

Image Feature Extraction on Connection Machine CM-5 *

Viktor K. Prasanna and Cho-Li Wang

Department of EE-Systems, EEB-244
University of Southern California
{prasanna, chowang}@halcyon.usc.edu

(Summary of Results)

Abstract

In this paper, we present a parallel implementation of an image feature extraction task on Connection Machine CM-5. We show that, given a 2048×2048 grey level image as input, the extraction of image features, which includes edge detection, thinning, linking, and linear approximation, can be performed in less than 1.2 seconds on a partition of CM-5 having 512 processing nodes. A serial implementation on a Sun Sparc 400 takes more than 8 minutes. Experimental results on various sizes of images using various partitions of CM-5 are also reported. The software has been developed in a modular fashion to permit various techniques to be employed for the individual steps of the processing. Our technique starts by modeling the communication and computation features of the machine. Using this model, scalable algorithms are designed.

1 Introduction

Image feature extraction is a fundamental process in vision. In general terms, the primary task of feature extraction is to extract a subset of pixels from the original image array, and to generate a representation of image content which is appropriate for symbolic manipulations at a higher level.

Image feature extraction has been known to be ideally suited for parallel implementations. In the past, fast parallel algorithms and implementations have been developed for several image feature extraction tasks. However, most of the contributions have been at the individual task level. Our motivation for this research is to build a high-speed image feature extraction system which provides the desired inputs to higher level analyses on CM-5, a state-of-the-art coarse grain MIMD machine.

Even though, the extraction of image feature has been well understood and the real challenge lies in devising approaches to image understanding, to our knowledge no parallel system for image feature extraction is available to the vision community. Indeed, image feature extraction operations can be very time consuming. For example, the well-known Nevatia-Babu line finder implemented in LISP can take several hours to extract features

in a 2048×2048 image on a state-of-the-art Sun Sparc station. Even though we reported [18] a few hundreds of millisecond performance to perform a probe in object recognition task on CM-5, without a fast feature extraction system, realizing a complete vision system remains to be a distant goal.

Design of scalable algorithms for extracting image features is somewhat challenging since the computational requirements of feature extraction tasks depend on the input and output data structures and algorithms used to implement specific tasks. In this paper, we discuss the parallel implementation of a typical image feature extraction application proposed in [12]. This is used to extract linear features from input image. The algorithm proceeds by extracting neighboring edges of similar orientation into contours and then approximating the shape of the contours with piecewise linear segments. The process of extracting linear features consists of two major procedures [12]: *contour detection* and *linear approximation*. The contour detection involves detection of edges using convolution, removal of the “false” edges using a thinning operation, and removal of the “weak” edges using a thresholding operation. Upon detection of the edge pixels, a linear approximation is performed to group the detected edge pixels into line segments. The sequential algorithm in [12] and its variants have been widely used by the vision community.

While parallelizing the feature extraction task, the contour detection and linear approximation tasks exhibit different types of computational characteristics. In contour detection, the operations can be easily performed in a synchronized fashion with data communication between the pixels that is regular and local to each pixel. This type of operation maps well onto SIMD machines. However, in linear approximation, the communication can extend over large windows and is often irregular. This maps well onto MIMD machines. In our implementation, CM-5, a *synchronized* MIMD machine, is operated in SPMD (Single-Program Multiple-Data) mode, half way between the highly synchronized SIMD model and the message-passing MIMD model. This provides the desired capabilities to build a image feature extraction system.

We employ a realistic model of CM-5 to predict the performance of our algorithms. This model considers the communication overheads. We assume that in a unit

*This research was supported in part by NSF under grant IRI-9217528 and in part by ARPA under grant F49620-93-1-0620.

Figure 1: An overview of the Neva-Babu line finder.

The above tasks have been implemented using C and LISP on several sequential machines, including Sparc workstations and Symbolics machine. This system is widely used to produce linear features for higher level vision processing such as image matching, perceptual grouping, object recognition, etc.

