SIMD and Mixed-Mode Implementations of a Visual Tracking Algorithm

Mark B. Kulaczewski
Laboratorium für Informationstechnologie
University of Hannover
30167 Hannover, Germany
mbk@mst.uni-hannover.de

Howard Jay Siegel
Parallel Processing Laboratory
School of ECE, Purdue University
West Lafayette, IN, 47907-1285, USA
hj@ecn.purdue.edu

Abstract

This paper describes the implementation of a feature-based visual tracking algorithm on a SIMD MasPar MP-1 and the mixed-mode PASM prototype. The sequential algorithm is introduced and a parallel version of the algorithm is described. Aspects of the mapping of this algorithm onto both machines are presented. Real image data was used to obtain timing results for all experiments. For the SIMD implementation, the performance of each subtask of the algorithm is measured as a function of submachine size used. For the subtask of finding the best match of a feature in an image, different implementations using architectural properties are compared, and a load balancing approach is examined. For the mixed-mode implementation, the best execution mode for each subtask is determined and the results are analyzed.

1. Introduction

As an example of an image processing application with possible real-time constraints, a feature-based visual tracking algorithm was mapped onto four different parallel machines. The implementations on two commercial MIMD machines were presented earlier in [5]. This paper is a summary of parts of [4] and presents the implementation on the SIMD MasPar MP-1 and the mixed-mode PASM prototype. Examples of existing or recently proposed SIMD architectures specifically designed for image processing applications are CPP's Gamma [2], HiPAR [7], and MPA [3]. Image processing is also one of the target applications for the many prototyped mixed-mode machines, e.g., PASM, Triton, Execube, and MeshSP [8]. It is, therefore, of interest to investigate the architectural implications and trade-offs experienced when porting an image processing application to SIMD and mixed-mode systems.

This research was supported in part by the NRaD Naval Laboratory under contract N66001-96-M-2277, and by the DARPA/ITO Quorum Program under NPS subcontract N62271-97-M-0900. The equipment used was supported in part by NSF under grant CDA-9015696.

2. The Visual Tracking Algorithms[1]

2.1. Sequential Visual Tracking

The algorithm for this application study is based on work presented in [9]. Gray-scale image sequences to be analyzed are obtained using a static camera. Images are of size $\underline{C}$ columns by $\underline{R}$ rows. $\underline{I_m}$ denotes the m-th <u>source</u> <u>image</u> in the sequence, with $I_m(x,y)$ the gray-scale value of the pixel at position (x,y). The <u>ground</u> <u>image</u> $\underline{G_m}$ is a weighted average of past source images, defined as $G_0 = I_0$ and

$$G_m(x,y) = (1-a)G_{m-1}(x,y) + aI_m(x,y) \qquad (1)$$

for $m \geq 1$, with $0 \leq a < 1$. G_m incorporates all objects that belong to the background. Typical values for a are close to zero. The binary <u>figure</u> <u>image</u> $\underline{F_m}$ is defined as

$$F_m(x,y) = \begin{cases} 1 & \text{if } |G_m(x,y) - I_m(x,y)| > T \\ 0 & \text{otherwise} \end{cases} \qquad (2)$$

with $\underline{T}$ being a <u>threshold</u> <u>value</u>. If $F_m(x,y)$ is equal to 1, it is assumed that the pixel at position (x,y) in I_m corresponds to a part of a moving object to be tracked. A <u>feature</u> $\underline{W}$ is defined as an array of gray-scale values of size $\underline{W_C}$ columns by $\underline{W_R}$ rows. The reference point of a feature is the left upper corner with pixel coordinates $(0,0)$. A feature of an object is used in this algorithm to track the object. The measure used to quantify how well a feature matches an image area is the <u>s</u>um-of-the-<u>s</u>quared-<u>d</u>ifferences (<u>SSD</u>). The SSD for a feature and a rectangular subimage of I_m with its left upper corner at position (x,y) is defined as

$$\text{SSD}(I_m, W, x, y) = \sum_{i=0}^{W_C-1} \sum_{j=0}^{W_R-1} (I_m(x+i, y+j) - W(i,j))^2. \qquad (3)$$

