Sādhanā, Vol. 18, Part 2, June 1993, pp. 209–221. © Printed in India.

Mapping stereo and image matching algorithms onto parallel architectures

ASHFAQ KHOKHAR[†] and VIKTOR K PRASANNA[†]

[†]Department of EE-Systems, EEB 244, University of Southern California, Los Angeles, CA 90089-2562, USA

Abstract. In this paper we present parallel implementations of two vision tasks; stereo matching and image matching. Linear features are used as matching primitives. These implementations are performed on a fixed size mesh array and achieve processor-time optimal performance. For stereo matching, we propose $O(Nn^3/P^2)$ time algorithm on a $P \times P$ processor mesh array, where N is the number of line segments in one image, n is the number of line segments in a window determined by the object size, and $P \leqslant n$. The sequential algorithm takes $O(Nn^3)$ time. For image matching, a partitioned parallel implementation is developed. $O[((nm/P^2) + P)nm]$ time performance is achieved on a $P \times P$ processor mesh array, where $P^2 \leqslant nm$. This leads to a processor-time optimal solution for $P \leqslant (nm)^{1/3}$.

Keywords. Stereo matching; image matching; linear features, parallel algorithms; fixed size arrays; object recognition; shape from depth.

1. Introduction

Parallel processing has been used in computer vision over the past two decades. However, most of these solutions have addressed problems in low-level and mid-level vision (Prasanna Kumar 1991). This paper presents processor-time optimal parallel implementations for two vision tasks; stereo and image matching.

Several sequential techniques have been proposed for stereo and image matching. One of the well-known methods for stereo and image matching uses linear features as matching objects. The key advantage of this method is its intrinsic merit with respect to accuracy and sensitivity due to photometric variations (Medioni & Nevatia 1984, 1985).

For stereo matching, we propose $O(Nn^3/P^2)$ time algorithm using a $P \times P$ processor mesh array, where N is the number of line segments in one image, n is the number of line segments in a window determined by the object size, and $P \leqslant n$. For image matching, based on the sequential algorithm presented in Khokhar *et al* (1992), a partitioned implementation is developed. $O[((nm/P^2) + P)nm]$ time performance is achieved on a $P \times P$ mesh array, where $P^2 \leqslant nm$. Both implementations achieve linear speed-up compared with the corresponding sequential algorithms.

The organization of this paper is as follows. In § 2, a model of the architecture used for our implementations is described. Sections 3 and 4 provide parallel implementations for stereo matching and image matching problems respectively. Conclusions and open problems are presented in § 5.

2. Fixed size mesh array

A *fixed size mesh array* is a two-dimensional array of $P \times P$ processors, where P^2 is less than or equal to the problem size. The processors (or processing elements, PE) are connected through bidirectional local links and the array operates in single instruction multiple data (SIMD) mode. Each processor PE_{ij} is connected to PE_{i+1j}, PE_{i-1j}, PE_{ij-1}, and PE_{ij+1}, if they exist. A memory plane of $P \times P$ memory modules (MM) is provided such that each memory module is connected to exactly one processor in the array. Each PE_{ij} is attached to memory module MM_{ij} to store the relevant data. The architecture is shown in figure 1.

The following assumptions are made regarding computations in this model:

- Each arithmetic/logic operation performed in a PE takes $O(1)$ time.
- Each access by PE_{ij} to memory module MM_{ij}, $0 \leqslant i,j < P$, takes $O(1)$ time.
- A unit data transfer between adjacent processors takes $O(1)$ time.

3. Stereo matching on fixed size mesh arrays

Stereo matching is one of the well-known methods for extraction of depth information. Two images, left image and right image, captured at the same time but at different angles are matched. Various stereo matching algorithms differ with respect to the primitives used for matching. Techniques include intensity/area-based matching, feature-based matching, and hierarchical matching (Dhond & Aggarwal 1989). Each technique has its own advantages and disadvantages. Stereo matching using linear features is capable of handling more complex scenes (such as those containing repetitive structures). In this section we provide fast parallel implementation of the

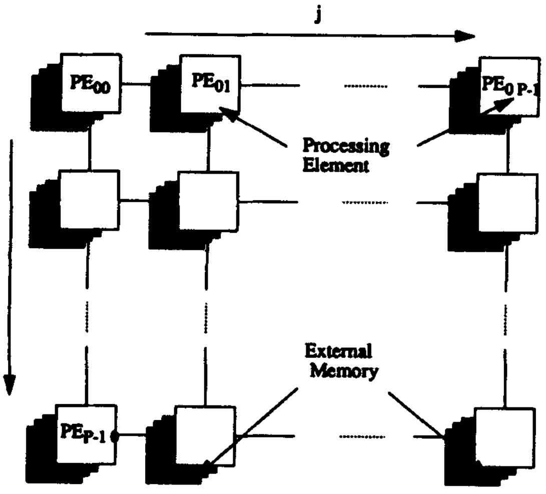

Figure 1. A fixed size mesh array.