3 A Model of CM-5

A Connection Machine Model CM-5 system contains between 32 and 16,348 processing nodes (PNs). Each node is a 32 MHz Sparc processor with upto 32 Mbytes of local memory. The peak performance of a node having 4 vector units is 128 MFLOPS. The PNs are interconnected by three networks: a data network, a control network, and a diagnostic network. The data network provides point-to-point data communication between any two PNs. Communication can be performed concurrently between pairs of PNs and in both directions. The data network is a 4-ary *fat tree* [8]. The bandwidth continues to scale linearly up to 16,384 PNs [9]. The control network provides cooperative operations, including broadcast, synchronization, and scans (parallel prefix and suffix). The control network is a complete binary tree with all the PNs as leaves.

For our analysis, we will model the CM-5 as a set of high performance SISD machines interacting through the data and control networks. We assume *cooperative message passing* [17]: the sending and receiving PNs must be synchronized before sending the message. Thus, the *startup* cost including software overhead and synchronization overhead is associated with each message. We assume SPMD (*Single Program Multiple Data*) mode execution in which each PN runs the same part of a program asynchronously until a synchronization point is reached. Synchronization points are inserted before a data communication step. The synchronization cost can be counted as part of the startup time. In our analysis, we consider machine sizes that are not “large.” The hardware latency (network interface overhead and net-

work latency) in data network is small and is *hidden* by the software overheads.

Let T_d denote the startup time (seconds/message) for sending a message using data network. Let τ_d denote the transmission rate (seconds per unit of data) for data communication using the data network. We make the following assumptions for our analysis:

1. In a unit of time, a PN can perform an arithmetic/logic operation on local data.
2. Sending a message containing m units of data from a PN to another PN or exchanging a message of size m between a pair of PNs takes $T_d + m\tau_d$ time using the data network.
3. Suppose each PN has m units of data to be routed to a single destination using the data network and the set of all destinations is a permutation, then the data can be routed in $T_d + m\tau_d$ time.

It has been measured [5], the startup time T_d is around 40-90 μsec which depends on the use of communication primitives. These times are measured by sending a 0 byte message between two PNs using the data network. For regular data communication pattern, τ_d is at the range of 0.100 to 0.123 $\mu\text{sec}/\text{byte}$ [5]. This model favors communicating long messages to communicating large number of short messages. A unit of data is defined as a fixed size data structure to contain image data (a contour pixel, a label etc.) in our analysis. Using this model, we can quantify the communications time and predict the running times of our implementations. We have defined a related model in [18]. A framework for understanding communication costs in terms of the communication patterns and the message sizes of the computation is also proposed in [10].

4 Scalable Parallel Algorithms

A parallel algorithm is considered scalable if the execution time of the algorithm on a machine with P processors varies as $\frac{1}{P}$ [4]. Our goal is to design scalable algorithms which provide high-speed execution on available partition sizes of machines for problem sizes useful to the vision community.

The contour-detection task exhibits parallelism in a natural way. However, the linear approximation task does not have explicit parallelism; in the absence of efficient partitioning and data movement techniques, the data communication overheads may dominate the overall execution time.

4.1 Contour Detection

The contour-detection task consists of edge detection, thinning, and linking steps. These are *window operations*, in which the output at a pixel is based on the value of the input pixel and the value of its neighboring pixels. The neighborhood is defined by the window size. Several types of window operations with different window sizes are performed in the image feature extraction system [12].

The image array is divided into P blocks, where P is the number of processors available. Each block is of size

$\frac{n}{\sqrt{P}} \times \frac{n}{\sqrt{P}}$. The blocks are numbered in a shuffled row major order. PN_i receives the i th block, $0 \leq i \leq P - 1$. This mapping reduces the communication towards the root of the tree.

An important step of the implementation is the *image boundary padding* which involves communication between PNs. This step is required because the window operations performed on the boundary pixels of an image block may need pixel data stored in the neighboring PNs. The communication time depends on the size of the window.

