Load Balancing and Task Decomposition Techniques for Parallel Implementation of Integrated Vision Systems Algorithms

Alok N. Choudhary and Janak H. Patel

Coordinated Science Laboratory
University of Illinois
1101 W. Springfield
Urbana, IL 61801

Abstract

Integrated vision systems employ a sequence of image understanding algorithms in which the output of an algorithm is the input of the next algorithm in the sequence. Algorithms that constitute an integrated vision systems exhibit different characteristics, and therefore, require different data decomposition techniques and efficient load balancing techniques for parallel implementation. However, since input data of a task is produced as output of the previous task, this information can be exploited to perform knowledge based data decomposition and load balancing. This paper presents several techniques to perform static and dynamic load balancing schemes for integrated vision systems. These techniques are novel in the sense that they capture the computational requirements of a task by examining the data when it is produced. Furthermore, they can be applied to many integrated vision systems because many algorithms in different systems are either same or have similar computational characteristics. These techniques are evaluated by applying them to the algorithms in a motion estimation system. It is shown that the performance gains when these techniques are used are significant and the overhead of using these techniques is minimal. The performance is evaluated by implementing the algorithms using the presented techniques on a hypercube multiprocessor system.

1. Introduction

Computer vision and image understanding tasks employ a broad range of algorithms. In an integrated systems environment many algorithms with different characteristics and varying computational requirements are used in a sequence where output of one algorithm becomes the input of the next algorithm in the sequence [1]. An example of such a system is motion estimation systems. In this system sequence of images are used to compute motion parameters of a moving object in the scene. Figure 1 shows the computational flow for motion estimation system in which stereo images (*Lim,Rim*) at each time frame are used as the input to the system. Briefly, the involved tasks in this system are as follows. The first algorithm is computation of zero crossings of the images (edge detection (*Lzc* and *Rzc*)). Zero crossings are used to perform stereo matching between the two images of the same time frame. Stereo match algorithm provides points to compute 3-D information about the object in the scene. Using these matched points (*Lsm* and *Rsm*), corresponding points in the image from the next time frame (*Ltm*) are located and this task is performed by time match algorithm. Stereo matching is again used to obtain the corresponding 3-D points in the next image frame. These two sets of points provide information to compute the motion parameters.

Computational requirements for such integrated vision systems are tremendous [2]. Not only does such a system require powerful parallel processing capabilities but other issues such as task scheduling, load balancing and efficient utilization of processors become important.

In this paper we present techniques to perform efficient data decomposition and load balancing for integrated vision systems for medium to large grain parallelism. Two important characteristics of these techniques are that they are general enough to apply to any such integrated system, and that they use statistics and knowledge from the execution of a task to perform load balancing and scheduling for the next task in the system. For example, in the motion estimation system sufficient knowledge can be obtained about the output data from the zero crossing computation step to perform efficient data decomposition and load balancing for the stereo match step. Knowledge from each step is used to perform load balancing in the next step. Advantages of such schemes are as follows. First, these techniques use characteristics of tasks and data, and therefore, work well no matter how data changes. Second, many integrated vision systems consist of such tasks and exhibit the above described computation flow, and hence, these techniques can be used in any system (e.g., object recognition, optical flow etc.) [3]. Also, we show that these schemes perform better than completely dynamic load balancing schemes as the number of processors increases because the overhead of dynamic load balancing grows with the number of processors.

This paper is organized as follows. In Section 2 we present the algorithms for each step in the motion estimation system [4]. These algorithms will provide insight into the involved computations in the above system, any other such system, and provide a framework for the discussion in the following sections. Section 3 describes the proposed load balancing and data decomposition techniques. In Section 4 we present parallel implementation of these algorithms in an integrated environment, discuss the performance results for each of these algorithms, and that of data decomposition and load balancing schemes. The multiprocessor machine on which we have implemented these algorithms is intel iPSC/2 hypercube. Some of these techniques have been applied to other integrated vision systems and have been shown to work well [5]. Finally, summary and conclusions are presented.

2. Motion Estimation Algorithms

This section describes steps in the motion estimation system.

This research was supported in part by National Aeronautics and Space Administration Under Contract NASA NAG-1-613.

