipelines. The pipeline was tested for two to eight stages. The workload was devised so that all stages did exactly the same amount of work in order to observe the maximum speedup available. Image sizes ranging from 60x512 pixels to 480x512 pixels were transferred between stages to observe whether larger communication requirements affected the results from the system. The speedups observed are shown in Figure 13. It shows that significant speedups are seen as the number of stages increases. Larger communication requirements do not seem to affect the speedup significantly.

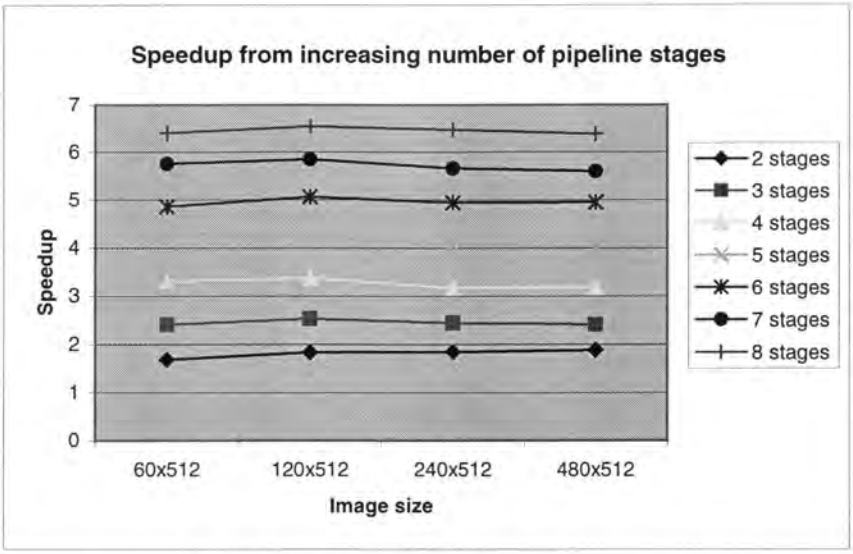


Figure 13. Speedups in xCat cluster as pipeline stages increase.

Another useful measure is the efficiency of the pipeline. It is important to know how well the processing nodes are being utilized in the pipelined system. For example, if the system has low efficiency, it may not be worthwhile to pipeline the system, because much of the processing ability is being wasted. Figure 14 shows the efficiency of the various pipeline lengths. Efficiency is defined as the actual speedup realized in the system divided by the maximum theoretical speedup.

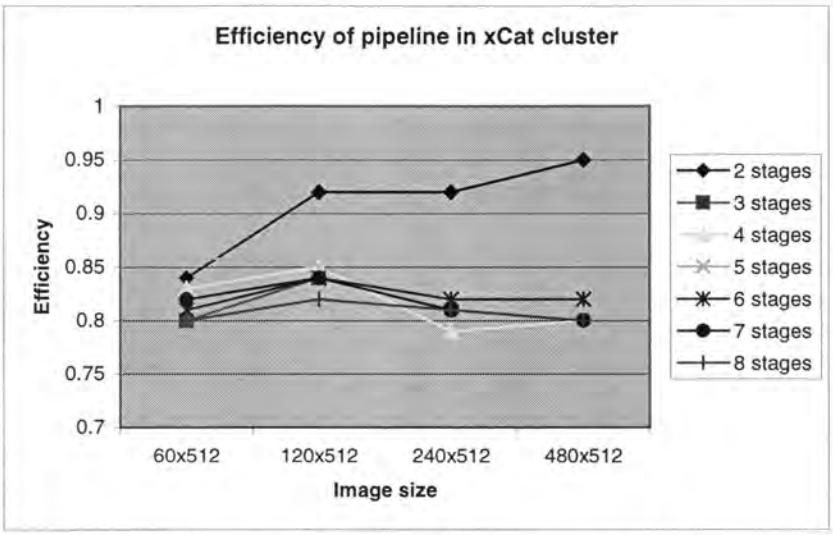


Figure 14. Efficiency of hardware usage in xCat cluster as pipeline stages increase.