Following the techniques described in [12], in edge detection, the input image is convolved with six masks. Each mask is of size 5×5 . Boundary padding is needed to exchange the image data of boundary pixels before executing the convolution step. The outputs of the convolution step are the magnitude and the orientation of the edges. After the convolution step, boundary padding is needed. The edge magnitude and orientation information are then used to perform edge thinning which can be considered as a 3×3 window operation. Again, edge pixel information established by the thinning step needs to be updated. In edge linking, an edge pixel is linked with its neighbors, in the direction of its orientation, if they are of similar orientation. Some edge refining procedures are performed in this step to bridge the gap between disconnected edges pixels. This operation can be regarded as a 5×5 window operation.

Procedure : Contour-Detection

Input: $n \times n$ image and each PN has image block of size $\frac{n}{\sqrt{P}} \times \frac{n}{\sqrt{P}}$

Step 1: Perform *Boundary Data Padding*

Step 2: Perform *Edge Detection*

Step 3: Perform *Boundary Data Padding*

Step 4: Perform *Thinning and Thresholding*

Step 5: Perform *Boundary Data Padding*

Step 6: Perform *Edge Linking*

end

Figure 2: A skeleton of contour detection procedure.

The processing steps for the contour detection are outlined in Fig. 2. Given an $n \times n$ image and P PNs, we assume that each PN contains an image block of size $\frac{n^2}{P}$ and the largest window is of size $m \times m$. The serial complexity of contour detection is $O(n^2m^2)$, assuming a constant number of window operations are performed.

Theorem 1 *Given an image of size $n \times n$, the contour detection procedure can be performed in $O(n^2m^2/P)$ computation time and $12T_d + (12\lfloor \frac{m}{2} \rfloor \frac{n}{\sqrt{P}} + 24\lfloor \frac{m}{2} \rfloor)\tau_d$ communication time using a partition of CM-5 having P PNs.*

The total computation time for Steps 2, 4, and 6 is $O(\frac{m^2n^2}{P})$. The communication time for each Boundary Data Padding step is $4T_d + (4\lfloor \frac{m}{2} \rfloor \frac{n}{\sqrt{P}} + 8\lfloor \frac{m}{2} \rfloor)\tau_d$. This assumes $\lfloor \frac{m}{2} \rfloor < \frac{n}{\sqrt{P}}$. Note that in practice, $n \geq 256$, $m \leq 10$, and $P \leq 1K$.

is applied starting at p_{b-1} . If p_b is the last pixel, then $\overline{p_a p_b}$ is the approximation to the contour from p_a to p_b and the procedure stops. Assuming that the total number of pixels on the contour as l , it is easy to verify that the strip-based algorithm runs in $O(l)$ time on a serial machine.

Most of the known heuristics for linear approximation are inherently sequential in nature. One of the solutions to speed up the process is to let each PN concurrently approximate the contours whose starting pixels are located in its image block; if the contour crosses over the boundary, the approximation is continued by the PN containing the the next part of the contour. The process continues until all contours are approximated. However, in this approach, some PNs can become bottlenecks, if many contours pass through them and the approximation processes for each of these contours arrive at these PNs at the same time.

Our approach is to assign the same label to each pixel of a contour and group all the pixels having the same label to a single PN to perform the linear approximation. Using this data allocation strategy, the contour redistribution is performed once to localize the contours into the PNs and the communication needed during the execution of the approximation procedure is minimal. An advantage of this approach is that any linear approximation heuristic (that has been proposed for serial machines) can be adapted without having to pay a communication penalty after the contours have been localized. We define *workload* on a PN as the total number of contour pixels to be processed by the PN. We employ a *load balancing* strategy that relocates the contour pixels such that all the connected contour pixels are moved to the same PN (or at most two PNs) and each PN has no more than $O(\frac{n^2}{P})$ *workload*. The processing steps for the linear approximation on CM-5 are outlined in Fig. 4.