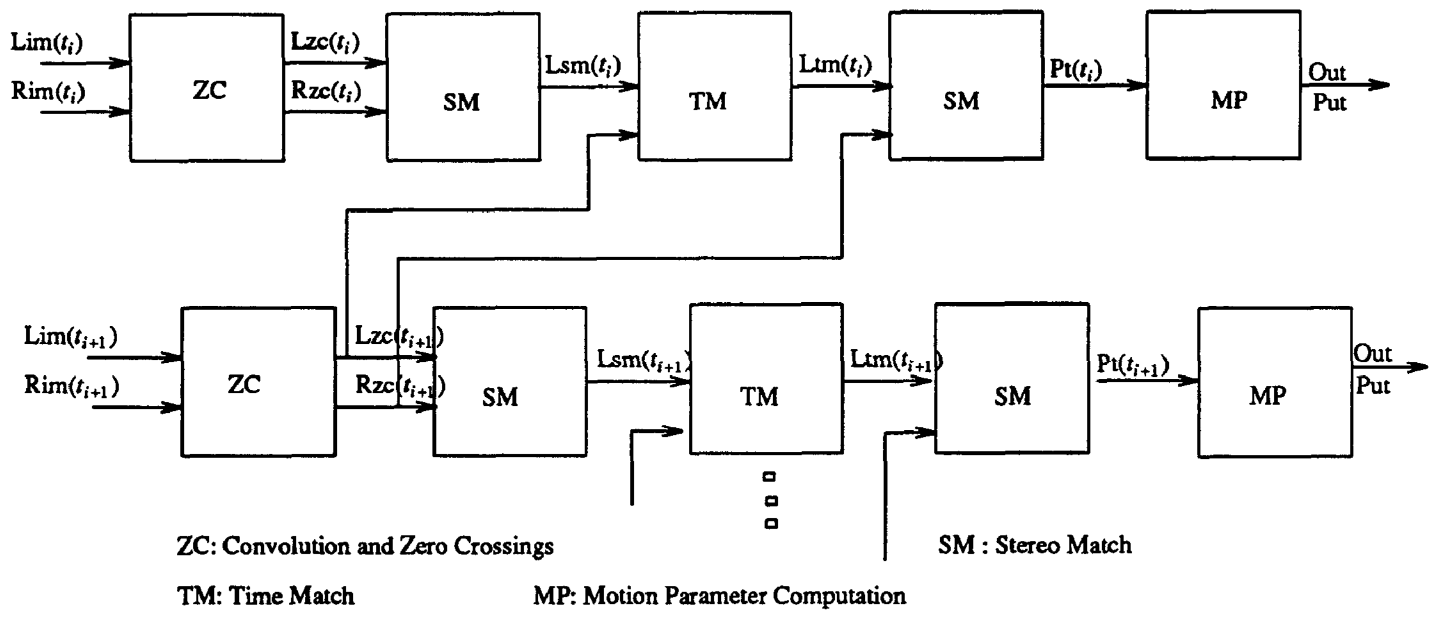

Figure 1 : Computation Flow for Motion Estimation

A detailed description of the involved computations is included in order to understand the characteristics of such algorithms. A detailed description of the involved computations is included in order to understand the characteristics of such algorithms. The motion estimation algorithm consists of two processes. The first process is feature points extraction. Since the feature points used in our algorithm are edge points, we can extract them by locating the zero crossings of an image. The second process is matching and has three sub-processes which are i) stereo matching, ii) time matching and iii) elimination of multiple matches. The basic evidences exploited in these sub-processes to obtain unambiguous matched point pairs are the normalized correlation coefficient and the zero crossing patterns [2].

2.1. Feature Points

The feature points used in this algorithm are zero crossing points of an image which are computed using Laplacian-Gaussian masks [6]. In order to eliminate non-significant zero crossing points and maintain enough details, we threshold the zero crossing image based on the intensity gradient at each zero crossing point. Figure 2(b) depicts the thresholded zero crossing images of the pictures shown in Figure 2(a).

Each zero crossing point is associated with one of the sixteen possible zero crossing patterns as suggested in [7]. The similarity between any two zero crossing points is based on the directional difference of their zero crossing patterns. The directional difference between any two direction values (e.g. D_1 and D_2) is calculated as follows [2]:

$$DIFF = | D_1 - D_2 |$$
$$\text{if } (DIFF > 4), DIFF = | 8 - DIFF |$$

In the matching process, the use of directional difference (or zero crossing pattern values) in finding matched point pairs is through the expression of directional difference weight as shown below :

$$w_{ddif} = \frac{1}{1 + DIFF} \qquad (1)$$

2.2. Matching

Once zero crossings are extracted in all the involved images, the matching process is applied to find point correspondences among the images (two stereo image pairs at two consecutive time instants, i. e. t_{i-1} and t_i). The evidences used in this process to obtain matched point pairs are the normalized correlation coefficient and the directional difference weight as mentioned above; furthermore, in order to limit the search space, the heuristic of limited displacement or disparity between frames is exploited. The matching processes in motion estimation consists of six steps as follows :

1) Perform stereo (from left to right) matching in the t_{i-1} stereo image pair.
2) Obtain unambiguous matched point pairs by eliminating multiple matches.
3) Perform time matching between the unambiguous matched points in the left t_{i-1} image and the feature points of the left t_i image.
4) Obtain unambiguous matched point pairs from the time matched points by eliminating multiple time matches.
5) Perform stereo matching between the unambiguous matched points (obtained in step (4)) in the left t_i image and the feature points of the right t_i image.
6) Obtain unambiguous matched point pairs from the results of t_i stereo matching by eliminating multiple matches.

The results of the above steps are two sets of unambiguous stereo matched point pairs at time instant t_{i-1} and t_i. These two sets are related through step (3) and step (4), the matching over time; therefore, we can pick out all the unambiguous matched points that correspond to each other among the two stereo image pairs at time instants t_{i-1} and t_i.

2.2.1. Stereo Matching

This is the sub-process to obtain the matched point in the right image for each matchable zero crossing point in the corresponding left image of the same stereo pair.