Figure 14 shows that the efficiency of the pipelines generally hovers between 0.80 and 0.85 for all pipeline lengths, except for a length of two. The efficiency of the two-stage pipeline is higher, due to the simplified communication requirements in the two-stage system. In the two-stage system, the first stage only sends its finished work to the last stage, and the last stage only receives the frame from the first stage. For pipeline lengths greater than two, there are intermediate nodes that must both send and receive data at each step, which complicates communication. In reality, efficiency would probably not be as high in a real system, because it is highly unlikely that workloads would be divided perfectly equal between the processing nodes. However, the figure shows that efficiency remains relatively high as the number of pipeline stages increases, meaning the system should scale well as long as workloads are fairly well matched.

3.2.3. Conclusions

It has been shown that pipelining of algorithms over multiple processing nodes can be a very efficient way of gaining speedups in computation. The results from both the NIU system and the xCat cluster show that even a short pipeline can significantly increase the throughput of algorithms that are inherently linear. The xCat cluster also showed that this paradigm performs well, even as the number of pipeline stages increases, maintaining an efficiency above 80% even as the number of pipeline stages increased to eight.

4. Communication Hiding in ALOA

The previous section showed that large speedups may be gained by pipelining the ALOA system. However, there are situations where communication costs may negate any reduction in processing time, such as in the case when the tasks at each node take a very short amount of time relative to communication times. Ideally, the communication should be reduced as much as possible. However, the communication requirements are basically constant for a given frame size in the pipelined system. Since the communication time cannot be reduced, another method is to "hide" the communication behind the computation.

Communication hiding is a common way to optimize distributed programs. Communication can be a very expensive operation, because accessing data on a remote machine can take many times longer than accessing data in local memory.

Communication hiding requires that the network or software interface be able to perform both sends and receives in the background. This means that the interface must allow for non-blocking sends on the transmit side, and sufficient buffering on the receive side such that the receiver may wait to process incoming data until it is ready. The authors of [17] show that if the total incoming and outgoing communication requirements take less time than the computational requirements of the given task, then the communication time can be completely hidden. Sends should be initiated as early as possible, and receives as late as possible in a communication cycle to maximize the communication hiding [18].

4.1. Previous work in communication hiding

Much previous work has been done in both the hardware and the software realms of communication hiding. The authors of [19] developed a parallelizing compiler that attempted to automate communication hiding. The authors note that there is no linguistic support for automatically doing communication hiding in data parallel languages. Also, there is no formal asynchronous data

transfer capability in the compilers for these languages. Consequently, a compiler was developed to automate this process using a structure called an N-level message queue. The message queue is an intermediary between the high-level application and hardware that handles sending and receiving messages in the background. The N-level queue also implements message priority, which could possibly be utilized in real-time applications. Utilizing overlapped communication, the authors found a speedup from 0.2% to 11.9%, depending on the application and the number of processors in the distributed system.

In [17], communication hiding was implemented on a distributed system called Proteus. Proteus is implemented as a group of clusters, where each cluster is connected to another cluster through a crossbar network for inter-cluster communication, and a VMEbus for control signals. Inside each cluster are multiple processing elements that utilize shared memory for communication. Each cluster also contains a cluster controller that manages the resources in the cluster. The authors state that this system of groups of clusters allow for greater scalability than a general distributed system.

Proteus is able to overlap communication and computation because the cluster controller handles communication functions instead of individual processing elements. The processing element is allowed to post a communication request with the cluster controller and continue its work while the controller does the work of setting up the actual communication.

To test the performance gains due to background communication on Proteus, a parallelized FFT algorithm was implemented with the focus on optimizing the communication between processing elements. The authors found that the communication requirements for processing parts of the FFT algorithm approached zero, meaning the communication was almost completely hidden behind the computation.

In [20], the ability of two specific distributed systems to overlap computation with communication was analyzed. The EM-X [21] multiprocessor system and the IBM-SP2 [22]

distributed system were tested. The EMX has 80 processing nodes connected by an Omega network and is designed to lower communication costs in distributed systems with three methods: latency reduction, latency hiding, and latency minimization when accessing remote memory. First, latency is reduced by joining the communication pipeline with the execution pipeline. Second, latency is hidden by multi-threading tasks. Finally, latency is minimized during remote memory accesses by optimizing packet routing and throughput in the communication network. The SP-2 from IBM uses a distributed memory passing architecture and can have anywhere from 2 to 128 processing nodes. The design goal of the SP-2 was to be a general purpose processing cluster with the ability to scale well as processing nodes increase.