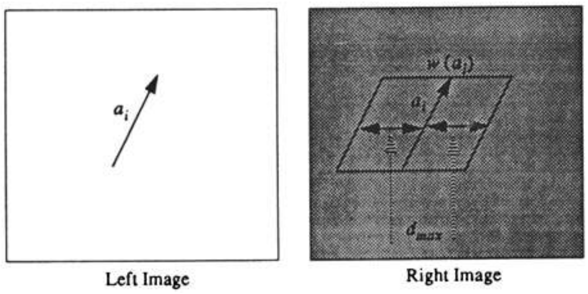

Figure 2. Determining a *stereo-window*.

stereo matching algorithm (also called the Minimum Disparity Algorithm) described in Medioni & Nevatia (1985). For the sake of completeness, the main ideas of this algorithm are presented in § 3·1.

3.1 *Minimum disparity algorithm*

The technique attempts to match overlapping segments detected along the same epipolar line, having similar contrast and orientation. Following the terminology in Medioni & Nevatia (1985), for each segment a_i in the left image, a match is found in a window $w(a_i)$ defined in the right image. Similarly, for each segment b_j in the right image, a match is found in $w(b_j)$ defined in the left image. The shape of the window is a parallelogram. This is shown in figure 2 for left to right match, one side corresponds to a_i, and the other side is a horizontal vector of length $2d_{\max}$, where $d_{\max}$ is the maximum disparity. The number of segments in each window is assumed to be at most n and both the images are assumed to have N segments each.

For each a_i, a set $S_p(a_i)$ of possible matches in window $w(a_i)$ is defined based on the contrast, overlap, and orientation. Similarly for each b_j, a set $S_p(b_j)$ is defined. To assign unambiguous matches, a set of matches is considered together for each segment in the image. For each possible element j in $S_p(a_i)$, an evaluation function $v(i,j)$ is computed. This function is dependent on how well the disparities of the other matching pairs in $w(b_j)$ agree with the average disparity d_{ij} of the matching pair. A set of preferred matches $Q^t(a_i)$, is constructed for each i during iteration t, if the following holds:

$$\forall k \in S_p(a_i), \quad \text{such that} \quad b_k \leftrightarrow b_j, v^t(i,j) < v^t(i,k), \tag{1}$$

and

$$\forall h \in S_p(b_j), \quad \text{such that} \quad a_h \leftrightarrow a_i, v^t(i,j) < v^t(h,j). \tag{2}$$

The relation $b_k \leftrightarrow b_j$ is true if b_k overlaps b_j. $v(i,j)$ is defined as follows:

$$v^{t+1}(i,j) = \sum_{a_h \text{ in } w(b_j)} \min_{b_k \text{ verifies } C_1(a_h)} \lambda_{ijhk} |d_{hk} - d_{ij}| / card(b_j)$$

$$+ \sum_{b_k \text{ in } w(a_i)} \min_{a_h \text{ verifies } C_2(b_k)} \lambda_{ijhk} |d_{hk} - d_{ij}| / card(a_i). \tag{3}$$

In the above equation, $t+1$ indicates the iteration number and $\lambda_{ijhk} = \min$ (overlap(i,j), overlap(h,k)) and $card(a_i)$ is the number of segments in $w(a_i)$. The

```
1. repeat
2.     change ← 0;
3.     for $i = 1$ to $N$ do
4.         for each $j$ such that $j \in w(a_i)$ do
5.             i-pref($i, j, QT(a_i)$)
6.         end
7.     end;
8.     for $j = 1$ to $N$ do
9.         for each $i$ such that $i \in w(b_j)$ do
10.            j-pref($j, i, QT(b_j)$)
11.        end
12.    end;
13.    for $i = 1$ to $N$ do
14.        for each $j$ such that $j \in w(a_i)$ do
15.            Q-update($i, j$)
16.        end
17.    end
18.    for $i = 1$ to $N$ do $Q(a_i) \leftarrow Q'(a_i)$;
19.    for $j = 1$ to $N$ do $Q(b_j) \leftarrow Q'(b_j)$;
20. until (change = 0)
```

Figure 3. Sequential algorithm for stereo matching.

relations C_1 and C_2 are defined as follows. We say b_k verifies $C_1(a_h)$ if:

1. $Q^t(a_h) \neq \emptyset$, b_k is in $Q^t(a_h)$ else b_k is in $S_p(a_h)$;
2. Either $b_k \neq b_j$, or a_h and a_i do not overlap.

