Background Compensation and an Active-Camera Motion Tracking Algorithm

Rohit Gupta, Mitchell D. Theys, and Howard Jay Siegel

Parallel Processing Laboratory, School of Electrical and Computer Engineering
Purdue University, West Lafayette, IN 47907-1285, USA

Abstract

Motion tracking using an active camera is a very computationally complex problem. Existing serial algorithms have provided frame rates that are much lower than those desired, mainly because of the lack of computational resources. Parallel computers are well suited to image processing tasks and can provide the computational power that is required for real-time motion tracking algorithms. This paper develops a parallel implementation of a known serial motion tracking algorithm, with the goal of achieving greater than real-time frame rates, and to study the effects of data layout, choice of parallel mode of execution, and machine size on the execution time of this algorithm. A distinguishing feature of this application study is that the portion of each image frame that is relevant changes from one frame to the next based on the camera motion. This impacts the effect of the chosen data layout on the needed inter-processor data transfers and the way in which work is distributed among the processors. Experiments were performed to determine for which image sizes and number of processors which data layout would perform better. The parallel computers used in this study are the MasPar MP-1, Intel Paragon, and PASM. Different modes are examined and it is determined that mixed mode is faster than SIMD or MIMD implementations.

1. Introduction

Motion tracking refers to a method of following a moving object and determining the exact location of the object relative to the observer at any given instant. There are several methods for performing motion tracking. The simplest is to use a static camera. In this approach, the observer (camera) is held stationary and each frame is subtracted from the next frame. In this manner, information about the objects that are moving can be extracted. The simplicity of this algorithm prevents the system from tracking objects once they move outside the field of view of the camera. The advantage of such a tracking system is that high frame rates can be achieved without using special-purpose hardware.

Another method, active motion tracking, involves using a camera mounted on a moving platform. Images from the camera are processed and the camera is then repositioned to keep the target in the center of the field of view. Tracking objects using an active camera necessitates high-speed processing so that fast moving targets can be tracked.

In an attempt to study the effects of data layout, mode of parallelism, and machine size on computation and communication times, parallel implementations of an existing serial motion tracking algorithm were executed on three parallel machines: the PASM prototype [13], the MasPar MP-1 [11], and the Intel Paragon XP/S [1]. A distinguishing feature of this application study is that the portion of each image frame that is relevant changes from one frame to the next based on the camera motion. This impacts the effect of the chosen data layout on the needed inter-processor data transfers and the way in which work is distributed among the processors.

In Section 2, background information, definitions, and related work associated with motion tracking are presented. An overview of the serial algorithm appears in Section 3. Section 4 contains implementation details of the background compensation portion of the tracking algorithm. This computationally intensive section of the overall algorithm was the focus of this study. Edge detection is examined in Section 5 so that the entire algorithm can be analyzed for each machine. In Section 6, results are compared and summarized from the three machines used in this study.

2. Motion Tracking

2.1 Background Information

There are two classifications into which a number of the present methods of motion tracking fall: recognition-based and motion-based [10]. In recognition-based motion tracking, the object being tracked is first recognized and then the position of the object is determined. This allows tracking to be done in three dimensions and also allows the estimation of rotation and translation of the object. A limitation of this method is that it can only be used to track recognizable objects. Furthermore, recognition is a computationally complex task and, hence, the overall speed of the algorithm is reduced compared to motion-based tracking.

Motion-based tracking is a less computationally complex alternative. In motion-based tracking systems, the object being tracked does not need to be recognized. Instead, motion within a frame is detected using a temporal derivative of the images to find areas of motion. If the sampling rate is high, the derivative can be approximated by a simple difference operation between successive frames followed by a thresholding operation. If it is assumed that motion is uniform within discrete objects, then only the edges of objects are important. Tracking only edges, rather than entire objects, decreases the computational burden, if finding the edges of objects is com-

This research was supported in part by NRaD Naval Laboratory under contract number N66001-96-M-2277 and by Architecture Technology Corp. under contract number 6005. Some of the equipment used was supported by the National Science Foundation under grant number CDA-9015696.

putationally simple. To highlight the edges of the moving portion of the frame, the difference image is ANDed with the edge image of the frame. This yields a frame that contains the moving edges (and other noise components) only. The computational complexity required in this approach is lower than the recognition-based approach, and the algorithms contain parallelizable features, e.g., the difference and AND operations. Therefore, a motion based method presented in [10] was selected as a basis for the development of a parallel implementation for this application study.

2.2 Related Work

Parallel implementations of active-camera motion tracking algorithms have been studied. In [17], a system was introduced that incorporated a sensing element (SE) into each PE. The PEs were arranged as a 16-by-16 square mesh SIMD machine with four nearest-neighbor interconnections. The 256-PE mesh was then divided into four quadrants. Each quadrant processes one quarter of the image and determines the location of the object, which was conveyed to the control computer. The camera was then repositioned and the cycle repeats.

The approach is novel in the fact that the PEs receive the image directly from the SEs. In [17] background compensation was not performed, so the algorithm is susceptible to the effects of apparent motion.

Because of the simplicity of the algorithm, very high frame rates (1492 frames per second) were attained. However, the authors acknowledge that the actuators used to move the camera are not capable of such high speeds. It should be noted that the focus of that work was not to study issues of parallel processing.

Another parallel implementation of a motion tracking algorithm was introduced in [9]. There a tracking system is implemented on transputers in a MIMD organization. Three transputer sets were used in parallel. The Host Interface (HI) consists of two transputers that are used to communicate the results and progress of the system. The HI was connected to the two other transputer sets, the Edge Extraction Engine (EEE) and the Tracking Engine (TE). The EEE consisted of up to 16 transputers interconnected in a mesh configuration. The TE was arranged in a tree configuration of up to ten transputers. The master controllers of the three sets of transputers are interconnected using a system bus. All communication between these sets of transputers was performed using this bus.