Procedure : Linear-Approximation

Input: PN_{*i*} has contour data extracted from the *i*th image block, $0 \leq i \leq P - 1$.

Step 1: Perform *Linear Approximation*

Step 2: Perform *Connected Component Labeling*

Step 3: Group contour data having the same label to a PN

Step 4: Perform *Linear Approximation*

end

Figure 4: A skeleton of linear approximation procedure.

Theorem 2 *Given an image of size $n \times n$, the Linear Approximation step can be performed in $O(n^2/P)$ computation time and $(20P + 3 \log P)T_d + (24n + \frac{80n^2}{P})\tau_d$ communication time on a partition of CM-5 having P PNs, where $n \geq \frac{1}{2}P^{\frac{3}{2}}$.*

Given an $n \times n$ image and P PNs, we assume that each PN contains an image block of size $\frac{n^2}{P}$ and all the contour pixels in the image block have been detected.

Figure 3: Strip-based linear approximation.

The algorithm proceeds as follows [15]: given a starting pixel p_a of a contour and an error bound ϵ for controlling the quality of the approximation, it selects a pixel p_f on the contour which is at a distance $> \epsilon$ from p_a . If it can not find such a pixel, the algorithm stops and forms a line from p_a to the last pixel of the contour. Otherwise, draw a line using p_a and p_f . This line is referred to as the *critical line*. Next form two lines parallel to the critical line at a distance ϵ . These lines are referred to as the *boundary lines* (see Figure 3). Beginning with pixel p_{f+1} , examine the remaining pixels on the contour until a pixel p_b is found, where p_b is the first pixel lying outside the region formed by the boundary lines or p_b is the last pixel on the contour.

If p_b is outside the region formed by the boundary lines, then the line segment $\overline{p_a p_{b-1}}$ is the approximation to the contour from p_a to p_{b-1} and the same procedure

An outline of our scalable algorithm is shown in Figure 4.

In Step 1, any contour that does not cross an image boundary can be approximated within a PN. This approximation can be carried out for all the contours local to the PNs in $O(\frac{n^2}{P})$ time. In the remainder of this discussion, we only consider contours that cross image boundaries. In Step 2, a divide-and-conquer strategy can be used to complete the component labeling in $O(\frac{n^2}{P} + n\sqrt{P})$ computation time and $24n\tau_d + 3 \log PT_d$ communication time. In Step 3, a modified *column sort* algorithm [7] described in [18] can be used to move contours. The algorithm [18] repeats a data permutation step and a local sorting step for a constant number of times. Note that we only sort the labels. Contour data having the same label are moved along with the label. Thus, the computation time for sorting is $O(\frac{n \log n}{\sqrt{P}})$. If $n \geq \frac{1}{2}P^{\frac{3}{2}}$, the algorithm [18] requires 20 permutation steps. The total communication time is $\frac{80n^2}{P}\tau_d + 20PT_d$ communication time. At the end of the sorting step, the contour pixels are redistributed such that each PN has at most $\frac{n^2}{P}$ contour pixels and each contour is in at most two adjacent PNs and given two PNs at most one contour extends over them. The strip-based heuristic for approximating the contours can be applied within each PN. Step 4 can be completed in $O(\frac{n^2}{P})$ time. On a partition of CM-5 having P PNs, the computation time is $O(n^2/P)$ and the communication time is $(20P + 3 \log P)T_d + (24n + \frac{80n^2}{P})\tau_d$, where $n \geq \frac{1}{2}P^{\frac{3}{2}}$. Note that if $n \geq \frac{3}{2}P^{\frac{3}{2}}$, a modified *rotate sort* [11] described in [2] can be employed to reduce the communication time as it only requires 8 data permutation steps.