In order to expose the potential parallelism, the computations performed in the algorithm are shown in figure 3. The procedure *i-pref*$(i, j, QT(a_i))$ shown in figure 4 is used to find a temporary *preferred* set $QT(a_i)$ for a_i without the confirmation from the other image [i.e. satisfying (1) but not (2)]. The two nested loops (lines 4–8 and lines 9–13), correspond to the computations in (3). On the other hand, the procedure *j-pref*$(j, i, QT(b_j))$ shown in figure 5 is used to find a temporary *preferred* set $QT(b_j)$

```
procedure i-pref($i, j, QT(a_i)$)
1. for each $h$ such that $h \in w(b_j)$ do
2.     for each $k$ such that $b_k$ verifies $C_1(a_h)$ do
3.         $\min_h \leftarrow min(\min_h, \lambda_{ijhk}|d_{hk} - d_{ij}|)$;
4.     end;
5.     $\mathrm{sum1}(i,j) \leftarrow \mathrm{sum}(i,j) + \min_h$;
6. end;
7. $\mathrm{ave1}(i,j) \leftarrow \mathrm{sum1}(i,j)/card(b_j)$;
8. for each $k$ such that $k \in w(a_i)$ do
9.     for each $h$ such that $a_h$ verifies $C_2(b_k)$ do
10.        $\min_k \leftarrow min(\min_k, \lambda_{ijhk}|d_{hk} - d_{ij}|)$;
11.    end;
12.    $\mathrm{sum2}(i,j) \leftarrow \mathrm{sum}(i,j) + \min_k$;
13. end;
14. $\mathrm{ave2}(i,j) \leftarrow \mathrm{sum2}(i,j)/card(a_i)$;
15. $\mathrm{sum}(i,j) \leftarrow \mathrm{ave1}(i,j) + \mathrm{ave2}(i,j)$;
16. case:
17.    $\mathrm{sum}(i,j) < \mathrm{Min}(i)$:
18.        $QT(a_i) \leftarrow \{ j \}$;
19.        $\mathrm{Min}(i) \leftarrow \mathrm{sum}(i,j)$;
20.    $\mathrm{sum}(i,j) = \mathrm{Min}(i)$:
21.        $QT(a_i) \leftarrow QT(a_i) \cup \{ j \}$;
22. end
```

Figure 4. Finding partially *preferred* matches for the first image.

```
procedure j-pref(j, i, QT(b_j))
1.  for each $k$ such that $k \in w(a_i)$ do
2.      for each $h$ such that $a_h$ verifies $C_2(b_k)$ do
3.          $\min_k \leftarrow min(\min_k, \lambda_{ijhk}|d_{hk} - d_{ij}|)$;
4.      end;
5.      $\mathrm{sum1}(j,i) \leftarrow \mathrm{sum}(j,i) + \min_k$;
6.  end;
7.  $\mathrm{ave1}(j,i) \leftarrow \mathrm{sum1}(j,i)/card(a_i)$;
8.  for each $h$ such that $h \in w(b_j)$ do
9.      for each $k$ such that $b_k$ verifies $C_1(a_k)$ do
10.         $\min_h \leftarrow min(\min_h, \lambda_{ijhk}|d_{hk} - d_{ij}|)$;
11.     end;
12.     $\mathrm{sum2}(j,i) \leftarrow \mathrm{sum}(j,i) + \min_h$;
13. end;
14. $\mathrm{ave2}(j,i) \leftarrow \mathrm{sum2}(j,i)/card(b_j)$;
15. $\mathrm{sum}(j,i) \leftarrow \mathrm{ave1}(j,i) + \mathrm{ave2}(j,i)$;
16. case:
17.     $\mathrm{sum}(j,i) < \mathrm{Min}(j)$:
18.         $QT(b_j) \leftarrow \{ i \}$;
19.         $\mathrm{Min}(j) \leftarrow \mathrm{sum}(j,i)$;
20.     $\mathrm{sum}(j,i) = \mathrm{Min}(j)$:
21.         $QT(b_j) \leftarrow QT(b_j) \cup \{ i \}$;
22. end
```

Figure 5. Finding partially *preferred* matches for the second image.

for b_j without the confirmation from the other image [i.e. satisfying (2) but not (1)]. The third procedure, *Q-update(i,j)*, is then used to combine the results from the procedures, *i-pref* and *j-pref*, to determine the new sets, $Q(a_i)$ and $Q(b_j)$, for a_i and b_j, respectively (figure 6).

Notice that the execution of the *i-pref* procedure (or *j-pref* procedure) takes $O(n^2)$ time, and that of the *Q-update* procedure takes constant time. It is easy to verify that each **repeat** iteration takes $O(Nn^3)$ time. The complete algorithm terminates after a constant number of iterations (Medioni & Nevatia 1985).

3.2 *Parallel stereo matching*

In this section we present a parallel implementation of the stereo matching algorithm on a fixed size mesh array. A parallel version of the minimum disparity algorithm is given in figure 7. Procedures *Parallel-i-pref*(i) and *Parallel-j-pref*(j) determine the partial *preferred* matches, $QT(a_i)$ and $QT(b_j)$, for a_i and b_j respectively. The third procedure *Parallel-Q-update* is used to determine the new sets, $Q(a_i)$ and $Q(b_j)$, for a_i and b_j, respectively.

3.2a *Data partitioning*: In stereo matching, input to the algorithm is a set of N segments from the right image and a set of N segments from the left image. As described in § 3.1, each segment is represented by its length, contrast and orientation. With each

```
procedure Q-update(i, j)
1.  if (($j \in QT(a_i)$) AND ($i \in QT(b_j)$)) then
2.  begin
3.      change $\leftarrow$ 1
4.      $Q'(a_i) \leftarrow Q'(a_i) \cup \{j\}$;
5.      $Q'(b_j) \leftarrow Q'(b_j) \cup \{i\}$;
6.  end
```

Figure 6. Updating sets of *preferred* matches.

```
1. repeat
2.    change ← 0;
3.    for $i = 1$ to $N$ do
4.       Parallel-i-pref($i$);
5.    for $j = 1$ to $N$ do
6.       Parallel-j-pref($j$);
7.    for $i = 1$ to $N$ do
8.       Parallel-Q-update($i$);
9. until (change = 0)
```

Figure 7. Parallel algorithm for stereo matching.

pair (a_i, b_j), such that a_i, b_j overlap and have similar contrast and similar orientation, an average disparity d_{ij} is associated. In order to find a possible match for each segment a_i in the left image, a window $w(a_i)$ is defined in the right image. Similarly, for each segment b_j in the right image a window $w(b_j)$ is defined in the left image. Each window is assumed to contain at most n segments. In this section, we present a partitioning of these windows such that the communication overhead among the processors does not become a bottleneck and at the same time the data redundancy is minimized.