The important characteristics of the parallel computational model considered there were that the system was a transputer-based MIMD machine utilizing functional parallelism. The issues that were considered were the effect of interconnection topologies, quality of edge detection using Canny and Sobel edge detectors, and load balancing.

The frame rates attained in that research were less than one frame per second when real images were used and about 3.7 frames per second for synthetic images. These rates were much lower than real-time, and also much lower than the desired rates for the work presented here.

In [12], another transputer-based implementation of a tracking algorithm was described. The system consisted of 30 transputers, 24 of which were in the tracking engine. The remaining transputers constitute the image store, the corner store, and the controller. The image store consisted of four transputers, each containing one quarter of the image. The controller was a single transputer that coordinated the operation of the tracking engine. The tracking engine transputers were configured to work independently (e.g., MIMD mode) and it was possible that a single corner of an object may be tracked by multiple transputers. To avoid this situation, the corner store maintains a record of the image plane of each corner and only one transputer is allowed to track a corner within one image plane.

One of the important features of the work was that the search space was reduced so as to increase the speed of processing. The search space was reduced by selecting "regions of interest" that were calculated based on the movement of the object, e.g., the velocity and the direction of the object. The regions of interest were calculated for each corner being tracked and for each frame.

The issues of parallel processing that were relevant in the approach are that the tracking engine transputers are arranged in a MIMD configuration with a controller processor. Load balancing among the transputers was accomplished with the help of the corner store, such that all corners were distributed among the transputers equally. An "optimized" method of distributing the image based on the location of the corners in the image was developed and the execution time of the algorithm was compared to that in the non-optimized method.

The results of the above research have indicated that near-real-time tracking is achievable using this method. A frame rate of 11 frames per second was achieved using 24 transputers in the tracking engine when 64 corners were tracked.

It can be seen from the three approaches described above that none used a standard parallel computer and none performed background compensation. The work presented here includes implementations of the [10] algorithm on commercial parallel machines. This work explores the impact of data allocation strategies, solving complex communication needs, and performing a case study that focuses on the background compensation component. It also examines the entire algorithm while varying the number of PEs, mode choice, and data layout.

3. Overview of the Study

3.1. Serial Algorithm for Motion Tracking

In active vision systems, movement of the camera may cause apparent motion of objects in the image. Apparent motion is motion that is perceived by the camera due to its own movement, i.e., if the camera moves past a stationary object, the object's position changes between frames. This will result in the stationary object

being present in the difference image because it will not be canceled out in the subtraction of the two consecutive frames. For the difference operation to be applicable in active vision systems, a correspondence between successive frames must be established. Because the background is assumed to be stationary, this process is referred to as background compensation in [10]. In [6], a formula for calculating pixel correspondence based on camera rotations was developed. Using this formula, background compensation is performed. After background compensation, the two images can be compared directly using the simple difference method because the apparent motion of the stationary parts of the image is corrected. An absolute value of the difference between the pixel values is then taken. A block diagram of the complete serial algorithm is presented in Figure 1, where $I(t)$ represents the most recent image frame and $I(t-1)$ represents the previous image frame. In the figure, α, γ, and θ represent the pan angle, tilt angle, and inclination of the camera, respectively.

Because the background compensation algorithm is based on a model that assumes that the rotation of the camera occurs around the center of the lens, and all rotations are not performed this way, there is a small amount of error in the compensated image. Morphological opening is performed to filter the difference image to reduce the effect of these errors.

The next step of the algorithm, edge detection, is performed on the current frame by applying a Sobel 3-by-3 operator. The difference image and the edge image are then ANDed together resulting in an image that contains the edges of the moving object(s) in the current frame only. The relative displacement of the object is then calculated and the angles that the camera must rotate are determined. The camera is then rotated to keep the moving object in the center of the field of view.

The algorithm described in [10] was implemented using a camera mounted on a pan and tilt device controlled by a workstation. A CCD camera provided the frames to a real-time digitizer. The amount of pan/tilt was determined by a potentiometer connected to the platform supporting the camera. The system was shown to be capable of real-time motion detection and could extract moving edges from images when the pan and tilt angles between successive frames were as high as 3^o.

Although the serial algorithm is capable of real-time motion tracking, much higher frame rates (100 frames per second or more) and processing of larger images are often required, such as in the case of tracking missiles. Therefore, a parallel implementation of this algorithm was developed with the expectation of obtaining higher frame rates while processing larger images.

The above description of the serial algorithm is summarized from [10]. Several steps of the serial algorithm are not discussed in this paper. The subtraction of the compensated image from the current image, the AND operation, and the thresholding operation are excluded from this paper because they are simple. Parallelizing the morphological operators is discussed in [16] and is therefore not re-examined here. Thus, this paper focuses on the computationally intensive step of background compensation. Furthermore, it is the background compensation step that is directly affected by the camera motion, which moves some of the previous frame's pixels outside the field of interest, and brings some new pixels (not in the previous frame) into the field of interest. Edge detection is not examined in as much detail here, as many people have already parallelized edge detection algorithms. Edge detection is briefly discussed because it is included in the execution times of the entire algorithm.

3.2. Parallel Algorithm and Computers Used

In the development of the parallel algorithm, several assumptions are made. It is assumed here that the layout of the images in memory is determined before execution and that it remains fixed during the entire process. The inputs to the system are two successive frames, which will be divided equally among the processing elements (PEs), each of which is a processor memory pair, and the camera rotation information, which is known by all the PEs. The output of the system is an image containing the moving edges in the latter frame. Determination of the center of the moving object is not done at this time because different schemes may be used to determine the center, particularly if more than one object is being tracked. Because external I/O varies from one machine installation to another, it is not considered here. This work concentrates on the impact of camera movement on the execution time of parallel implementations of the background compensation step on a commercial SIMD, a commercial MIMD, and an experimental mixed-mode machine.