5 Implementation Details and Experimental Results

The system was implemented on CM-5 partitions of 32, 64, 256, and 512 PNs. The code was written using C and CMMD 3.0 message passing library provided by the Thinking Machines Corporation. The total length of the code is around 2000 lines. The code has two parts. The code executed by the control processor deals with I/O, user-system interaction (such as setting up the error bound ϵ used in linear approximation etc.), and gathering statistics from the PNs. The second part is executed by the PNs which perform the convolution, thinning, linking, and linear approximation with barrier synchronization inserted between successive steps. The modular design of this implementation allows users to replace the processing modules by user's code with very little effort.

The program starts by reading the image data to the control processor and then evenly distributing it to the PNs. Results are stored in the PNs and can be output through the control processor. In the current implementation, parallel I/O and vector units are not employed.

At the time of this writing, the scalable linear approximation algorithm discussed in Section 4.2 was not implemented. We implemented an alternate approach which has two main steps: (1) Perform local approxi-

Total Execution Time (in seconds) on CM-5					
Machine Size (No. of PNs)	Image Size				
	128X128	256X256	512X512	1KX1K	2KX2K
32	0.097	0.347	1.001	3.526	14.958
64	0.065	0.239	0.537	1.853	7.602
256	0.053	0.170	0.086	0.599	2.010
512	0.052	0.179	0.123	0.374	1.118

Table 1: Execution times for extracting linear features on various sizes of image, using various partitions of CM-5. We let $\epsilon=2.0$ for performing linear approximation.

mation (2) Exchange boundary information. These two steps are repeated until there are no more contours to process. Indeed, Step (1) was performed until all the PNs had finished their local work to reduce frequent communications. The rationale for choosing this approach is as follows. Note that the convolution step takes up a major part of the total execution time in the serial implementation. Thus, a reasonably good non-optimal parallel algorithm for implementation of a linear approximation task probably will not lead to severe degradation of the overall performance. Note that, the communication time of our scalable algorithm is $(20P + 3 \log P)T_d + (24n + \frac{80n^2}{P})\tau_d$. The serial time for linear approximation on a $2K \times 2K$ image (the largest image we processed) is only 54 seconds. If $P = 512$ and we let $T_s = 60 \mu\text{sec}$ and $\tau_d = 0 \mu\text{sec}$, the communication time is $\approx 600 \text{ msec}$. The speed-up achievable is less than 100. For the values of P (≤ 512) and n (≤ 2048) considered here, the alternate approach seems to lead to a reasonably fast implementation.

In Table 1, we shows the total execution times for extracting linear features on various images using various partition sizes of CM-5. For the sake of comparison, the execution times of individual steps in contour detection and linear approximation procedures on Sun Sparc 400 are shown in Table 2. The times reported are the total CPU time measured by the CM-5 timer which has a resolution of 1 microsecond. The times reported on the serial machine are the total CPU time used by the user process. The resolution of the clock is 16.667 milliseconds. All the reported times do not consider I/O time. The largest image we have processed is of size 2048×2048 . The total execution time for processing such an image (including contour detection and linear approximation) is 1.118 seconds. The same image processed by a Sun Sparc 400 station operating at 32 MHz using a optimized code written in C takes more than 8 minutes.

Figure 5 depicts the speed-up curves for performing the linear features extraction task on various sizes of images. Speedup is calculated as the ratio of the execution time on Sparc 400 divided by the execution time on CM-5 for the same image size. For larger image sizes, linear speed-up is observed. We further analyze the effect of the granularity of image size on the execution time for performing the feature extraction task on various partition sizes of CM-5. Figure 6 depicts the efficiency curves for various grain sizes of images. The grain size is defined as the size of image block accessed by a PN which is $\frac{n^2}{P}$, if given an image of size $n \times n$ image and P PNs.