As described earlier, for each segment a_i there is a window defined in the other image. Thus, there is a total of N windows. Each window has at most n segments. In order to reduce the data redundancy while mapping these windows onto a mesh array of P^2 processors, the following constraint should be satisfied.

- There should be at most N/P^2 windows mapped onto any PE.

At any time, if the above condition is satisfied, no further windows should be assigned to that PE.

As shown in lines 5 and 10 of the algorithm given in figure 3, it is clear that the windows defined in the left image are accessed by the segments in the right image and vice versa. Therefore, while processing a segment a_i in the right image, the algorithm in figure 8 ensures that for a given segment a_i, each PE is assigned no more

```
procedure Map-windows-left-image(right-image, left-image)
1. Initialize:
2. for $k = 0$ to $P - 1$ do
3.    for $l = 0$ to $P - 1$ do
4.    counter1[$k$][$l$], counter2[$k$][$l$] ← 0;
5. Mark all PEs as available
6. for each segment $a_i$ in the left image do
7.    in parallel for each $b_j$ in the right image such that $j \in w(a_i)$ do
8.       for each $a_h$ in the left image such that $h \in w(b_j)$ do
9.          if $w(a_h)$ is not assigned then
10.            assign $w(a_h)$ to an available PE, say PE$_{kj}$,
               such that counter2[$k$][$j$] $< n/P$, $0 \le k \le P - 1$;
11.            counter1[$k$][$j$]$^{++}$, counter2[$k$][$j$]$^{++}$
12.            if counter1[$k$] = $\lceil N/P^2 \rceil$
13.               mark PE$_k$ as not_available
14.       else counter2[$k$][$j$]$^{++}$, assuming $w(a_h)$ is assigned to $PE_{kj}$
15.    end;
16.    for $k = 0$ to $P - 1$ do
17.       for $l = 0$ to $P - 1$ do
18.          counter2[$k$][$l$] ← 0;
19. end;
```

Figure 8. Mapping windows of left image onto fixed size mesh array.

```
procedure Parallel-i-pref(i, QT(a_i))
1. in parallel for each h, j such that $h \in w(b_j)$ and $j \in w(a_i)$ do
2.     for each k such that $b_k$ verifies $C_1(a_h)$ do
3.         $\min(i,j,h) \leftarrow \min(\min(i,j,h), \lambda_{ijhk}|d_{hk} - d_{ij}|)$;
4.     end;
5.     $\mathrm{sum1}(i,j) \leftarrow \mathrm{sum}(i,j) + \min(i,j,h)$;
6. parallel end;
7. $\mathrm{ave1}(i,j) \leftarrow \mathrm{sum1}(i,j)/\mathrm{card}(b_j)$;
8. in parallel for each k, j such that $k \in w(a_i)$ and $i \in w(b_j)$ do
9.     for each h such that $a_h$ verifies $C_2(b_k)$ do
10.        $\min(i,j,k) \leftarrow \min(\min(i,j,k), \lambda_{ijhk}|d_{hk} - d_{ij}|)$;
11.    end;
12.    $\mathrm{sum2}(i,j) \leftarrow \mathrm{sum}(i,j) + \min(i,j,h)$;
13. parallel end;
14. $\mathrm{ave2}(i,j) \leftarrow \mathrm{sum2}(i,j)/\mathrm{card}(a_i)$;
15. $\mathrm{sum}(i,j) \leftarrow \mathrm{ave1}(i,j) + \mathrm{ave2}(i,j)$;
16. case:
17.    $\mathrm{sum}(i,j) < \mathrm{Min}(i)$:
18.        $QT(a_i) \leftarrow \{ j \}$;
19.        $\mathrm{Min}(i) \leftarrow \mathrm{sum}(i,j)$;
20.    $\mathrm{sum}(i,j) = \mathrm{Min}(i)$:
21.        $QT(a_i) \leftarrow QT(a_i) \cup \{ j \}$;
22. end
```

Figure 9. *Parallel-i-pref*(*i*) on a mesh array.

than n/P windows. Also, no window is assigned to more than one PE. A PE is marked *available* if it is taking part in the assignment. During the assignment if a PE has been assigned N/P^2 windows in total, it is marked as *not-available* and no more windows are assigned to that PE. At the beginning of the algorithm all the processors are marked *available*. The mapping algorithm is outlined in figure 8.

The mapping algorithm described in figure 8 guarantees that, while processing any segment for matching, there will be no more than $O(n)$ data communications per potential match. This fact will become clear when the parallel algorithm is explained in §3·2b below. The running time of the mapping algorithm is $O(Nn)$.