The Intel Paragon XP/S is a distributed memory commercial MIMD system [1]. The system used for this study has 140 compute nodes, each of which includes a 50 MHz Intel i860 XP microprocessor. The compute nodes have 32 MB of DRAM each, and are arranged in a two-dimensional rectangular mesh with each node connected to its four nearest neighbors. Each node contains another i860 XP processor, called the message co-processor, that is dedicated to handling message-passing operations. The Paragon was programmed using C extended with parallel constructs for communication and synchronization.

The MasPar MP-1 is a distributed memory commercial SIMD system [11]. The system used for this study has 16,384 custom PEs, each of which has a four-bit arithmetic and logic unit (ALU) and 16 KB of memory. The PEs are arranged in a 128-by-128 mesh, with each PE connected to its eight nearest neighbors. Communications are facilitated by a global router multistage network that allows a given PE to establish a path to any other PE in the system. The PEs receive instructions from and are controlled by the Array Control Unit (ACU). MPL was the language used for the programs executing on the MasPar MP-1.

The PASM (partitionable SIMD/MIMD) system is a parallel computer system designed at Purdue University [13]. It is a distributed memory mixed-mode machine that can dynamically switch between SIMD and MIMD modes of parallelism at instruction-level granularity and with generally negligible overhead. PASM can be dynamically reconfigured to form independent or communicating submachines of various sizes. A flexible multistage cube interconnection network allows the connection scheme between the processors to be varied. Therefore, PASM is reconfigurable along three dimensions: partitionability, mode of parallelism, and interprocessor communication. A small-scale proof-of-concept prototype (with 16 PEs in the computational engine) has been built and is being used to study various aspects of parallel processing. Each PE consists of an MC68010 processor and 2 MB of dual-ported DRAM. An extra-stage cube network connects the PEs in the prototype. It can be seen that the hardware used is not competitive in terms of performance with current high-end workstations, however, the studies conducted on PASM can be used to compare the relative performance of an algorithm under different modes of parallelism. PASM is programmed using an explicit language for parallelism called ELP [13].

3.3. Parameters of the Study

The focus of this research is to study the effect of parallel machine characteristics and algorithm mapping on the computation and communication times for a practical application. The results of this study should be applicable to other programs that exhibit similar computation and communication characteristics. Furthermore, insights into the architecture and design of parallel machines can be obtained.

Many studies on the effects of data layout for different application programs have been performed and it has been shown that layout schemes have a significant impact on the computation and communication times of parallel programs. No one data layout is best for all applications.

A goal of this research is to study the effect of two different data layout schemes on the execution time of the motion tracking algorithm. The two data layout schemes considered here are row-striping and rectangular distributions. In row–striping, each PE memory contains a section of an image such that all PEs have the same number of contiguous rows of the image and all the columns. Consider an image of size M-by-M pixels and a parallel computer system having N processors. Each PE then has M/N rows and M columns of each image.

In the rectangular distribution scheme, each PE memory contains a rectangular portion of an image, i.e., for an image with M-by-M pixels in a system of N PEs arranged in an X-by-Y logical mesh, each PE contains M/Y rows and M/X columns of the image. Although these data distribution schemes are not new, what is new is examining how they influence the execution time of the motion tracking algorithm.

An important area of research in parallel computers is to analyze program behavior under different modes of parallelism (i.e., SIMD vs. MIMD). This is a complicated research area because there are very few machines that are capable of executing programs in different modes of parallelism. Examples of such machines are PASM, TRAC [8], OPSILA [2], Triton [4], MeshSP [5], and EXECUBE [7]. Most programs contain segments that are better suited to a particular parallel mode of operation and machines that allow mixed-mode computation can provide the better mode for each segment. Analysis of the execution times can provide insights into how the choice of the mode of computation affects the execution times of parallel programs.

For all test images, random pixel values were generated. This is because the number and type of operations performed in this application, and hence the execution time, are independent of the values of the pixels. Each entry in Tables 1 through 5 is the average of ten trials. Each machine was in single user mode when conducting the experiments. The communication times for the three machines are somewhat unpredictable (even when using the machines exclusively) due to subtleties in the network operating routines (which are not readily accessible to normal users). Images ranged from 128-by-128 pixels to 1024-by-1024 pixels. A selective set of results is presented in the tables.

4. Background Compensation

4.1. Overview

The purpose of background compensation is to remove the apparent motion from the images being analyzed. Because of the assumption that the camera rotates about two axes (horizontal and vertical), Kanatani's formula [6] can be used. In the equations below, α corresponds to the pan angle, γ, to the tilt angle, and θ, to the initial inclination of the camera. The focal length of the camera is denoted by f in the equations. For each pixel location (x_t, y_t) in the latest image, the corresponding pixel (x_{t-1}, y_{t-1}) in the previous image is located. The (x_{t-1}, y_{t-1}) must be calculated for each pixel (x_t, y_t) because not every pixel is displaced by the same distance, in Cartesian coordinates.

$$x_{t-1} = f\frac{x_t + \alpha \sin\theta\, y_t + f\alpha\cos\theta}{-\alpha\cos\theta\, x_t + \gamma y_t + f} \quad (1)$$

$$y_{t-1} = f\frac{-\alpha\sin\theta\, x_t + y_t - f\gamma}{-\alpha\cos\theta\, x_t + \gamma y_t + f} \quad (2)$$

The main focus of this research is to parallelize the background compensation algorithm so that high frame rates are achieved. The effects of data layout choice, machine size, and mode choice were studied so as to achieve the aforementioned goal. The serial algorithm from [10] was parallelized for the three machines used. It should be noted that equations (1) and (2) were derived using the "small angle" assumption, i.e., $\sin\beta = \beta$ for

small β (in radians). In the implementation of the parallel algorithm, α and γ were assumed to be 3^o each, θ was assumed to be 0^o, the focal length was assumed to be 890 and gray scale images were used as inputs (all as was done in [10]).