To test the capability of the EM-X and SP2 systems to overlap communication and computation, a distributed bitonic sorting algorithm was implemented. The algorithm consists of two steps. The first step is a local sort and contains no interprocessor communication. The second step merges the results from the individual processing nodes and does require communication between processors. It was found that both systems were able to reduce the communication overhead by 30% to 40% in the case that the message size was 1000 integers, resulting in significant speedups in the algorithm.

4.2. Communication hiding in ALOA and its relevance in high-speed networks

The NIU interface is very well suited for communication hiding. Both the transmit and receive queues contain circular buffers, which allow the sender and receiver to do work on one buffer of data while another buffer is being sent or received. Secondly, the NIU utilizes DMA such that no processing time is utilized when transferring the communicated data into the circular buffers. Finally, all the communications are non-blocking. Transmitting a message entails organizing a data structure to tell the hardware which data buffer to send, then initiating the send. After initiation, the host returns to the task it was working on, and the message is sent completely in the background.

The MPI programming library used in the xCat cluster is also suited for background communication. Received messages are buffered until the processing node accesses them, and the library supports both non-blocking sends and receives.

Communication hiding is a very important idea, especially when communication speeds are relatively slow compared to the computation time of a parallel task. However, communication speeds have increased dramatically recently with the advent of high-speed Ethernet, fibre-channel, and other transfer media. Therefore, an interesting question to ask is, "Given a high-speed communication link, is communication hiding even necessary?" It may be possible that communication hiding is difficult to implement on a given system, so the tradeoff between program complexity and reduced communication time may not be advantageous for high-speed networks.

To look at the problem, two types of situations were analyzed. The first situation is the data pipelining algorithm introduced in the previous section. The second situation is hypothetical; it involves a master node that distributes frames of data to be computed by other processors. In the ALOA system, this master node may be considered the image formation node, which receives radar data and transforms it into a visual image. The master node supplies the processing nodes with the next set of data when completion of the current set of data has occurred. This round-robin technique of work assignments allows for temporal parallelism [23]. A diagram of this system can be seen in Figure 15.

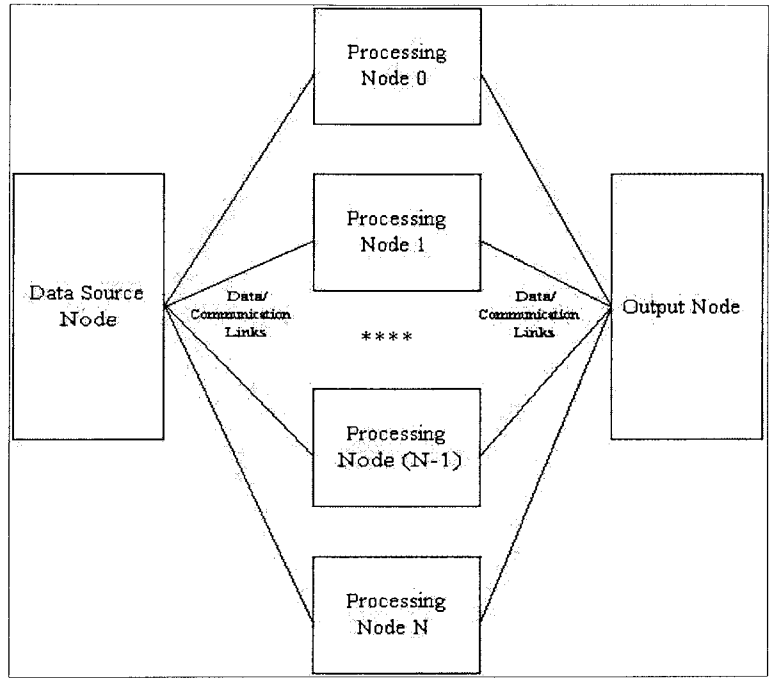


Figure 15. Theoretical distributed system with data source node providing data to all processing nodes.

These situations were analyzed on two networks. The first was the fibre-channel based NIU system that was introduced in the previous section. This network has two processing nodes connected by a 1 Gb/s fibre-channel interface. The circular buffering and the non-blocking communication features of the NIU were utilized to move the communication to the background.