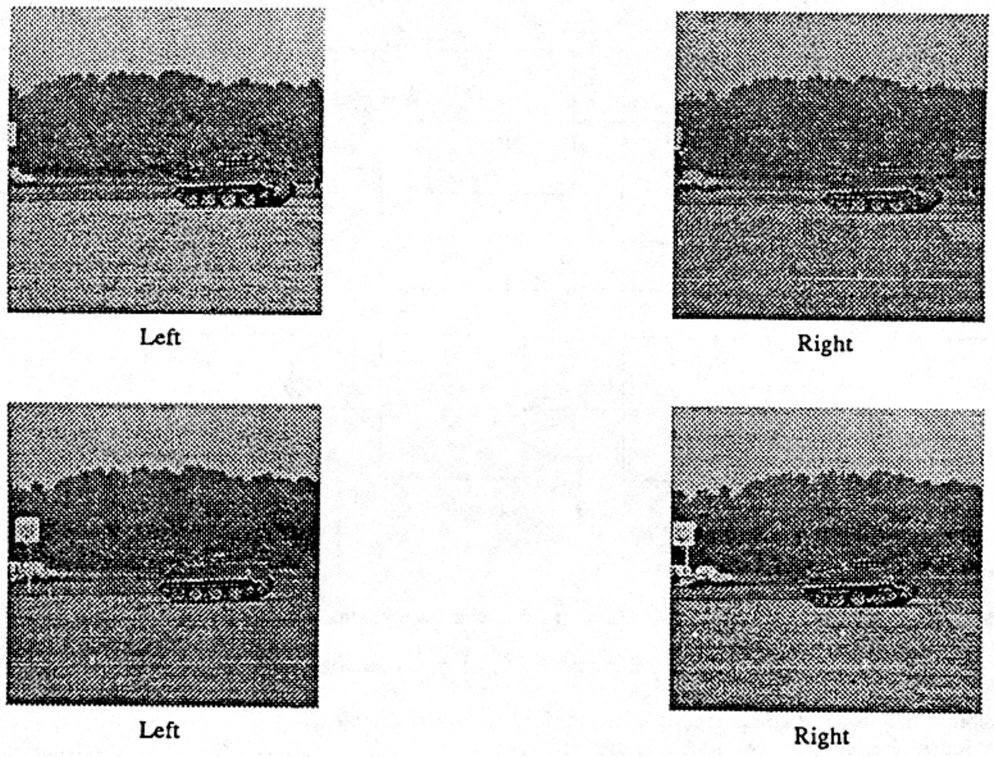

Figure 2(a) : Stereo Image Pairs at t_3 and t_4

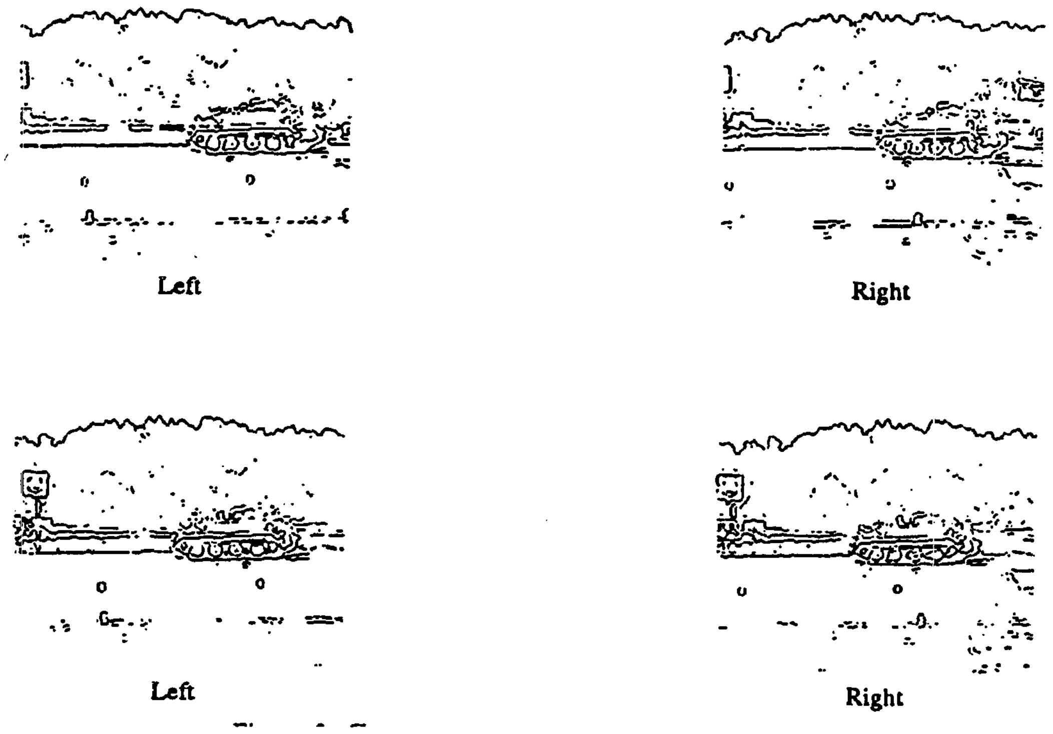

Figure 2(b) : Zero Crossings of the Images in Figure 2(a)

Since the imaging setup is the parallel axis method, we can exploit the epipolar line constraint in solving the stereo matching problem. As the result, we have 1-dimensional search space instead of 2-dimensional search space in the stereo matching process. A typical search space in the right image for a matchable zero crossing point in the left image is on the left side of the transferred location of that particular left image zero crossing point; however, by using the heuristic of limited disparity between frames, the search space is limited to d_{max} (the maximum possible disparity).

Let S_{rl} be the set of all non-horizontal zero crossing points in the right image within the search space of a zero crossing point in the left image. The stereo matching process is as follows :

For each point in S_{rl},

i) Calculate the normalized correlation coefficient with a template size of $s \times s$ between the grey level images of left and right at the corresponding locations. The normalized correlation coefficient is calculated by using the following expression :

$$\rho_s = \frac{\sum_i\sum_j (lx_{ij} - \overline{lx})\,(rx_{ij} - \overline{rx})}{\sqrt{\sum_i\sum_j (lx_{ij} - \overline{lx})^2}\ \sqrt{\sum_i\sum_j (rx_{ij} - \overline{rx})^2}} \quad (2)$$

where

lx_{ij} : value at point (i,j) in the left image; rx_{ij} : value at point (i,j) in the right image; $\overline{lx}$: mean value of the template in the left image; $\overline{rx}$: mean value of the template in the right image.

ii) If the normalized correlation value ρ_s is less a threshold value $thrsh_{\rho_s}$, we discarded that particular point in the search space in the remaining steps.

iii) Calculate the directional difference weight between the left and the right zero crossing point (within the search space) according to equation (1) $w_{ddif(stereo)}$.

iv) Obtain the total weight as the combination of the correlation coefficient and the directional difference weight.

$$w_s = a \times \rho_s + b \times w_{ddif(stereo)};\ a + b = 1. \quad (3)$$

v) Among all elements of S_{rl}, the point with the maximum total weight w_s is considered as the matched point for the corresponding zero crossing point in the left image.

2.2.2. Time Matching

This is the sub-process to obtain the matched point in the left t_i image for each candidate zero crossing point in the corresponding left t_{i-1} image. Similar to the stereo matching process, we exploit the heuristic of limited displacement (instead of disparity) between frames in solving the time matching problem. We assume that the total motion between the t_{i-1} and t_i frames is within f pixels in vertical direction and h pixels (from right to left) in horizontal direction. Hence, the search space for each candidate zero crossing point in the left t_{i-1} image is a window of size $f \times h$ pixels on the left side of its transferred location in the left t_i image. Any zero crossing point (except horizontal ones) inside this window is a potential match point for the corresponding candidate zero crossing point in the left t_{i-1} image. The time matching process is similar to the stereo matching process and is listed as follows :

For each non-horizontal zero crossing point in the left t_i image within the search space of a zero crossing point in the left t_{i-1} image,

i) Calculate the normalized correlation coefficient with a template size of $t \times t$ between the grey level image of t_{i-1} and t_i at the corresponding locations. The normalized correlation coefficient is calculated by using the following expression :

$$\rho_t = \frac{\sum_i\sum_j (x_{ij} - \overline{x})\,(y_{ij} - \overline{y})}{\sqrt{\sum_i\sum_j (x_{ij} - \overline{x})^2}\ \sqrt{\sum_i\sum_j (y_{ij} - \overline{y})^2}} \quad (4)$$

where

x_{ij} : value at point (i,j) in the t_{i-1} image; y_{ij} : value at point (i,j) in the t_i image; $\overline{x}$: mean value of the template in the t_{i-1} image; $\overline{y}$: mean value of the template in the t_i image.

ii) If the normalized correlation coefficient ρ_t is less than a threshold value $thrshd_{\rho_t}$, we discard that particular point in the remaining steps.

iii) Calculate the directional difference weight between the left t_{i-1} and the left t_i zero crossing point (within the search space) according to equation (1) $w_{ddif(time)}$.

iv) Obtain the total weight as the combination of the correlation coefficient and the directional difference weight.

$$w_t = c \times \rho_t + d \times w_{ddif(time)};\ c + d = 1. \quad (5)$$

v) Within a search window in the left t_i image, the zero crossing point with the maximum total weight w_t value is considered as the match point for the corresponding zero crossing point in the left t_{i-1} image.