4.2. Parallel Implementation

Because each pixel has to be mapped from the previous frame to the current frame based on equations (1) and (2), the process of background compensation is computationally complex. For an image of size M-by-M distributed among N PEs, both equations and their inverses need to be evaluated M^2/N times per PE. Using equations (1) and (2), all pixels in the current image can be mapped to those in the previous image, except those pixels in the current image that correspond to sections in the image that have just come into the field of view.

If (x_{t-1},y_{t-1}) and (x_t,y_t) are in different PEs, (x_{t-1},y_{t-1}) is sent to the PE containing (x_t,y_t) to perform the difference operation. For a given PE i, the number of pixels transferred and the number of PEs that receive data from PE i is a function of the camera rotation, the focal length of the camera, and the data layout. It is assumed that the focal length of the camera remains constant but the pan and tilt angles can change between frames (within the range of the formula governing the movement of the pixels as discussed above). Therefore, the number of pixels transferred, the number of PEs that will receive data from a given PE, and the identities of the receiving PEs cannot be known *a priori*. To reduce the cost of network operations, pixels are transferred in blocks. This requires only one network setting per block. For MIMD implementations on the Paragon and PASM, the source PE needs to initiate a "send" and the receiving PE (and only receiving PEs) must initiate a "receive" for each block.

Given that the data layout is assumed to be fixed, the number of the PE that contains each $(x_{t-1};y_{t-1})$ pixel location can be calculated. Using such a calculation, each PE can determine the destination PE numbers that correspond to all pixels in its own memory. Also, each PE can determine which PEs contain the pixel values it needs (based on the inverses of Equations (1) and (2)). Thus, a list of PEs that will send data to a particular PE i can be constructed. PE i then needs to wait for data only from PEs in this list. Using this method reduces the number of transfers to the minimum possible for the given data layout.

After the pixel mapping calculations (Equations (1) and (2)) and source/destination PE calculations are complete, each PE sends blocks of pixels to the appropriate PEs. The pixel value for a given (x_{t-1},y_{t-1}) and its location in the destination PE (i.e., corresponding to (x_{t-1},y_{t-1})) are sent for each pixel in the block. After a PE receives all the blocks it is expecting, it moves the received pixels and any local pixels, (x_{t-1},y_{t-1}), into their correct position in an intermediate array so a simple subtraction can be performed between (x_t,y_t) and the shifted (x_{t-1},y_{t-1}).

4.3. Communication Time Analysis

One means to reduce the overall communication time is to analyze the effects of different data layout schemes on the communication time of the algorithm. For a given image size, the number of pixels stored in each PE decreases with increasing numbers of PEs in the system. This means that, in general, the total number (i.e., summed over all PEs) of pixel transfers required will increase as well. It should be noted that the maximum total number of pixels that need to be transferred is the number of pixels in the previous image minus the number of pixels that travel outside the scope of the current image. Therefore, it is expected that the total number of pixels that need to be transferred among the PEs will first increase with the number of PEs in a machine and then level out at a maximum. To confirm this hypothesis, a simulation study was conducted. The total number of pixels that need to be transferred in the rectangular distribution scheme is always less than or equal to the number of pixels transferred in the row-striping method. Another interesting result from the simulation is that in the row-striping method, the total number of pixels transferred is the same for systems containing 32 or more PEs, while in the rectangular distribution scheme it is the same for systems with 512 or more PEs.

The difference between the number of pixels transferred in the the two methods decreases as the number of PEs increases in the system. However, as the number of PEs increases, the number of inter-PE transfers increases. This means that there might be more conflicts in the network and the overall communication time could increase. In addition to this, each transfer incurs network setup costs that add to the communication cost. In Figure 2, a sample communication pattern is presented where M and N, α and β are such that just nearest neighbor communications are needed. Pixels in the image are moved by approximately the same amount under both schemes. In the row-striping case, only one network setting is needed and there are no conflicts. In the rectangular distribution scheme, three network settings are needed per PE and multiple PEs send pixels to one PE, thereby creating conflicts. These extra settings will cause the rectangular distribution to be slower than the row-striping scheme on machines where the overhead to set up the network is large, as in asynchronous communication on the Paragon. Also, more overhead is required to compose the blocks that will be transferred. Thus, there are trade-offs that must be considered for a given M, N and architecture. On the average, row-striping requires more pixels to be transferred per PE until the maximum value is reached for both data layouts. However, the data block construction time, network settings per PE, and number of PEs that send to a given PE (creating conflicts) is worse for rectangular subimages.

4.4. Computation Time Analysis

In both data layout schemes, the number of pixels each PE holds is the same. It is expected that the compu-

tation time of the background compensation algorithm will be similar under both layout schemes. To support transferring pixels in blocks, a certain amount of computation to determine source and destination PE numbers, array indexing, etc., is performed by each PE. If a PE maps a previous frame pixel to a point outside the scope of the current frame, the pixel value does not need to be transferred and the computation of the destination PE number and the pixel location in the destination PE are skipped. Such pixels are denoted as rejected. Current frame pixel locations which correspond to points outside the scope of the previous frame are set to zero (i.e., pixels that enter the current frame due to camera movement that are not in the previous frame). Such pixels are denoted as initialized. An illustration shows rejected and initialized pixels in Figure 3.