The search for the best match of a given feature is restricted to a subimage of I_m. The <u>search</u> <u>region</u> $\underline{A}$, centered at (x,y), is the set $A(x,y) = \{(u,v) : |u-x| \leq A_C \wedge |v-y| \leq A_R\}$. A_R and A_C are determined by the application. Finding the the best match of a feature window W in I_m, assuming the best match in image I_{m-1} was at (x,y), is equivalent to

$$\begin{array}{l} \text{minimize SSD}(I_m, W, u, v) \\ \text{subject to } (u,v) \in A(x,y). \end{array} \qquad (4)$$

For every source image, the ground image is updated ac-

[1] This section summarizes a part of [5].

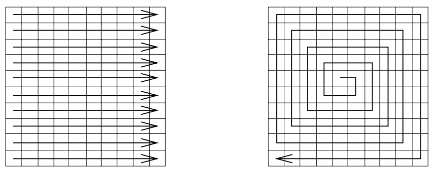

Figure 1. SSD calculation patterns.

```
DATA DISTRIBUTION PHASE:
Distribute $I_m$ across PEs for all $m$
Initialize local subimage of $G_0$ (for $m = 0$)
Calculate local subimage of $G_m$ (for $m \geq 1$)
DETECTION PHASE:
Local Segmentation Step:
Calculate local subimage of $F_m$
Partition local subimage of $F_m$ into connected components
            and calculate bounding box for each component
Merge overlapping local bounding boxes
Delete bounding boxes below threshold size
          (if not on a subimage edge)
Extract edge connectivity information
Combine Step:
Combine local components based on edge information
If new features needed:
          SELECTION PHASE:
          Determine set of features for each resulting bounding box
TRACKING PHASE:
Local Tracking Step:
Find best match for features on subset of search region
Combine Step:
Combine local search results to find global best match
```

Figure 2. Parallel tracking algorithm.

cording to Equ. (1). Every p images, the object detection phase is executed. The figure image F_m is calculated (Equ. (2)), the connected components in F_m are computed, and the bounding box (smallest rectangle that encloses all points) for each connected component is determined. Overlapping bounding boxes are merged, and resulting bounding boxes below a threshold size are deleted from further analysis. In the feature selection phase, a suitable subimage from within an object's bounding box is selected as a feature. For this work, features are selected on a monoprocessor and made available to all processors of the parallel machine (this time is negligible). During the tracking phase, the best match for a feature is determined by solving the minimization problem of Equ. (4), using an exhaustive search.

To reduce computation time, the calculation of the SSD measure for a given position is provided with the minimum SSD value observed so far. If the running sum of Equ. (3) exceeds this value, the calculation for this position is aborted, and the next position is checked. Two different methods to evaluate the double sum are illustrated in Fig. 1, the row-major and the spiral pattern [9]. The best performance on a monoprocessor was found for a spiral SSD calculation.

2.2. Parallelized Visual Tracking

For the parallelized version of the tracking algorithm, the object detection phase is executed for every image I_m (i.e., $p = 1$). This allows detection of a new object in every image. A distributed memory architecture parallel machine with N=2^n PEs (processor and memory pairs) is assumed. The PEs are arranged logically (not necessarily physically) in a mesh with V rows and H PEs per row and numbered in row-major order from 0 to $N - 1$. Image data distribution refers to the process of storing image data into PEs. Data layout describes what data is stored in each PE. Work layout describes how the work is assigned to PEs.