2.2.3. Elimination of Multiple Matches

After the sub-process of either stereo matching or time matching, there may be multiple matches for some zero crossing points in either the left image or the left t_{i-1} image. We use the same procedure in eliminating both types (stereo or time) of multiple matches except different sizes of the search window are exploited. For stereo matching, we use a 1-D search window of size d_{max} on the left side of a multiple matched point; on the other hand, for time matching, we use a search window of size $2f \times h$ on the left side of a multiple matched point. The remaining steps of the procedure for determining unambiguous matched points are listed as following :

i) At the position of a multiple match in either the right image for stereo matching or the left t_i image for time matching, open a search window either with a size of d_{max} or with a size of $2f \times h$ respectively.

ii) Within the search window, locate all the positions that have the same (multiple) match.

iii) If all the positions are within the neighborhood region r_{neigh}, calculate the total weight (w_s or w_t) according to equation (3) or (5) (depending on whether we try to eliminate stereo multiple matches or time multiple matches). The position with the highest value in its total weight is regarded as the correct match.

iv) If one or more positions are outside the neighborhood region r_{neigh}, we use the disambiguation procedure described in Section 2.2.3.1 to resolve the multiple matches.

v) If multiple matches still exist after the application of the above steps, they all are discarded from the match set.

2.2.3.1. Disambiguation

We only use this procedure, if step (iv) in the above discussion is true. In this procedure, the neighboring unambiguous matched points around a multiple matched point are used as one of the supporting evidences in determining the correct match. The other evidences used are the normalized correlation coefficient (*ncc*) and the directional difference weight (*ddw*). The steps are listed as follows :

i) At each position, calculate *ncc* (ρ_s for stereo matching or ρ_t for time matching) and the *ddw* ($w_{ddif(stereo)}$ for stereo matching or $w_{ddif(time)}$ for time matching).

ii) Assign a correlation coefficient rank, R_{cc}, and zero crossing pattern rank, R_{zcp}, to each position according to its normalized correlation coefficient (ρ_s or ρ_t) and *ddw* ($w_{ddif(stereo)}$ or $w_{ddif(time)}$). The position with the highest value in *ncc* or *ddw* has the highest rank in R_{cc} or R_{zcp}.

iii) At each position, check for unambiguous matched neighbors. If it has two attached unambiguous matched neighbors, a neighbor weight $neigh_{wt}$ of 3 is assigned. On the other hand, if it has only one attached unambiguous matched neighbor, a neighbor weight $neigh_{wt}$ of 2 is assigned.

iv) At each position, open a check window of size 5×5 but excluding the center 3×3 region and count the number of unambiguous matched points, num_{smp}.

v) At each position, calculate the total possibility as the sum of the ranks, the weight and the number.

$$poss_{tot} = R_{cc} + R_{zpw} + neigh_{wt} + num_{smp} \quad (6)$$

vi) The position with the highest $poss_{tot}$ value is considered as the correct match.

Figure 3 shows the unambiguous point of the final result after applying all the algorithms.

3. Data Decomposition and Load Balancing

In a multiprocessor system an obvious and simplest method to implement a task in parallel is to decompose the data and and the underlying computations equally among the processors. In a completely deterministic computation where computation is independent of the input data such schemes perform well and normally the processing time is comparable on all the processors. That is, efficient utilization and load balancing can be obtained. For example, regular algorithms such as convolutions, filtering or FFT exhibit such properties. The amount of computation to obtain each output point is the same across all input data. Therefore, uniform decomposition of data results in load balanced implementation.

Most other algorithms do not exhibit the regular structure because computation is data dependent. Furthermore, computation is not uniformly distributed across the input domain. In such cases, a simple decomposition does not provide efficient mapping and results in poor utilization and low speedups. Also, the performance cannot be predicted for a given number of processors and data size because computation varies as type of data and its distribution varies. For example, in the stereo match algorithm, computation is more where feature points are dense and is comparatively small where number of features is small and sparsely distributed (Figure 2). Hence uniformly partitioning the input data among processors is not expected to provide good speedups and utilization.

In an integrated vision system, it is important to efficiently allocate resources and perform load balancing at each step to obtain any significant performance gains overall. An important characteristic of such systems is that input data of a task is the output of the previous task. Therefore, while computing the output in the previous task enough knowledge about data can be obtained to perform efficient scheduling and load balancing. In the following we discuss such techniques and in the next section we present performance results for these techniques using algorithms in the motion estimation system.

Consider a parallel implementation of a task on n processor parallel machine. Let T_i ($1 \le i \le n$) denote the computation time at processor node i. Then overall computation time for a task is given by

$$T_{max} = \max\{T_1, \ldots, T_n\} \quad (7)$$

The total wasted time (or idle time) T_w is given by

$$T_w = \sum_{i=1}^{i=n} (T_{max} - T_i) \quad (8)$$

If $T_{max} = T_i$ for all i, $1 \le i \le n$, then the task will be completely load balanced. Another measure of imbalance is given by the variation ratio V,

$$V = \frac{T_{max}}{T_{min}}, \quad T_{min} = \min\{T_1, \ldots, T_n\} \quad (9)$$

The goal in performing load balancing is to minimize T_w, or move V as close to 1 as possible. In the best case, $T_w = 0$ or $V = 1$.