The second network utilized was an xCat cluster of workstations, also introduced in the previous section. The cluster consisted of 32 processing nodes, with each node have two processors. A Myrinet network connected the processing nodes. The features of the MPI programming interface were utilized to move the communication to the background. Specifically, the ability to perform non-blocking sends and receives, and the automatic buffering of messages on the receive end allowed the attempt to hide communication behind computation.

4.2.1. Results of communication hiding in NIU system

To perform this test, the runway detection algorithms of the ALOA system were performed on each system for a variety of processor numbers and image sizes. The number of processors in the NIU system was limited to two because of limitations on buffering in the kernel space, which is necessary for the NIU interface to buffer incoming and outgoing data. Communication hiding was tested for the two-stage pipeline system and the theoretical distributed system with a node passing out data to other processing nodes. The results for the pipelined system are shown in Figure 16, while the results for the distributed system are shown in Figure 17.

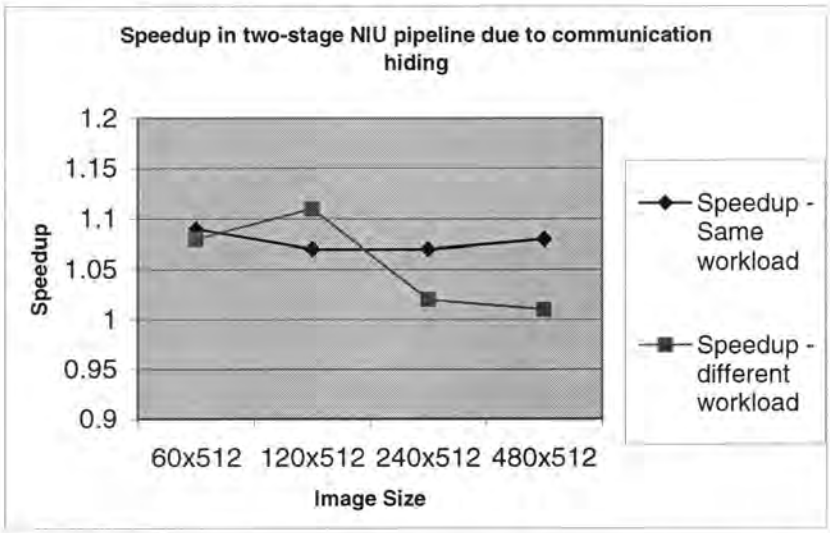


Figure 16. Speedup of two-stage NIU pipeline due to communication hiding.

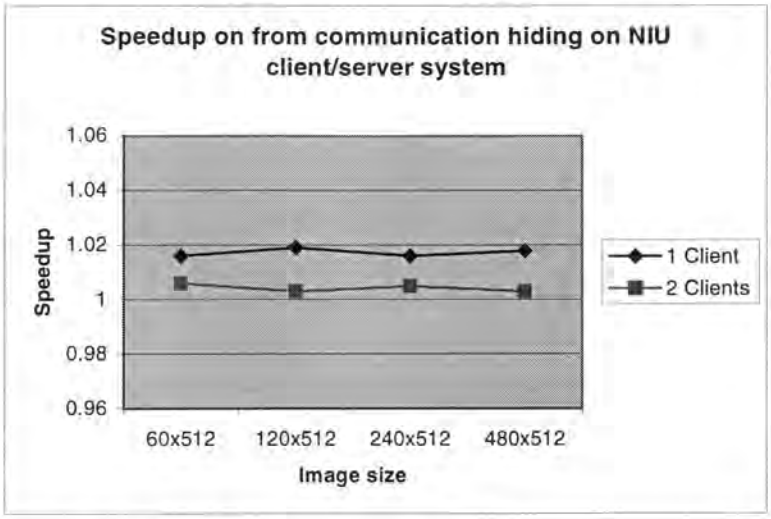


Figure 17. Speedup in NIU client/server system due to communication hiding.

The pipelined system showed speedups ranging from 1.01 to 1.11. The distributed system showed speedups around 1.02 for the single client case, and the dual processor case was limited to speedups of about 1.01. In the case of the dual processor distributed system, the source node may have trouble keeping up with requests from the rest of the consumer nodes, which limits the speedup possible as the number of consumer nodes increases. In all of these cases, speedups are seen, but the speedups are relatively small. The argument is that the communication times are so small now due to high-speed networking interfaces that the amount of speedup available is much lower than it used to be.