A pseudocode version of the parallelized visual tracking algorithm is shown in Fig. 2. The algorithm for the object detection phase is based on a divide-and-conquer approach [1]. Two intuitive ways exist to divide the image into equal subimages. For row striping, each PE works on a subimage of size C columns by R/N consecutive rows. For rectangular subimages, each PE is assigned a subimage of size $C/H \times R/V$. The work layout for the local segmentation step determines what part of the source image I_m must be present in each PE. Each PE also maintains a subimage of G_m and F_m with the same data layout. For every I_m, each PE determines the connected components of its associated subimage of F_m, using a sequential labeling algorithm. For each connected component, its associated bounding box is computed, overlapping bounding boxes are merged, and irrelevant bounding boxes are deleted. For the subsequent combination of information about subimages, the required connectivity information is derived. Recursive doubling is used to combine all local results efficiently. For recursive doubling with the result-to-every-PE technique, in each step, $N/2$ distinct pairs of PEs exchange intermediate results and combine these. After $\log_2 N$ steps, all PEs have the bounding boxes for the whole image. Details for the combine process can be found in [4].

Depending on the data layout of I_m, different implementations of the tracking phase are possible. For whole images on all PEs, every PE could calculate the SSD for any position in the search region without any additional communication. A simple approach to distribute the work across PEs in this case is to divide the search region evenly among PEs, e.g., using row striping. If subimages are chosen as the data layout, an additional number of overlap pixels from adjacent subimages is needed to perform the matching of a feature for all positions corresponding to a subimage. With subimages on PEs, two tracking approaches were examined. Without load balancing, each PE calculates the best match for a feature for the subset of the whole search region it can work on using local and overlap data only. If load balancing is desired, each PE is assigned a disjoint subset of the search region. For a given data layout, this also involves the sending of image data. For both approaches, more than one PE may search a region A for a given feature. Recursive doubling is used to combine intermediate results of all local best match searches.

The workload per PE for this algorithm is highly dependent on image content. Realistic timing data can, therefore, only be obtained with real image data. Similar to [9], some scenes showing pedestrian traffic were used. The feature window size is $W_C = W_R =$ 10. A search region A of size 41 $\times$ 41 positions, i.e., $A_C = A_R =$ 20, was used. All results shown were obtained using input sequences with one moving object and one feature per object to be tracked. An analysis for tracking multiple features using the presented techniques can be found in [4].

3. SIMD Experiments

3.1. MasPar MP-1 Architecture

The MasPar MP-1 [6] is a massively parallel, distributed memory SIMD architecture. The MP-1 used for these experiments has 2^{14} PEs organized in a 128 $\times$ 128 mesh. Each PE is a 4-bit processor and is connected with its eight nearest neighbors via the X-Net network. A multistage network, the global router, provides arbitrary communication patterns. The PE array is controlled by the Array Control Unit (ACU), which can also be used to broadcast data to all PEs. PEs can send data to the ACU using a global-or tree that combines data from all enabled PEs. The algorithm was implemented in MPL. A variable in MPL belongs to one of the storage classes singular or plural. Storage for singular variables is allocated on the ACU; a singular variable, therefore, always has a unique value. A plural variable has storage allocated on each PE and will have, in general, different values across PEs.

For the images given (C = 512 by R = 512), row striping can not be used, because the number of PEs for the full machine would exceed the number of rows in the input image. Thus, rectangular subimages are used exclusively as the data layout and the work layout for object detection. All experiments were performed on submachines of size $16 \times 16, 32 \times 32$, and 64×64, and the full 128×128 machine. Execution on smaller submachines was not possible due to PE memory constraints. All results were calculated based on an image sequence with 60 single images. It is assumed that at start of the detection phase all required data, including the overlap pixels, reside on the respective PEs.

3.2. Object Detection

Local Segmentation. The MP-1 allows local memory references using a PE-local register as an index. This offers the possibility of creating dynamic data structures on each PE, and the sequential labeling implementation uses this feature extensively. Fig. 3 shows the timing results for the object detection phase. The time needed for the local segmentation step is mainly determined by the local subimage size. With increasing N, the subimage size decreases, and so does the time required for this computation step.