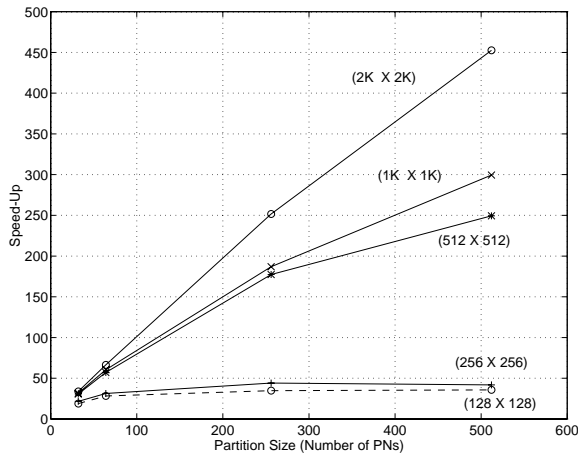


Figure 5: Speed-up curves for performing linear feature extraction on various sizes of images.

The efficiency is defined as the ratio of speed-up over P . We plot the efficiency curve versus $\log_2(\frac{n^2}{P})$. Because CM-5 is a coarse-grain machine, we observed that efficiency higher than 0.5 can be achieved if $\frac{n^2}{P} \geq 32^2$ (i.e. $\log_2 \frac{n^2}{P} = 10$) in our implementations.

The execution times of individual steps in the contour detection and linear approximation on various partition sizes of CM-5 are shown in Tables 3-7. Note that, in Table 7, if the number of PNs increases the time for linear approximation increases in some cases. This is due to the increased communication time (resulting from increased number of iterations). Even though the local work on each PN reduces, we observe that the time for communication is much larger than the time for local computation.

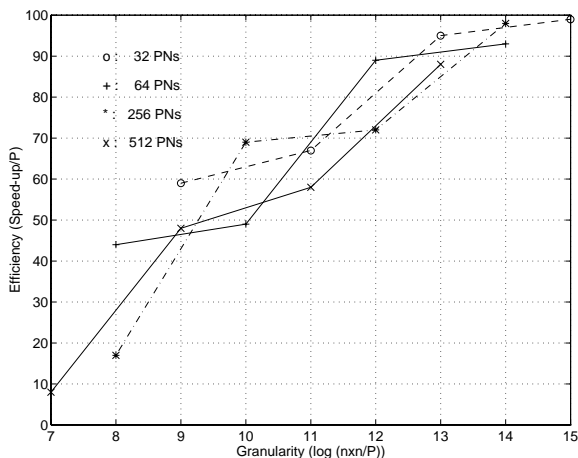


Figure 6: Efficiency versus granularity. The efficiency is defined as $\frac{Speedup}{P}$ and the grain size is $\frac{n^2}{P}$, where n^2 is the image size and P is the partition size of CM-5. The curve for efficiency versus $\log_2(\text{grain size})$ is plotted.

Figures 7-12 shows the raw images and the extracted line segments for different sizes of images on a partition of CM-5 having 32 PNs. These images are used by the ARPA vision community.

6 Summary

We have presented scalable algorithms for image feature extraction and fast parallel implementations of it. The experimental results are very encouraging.

We have been working on the design of scalable algorithms for various vision problems. In [14], we presented scalable parallel algorithms for perceptual grouping on CM-5. Perceptual grouping is a key step in vision to organize image data into structural hypotheses to be used for high-level analysis. We proposed data allocation and load balancing strategies which reduce the communication cost and evenly distribute the grouping operations among the PNs. Our implementations show that given a $1K \times 1K$ input image, extraction of line segments and several perceptual grouping steps can be performed in 5.0 seconds using a partition of CM-5 having 32 PNs. A serial implementation of these steps on a Sun Sparc 400 takes more than 2 minutes.

In [18], we have presented scalable algorithms for recognizing flat object using geometric hashing [6]. Object recognition is a key step in an integrated vision system. Recently, geometric hashing [6] has been proposed as an alternate approach for object recognition. However, parallel techniques are needed to perform geometric hashing in high-speed [3]. We have shown that using a 4M entries model database distributed over the entire processor array, a probe on a scene consisting of 256 feature points can be performed in less than 200 msec using a partition of CM-5 having 32 PNs. The earlier implementations in [16] takes 1.52 seconds on the same problem size.