4.2.2. Results of communication hiding in xCat cluster

Timings were also taken in the xCat cluster for both the cases tested in the NIU system. Results from the pipelined system are shown in Figure 18. Results from the distributed system with a producer and multiple consumers are shown in Figure 19.

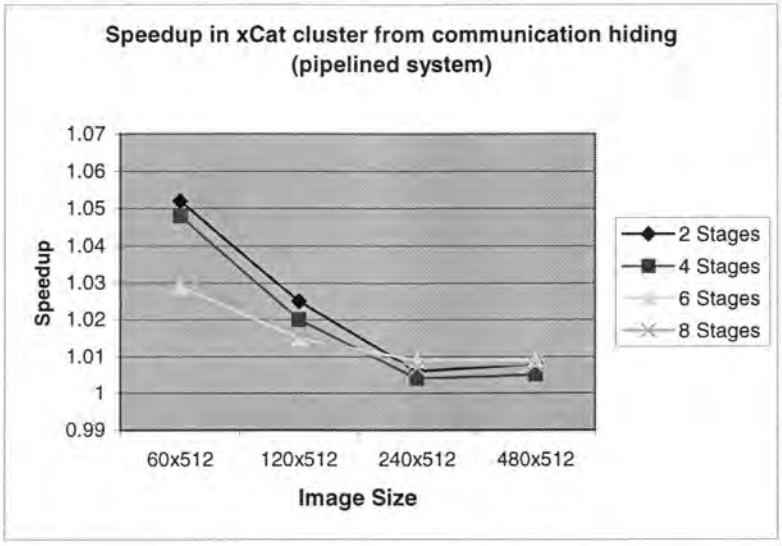


Figure 18. Speedup in pipelined system on xCat cluster due to communication hiding.

The pipelined system implemented on the xCat cluster showed very limited speedups due to communication hiding. Speedups ranged from 1.004 to under 1.052, meaning gains due to communication hiding were minimal. The smallest image benefited the most from communication hiding, although even these results showed limited gains. It can be concluded that any effort to speed up the pipelined system should be spent elsewhere.

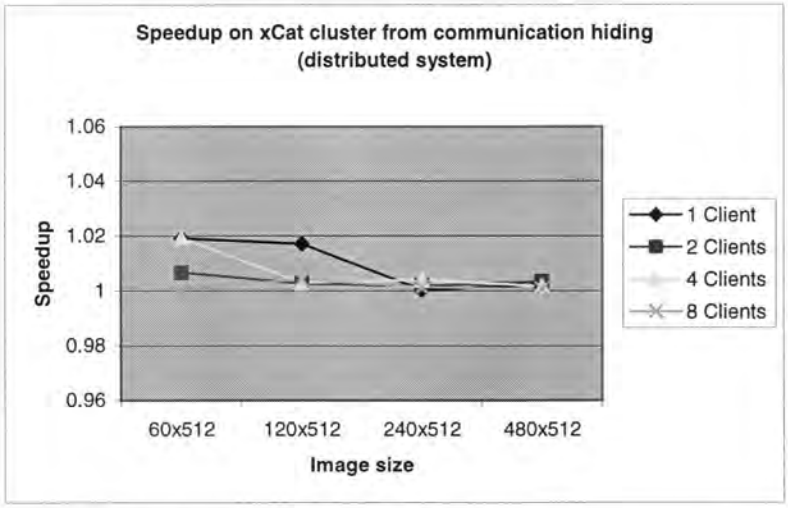


Figure 19. Speedup in distributed system on xCat cluster due to communication hiding.

Like the pipelined system, the distributed system showed limited speedups across the board for all numbers of consumer nodes and image sizes. No speedup was seen larger than 1.02, meaning the advantage gained by attempting to hide the communication in the background was minimal. Again, given the high-speed interconnection network between processing nodes, the advantages gained by communication hiding may be limited.

4.2.3. Conclusions

The time trials on the NIU system and xCat cluster showed that there are speedups to be found when implementing communication hiding. However, the speedups found in the sample systems were minimal. Except for a couple of specific cases, the NIU system was limited to at most a 1.10 speedup from communication hiding, while the xCat cluster never saw more than a 1.052 speedup due to communication hiding in either the pipelined or distributed system. It is believed that this is due to lower communication times because of high-speed networking links. The low communication times limit the effectiveness of communication hiding.