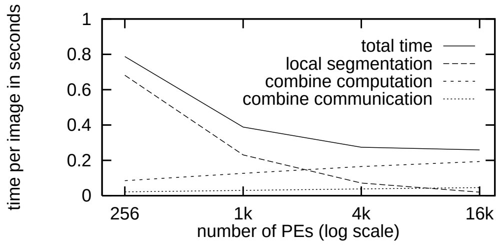

Figure 3. MasPar object detection phase.

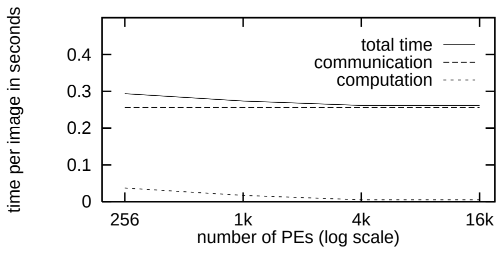

Figure 4. Balanced tracking on MasPar.

Combining. The fastest implementation was achieved by using recursive doubling with the result-to-one-PE technique, where a sending PE is disabled after the communication, and the combine operation is performed by the receiving PE. The final result is broadcast via the ACU to all PEs. For submachines of size $N \geq$ 4k (64×64), the computation time for the combine operation accounts for more than 50% of the time needed for the object detection phase (Fig. 3).

3.3. Feature Tracking

For tracking with load balancing, the image data necessary to determine the best match of the current feature is broadcast to all PEs, and the search region A is distributed evenly among PEs (Fig. 4). The time for balanced tracking is the sum of the time for broadcasting the required image data and the tracking operation itself. This broadcasting time is constant, because the instruction broadcast bus is used to send the data to all PEs in parallel. For the parameters used here, an image area of 50 $\times$ 50 pixels is broadcast. The time for combining of results to obtain the position of the best match is negligible and is not included in the graphs.

Two implementation variations of the tracking phase without load balancing were examined: singular tracking and plural tracking. For plural tracking, all pointer variables used have storage class plural, i.e., they are stored and all pointer arithmetic is performed on the PEs. For singular

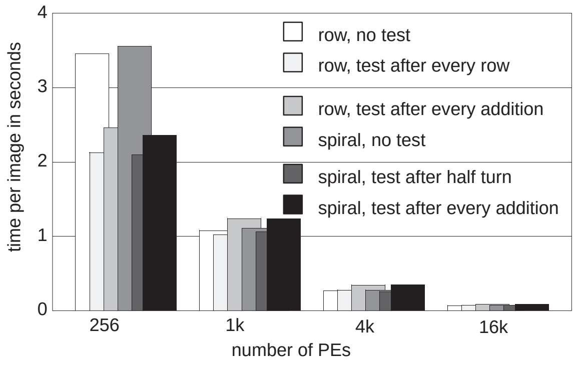

Figure 5. Plural tracking on MasPar.

tracking, all pointer arithmetic is performed on the ACU, where the pointer's storage is allocated. It can be shown [4] that for plural tracking, the number of times the SSD calculation must be executed is determined by the PE with the maximum number of positions to check. For singular tracking, this number is, in general, bounded below by the number for plural tracking and bounded above by the maximum of the subimage size and the size of the search region.

For calculating the SSD, row-major and spiral patterns were examined. For the row-major pattern, the running sum of the SSD measure on each PE was compared with the current minimum on a PE so far (1) after every addition, (2) after every row, or (3) at the end only. A PE exits the calculation if the running sum exceeds the minimum so far. For the spiral calculation pattern, this comparison was done (1) after every addition, (2) after every half turn of the spiral, or (3) at the end only. The results for the different implementations for plural pointers are shown in Fig. 5. The relative results are similar for singular and plural tracking. In both cases, testing after each row or half turn is faster than testing after each addition. The overhead created by comparing after each addition outweighs the possible benefit of terminating the SSD calculation earlier. For smaller machine sizes, the implementation with a test after every row or half turn is up to 40% faster than not testing against the current minimum at all. With increasing machine size, and, therefore, more PEs involved in a search, this effect disappears, and tracking with or without testing after every row or half turn is almost equally fast. That is, because the more PEs take part in a search for the best match, the higher the probability is that at least one PE has to calculate the full SSD.