If T_{seq} is the time to execute the same task on a sequential machine then the speedup is given by

$$S_p = \frac{T_{seq}}{T_{max}} \quad (10)$$

3.1. Uniform Partitioning

Data decomposition using uniform partitioning performs well as a load balancing strategy for input data independent tasks because equally dividing the data distributes the computation equally. If the total input data size is D then the total computation time to execute the task is $T = k \times D$, where k is determined by computation at each input data point. For example, in convolution of an image with $m \times m$ kernel, $k = 2 \times m^2$ floating point operations. Hence, for an n node multiprocessor the data decomposition methods to balance the computation is to make the granule size to

$$d_i = \frac{D}{n} \quad (11)$$

3.2. Static

When computation is not uniformly distributed across input domain and is data dependent, uniform partitioning does not work well for load balancing. Normally, the computation depends on significant data or type of data in a partition. Many image processing and vision algorithms exhibit this behavior. For example, in stereo match, hough transform etc., the computation is proportional to number of features (edges) or significant pixels in a granule rather than on granule size. Therefore, equal size granules do not guarantee load balanced partitioning. In fact, the variation can be very significant as we shall observe in the next section when we discuss the performance. In many such algorithms, the computation time for a granule (i), denoted as T_i, is proportional to a certain extent on the granule size (fixed overhead to process a granule) and to the number of significant data in a granule. That is,

$$T_i = A \times d_i + B \times f_i \quad (12)$$

where, d_i is granule size, f_i is a measure of significant data in granule (i), and A and B are arbitrary constants which depend on the algorithm. Therefore, the objective is to divide the computation equally among the processors such that each processor receives equal measure of computation.

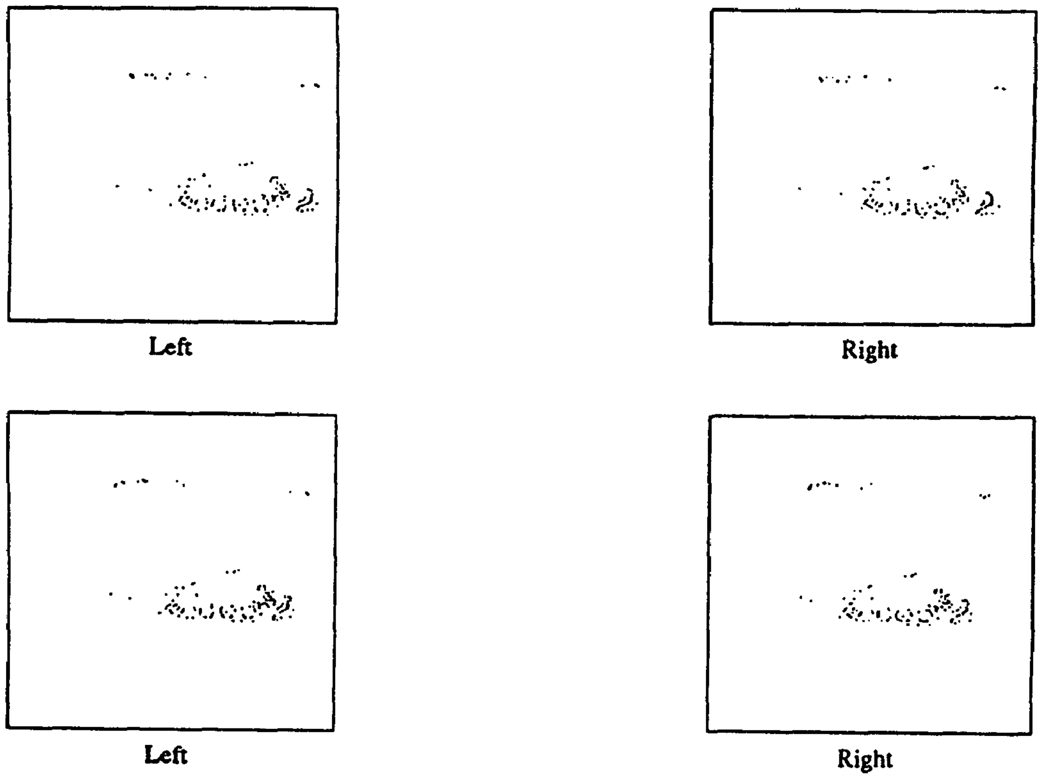

Figure 3 : Unambiguous Matched Points of the Images in Figure 2

One way to assign a granule to a processor is to compute the total measure of computation and partition as follows:

$$T_i = \frac{\sum_{i=1}^{i=g} A \times d_i + B \times f_i}{n}$$

where, g is the total number of granules in the input domain (Note that the number of granules for the current task is n for n processor system). For example, consider computing hough transform of an edge image. The algorithm involves computing the parameters for line segments in an image. If there exists a line whose normal distance from the origin is r, the normal makes an angle θ with the x-axis then if the point (x,y) lies on that line, the following equation is satisfied.

$$r = x\cos\theta + y\sin\theta$$

r and θ are quantized for desired accuracy and for each significant pixel (where there exists an edge), r is computed for all quantized θ values. If two partitions of equal size contain different number of edge pixels then the amount of computation will be different for the two partitions despite them being equal in the size. In fact, the computation is directly proportional to the number of edge pixels in a partition. One way to perform static load balancing is to decompose the input data such that each partition contains equal number of edge pixels. The computation to recognize this partioning can be performed within the task in which edges are detected by keeping a count of the number of edges detected by a processor. Note that it is important to compute the statistics at run time when edges are detected to guarantee low overhead. If the same statics are gathered by sequentially scanning the input data, the overhead can be significant. Once a task is completed, data can be reorganized such that number of edges with each processor is in the interval $(\frac{Z_a}{n} - \delta, \frac{Z_a}{n} + \delta)$, where Z_a is total number of edges detected and δ is determined by the minimum granule size from fixed overhead considerations.

3.3. Weighted Static

When computation in a granule not only depends on the number of significant data points in the input domain but also depends on their spatial relationships then data distribution also needs to be taken into account as a measure of computation to perform load balancing. For example, from the previous section it is evident that in stereo matching, not only does the computation depend on the number of zero crossings but it also depends on their spatial distribution. If the zero crossings are densely spaced then the computation will be more than that if the same number of zero crossings are sparsely distributed (Refer to Figure 2(b)). The reason being that if zero crossings are densely packed, more number of zero crossings need to be matched with each corresponding zero crossing in the other image; where as less number of zero crossings need to be matched if they are sparsely distributed. Hence, the computation also depends on the spatial density (such as features/row if one dimensional matching is performed). That is,

$$T_i = A \times d_i + B \times w_i \times d_i$$

where, w_i is the feature dependent spatial density. For example, if the minimum granule size is a row of the input data then $w_i = r_i^{\beta}$, where r_i is the number of features in row i and β is a parameter, $0 \le \beta \le 1$. $\beta = 0$ means that the computation is independent of how features are distributed within a row. Therefore, to divide the computation equally among n processors, the following heuristic can be used.

$$T_i = \frac{\sum_{i=0}^{i=R} A \times d_i + B \times w_i \times d_i}{n}$$

where, R is the number of rows in the image. Note that the above heuristics approximate the load and do not exactly divide the computation among processors. However, in the next section we will show that these schemes perform well.