The distribution of rejected pixels can have a significant impact on the execution time of the background compensation algorithm. This is because if a certain PE has more pixels that will be rejected, it has to perform less computations than other PEs that have fewer rejected pixels. This means that PEs with more rejected pixels will finish computations sooner than PEs with fewer rejected pixels.

Consider the following situation. Let the camera pan by an angle α and tilt by an angle γ, and the corresponding pixel movement in the image be at least r rows and c columns. If the image size has M-by-M pixels and there are N PEs, the maximum number of pixels that any PE will have after determining the rejected pixels is M/N-by-$(M-c)$ in the row-striping scheme. In the rectangular distribution scheme, the maximum number of pixels in any PE is $M/\sqrt{N}$-by-$M/\sqrt{N}$ (because there are PEs with no rejected pixels). Therefore, in the row-striping scheme, the maximum number of pixels that any of the PEs have is cM/N pixels less than the maximum number of pixels in any PE in the rectangular distribution scheme. This is because the rejected pixels are shared among more PEs in the row-striping scheme than in the rectangular distribution scheme, and therefore, the remaining pixels are more uniformly distributed in row-striping. Also, in the rectangular distribution scheme, the maximum number of pixels that any PE processes is more than the maximum in the row-striping case. Therefore, the maximum computation times of the background compensation algorithm under row-striping are expected to be lower. In addition, the more uniform distribution of pixels in row-striping, causes the load to be better balanced across all PEs.

4.5. Parallel Mode Analysis

Certain characteristics of parallel programs make a particular mode (i.e., SIMD, MIMD, or mixed) more suitable than the others. The advantages and disadvantages of executing different program constructs in SIMD and MIMD modes can be found in [14]. For example, interprocessor communication is more efficient in SIMD mode, because the single program synchronization simplifies the needed transfer protocol overhead. As another example, the execution of conditional statements is more efficient in MIMD mode because in SIMD mode the control unit must broadcast all "then" and "else" instructions to all PEs, and then some PEs are disabled for the "then" or "else" or both. Mixed mode allows each segment of an algorithm to be executed in the most appropriate mode.

The kernel of the background compensation phase is a doubly-nested loop that goes through all rows and columns of the subimage contained by a PE. The best mode for each part of this kernel is now considered.

The loop control is more efficient in SIMD mode and is expected to decrease the execution time (because the control unit can do it concurrently with PE operations). Due to the multitude of conditional statements in the body of the loop (including deciding if a pixel is rejected and therefore needs no further processing), it is expected to perform better in MIMD mode. The computation time of this kernel is expected to be better under the row-striping data layout scheme, as was discussed in Subsection 4.4. The communication times are expected to be higher in MIMD mode than in SIMD mode, however, for the reasons mentioned above.

Using PASM's mixed-mode computation capability, the parts of the program that are expected to perform better in SIMD mode can be executed in SIMD mode, and those that are expected to perform better in MIMD mode can be executed in MIMD mode. From the above analysis of the algorithm in SIMD and MIMD modes, it follows that the loop control and inter-PE communications of the algorithm should be performed in SIMD mode and the body of the algorithm should be executed in MIMD mode. These mode choices are based on typical parallel programs and are applicable when considering mode choices in general. In this case, these choices are not necessarily optimal. This is because of the nature of the background compensation algorithm.

If this kernel were executed completely in MIMD mode, then if a PE rejects a pixel because it moves outside the image, it can go on to the next pixel. This cannot happen if the loop control is in SIMD mode. By performing the loop control in SIMD mode, the PEs are forced to synchronize at the end of each loop iteration. Thus, no PE can begin to execute the next iteration of the loop (i.e., the next pixel) until all PEs have completed the current iteration. The temporal juxtaposition of loop iterations that occurs if the loop control is in MIMD mode is not possible if the loop control is in SIMD mode. Further details of this phenomenon can be found in [3]. Forcing PEs to synchronize to switch to SIMD mode for inter-PE communications within the body of the loop leads to the same problem. Therefore, mixed mode is not the ideal choice for this phase of the algorithm.

4.6. Results

The algorithm was first implemented on PASM. The algorithm was implemented on four to 16 PEs in SIMD, MIMD, and mixed mode with image sizes of 64-by-64, 128-by-128, and 256-by-256 pixels each. The image

sizes used on PASM were smaller than on the other machines because of memory limitations. Potential mixed-mode implementations are to use SIMD for loop control and/or for inter-processor communication. Executing the loop control in SIMD would yield poor performance as discussed in Section 4.5. Executing the inter-processor communications in SIMD has the potential for little improvement relative to computation time, and forcing the synchronizations would yield the same problems as discussed for loop overhead in Section 4.5. Therefore, for the mixed-mode version, the background compensation phase is performed entirely in MIMD mode. (Because other portions of the program are performed in SIMD, the entire program can be thought of as mixed mode.)

To analyze the behavior of the algorithm in an SIMD machine, the algorithm was implemented on the MasPar MP-1. The image sizes used were 256-by-256, 512-by-512, 768-by-768, and 1024-by-1024. The number of PEs used in the study was varied from 16 to 16,384. All of these image sizes could not be used with the chosen numbers of PEs considered because of limiting factors. The first is that in row-striping, the maximum number of PEs that can be used is limited by the dimensions of the image, e.g., a maximum of 256 PEs can be used to process a 256-by-256 pixel image, because in row-striping the minimum number of rows that can be contained by a single PE is one. The other constraint is that each PE has 16 KB of memory, which is not enough to hold the larger subimages when dealing with large images and a small number of PEs. Block transfers were implemented using the global router. Using the global router simplified the calculations needed because determining the number of steps in the relevant directions is not necessary, as would have been required had the X-Net been used.