Indeed, vision computations have significantly different characteristics compared with other grand challenge problems in scientific and numerical computations. We believe modeling features of parallel machines, designing data partitioning techniques, and mapping the computations to balance the load using explicit message passing is a feasible approach to solve vision problems on coarse-grain message-passing parallel machines.

References

- [1] J. Dunham, "Optimum Uniform Piecewise Linear Approximation of Planar Curves", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 8, No. 1, pages 67-75, 1986.
- [2] J. Jájá and K. Ryu, "The Block Distributed Memory Model for Shared Memory Multiprocessors," *Proc. of International Parallel Processing Symposium*, pages 752-756, 1994.
- [3] A. Khokhar, Scalable Data Parallel Algorithms and Implementations for Object Recognition, Ph.D. Thesis, Department of EE-Systems, University of Southern California, April 1993.
- [4] V. Kumar, A. Grama, A. Gupta, and G. Karypis, *Introduction to Parallel Computing: De-*

sign and Analysis of Parallel Algorithms, Benjamin/Cummings, 1994.

- [5] T. Kwan, B. Totty, and D. Reed, "Communication and Computation Performance of the CM-5," *Proc. of Supercomputing '93*, pages 192-201, 1993.
- [6] Y. Lamdan and H. Wolfson, "Geometric Hashing: A General and Efficient Model Based Recognition Scheme," *International Conference on Computer Vision*, pages 218-249, 1988.
- [7] F. Leighton, "Tight Bounds on the Complexity of Parallel Sorting," *IEEE Trans. on Computers*, Vol. 34, No. 4, pages 344-354, 1985.
- [8] C. Leiserson, "FAT-TREES : Universal Networks for Hardware Efficient Supercomputing", *IEEE Transactions on Computers*, Vol. 34, No. 10, pages 892-901, 1985.
- [9] C. Leiserson et.al., "The Network Architecture of the Connection Machine CM-5", Technical Report, Thinking Machines Corporation, 1992.
- [10] C. Lin and V. Prasanna, "Analysis of Communication Costs of Parallel Machines with Various Communication Mechanisms," Manuscript, *Department of EE-Systems, University of Southern California*, July 994.
- [11] J. Marberg and E. Gafni, "Sorting in Constant Number of Row and Column Phases on a Mesh," *Algorithmica*, Vol. 3, pages 561-572, 1988.
- [12] R. Nevatia and K. Babu, "Linear Feature Extraction and Description", *Computer Graphics and Image processing*, Vol. 13, pages 257-269, 1980.
- [13] V. Prasanna, C. Wang, and A. Khokhar, "Low Level Vision Processing on Connection Machine CM-5," *Workshop on Computer Architectures for Machine Perception*, pages 117-126, 1993.
- [14] V. Prasanna and C. Wang, "Scalable Parallel Implementations of Perceptual Grouping on Connection Machine CM-5," to appear in *International Conference on Pattern Recognition*, 1994.
- [15] J. Roberge, "A Data Reduction Algorithm for Planar Curves," *Computer Vision, Graphics, and Image Processing*, Vol. 29, pages 168-195, 1985.
- [16] I. Rogoutsos and R. Hummel, "Massively Parallel Model Matching: Geometric Hashing on the Connection Machine," *IEEE Computer*, pages 33-42, 1992.
- [17] Thinking Machines Corporation, CMMD Reference Guide Version 3.0, 1992.
- [18] C. Wang, V. K. Prasanna, H. Kim, and A. Khokhar, "Scalable Data Parallel Implementations of Object Recognition using Geometric Hashing," *Journal of Parallel and Distributed Computing*, pp. 96-109, March 1994.
- [19] Cho-Li Wang, *Scalable Parallel Algorithms and Implementations for Integrated Vision System*, Ph.D. Thesis, in preparation, Department of EE-Systems, University of Southern California.