As an example, consider the stereo match computation. While partitioning the data among processors a weight can be assigned to each row as a function of number of features in the row. This weight represents the feature density. Note that using a row as the smallest granule avoids the communication overhead because search space for stereo matching is one dimensional, and therefore, if the granule boundary is one row then there is no need for communication.

3.4. Dynamic

The above three methods use knowledge about data when it is produced to perform load balancing for the next task. However, once decomposition is done, the data is not reshuffled. Therefore, we consider the above methods as knowledged based static load balancing schemes. In dynamic scheme, however, data is decomposed into finer granules such that the number of tasks (that is number of independent granules) M is much larger than the number of processors.

At execution time, processors are assigned these tasks dynamically by a designated scheduler from a task queue which contains these tasks. Processors are assigned new tasks as they finish their assigned tasks and if there are more tasks left to be assigned. However, knowledge obtained from the previous step again can be used to anticipate completion of the existing task and assign a new task without delay. That is, tasks can be pipelined, and therefore, overhead of dynamic load balancing can be reduced. Communication overhead of dynamically assigning tasks is not incurred in the previous three schemes.

4. Parallel Implementation and Performance Evaluation

In this section we present parallel implementation of the algorithms that are part of motion estimation system and describe performance of the algorithms and load balancing strategies.

The algorithms were implemented and evaluated on a hypercube multiprocessor. A typical commercially available hypercube multiprocessor system consists of a host processor and node processors. The host processor serves as the cube manager, provides interface with the external environment, provides input-output of data and program. We used Intel ipsc/2 hypercube multiprocessor consisting of 16 nodes. Each node consists of an Intel 80386 processor, Intel 80387 co-processor, 4 megabyte memory and a communication module.

4.1. Feature Extraction

Features used for stereo matching in our algorithms are the zero crossings of the convolution of the image with laplacian. Zero crossing computation involves 2-D convolution and template matching. Since convolution is a data independent algorithm uniform partitioning is sufficient to evenly distribute the computation. The mapping is a division of $N \times N$ image onto P processors. Each processor computes zero crossings of share of N^2/P pixels (eqn. 11). Data division onto the processors is done along the rows. This mapping reduces communication to only in one direction. The reason is as follows. 2-D convolution can be broken into two 1-D convolution [6]. This not only reduces the computation from W^2 sum of products operations per pixel to $2 \times W$ sum of product operations per pixel (W is the convolution kernel size) but also reduces the communication requirements in a parallel implementation if the data partitioning is done along the rows. There is no need for communication when convolution is performed along the rows.

Table 1 shows the performance results for the above implementation for an image of size 256×256 and convolution window of size 20×20. First column shows the number of processors in the cube(P). Second column represents the total processing time (t_{proc}) for convolution. Column 3 shows the number of bytes communicated by a processor to the neighboring processor and column 4 shows the corresponding communication time which is small compared to the computation time. The second half of the table shows computation time for extracting zero crossings from the convolved image. The corresponding speedups are also shown.

We observe that almost linear speedup is obtained for convolution. Two factors which contribute towards this result are that the communication overhead is relatively small and it is constant as the number of processors increases. However, the speedup obtained in the elapsed time (which includes program and data load time also) is sub-linear because the hypercube multiprocessor's host does not have a broadcast capability, and therefore, overhead of loading a program increases linearly with the number of processors. But data load time increment with increase in number of processors is comparatively small because amount of data to be loaded to one processor decreases as the number of processors increases. Only increment in data load time results from the number of communication setups from the host to the node processors which increases linearly with the number of processors.

4.2. Stereo Match

This task involves matching features in stereo pair of images. As discussed in section 2, the epipolar constraint limits search for a match in the corresponding image to only in horizontal direction, i.e., along rows in the zero crossings of an image. Thus data partioning along the rows for parallel implementation results in no communication between node processors as long as each partition contains an integral number of rows.

The computation involved in stereo matching algorithm is data dependent and varies across the image because it depends on the number of zero crossings, distribution of zero crossing across the image and distribution of zero crossings along the epipolar lines. Therefore, partioning data uniformly among processors (i.e. assign each processor equal number of rows) may not yield expected speedups and processor utilization. A processor which has very few zero crossings and sparsely distributed zero crossings will be under utilized where as a processor with large number of zero crossings and densely distributed zero crossings will become a bottleneck and this imbalance of load will result in poor performance. We used uniform partitioning, static load balancing, weighted static and dynamic load balancing schemes to decompose computation on the multiprocessor.

Static load balancing can be achieved by keeping a count of the zero crossings with each processor when the previous task (convolution and feature extraction) is executed. At the completion of a task, data is reorganized using this information and the techniques described in the previous section.

Figure 4 shows the distribution of the computation times for 8 processor case. The X-axis shows the processor number and the Y-axis shows the computation time for each scheme. As we can observe, uniform partitioning does not perform well at all because the variation in computation time is tremendous, and therefore, performance gains are minimal. Static load balancing scheme (shown as dashed bars) performs much better than uniform partitioning but variation in computation times is still significant because the computation also depends on the distribution of zero crossings. The Weighted static scheme does perform better than static and further reduces the variation in computation times. Note that these schemes only measure the load approximately, and therefore, will not divide the computation exactly uniformly. Furthermore, minimum granularity is a row boundary in order to avoid communication between processors.

Table 1 : Performance for feature Extraction (Zero Crossings)

Computation for Convolution and Zero Crossings							
	Convolution Window Size = 20x20						
No. Proc.	Conv. Comp. Time(sec.)	Conv. Comm. Bytes	Conv. Comm. Time(ms.)	Conv. Total Time(sec.)	Conv. Speed Up	ZC Comp. Time(sec.)	ZC Speed Up
1	109.0	0	0	109.0	1	6.47	1
2	54.76	2816	13	54.78	1.98	3.23	1.99
4	27.51	5632	36	27.55	3.95	1.66	3.89
8	13.88	5632	36	13.92	7.83	0.85	7.60
16	7.07	5632	36	7.11	15.33	0.42	15.25

Feature Extraction Performance (Elapsed Time)		
No. Proc.	Elapsed Time(sec.)	Speed up
1	116.2	1
2	58.8	1.97
4	30.1	3.86
8	16.1	7.22
16	9.6	12.1