The algorithm was then implemented on the MIMD Intel Paragon. The algorithm was implemented on four to 128 PEs with image sizes of 256-by-256, 512-by-512, 768-by-768, and 1024-by-1024 pixels. Communication on the Paragon can be overlapped with computation and may result in a significant reduction in the total execution time of the program. However, because the communication is overlapped with computation, it is not easy to calculate the time each PE spends communicating. Therefore, in this study the communication times were not calculated separately.

The communication time of the program is a small portion of the total execution time, so the execution times presented are not decomposed into communication and computation time. The first two columns of Tables 1 through 5 contain a representative selection of the data collected for the background compensation experiments. The difference between the communication times for the two data schemes was low (less than five percent for PASM implementations and less than ten percent for MasPar implementations). This difference will have little influence on the differences between the two schemes' total execution time, so the communication time alone is not presented. The execution times for all implementations show that the row-striping scheme is the better data layout scheme on all three machines, for the reasons outlined in Sections 4.3 through 4.5.

5. Edge Detection

5.1. Overview

Edges in images are characterized by sharp changes in intensity. Taking a two-dimensional spatial derivative of the image results in high values at points in the image that have large and abrupt changes in intensity. An edge detector that is based on this derivative principle is the Sobel edge detector and is used in this study.

In a parallel implementation of edge detection using a Sobel operator, the image is divided among the PEs. Therefore, to process some of the pixels in a PE, pixels located in other PEs are required. The number of pixels that need to be transferred to each PE is a function of the data layout scheme used, as is explained in the next subsection. The pixels around the edge of the image are initialized to zero, because they do not contain a full 3-by-3 region of pixels. This initialization ensures that the edge detection algorithm is not influenced by pixels that do not get operated on by the full Sobel operator.

Once edge detection is completed, the resulting image is ANDed with the image from background compensation (which was performed prior to edge detection). If a different data layout was used in each portion of the tracking algorithm, the data would have to be redistributed so that the two images could be ANDed (i.e., so each PE would contain corresponding subimages to AND). Therefore, edge detection was investigated to see if row-striping would perform better, and no redistribution would be required, or if the rectangular distribution performed better and the data would have to be redistributed.

5.2. Effect of Data Layout

Consider an image of size M-by-M in a system of N PEs. Assume that $X = Y = \sqrt{N}$, which means that square subimages are used. The following analysis is thoroughly examined in [15] and its application here is summarized for completeness. In the row-striping data layout scheme, each PE contains M/N rows and M columns. Because the Sobel operation is based on a 3-by-3 window, with the center defined as the center pixel, PE i $(0 < i < N-1)$ requires the bottom row of pixels from PE $i-1$ and the top row of pixels from PE $i+1$. Therefore, a total of $2M$ pixels need to be transferred. It should be noted that PE 0 only requires a row of pixels from PE 1 and PE $N-1$ only requires a row of pixels from PE $N-2$.

In the square distribution scheme, with the same image and machine sizes as in the row-striping example, each PE contains $M/\sqrt{N}$ rows and columns. In this case, a PE requires pixels from eight neighboring PEs, $M/\sqrt{N}$ pixels are required from each of the four adjacent PEs, and one pixel is required from each of the PEs diagonally adjacent. Therefore, a total of $4M/\sqrt{N} + 4$ pixels need to

be transferred. When processing pixels located in rows 0 and $M-1$, and columns 0 and $M-1$, transfers that would correspond to pixels outside the image do not need to be performed. For example, for an image of size 256-by-256, in a 16-PE system, row-striping would require 512 pixels to be transferred per PE, while in the square distribution scheme, 260 pixels need to be transferred. Therefore, in the square distribution scheme, less pixels need to be transferred.

A disadvantage of the square distribution scheme is the processing of pixels located on the edge of the image. In edge detection, pixels located in the first and last rows and columns of the image do not need to be processed. This is because a complete 3-by-3 window does not exist around these pixels. In row-striping, PEs 0 and $N-1$ contain rows 0 and $M-1$. Pixels in these rows do not need processing. Pixels located in the first and last columns are distributed among all the PEs equally. Therefore, all PEs save on the processing of $2M/N$ pixels. Additionally, PEs 0 and $N-1$ save on processing $M-2$ pixels each. The maximum number of pixels that any PE needs to process is $M^2/N - 2M/N$.

In the square distribution scheme, row 0, row $M-1$, column 0, and column $M-1$ are not distributed evenly among the PEs. Therefore, PEs that do not possess pixels in any of these locations do not save on any processing of pixels. The maximum number of pixels that are processed by any PE in this scheme is M^2/N.

It is shown that the number of pixels transferred in the square distribution scheme is less than in the row-striping scheme ($4M/\sqrt{N}+4$ versus $2M$). However, the maximum number of pixels that need to be processed in the row-striping scheme is less ($M^2/N - 2M/N$ versus M^2/N). Let the time to transfer one pixel be τ times the time to process one pixel. Therefore, if $2M/N > \tau\times[2M - (4M/\sqrt{N}+4)]$, the row-striping method is better, and if $\tau < 2M/N\,/\,(2M - (4M/\sqrt{N}+4))$, then row-striping is better, otherwise, square distribution is better.

The above analysis is based on the assumption that the communication time is directly proportional to the number of pixels transferred. Factors such as transferring all pixels to a particular PE in a block and the number of network operations required will affect the value of τ in the expression derived above. Also, poor synchronization among the PEs can result in higher than expected communication times in MIMD mode. The presence of these variables, can therefore impact the relative performance of using either data layout scheme.

5.3. Results

To accurately study the entire motion-tracking algorithm, image and machine sizes were the same as discussed in the background compensation portion, Section 4.6. The third and fourth column of Tables 1 through 5 are a selection of the results from the experiments.