3.2b *Partitioned implementation on a fixed size mesh array*: Based on the algorithm shown in figure 7, parallel implementation on a $P \times P$ mesh array is developed. The procedure *Parallel-i-pref*(*i*) is presented in figure 9. Similar procedure for *Parallel-j-pref*(*j*) can be developed. In the following, floor function ($\lfloor\ \rfloor$) is assumed for all indices of the form a/b.

In each i-loop, $\mathrm{PE}_{(h/P)(j/P)}$, $0 \leqslant h, j < n$ (or $\mathrm{PE}_{(k/P)(j/P)}$ for the second part of the j-loop) is used for the computation of $\min(i,j,k)$ (or $\min(i,j,h)$). The information required to accomplish the computation in $\mathrm{PE}_{(h/P)(j/P)}$ (or $\mathrm{PE}_{(h/P)(j/P)}$) includes:

1. if $t = 0$, then $S_p(a_h)$ (or $S_p(b_k)$), else $Q^t(a_h)$(or $Q^t(b_k)$),
2. $d_{hk}, 0 \leqslant k < n$ (or $d_{hk}, 0 \leqslant h < n$), and
3. $\lambda_{ijhk}, 0 \leqslant k < n$ (or $\lambda_{ijhk}, 0 \leqslant h < n$).

The main steps of procedure *Parallel-i-pref*(*i*) are briefly discussed in the following, with the corresponding execution time indicated within parentheses.

1. $\forall j \in w(a_i)$, load d_{hk} and λ_{ijhk}, $O(n)$ data to $\mathrm{PE}_{(h/P)(j/P)}$, $O(n^3)$ data in all. ($O(n^3/P^2)$).

```
procedure Parallel-Q-update(i, j)
1. if (($j \in QT(a_i)$) AND ($i \in QT(b_j)$)) then
2. begin
3.     change ← 1;
4.     $Q'(a_i) \leftarrow Q'(a_i) \cup \{j\}$;
5.     $Q'(b_j) \leftarrow Q'(b_j) \cup \{i\}$;
6. end
```

Figure 10. *Parallel-Q-update* for updating sets of preferred matches.

2. PE_{11} broadcasts d_{ij} to all the processors in the array. $(O((n/P)P))$
3. Perform the *min* operation over all k to determine $\min(i,j,h)$ in $\mathrm{PE}_{(h/P)(j/P)}$. $(O((n/P)P))$
4. Along each column of processors, i.e. $\forall j$, all the $\min(i,j,h)$ values are summed up and saved in the last processor $\mathrm{PE}_{P(j/P)}$. $(O((n/P)P))$
5. In each $\mathrm{PE}_{P(j/P)}$, compute the average.
6. $\forall j \in w(a_i)$, load d_{hk} and λ_{ijhk}, $O(n)$ data to $\mathrm{PE}_{(k/P)(j/P}$, $O(n^3)$ data in total. $(O(n^3/P^2))$.
7. Perform the *min* operation over all h to determine $\min(i,j,k)$ in $\mathrm{PE}_{(k/P)(j/P)}$. $(O((n/P)P))$
8. Along each column of processors, i.e. $\forall j$, all the $\min(i,j,k)$ values are summed up and saved in the last processor $\mathrm{PE}_{P(j/P)}$. $(O((n/P)P))$
9. In each $\mathrm{PE}_{P(j/P)}$, $\forall j \in w(a_i)$, compute the average of the sum obtained in step 8 and add to the average obtained in step 5. $(O(1))$
10. Along the last row of processors, find the minimum of all the values obtained in step 9, and store the corresponding b_j back in the memory, which is $QT(a_i)$. $(O((n/P)P))$

Figure 11 provides a pictorial representation of the execution of the procedure *Parallel-i-pref*(i). Similar steps can be designed for the procedure *Parallel-j-pref*(j).

It can be easily verified that for each a_i the procedure *Parallel-i-pref*(i) runs in $O(n^3/P^2)$ time with each time unit corresponding to a simple arithmetic/logic operation. The resulting $QT(a_i)$'s and $QT(b_j)$'s can then be combined in constant time by using the procedure *Parallel-Q-update* given in figure 10. Therefore, each iteration takes $O(Nn^3/P^2)$ time.

4. Image matching on a fixed size mesh array

The image matching problem plays a key role in object recognition. In the past, several approaches have been proposed for this problem (Clark *et al* 1978; Shapiro & Haralick 1981; Price 1982, pp. 105–112), which, in general, differ with respect to the primitives used for matching. In this section, we consider image matching using linear features for parallel implementation. Readers can refer to Medioni & Nevatia (1984) for additional details of the matching technique. We begin with the basic idea of this approach and then present a processor-time optimal parallel implementation of the algorithm on a fixed size mesh array.