Plural tracking is faster than singular tracking on smaller submachines. With increasing number of PEs, singular tracking becomes a slightly faster solution. Plural tracking for smaller submachines benefits from the fact that the number of SSD calculations to be performed is determined by the maximum number of match positions to be checked across all PEs. For larger submachines, subimages can become small enough, and at least one single PE needs to calculate

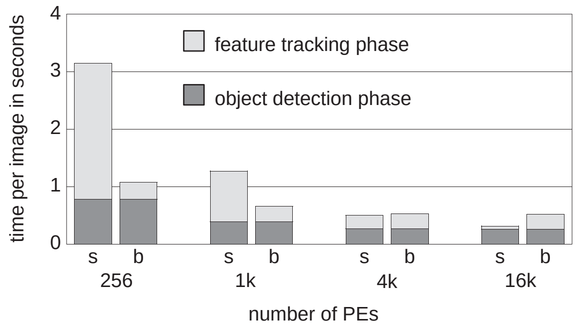

Figure 6. <u>S</u>ingular and <u>b</u>alanced MasPar tracking.

the SSD measure for all possible $R/V \cdot C/H$ positions, i.e., singular and plural tracking perform the same number of SSD calculations for a feature. In that case, singular pointer usage is faster because plural pointer operations are substantially slower on the MP-1.

3.4. SIMD Results

The average processing time for different submachine sizes, using the fastest implementation for singular tracking and load balancing, is shown in Fig. 6. For singular tracking and smaller submachine sizes, the processing time is dominated by the tracking phase. Balanced tracking provides a faster solution for $N < 4$k, whereas, for $N \geq 4$k, the unbalanced version results in a smaller execution time.

The object detection phase scales well only up to $N = 1$k PEs. This is because the implementation of the combine operation makes extensive use of plural pointer operations, which are relatively expensive compared to integer operations. A different algorithm for a massively parallel system like the MP-1 might have been a better choice. Investigating different algorithm choices, however, was outside the scope of this study. The same object detection algorithm was used for all four machines (here and in [5]).

4. Mixed-Mode Experiments

With <u>mixed-mode</u> parallelism, a submachine can switch between SIMD execution and MIMD execution at instruction level granularity. The <u>PASM</u> prototype built at Purdue contains 16 PEs (MC68010) [8]. PEs are connected by a multistage cube class network. For the prototype, submachines with full mixed-mode capabilities have four, eight, or 16 PEs. Although not competitive with respect to absolute computing power, PASM allows experimentation to obtain relative performance data for the different execution modes. Other mixed-mode machines that have been built are Triton, Execube, and MeshSP [8]. Because the focus of the experiments on PASM was to determine the best execution modes, whole images ($C = 640$ by $R = 480$) on each PE was selected as the data layout. All results were obtained using 20 images.

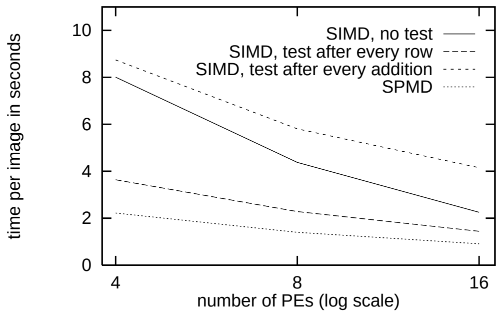

Figure 7. Tracking times on PASM.

The implementation of the local segmentation step is characterized by many nested conditional statements that are, in general, executed faster in SPMD mode than SIMD [8]. This behavior was confirmed with experimental data, with execution in SPMD mode up to five times faster than execution in SIMD mode.