5. Summary

This thesis introduced the ALOA system, which is an image processing system that analyzes radar images and predicts where the runway is located in a radar image and if obstacles are in the runway. It is designed for aircraft attempting to land in low visibility.

Because of high computation requirements, various methods of speeding up the ALOA algorithms were analyzed. The first methods analyzed consisted of speeding up the software of the system. The Hough transform was used in the system to find the edges of the runway, and it was very inefficient in the beginning. To speed up the Hough transform, a switch was made to the Duda/Hart version of the algorithm, and speedups were found ranging from 4.51 to 4.96. Next, the algorithm was converted to all-integer arithmetic through an arithmetic code, which resulted in further speedups from 1.56 to 1.82. Using binary shifts instead of division in the calculations optimized the arithmetic code, and speedups from 1.41 to 1.48 were found. The last optimization consisted of increasing the incremental value of the angle in the Duda/Hart algorithm, which resulted in large speedups. In fact, scaling the incremental value by N resulted in a speedup larger than N due to more efficient caching.

Because the entire system would still be too slow to meet minimum quality of service requirements, the algorithms were moved to a distributed system to increase throughput. A pipelined implementation of the algorithms was introduced with each stage of the pipeline assigned a subset of the ALOA algorithms. This system was tested on the fibre-channel-based NIU system and a larger cluster of workstations. Both systems found significant speedups due to pipelining. The speedup in the NIU system ranged from 1.48 to 1.61, while the cluster maintained an efficiency between 0.80 and 0.85 even as the number of pipeline stages increased to eight. We can conclude that the pipelined system is a very good way to parallelize algorithms in a system with linear data flow.

The last part of the thesis looked to optimize communication in the distributed system via communication hiding. Two systems, the pipelined system and a theoretical producer-multiple-

consumer system, were tested for speedups gained by communication hiding. In all examples, communication hiding never resulted in large speedups for either test system. The xCat cluster had especially disappointing results, with the speedup never being larger than 1.052 for any case. It was concluded that the high-speed communication networks result in very low communication times, even without communication hiding. Because of this, any speedups from communication hiding are going to be limited.

References

[1] Kaliyaperumal, K.; Lakshmanan, S.; Kluge, K. *An Algorithm for Detecting Roads and Obstacles in Radar Images.* IEEE Transactions on Vehicular Technology. Vol. 50, Issue 1. January, 2001. Pp. 170-182.

[2] Ito, M.R.; Duong, S.; McFee, J.E.; Russell, K.L. *Towards Real-Time Detection of Landmines in FLIR Imagery.* Proceedings of the 11th IEEE Signal Workshop on Statistical Signal Processing, 2001. Pp. 154-157.

[3] Hellwich, O.; Günzl, M. *Landuse Classification by Fusion of Optical and Multitemporal SAR Imagery.* Proceedings of the IEEE 2000 International Geoscience and Remote Sensing Symposium. IGARSS 2000. Vol. 6. Pp. 2435-2437.

[4] Katartzis, A.; Sahli, H.; Pizurica, V.; Cornelis, J. *A Model-Based Approach to the Automatic Extraction of Linear Features from Airborne Images.* IEEE Transactions on Geoscience and Remote Sensing. Vol. 39, Issue 9. September, 2001. Pp. 2073-2079.

[5] Hough, P.V.C. *Method and Means for Recognizing Complex Patterns.* U.S. Patent 3,069,654. December 18, 1962.

[6] Leavers, V.F. *Shape Detection in Computer Vision Using the Hough Transform.* Springer-Verlag London Limited. 1992.

[7] Duda, R.O.; Hart, P.E. *Use of the Hough Transformation to Detect Lines and Curves in Pictures.* Communications of the ACM, Vol. 15, No. 1. January, 1972.

[8] Olmo, G.; Magli, E. *All-Integer Hough Transform: Performance Evaluation.* Proceedings of the 2001 International Conference on Image Processing. Vol. 2. Pp. 338-341.

[9] Zhang, M. *On the Discretization of Parameter Domain in Hough Transformation.* Proceedings of the 13th International Conference on Pattern Recognition, 1996. Vol. 2. Pp. 527-531.