The communication times for the following implementations were orders of magnitude less than the computation time and are therefore not shown in a graph: SIMD implementation on PASM, mixed-mode implementation on PASM, MasPar MP-1 implementation, and the Paragon implementation. For the MIMD implementation on PASM the communication time is a significant portion of the total execution time, and as the number of PEs increases from four to 16 the row-striping scheme achieves the lower communication times and the lower computation times. With four PEs, the row-striping scheme and the rectangular distribution require similar numbers of network settings per PE, two and three respectively. With eight or 16 PEs, the row-striping scheme still only needs two settings, where the rectangular distribution now requires five or eight, respectively. This increase in the number of network settings required is why the row-striping scheme is better for more than four PEs on PASM.

A mixed-mode implementation of the edge detection portion of PASM does not suffer the same limitations that the background compensation portion faced (Sections 4.4 and 4.6). Therefore, the mixed-mode implementation performed the portions of the loop body that required multiple conditionals in MIMD, and the rest of the algorithm in SIMD. All PEs send to their nearest neighbors before any communications begin. The difference between any two PEs completing the communication is not large, and so the synchronization of SIMD communications are not detrimental. Also, each PE performs similar numbers of computations with the simpler edge detection process, and so the synchronizations after each loop iteration are not increasing the execution time.

6. Conclusions

The execution times of the entire motion tracking algorithm, minus the time for the morphological opening which was not studied here, are shown in the fifth and sixth column of Tables 1 through 5. These results show that the row-striping data scheme is the better data scheme for this task. (In [16], it was shown that the row-striping layout scheme was better for the morphology tasks on the three machines studied in this paper.) The mixed-mode implementation on PASM showed a small improvement over the MIMD version, which performed much better than the SIMD version, for the reasons discussed earlier. In the mixed-mode version, the edge detection portion of the algorithm used both SIMD and MIMD operations, while the other portions used only MIMD operations. The data show that for the larger machine sizes, frame rates which are greater than real-time, as defined in [10], can be achieved by implementing the algorithm on a commercial parallel machine.

In summary, active camera motion tracking algorithms are computationally complex and achieving high frame rates is not possible with conventional serial computer systems. Parallel computers can provide the computational resources that are necessary to allow high-speed implementations of these algorithms, allow an easy means to revise the algorithm, and support the execution of other useful computations. When serial algorithms are ported to parallel computers, some of the issues that need

to be addressed are the data layout schemes available and the parallel modes of operation. In this study, parallel implementations of an existing serial algorithm for performing motion tracking with an active camera were developed for different parallel computers. A detailed analysis of the different components of the algorithm was conducted and execution times were obtained for each component on each of the three parallel machines. A distinguishing feature of this application study is that the portion of each image frame that is relevant changes from one frame to the next based on the camera motion. This impacts the effect of the chosen data layout on the needed inter-PE data transfers and the way in which work is distributed among the PEs.

The implementations of the algorithm proved that high-speed active motion tracking is possible using commercially available parallel computers. The performance of the algorithm is dependent on the data layout scheme used and the image and machine sizes. The degree of effect of different data layout schemes on the execution time of the algorithm is governed by the parallel mode of operation as well. By carefully tuning the application implementation to a particular machine, a significant reduction in overall execution time is obtained, and high frame rates can be achieved.

Acknowledgement: The authors thank Janet M. Siegel and the reviewers for their comments.

References

[1] G.S. Almasi and A. Gottlieb, *Highly Parallel Computing, 2nd Edition,* Benjamin/Cummings, Redwood City, CA, 1994.

[2] M. Auguin and F. Boeri, "The OPSILA computer," in *Parallel Languages and Architectures,* M. Consard, ed., Elsevier Science Publishers, Holland, 1986, pp. 143-153.

[3] T.B. Berg, S.D. Kim, and H.J. Siegel, "Limitations imposed on mixed-mode performance of optimized phases due to temporal juxtaposition," *J. of Parallel and Distributed Computing,* Oct. 1991, pp. 154-169.

[4] C.G. Herter, T.M. Warschko, W.F. Tichy, and M. Philippsen, "Triton/1: A massively-parallel mixed-mode computer designed to support high level languages," *Heterogeneous Computing Workshop (HCW '93),* Apr. 1993, pp. 65-70.

[5] ICE, Inc., *The MeshSP,* Technical Report, Waltham, MA, July 1995.

[6] K. Kanatani, "Camera rotation invariance of image characteristics," *Computer Vision, Graphics and Image Processing,* Vol. 39, No. 3, Sep. 1987, pp. 328-354.

[7] P.M. Kogge, "EXECUBE - A new architecture for scalable MPPs," *1994 Int'l Conf. on Parallel Processing,* Vol. I, Aug. 1994, pp. 77-84.

[8] G.J. Lipovski and M. Malek, *Parallel Computing: Theory and Comparisons,* John Wiley & Sons, New York, NY, 1987.

[9] M. Mirmehdi and T.J. Ellis, "Parallel approach to tracking edge segments in dynamic scenes," *Image and Vision Computing,* Vol. 11, No. 1, Jan. 1993, pp. 35-48.

[10] D. Murray and A. Basu, "Motion tracking with an active camera," *IEEE Trans. on Pattern Analysis and Machine Intelligence,* Vol. 16, No. 5, May 1994, pp. 449-459.

[11] J.R. Nickolls, "The design of the MasPar MP-1: A cost effective massively parallel computer," *IEEE Compcon,* Feb. 1990, pp. 25-28.

[12] J.M. Roberts and D. Charnley, "Parallel attentive visual tracking," *Engineering Applications of Artificial Intelligence,* Vol. 7, No. 2, July 1994, pp. 205-215.