4.1 *Matching technique*

In general, in the image matching problem, we have n objects, $\{o_1, o_2, \ldots, o_n\}$, in the scene and m labels, $\{l_1, l_2, \ldots, l_m\}$, in the model. Here, the objects are *segments* in the scene derived from edge detectors and are described by the coordinates of their end

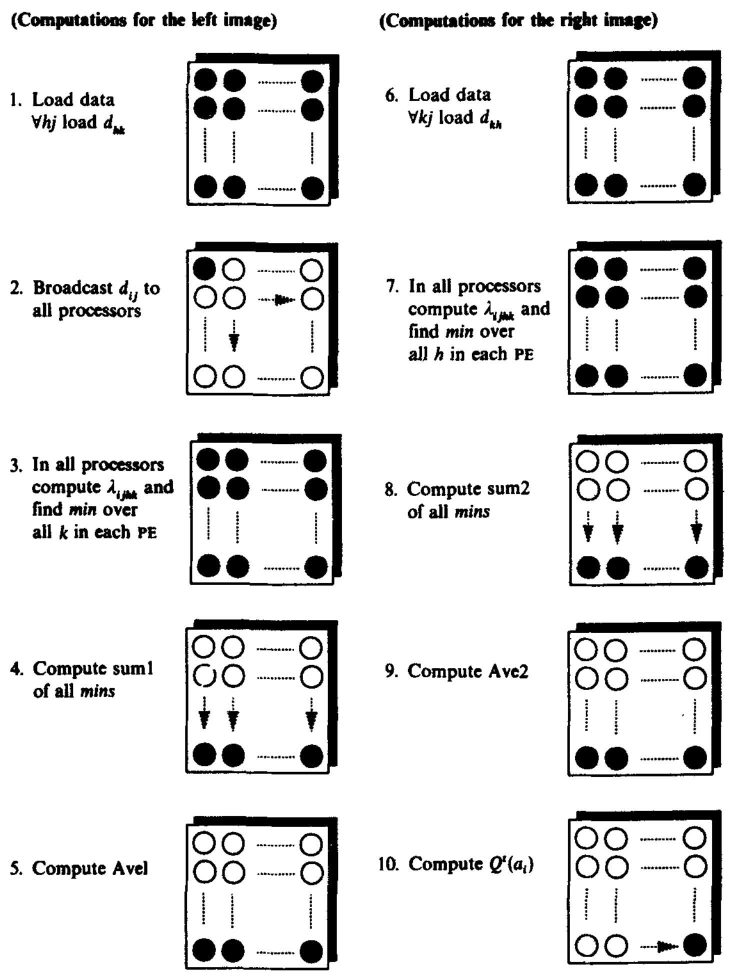
(Computations for the left image)
(Computations for the right image)
1. Load data $\forall hj$ load d_{hk}
2. Broadcast d_{ij} to all processors
3. In all processors compute λ_{ijhk} and find *min* over all k in each PE
4. Compute sum1 of all *mins*
5. Compute Ave1
6. Load data $\forall kj$ load d_{kh}
7. In all processors compute λ_{ijhk} and find *min* over all h in each PE
8. Compute sum2 of all *mins*
9. Compute Ave2
10. Compute $Q^t(a_i)$

Figure 11. Data flow for procedure *Parallel-i-pref*(i).

points, orientation and average contrast. The matching technique computes the quantity v_{i_p}, in $\{0,1\}$, which is the possibility of assigning label l_p to object o_i.

The method (Medioni & Nevatia 1984) relies on geometric constraints, which means that when a lebel l_p is assigned to object o_i, we expect to find an object o_j with assigned label l_q in an area depending on i, p, q. The *match-window* $W(i,p,q)$ denotes the area described by the parameters i, p, q. By representing the object o_i with a vector $\mathbf{A}_i\mathbf{B}_i$, the label l_p with $\mathbf{C}_p\mathbf{D}_p$ and label l_q with $\mathbf{C}_q\mathbf{D}_q$, we can determine the four extreme points, W_1, W_2, W_3, W_4, of the induced *match-window* $W(i,p,q)$ using the following relations: (μ denotes a given scaling factor).

- $\mathbf{A}_i\mathbf{W}_1 = \mu \cdot \mathbf{C}_p\mathbf{C}_q, \mathbf{W}_1\mathbf{W}_2 = \mu \cdot \mathbf{C}_q\mathbf{D}_q$:
- $\mathbf{B}_i\mathbf{W}_3 = \mu \cdot \mathbf{D}_p\mathbf{C}_q, \mathbf{W}_3\mathbf{W}_4 = \mu \cdot \mathbf{C}_q\mathbf{D}_q$.

Figure 12 shows the relationship between the window and the segments.

The meaning of *compatibility* is defined as follows:

$\langle i,p\rangle$ is *compatible* with $\langle j,q\rangle$ *iff* o_i in $W(j,q,p)$ and o_j in $W(i,p,q)$.

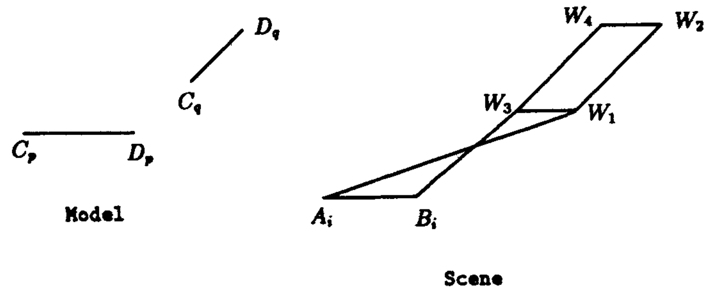

Figure 12. Determining a *match-window*.

Let $\Omega^x_{ij}[p,q]$ denote the *compatibility* of assigning label l_p to object i and label l_q to object j. A weak notion of consistency is used to determine whether an assignment is *feasible*. A predetermined confidence factor, $\delta \leqslant m$, is used to decide the *feasibility* of $\langle i,p \rangle$ as in the following update statement during an iteration*.