[10] Osterhaug, A. *Guide to Parallel Programming on Sequent Computer Systems.* Sequent Computer Systems, Inc., 1987.

[11] King, C.T.; Chou, W.H.; Ni, L.M. *Pipelined Data-Parallel Algorithms: Part I – Concept and Modeling.* IEEE Transactions on Parallel and Distributed Systems. Vol. 1, Issue 4. October, 1990. Pp. 470-485.

[12] Chong, M.N.; Soraghan, J.J.; Durrani, T.S. *Pipeline Functional Algorithms, Data Partitioning for Adaptive Transform Coding Algorithms.* IEEE Colloquium on Parallel Architectures for Image Processing Applications, 1991. Pp. 8/1-8/6.

[13] Chatterjee, S.; Strosnider, J. *Distributed Pipeline Scheduling: End-to-End Analysis of Heterogeneous, Multi-Resource Real-Time Systems.* Proceedings of the 15th International Conference on Distributed Computing Systems, 1995. Pp. 204-211.

[14] The MPI Standard. URL: http://www.mcs.anl.gov/mpi. Date of access: July 9, 2002.

[15] Turner, D.; Weiyi, C.; Kendall, R. *Performance of the MP_Lite Message-passing Library on Linux Clusters.* The Second International Conference on Linux Clusters: The HPC Revolution. 2001. URL: http://cmp.ameslab.gov/cmp/Papers/ddt_uiuc_MP_Lite.pdf. Date of access: July 9, 2002.

[16] Extreme Cluster Administration Toolkit. URL: http://www.x-cat.org. Date of access: July 9, 2002.

[17] Somani, A.K.; Sansano, A.M. *Achieving Robustness and Minimizing Overhead in Parallel Algorithms Through Overlapped Communication/Computation.* The Journal of Supercomputing. Vol 16, No. 1-2. May, 2000. Pp. 27-52.

[18] Fahringer, T.; Mehofer, E. *Buffer-Safe Communication Optimization based on Data Flow Analysis and Performance Prediction.* Proceedings of the 1997 International Conference on Parallel Architectures and Compilation Techniques. Pp. 189-200.

[19] Li, X.; Harada, K. *An Efficient Asynchronous Data Transmission Mechanism for Data Parallel Languages.* Proceedings of the 1996 International Conference on Parallel and Distributed Systems. Pp. 238-245.

[20] Sohn, A.; Ku, J.; Kodama, Y.; Sato, M.; Sakane, H.; Yamana, H.; Sakai, S.; Yamaguchi, Y. *Identifying the Capability of Overlapping Computation with Communication.* Proceedings of the 1996 Conference on Parallel Architectures and Compilation Techniques. Pp. 133-138.

[21] Kodama, Y.; Sakane, H.; Sato, M.; Yamana, H.; Sakai, S.; Yamaguchi, Y. *The EM-X Parallel Computer: Architecture and Basic Performance.* Proceedings of the ACM 22nd International Symposium on Computer Architecture. Pp 14-23. 1995.

[22] Agerwala, T.; Martin, J.L.; Mirza, J.H.; Sadler, D.C.; Dias, D.M.; Snir, M. *SP-2 System Architecture.* IBM Systems Journal Vol. 34, No. 2, 1995.

[23] Krikelis, A. *High-Performance Multimedia Applications and the Internet.* IEEE Concurrency, Vol. 6, Iss. 3, 1998. pp. 17-19.

Acknowledgements

First and foremost I would like to thank my major professor, Arun Somani. His help and guidance throughout my career at Iowa State, both as an undergraduate research assistant and later as a graduate student, have allowed me to take the next step in my educational and professional career. I especially appreciated his always-open door policy and his willingness to set aside anything he's doing to help me solve a problem.

A great deal of gratitude must go to the Research and Development group at Lockheed Martin in Eagan, Minnesota for their generous loan of test equipment used to write this thesis. Specifically, the logistical and technical support of John Esch and Rick Stevens made this thesis possible.

Finally, I need to say thanks to Jenny Hively, who has put up with more than anyone should be expected to during my stint as a graduate student. Between the long drives and the large phone bills, she has always provided me with the support I needed to make it through this process.