[13] H.J. Siegel, T. Braun, H.G. Dietz, M.B. Kulaczewski, M. Maheswaran, P.H. Pero, J.M. Siegel, J.E. So, M. Tan, M.D. Theys, and L. Wang, "The PASM project: A study of reconfigurable parallel computing," *2nd Int'l Symp. on Parallel Architectures, Algorithms, and Networks (ISPAN '96),* June 1996, pp. 529-536. Invited.

[14] H.J. Siegel, M. Maheswaran, D.W. Watson, J.K. Antonio, and M.J. Atallah, "Mixed-mode system heterogeneous computing," in *Heterogeneous Computing*, M.M. Eshaghian, ed. Artech House, Norwood, MA, 1996, pp. 19-65.

[15] H.J. Siegel, L. Wang, J.E. So, and M. Maheswaran, "Data parallel algorithms," in *Parallel and Distributed Computing Handbook*, A. Y. Zomaya, ed., McGraw-Hill, New York, NY, 1996, pp. 466-499.

[16] M.D. Theys, R.M. Born, M.D. Allemang, and H.J. Siegel, "Morphological image processing on three parallel machines," *The 6th Symp. on the Frontiers of Massively Parallel Computation,* Oct. 1996, pp. 327-334.

[17] Y. Yamada and M. Ishikawa, "High speed target tracking using massively parallel processing vision," *1993 IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems,* July 1993, pp. 267-272.

Figures

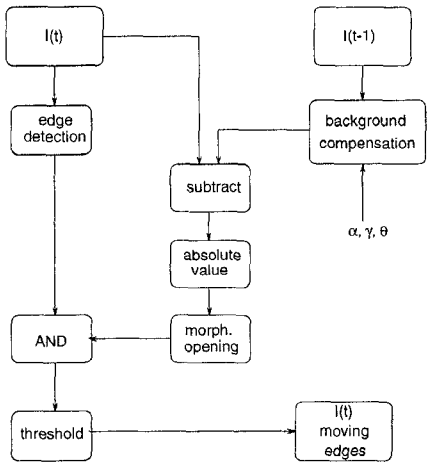

Figure 1: Block diagram of the serial algorithm.

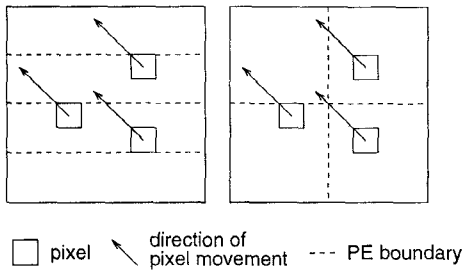

Figure 2: Row-striping communication pattern (no conflicts), left. Rectangular distribution communication pattern (two conflicts), right.

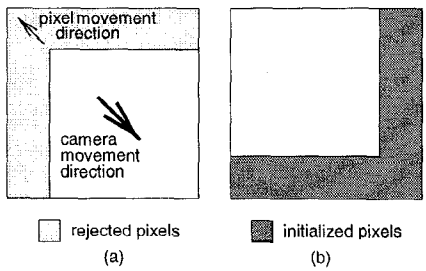

Figure 3: Location of (a) rejected pixels, and (b) initialized pixels.

Back. Comp.		Edge Detect.		Total Time		Num. PEs
Row	Rect.	Row	Rect.	Row	Rect.	
5.4586	5.7925	0.7459	0.7505	6.5785	6.9180	16
2.7319	2.9243	0.3759	0.3765	3.2948	3.4883	32
1.3684	1.4667	0.1908	0.1895	1.6528	1.7501	64
0.6868	0.7374	0.0983	0.0956	0.8320	0.8800	128
0.3286	0.3722	0.0522	0.0486	0.4042	0.4444	256

Table 1: Execution time in seconds on the MasPar MP-1 with a 256-by-256 pixel image.

Back. Comp.		Edge Detect.		Total Time		Num. PEs
Row	Rect.	Row	Rect.	Row	Rect.	
0.3830	0.3816	0.1138	0.1283	0.5253	0.5735	4
0.2316	0.2213	0.0568	0.0810	0.3100	0.3286	8
0.1292	0.1253	0.0266	0.0534	0.1642	0.1903	16
0.0781	0.0780	0.0158	0.0404	0.0981	0.1249	32
0.0478	0.0490	0.0079	0.0389	0.0580	0.0909	64
0.0368	0.0363	0.0071	0.0408	0.0451	0.0786	128

Table 2: Execution time in seconds on the Intel Paragon with a 256-by-256 pixel image.

Back. Comp.		Edge Detect.		Total Time		Num. PEs
Row	Rect.	Row	Rect.	Row	Rect.	
70.449	75.267	13.483	14.983	93.173	100.55	4
35.636	38.169	6.8576	7.6453	47.146	50.991	8
23.571	25.612	4.3995	5.1235	30.679	32.312	16

Table 3: Execution time in seconds on PASM in SIMD mode with a 128-by-128 pixel image.

Back. Comp.		Edge Detect.		Total Time		Num. PEs
Row	Rect.	Row	Rect.	Row	Rect.	
19.578	22.216	6.1522	8.7144	27.758	35.069	4
10.599	13.089	3.3399	6.3683	14.610	19.782	8
5.3330	6.4096	1.5574	3.9319	7.4050	10.627	16

Table 4: Execution time in seconds on PASM in MIMD mode with a 128-by-128 pixel image.

Back. Comp.		Edge Detect.		Total Time		Num. PEs
Row	Rect.	Row	Rect.	Row	Rect.	
19.317	22.939	4.0464	5.4570	25.913	32.583	4
10.600	13.217	2.1194	2.8109	13.873	18.212	8
5.3358	6.3577	1.1255	1.4760	7.0733	8.8693	16

Table 5: Execution time in seconds on PASM in mixed mode with a 128-by-128 pixel image.