Execution Time (in seconds) on Sun Sparc 400					
Processing Steps	Image Sizes				
	128X128	256X256	512X512	1KX1K	2KX2K
Convolution	1.23	4.90	19.73	80	314
Thinning	0.18	0.72	2.95	7.87	47
Linking	0.26	1.18	5.05	14.65	91
Approximation	0.18	0.70	2.95	8.41	54
Total Time	1.86	7.50	30.68	112	506

Table 2: Execution times on various image sizes using Sun Sparc 400 operating at 33MHz having 64 Mbytes of main memory. The code was written in C and optimized using an optimizing compiler. We let $e=2.0$ for performing linear approximation.

Convolution					
Machine Size (No. of PNs)	Image Size				
	128X128	256X256	512X512	1KX1K	2KX2K
32	38.03	145	605	2338	9479
64	18.24	73.13	299	1175	4751
256	4.75	18.48	74.08	293	1195
512	2.46	9.45	37.17	144	608

Table 3: Execution times (in msec) for the convolution step.

Thinning					
Machine Size (No. of PNs)	Image Size				
	128X128	256X256	512X512	1KX1K	2KX2K
32	5.23	19.30	78.07	295.87	1300
64	2.57	10.22	39.57	148.48	648
256	0.75	2.57	10.58	37.94	161
512	0.43	1.35	5.51	19.20	82

Table 4: Execution times (in msec) for the thinning step.

Linking					
Machine Size (No. of PNs)	Image Size				
	128X128	256X256	512X512	1KX1K	2KX2K
32	9.07	31.73	156.91	432.29	2366
64	5.06	16.60	83.14	230.99	1226
256	1.67	5.35	24.86	60.66	315
512	0.97	2.81	12.81	30.11	162

Table 5: Execution times (in msec) for the linking step.

Boundary Padding					
Machine Size (No. of PNs)	Image Size				
	128X128	256X256	512X512	1KX1K	2KX2K
32	15.17	19.76	29.80	48.91	93.35
64	14.79	15.58	24.87	38.53	68.93
256	13.42	16.96	18.09	25.23	39.21
512	13.43	14.23	16.52	21.23	30.70

Table 6: Execution times (in msec) for boundary communication during the window operations in convolution, thinning, and linking steps.

Linear Approximation					
Machine Size (No. of PNs)	Image Size				
	128X128	256X256	512X512	1KX1K	2KX2K
32	30	131	133	407	1721
64	25	131	90	260	905
256	44	128	45	181	297
512	35	151	47	159	234

Table 7: Execution times (in msec) for the linear approximation step with $e = 2.0$.



Figure 7: A 128 by 128 airport image.



Figure 10: Extracted line segments from contours of length ≥ 6 with $e=3.0$, using a CM-5 partition of 32 PNs. Execution time ≈ 96 msec



Figure 8: A 256 by 256 eye image.



Figure 11: Extracted line segments from contours of length ≥ 20 with $e=5.0$, using a CM-5 partition of 32 PNs. Execution time ≈ 347 msec



Figure 9: A 512 by 512 building image.

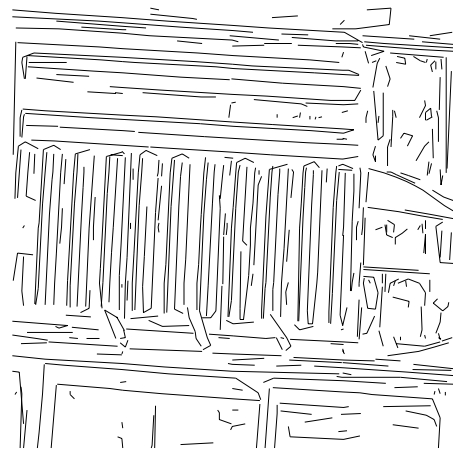


Figure 12: Extracted line segments from contours of length ≥ 30 with $e=5.0$, using a CM-5 partition of 32 PNs. Execution time ≈ 974 msec