On PASM, communication between PEs can be performed in two ways. MIMD asynchronous communication is provided by an implementation of message passing primitives. SIMD synchronous communication is implemented using network permutations and involves little overhead. A network permutation is a one-to-one mapping of sending and receiving PEs, i.e., every PE sends data to exactly one PE, and every PE receives data from exactly one PE. Three different implementations of the combine step were examined. Mixed-mode yielded the smallest execution times by combining the faster computation mode (MIMD due to conditionals) with the more efficient communication mode (SIMD due to low protocol overhead).

The distribution of the search region for a best feature match across PEs was performed using row striping. Because each PE maintained a copy of the whole source image, this method allowed a communication-free static load balancing for the tracking phase. For all implementations examined, the SSD measure was calculated using the row-major pattern. The SPMD implementation of the SSD compares the running sum against the minimum so far after every addition, and, for SIMD mode, three different versions perform the check for an early termination (1) after every addition, (2) after every row, or (3) not at all (Fig. 7). Similar to results on the MP-1, testing after every row provided the fastest version in SIMD mode. Testing for possible termination after every addition was, on PASM for all submachine sizes, slower in SIMD mode than not performing the test at all. This is mainly due to the expensive implementation of PE enabling/disabling in SIMD mode in the prototype. The SPMD implementation of the tracking phase outperformed all other implementations. A full SIMD implementation is approximately four times slower than SPMD and mixed-mode implementations, mainly due to low SIMD performance of the object detection phase. For a given machine size, the fastest implementation was achieved by performing all computations in SPMD mode and the communications using SIMD synchronous communication. The mixed-mode implementation is only slightly faster than the SPMD implementation because the time for the local segmentation step dominates the total time.

5. Conclusion

The implementations of a feature-based visual tracking algorithm on a SIMD and a mixed-mode machine were summarized from [4]. Different implementations for object detection and feature tracking were analyzed in detail. Case studies such as these are important for the fields of parallel processing, heterogeneous computing, and the application domain. For parallel processing, techniques for exploiting particular architectural attributes of different parallel machines have been presented and evaluated. For mixed-machine heterogeneous computing, where a set of machines are interconnected by high-speed links to form a metacomputer, studies such as this and [5] can help guide the choice of machine to use for each subtask in the application. Finally, for the application domain researchers, this study shows the potential for the use of different types of parallelism in solving practical problems of importance.

References

[1] H. M. Alnuweiri and V. K. Prasanna. Parallel architectures and algorithms for image component labeling. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 14(10):1014–1034, Oct. 1993.

[2] Cambridge Parallel Processing. Gamma technical overview. Tech. rep., Cambridge Parallel Processing, Irvine, CA, 1996.

[3] Y. Kim, T.-D. Han, S.-D. Kim, and S.-B. Yang. An effective memory-processor integrated architecture for computer vision. In *1997 Int'l Conf. Parallel Processing*, pp. 266–269.

[4] M. B. Kulaczewski. Parallel Implementations of a Visual Tracking Algorithm, and Dynamic Partitioning for a Mixed-Mode Programming Language. Master's thesis, School of ECE, Purdue University, Dec. 1996. Advisor: H. J. Siegel.

[5] M. B. Kulaczewski and H. J. Siegel. Implementations of a feature-based visual tracking algorithm on two MIMD machines. In *1997 Int'l Conf. Parallel Processing*, pp. 422–430.

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

[7] K. Rönner and J. Kneip. Architecture and applications of the HiPAR video signal processor. *IEEE Trans. Circuits and Systems for Video Technology*, 6(1):56–66, Feb. 1996.

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

[9] C. E. Smith, S. A. Brandt, C. A. Richards, and N. P. Papanikolopoulos. Visual tracking for intelligent vehicle-highway systems. *IEEE Trans. Vehicular Technology*. To appear.