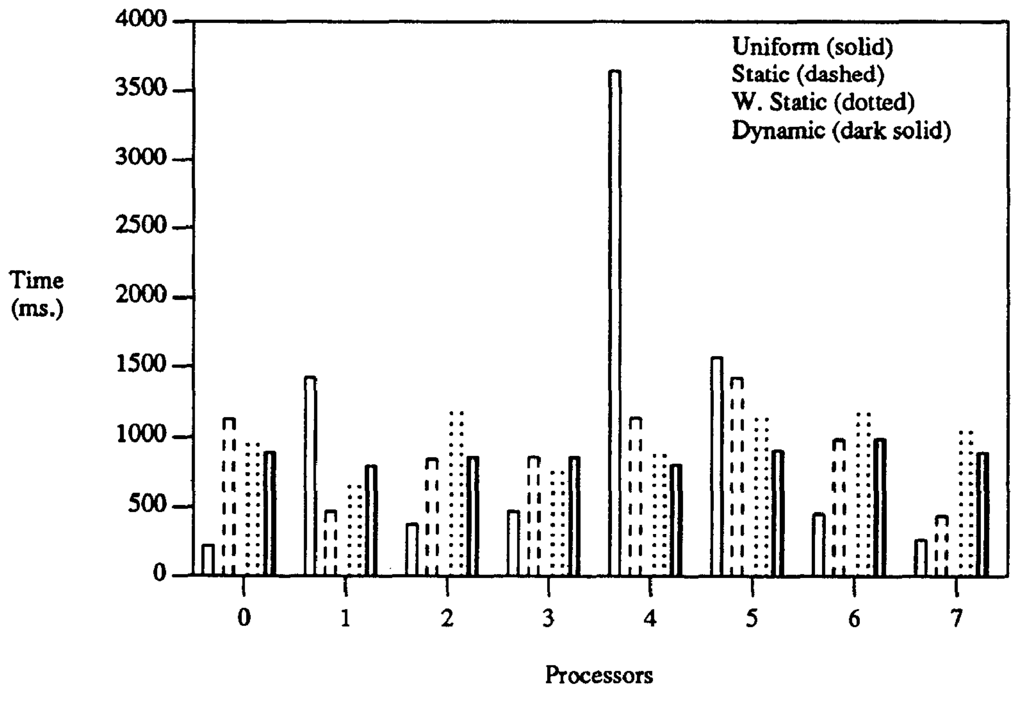

Figure 4 : Distribution of Computation Times for Stereo Match (P=8)

Finally, for 8 processor case, dynamic scheme performs very well. Table 2 summarizes the distribution for the 8 processor case. The table shows the computation time for each processor for all four methods. Speedup is computed as follows. If T_s is the sequential processing time and T_{max} is the maximum processing time of one processor among n processors, then speedup is $\frac{T_s}{T_{max}}$. Variation ratio is the ratio of the maximum processing time to the minimum processing time and it provides a measure of imbalance in the computation. For example, in Table 2 the variation ratio is 16.64 for the case of uniform partitioning, is 3.22 for the case of static load balancing, is 1.82 for weighted static and is 1.25 for dynamic load balancing. Improvement ratio is the ratio of speedup obtained with load balancing to that of uniform partitioning. The computation times shown in these tables include all the overhead of load balancing schemes.

Figure 5 shows the speedup graph for varying size of multiprocessor from 1 processor to 16. As we can observe, uniform partitioning does not provide any significant gains in speedup as number of processors increases. Dynamic scheme performs the best for smaller multiprocessors but as the number of processors increases further, weighted scheme out performs the dynamic scheme. This occurs due to the larger overhead of dynamic scheduling when number of processors is large.

Table 2 : Distribution of Computation Times for Stereo Match

Computation Time Distribution for Stereo Match (P=8)				
Proc. No.	Uniform Partitioning	Static	Static Weighted	Dynamic
	Time (ms.)	Time (ms.)	Time (ms.)	Time (ms.)
0	219	1128	941	893
1	1424	468	635	798
2	377	844	1160	861
3	471	861	736	860
4	3645	1134	861	808
5	1571	1428	1121	909
6	451	985	1159	991
7	259	443	1031	891
Max.	3645	1428	1160	991
Min.	219	443	736	798
Variation ratio	16.64	3.22	1.82	1.25
Improvement ratio	1	2.56	3.14	3.68

Table 3 : Distribution of Computation Time for Time Match

Computation for Time Match (Proc. = 16)						
Proc. No.	Uniform Partitioning			With Load Balancing		
	Matching (Sec.)	Total (Sec.)	No. Zcs	Matching (Sec.)	Total (Sec.)	No. Zcs
0	0.02	0.1	0	3.07	3.43	18
1	0.02	0.12	0	4.88	4.99	24
2	0.02	0.12	0	4.10	4.26	23
3	11.44	11.81	61	2.64	3.01	22
4	0.87	0.97	9	4.59	4.62	23
5	0.02	0.12	0	3.72	3.76	24
6	0.02	0.12	0	4.34	4.38	23
7	1.47	1.57	13	3.54	3.58	24
8	11.07	11.17	62	4.45	4.49	21
9	30.97	31.26	143	5.34	5.36	25
10	16.27	16.52	73	5.47	5.48	22
11	0.02	0.1	0	3.52	3.53	14
12	0.02	0.1	0	6.35	6.42	25
13	0.02	0.1	0	5.51	5.55	21
14	0.02	0.1	0	6.40	6.45	25
15	0.02	0.1	0	4.46	5.10	27
	Max. time(sec.)	Min. time(sec.)	Variation ratio	Speed up	Improvement ratio	
Uniform	30.97	0.02	1550	2.42		
Balanced	6.45	3.01	2.14	11.65	4.81	

4.3. Time Match

The computation in time match algorithm is similar to that in stereo match except that search space is two-dimensional and the input to the algorithm is stereo match output. Other difference is that the number of significant points in the input data is much smaller than that in stereo match because lot of input points get eliminated in stereo match. Table 3 shows the distribution of the computation times for 16 processor case. We only present uniform partitioning and static load balancing cases. The most important observation is that uniform partitioning performs worse than that in the case of stereo match and static load balancing performs better. The table shows how the measure of computation (number of zero crossings left from stereo match step) gets divided among the processors in the two cases. It is clear that the number of zero crossings are very evenly distributed (within the minimum granule of one row constraint) in the static case where as they are lumped with a few processors in the uniform partitioning case. Figure 6 shows the speedup graphs for the two schemes for a range of multiprocessor size. The speedup gains for the load balanced case is very significant over the uniform partitioning case. We computed the overhead of performing knowledge based static load balancing and the overhead was 3 ms., which is negligible compared to the computation time. Following important observations can be made from the above results. First, improvement in performance (such as utilization and speedup) itself increases using the load balancing schemes as number of processors increases. Therefore, performance gains are expected to be higher for larger multiprocessors. Secondly, in an integrated environment, the overheads of such methods are small because measure of load can be computed on the fly as a side result of the current task. Finally, though we