For every i,p, $v'_{ip} \leftarrow v_{ip}$ AND 'condition A', where 'condition A' is true if $(\exists S \subseteq \{1,2,\dots,m)$ and $\|S\| = \delta$, such that for every $q \in S, \exists j \in \{1,2,\dots,n)$ such that $v_{jq} = 1$ and $\Omega^x_{ij}[p,q] = 1)$ and is false otherwise.

The algorithm stops when for all $i,p,v'_{ip} = v_{ip}$. We can rewrite the above update statement as

$$v'_{ip} \leftarrow v_{ip} * \bigwedge_{q=1}^{\delta,m} \left[\sum_{j=1}^{n} (v_{jq} * \Omega^x_{ij}[p,q]) \right], \tag{4}$$

where

$$\bigwedge_{q=1}^{\delta,m} X_q = \begin{cases} 1, & \text{if } \Sigma^m_{q=1} X_q \geqslant \delta, \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

Note that the operation $\Sigma^n_{j=1}$ in (4) is a logical OR operation, while the operation $\Sigma^m_{q=1} X_q$ in (5) is an arithmetic ADD operation. With the modified update statement given by (5), we have designed a faster sequential algorithm (Khokhar *et al* 1992) which is easier to parallelize compared with the one proposed by Medioni & Nevatia (1984). This algorithm is an extension of the discrete relaxation algorithm developed by Lin (Lin 1991; Lin & Prasanna Kumar 1991) and it takes $O(n^2m^2)$ time. Each time unit corresponds to a simple arithmetic/logic operation. The original algorithm (Medioni & Nevatia 1984) runs in $O(n^2m^2dw)$ time, where d is the density of the segments and w is the window size. In the worst case, d and w can be n and m respectively. More details can be found in Khokhar *et al* (1992).

4.2 *Parallel image matching*

In this section, we present a parallel implementation of the image matching algorithm given in Khokhar *et al* (1992). Since the size of a *match-window* determined by any object and two labels is much smaller than the size of the complete image, the number of initially assignable segments in any of the *match-windows* for each object is much smaller than the total number of objects in the image. This allows us to obtain a

* $\|S\|$ denotes the cardinality of S

```
    { Initialization }
1.  Initialize all $\Omega^x_{ij}[p,q]$'s in parallel;
2.  parallel do (in $PE_{ip}$, $0 \le i \le n-1$, $0 \le p \le m-1$)
3.     $v_{ip} \leftarrow 1$;
4.     for $q = 0$ to $m-1$ do
5.        $N_{ip}[q] \leftarrow 0$;
6.        $T_{ip} \leftarrow 0$
7.        for $j = 0$ to $n-1$ do
8.           if $(\Omega^x_{ij}[p,q] = 1)$ then $N_{ip}[q] \leftarrow N_{ip}[q] + 1$;
9.        end;
10.       if $(N_{ip}[q] \neq 0)$ then $T_{ip} \leftarrow T_{ip} + 1$;
11.    end;
12.    if $(T_{ip} < \delta)$ then do
13.       $Send_{ip} \leftarrow 1$;
14.       $v_{ip} \leftarrow 0$;
15.    end;
13. parallel end ;

    { Iteration }
16. repeat
17.    parallel do (in $PE_{ip}$, $0 \le i \le n-1$, $0 \le p \le m-1$)
18.       if $(Send_{ip} = 1)$ then
19.          send Id $< i,p >$ to all the PEs;
20.       { if (no broadcast Id acknowledged)
21.         then stop;
22.         else a broadcast Id, say $< j,q >$, is acknowledged by all PE's;}*
23.       if $(< i,p > = < j,q >)$ then $Send_{ip} \leftarrow 0$;
24.       if $((v_{ip} = 1)$ AND $(\Omega^x_{ij}[p,q] = 1))$ then do
25.             $N_{ip}[q] \leftarrow N_{ip}[q] - 1$;
26.             if $(N_{ip}[q] = 0)$ then $T_{ip} \leftarrow T_{ip} - 1$;
27.             if $(T_{ip} < \delta)$ then do
28.                $Send_{ip} \leftarrow 1$;
29.                $v_{ip} \leftarrow 0$;
30.             end
31.          end
32.       end
33.    parallel end
34. forever
    *: the code inside braces is the broadcast operation.
```

Figure 13. Parallel algorithm for image matching.

partitioned implementation in which each processor is responsible for more than one ⟨object, label⟩ pair.

A parallel implementation of the algorithm is shown in figure 13. Each v_{ip} is associated with $m+1$ counter variables. There are $N_{ip}[q]$, $1 \leqslant q \leqslant m$, and T_{ip}. These counter variables have the following definitions:

- $N_{ip}[q]$ denotes the number of 1's in the n entries of $\Omega^x_{ij}[p,q]$, $1 \leqslant j \leqslant n$, and
- T_{ip} denotes the number of nonzero $N_{ip}[q]$'s, $1 \leqslant q \leqslant m$.