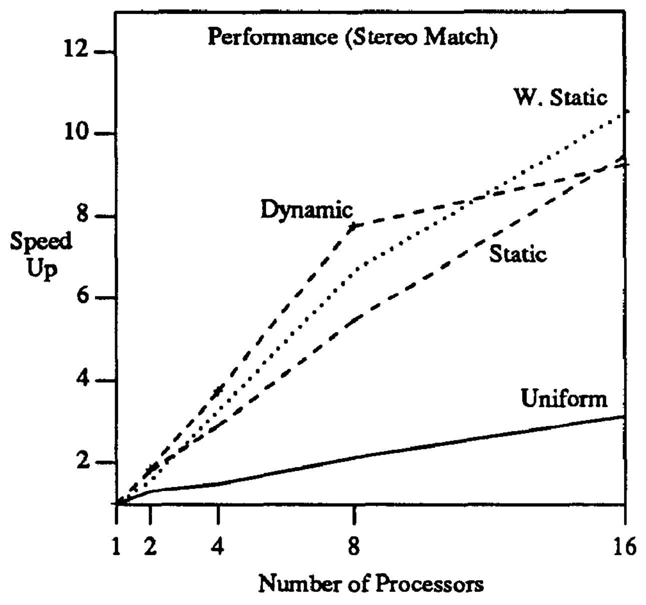

Figure 5 : Speedups for Stereo Match Computation

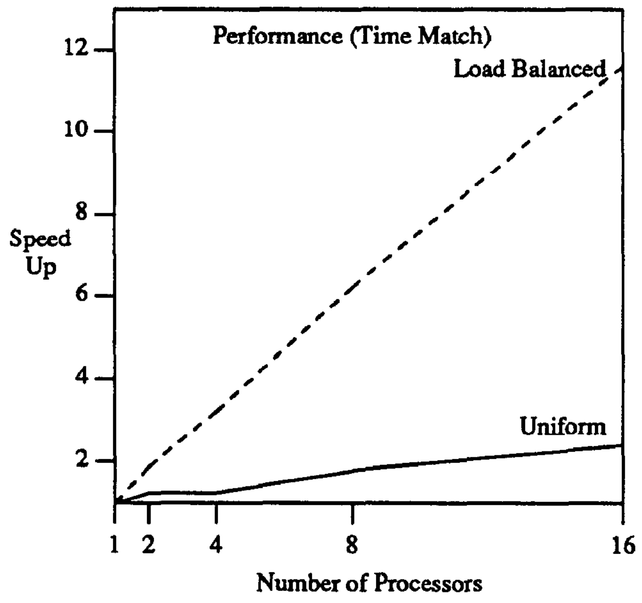

Figure 6 : Speedup for Time Match

showed the performance results of the implementation on the hypercube multiprocessor, these methods can be applied when algorithms are mapped on any medium to large grain multiprocessor system because these techniques are independent of the underlying multiprocessor architecture.

Consider the overall performance gains for the entire system. As the computation progresses from one step to the next, uniform partitioning performs worse because the data points reduce but the computation at each point increases. Therefore, the gains of using parallel processing are minimal. However, the load balancing techniques recognize the data distribution at each step and data is decomposed using the distribution. Therefore, performance gains are expected to improve as the computation progresses in an integrated systems environment. For example, consider zero crossing, stereo match and time match steps. In zero crossing computation uniform partitioning performs well and load is balanced. Hence, the improvement ratio is 1. For stereo match the improvement of static over uniform partitioning is 2.56 for 8 processor case, and is 2.94 for 16 processor case. Similarly, for time match step, the improvement of static load balancing for 8 processor case is 3.49 and for 16 processor case it is 4.81. Therefore, we can observe that the improvement in performance itself increases as the number of processors increases as well as when the computation progresses in an integrated vision system.

5. Summary

In this paper we presented several load balancing and data decomposition methods to implement tasks from an integrated vision system onto a multiprocessor architecture. The main feature of these schemes are that they exploit knowledge about the computation and use the current task to obtain information about data to perform load balancing in the next task. Many image processing and vision algorithms have similar characteristics, and therefore, we believe that these methods can be applied to most integrated vision systems. We show through an example of motion estimation system algorithms that these methods to balance computation on a multiprocessor perform extremely well with little overhead.

Acknowledgements

The authors would like to thank Mun Leung for providing the sequential algorithms for the motion estimation system.

REFERENCES

[1] C. Weems, A. Hanson, E. Riseman, and A. Rosenfeld, "An Integrated Image Understanding Benchmark: Recognition of a 2 1/2 D Mobile," in *International Conference on Computer Vision and Pattern Recognition*, Ann Arbor, MI, June 1988.

[2] Alok Choudhary and Janak Patel, "A Parallel Processing Architecture for Integrated Vision Systems," in *17th Annual International Conference on Parallel Processing*, St. Charles, IL, pp. 383-388, August, 1988.

[3] Alok N. Choudhary, "Parallel Architectures and Parallel Algorithms for Integrated Vision Systems," in *Ph.D. Thesis*,, University of Illinois, Urbana-Champaign, Agust, 1989.

[4] Mun K. Leung and Thomas S. Huang, "Point Matching in a Time Sequence of Stereo Image Pairs," in *Tech. Rep., CSL, university of Illinois*, Urbana-Champaign, 1987.

[5] Alok N. Choudhary, Subhodev Das, Narendra Ahuja, and Janak H. Patel, "Surface Reconstruction from Stereo Images : An Implementation of a Hypercube Multiprocessor," in *The Fourth Conference on Hypercubes, Concurrent Computers, and Applications*, Moneterey, CA, March, 1989.

[6] A. Huertas and G. Medioni, "Detection of Intensity Changes with Subpixel Accuracy Using Laplacian-Gaussian Masks," *IEEE TOPAMI*, vol. PAMI-8, pp. 651-664, Sept. 1986.

[7] Y. C. Kim and J. K. Aggarwal, "Positioning 3-D Objects using stereo images," *Computer and Vision Research Center, The University of Texas at Austin*.