For v_{ip}, each of the $m\,N_{ip}[q]$'s are used to determine an object for label l_q, i.e., if we can find any object to be labelled with l_q when o_i is labelled with l_p. T_{ip} is used to determine if there are at least δ such *compatible* labellings when o_i is labelled with l_p. Each infeasible pair $\langle i,p \rangle$ (having $T_{ip} \leqslant \delta$) is broadcast to all the processors. Each PE checks if it has any pair $\langle j,q \rangle$ such that $\Omega^x_{ij}[p;q] = 1$ and decrements $N_{ip}[q]$. At any time, if $N_{ip}[q]$ becomes zero, T_{ip} is decremented. As a result, if $T_{ip} \leqslant \delta$, pair $\langle j,q \rangle$ is marked *infeasible*. The broadcast operation can be implemented in a variety of ways depending upon the underlying parallel architecture.

4.2a *Partitioned implementation on a fixed size mesh array*: Based on the algorithm shown in figure 13, a partitioned implementation on a fixed size mesh array is obtained. Each of the P^2 processors process (nm/P^2) distinct v_{ip} values. The data stored in each of the P^2 memory modules include $(nm/P^2)v_{ip}$ values, corresponding $nm\,\Omega^x_{ij}[p,q]$ values, $m\,N_{ip}[q]$ counter values and the T_{ip} variable. Also, a flag is stored in each MM for each v_{ip} to indicate whether the *infeasibility* has been *acknowledged* by all the processors. Such an acknowledgement triggers the necessary update of the corresponding counters in each PE. Also, in each PE, an extra flag is used to indicate if at least one such *infeasible* assignment (among its nm/P^2 assignments) is yet to be *acknowledged*.

An initialization procedure is executed in each PE to initialize the m counter variables for each of its $(nm/P^2)v_{ip}$ values. Based on the condition defined in §4.1, each PE sets the corresponding flag for an *infeasible* assignment and retains the Id of one such assignment for later broadcast. This can be performed in $O(nm/P^2)$ time. During each iteration, a 'collect' operation is first executed. The purpose of this operation is to gather the Ids retained in all the processors at the end of the previous iteration. A designated PE (say PE_{00}) is responsible for collecting these Ids, and retaining one of them. This Id collection process is executed in each row by moving each *infeasible* Id to the processors in the left-most column and then moving it up to PE_{00}. The retained Id in PE_{00} is then broadcast to all the processors. The broadcast operation can be executed in $O(P)$ time. Once each PE receives such an Id (of an *infeasible* assignment), an update procedure is carried out to modify the corresponding counter variables and to set the corresponding *infeasibility* flag, if necessary. The algorithm terminates if there is no Id retained in PE_{00}. Since it takes constant time to update the affected counter variable of each assignment, $O(nm/P^2)$ time is sufficient for all the processors to complete the update operation in parallel. Thus, each iteration can be performed in $O((nm/P^2)+P)$ time. The total execution time is $O[(nm/P^2)+P)nm]$, since there can be at most nm iterations. This implementation leads to a processor-time optimal solution when $P \leqslant (nm)^{1/3}$.

5. Conclusions

We have presented processor-time optimal parallel implementations for image and stereo matching problems on fixed size mesh arrays. The sequential algorithms used in the implementations rely on linear features as primitives for matching. An extension of our image matching implementation provides efficient solution to the kernel matching algorithm given by Medioni & Nevatia (1984).

Image matching and stereo matching are key problems in image understanding. Several sequential approaches to these problems have been investigated. The proposed solutions vary mainly in terms of the primitives used for matching. Extensive work is needed to consolidate these approaches and provide a framework for parallel stereo and image matching problems.

This research was supported in part by NSF under grant IRI-9145810 and in part by DARPA and AFOSR contracts F-49260-89-C-0126 and F-49620-90-C-0078.

References

Clark C, Luk A, McNary C 1978 Feature based scene analysis and model matching. *Proc. of NATO Advanced Study Inst. for Pattern Recognition and Signal Processing,* France

Dhond U, Aggarwal J K 1989 Structure from stereo – A review. *IEEE Trans. Syst., Man Cybern.* 19: 1489–1510

Khokhar A, Lin W, Prasanna V K 1992 Stereo and image matching on fixed size linear arrays. *IEEE Trans. Pattern Anal. Mach. Intell.* PAMI (submitted)

Lin W 1991 *Mapping image algorithms onto window architectures,* Ph D thesis, Dept. of EE-Systems, University of Southern California, Los Angeles

Lin W, Prasanna Kumar V K 1991 Parallel algorithms and architectures for discrete relaxation technique. *Proc. of IEEE Conference on Vision and Pattern Recognition* (New York: IEEE)

Medioni G, Nevatia R 1984 Matching images using linear features. *IEEE Trans. Pattern Recogn. Image Process.* PAMI-6: 675–685

Medioni G, Nevatia R 1985 Segment-based stereo matching. *Computer Vision, Graph Image Process.* 31: 2–18

Prasanna Kumar V K 1991 *Parallel algorithms and architectures for image understanding* (Boston: Academic Press)

Price K 1982 Symbolic matching of images and scene models. *Proc. of IEEE Workshop on Computer Vision* (New York: IEEE)

Shapiro L, Haralick R 1981 Structural description and inexact matching. *IEEE Trans. Pattern Anal. Mach. Intell.* PAMI 3: 504–519