

DATA-PARALLEL ALGORITHMS FOR IMAGE COMPUTING

Mark J. Carlotto
TASC
55 Walkers Brook Drive
Reading, MA 01867

ABSTRACT

Data-parallel algorithms for image computing on the Connection Machine are described. After a brief review of some basic programming concepts in *Lisp, a parallel extension of Common Lisp, data-parallel programming paradigms based on a local (diffusion-like) model of computation, the scan model of computation, a general interprocessor communications model, and a region-based model are introduced. Algorithms for connected component labeling, distance transformation, Voronoi diagrams, finding minimum cost paths, local means, shape-from-shading, hidden surface calculations, affine transformation, oblique parallel projection, and spatial operations over regions are presented. An new algorithm for interpolating irregularly spaced data via Voronoi diagrams is also described.

1. INTRODUCTION

The Connection Machine (CM) system is a massively parallel, single instruction multiple data (SIMD) computer containing up to 64K physical processors. Its architecture and equally important, its parallel programming languages, have been shown to be well-suited to a variety of problems^{1,2}. This paper focuses on the use of the CM in image computing. Examples of image computing operations include: image processing (restoration, enhancement, interpolation, warping, and filtering), image understanding (shape-from-techniques, segmentation, spatial operations), and computer graphics (hidden surface algorithms, projections). Previous work includes parallel stereo³, scan-based models of computation⁴, algorithmic techniques for vision⁵, two-dimensional object recognition⁶, spatial reasoning^{7,8}, and terrain reconstruction and visualization⁹.

The organization of the paper is as follows: Section 2 reviews some basic programming concepts in *Lisp. After a brief discussion on the implementation of convolution and morphological operations, algorithms for computing connected components, distance transformations, Voronoi diagrams, and minimum cost paths based on a diffusion model of computation are described in Section 3. In diffusion, the computational wave spreads across the machine at a constant rate. This is contrasted with scan-based algorithms where the computation spreads across groups of processors in constant time. In Section 4, examples of scan-based algorithms for computing local means, connected components labeling, shape-from-shading, and hidden surface calculations are presented. The implementation of general input/output mappings, in particular input- and output-driven geometric transformations, using interprocessor communications operations is discussed in Section 5. Finally, a region-based model of computation implemented with interprocessor communications operations between virtual processor sets (i.e., different processing topologies simultaneously resident in the machine) is introduced. Algorithms for computing spatial properties over, and between connected regions are presented in Section 6. Appendix A describes a new algorithm for interpolating spatial data using Voronoi diagrams.

2. BASIC PROGRAMMING CONCEPTS

Currently, three high-level parallel programming languages are available for the CM: *Lisp, C* and CM-Fortran. *Lisp, a parallel extension of Common Lisp, involves the manipulation of parallel variables (pvars). Pvars are instantiated within virtual processor sets that define the number of dimensions as well as their size

(limited by physical memory). Thus, the CM hardware, which contains up to 64K physical processors (with 64K bits of memory per physical processor for the CM-2), can act like millions of virtual processors. In this paper, pvars (elements of pvars) are denoted by upper- (lower-) case Greek letters; e.g., the two-dimensional pvar, $A = \{\alpha(m,n)\}$. Several important data-parallel programming concepts in *Lisp are now briefly reviewed (see Ref. 10 for a more complete treatment).

Data parallel operations that operate upon and return pvars are indicated by !!. For example, the form (*set Γ (+! A B)) sets the pvar Γ to the result of adding pvars A and B together. Thus, operations such as adding two images (pvars) together are trivial on the CM. Reduction operations like *min, *sum, and *and return scalars, e.g., (*and A) returns t if all the $\alpha(m,n)$ are non-nil; otherwise it returns nil. Spreading does the opposite, setting some or all of the processors in a pvar to a scalar, e.g., (*set A (! 1)) sets all the processors in A equal to one. Although the CM is a SIMD machine, not all of its processors need be active at once. Conditionals such as if!!, and cond!! change the currently selected set (css). The syntax is like standard Lisp, e.g.,

```
(*set  $\Gamma$  (cond!! ((=! A B) (! 0)) (t! (! 1))))
```

sets $\gamma(m,n)$ to zero if $\alpha(m,n) = \beta(m,n)$, and one otherwise, for all (m,n). Conditionals like *if, *cond, and *when are used to manipulate pvars via side effects, e.g.,

```
(*cond ((=! A B) (*set  $\Gamma$  (! 0))) (t! (*set  $\Gamma$  (! 1))))
```

produces the same result as the previous expression. It is important to note that unlike standard Lisp all clauses in a conditional *Lisp expression are evaluated, each over a different set of processors.

In the sections to come several other important programming concepts are introduced during the course of the discussion: scans, interprocessor communications operations (including relative addressing for local operations), multiple virtual processor (vp) sets, and interprocessor communication between vp-sets.

3. LOCAL OPERATIONS

Local operations include a large number of image processing functions such as linear filtering, computing local statistics, and morphological operations. Direct implementations using relative addressing are described first. Then an iterated local model of computation similar to diffusion is introduced. The implementation of a number of important operations such as computing connected components, distance transforms, Voronoi diagrams, and finding minimum cost paths within this model is discussed.

3.1 CONVOLUTION

The operation news!! provides a means for addressing information locally, e.g., in the north, east, west, and south (NEWS) directions of a 2-d grid. The form (news!! A m n) returns a pvar that is equal to A offset by m and n in the x and y directions, respectively. Local operations using news!! can be performed in time proportional to the area of the window. Thus, linear convolution filters are easy to implement:

```
(*set B (! 0))
(dotimes (m M)
  (dotimes (n N)
    (*set B (+! B (*! (news!! A m n) (! (aref F m n))))))
```

where A and B are the input and output pvars and $F = \{f_{mn}\}$ is the kernel. Local statistics such as the mean, variance, min, and max can be implemented in a similar fashion.

3.2 MORPHOLOGICAL OPERATIONS

Dilations and erosions with arbitrary structuring elements can be computed in much the same way as linear convolutions. Let S be the structuring element. In two-dimensions, binary dilations and erosions of the image I are given by $\min \{i(m-m',n-n')\}$, and $\max \{i(m-m',n-n')\}$ for $\{(m',n') \in S\}$ and can be computed in $O(r^2)$ time where r is the radius of the structuring element. It is noted that if a morphological operation can be decomposed as

$$I \bullet S = I \bullet rS' = [(I \bullet S') \bullet S' \dots \bullet S']$$

r-1 times

where \bullet is either dilation or erosion, morphological operations with large structuring elements can be computed recursively in $O(r)$ time.

3.3 DIFFUSION MODEL OF COMPUTATION

Heat diffusion is often used as a model for interpolating sparse and irregularly spaced data. Diffusion is also a useful data-parallel model for computation. Diffusion defined on a 2-d grid is given by

$$u(m,n,t+1) = (1/4) \sum_{m',n'} u(m-m',n-n',t)$$

where t is time, and (m',n') are the four nearest neighbors. In diffusion, the "computational wave" propagates at a constant rate (Fig. 1) emanating from all processors simultaneously. A popular method for approximating large Gaussian convolutions is by diffusion. The rest of this section shows how diffusion-like algorithms can be used for computing connected components, distance transformations, Voronoi diagrams, and finding minimum path solutions.

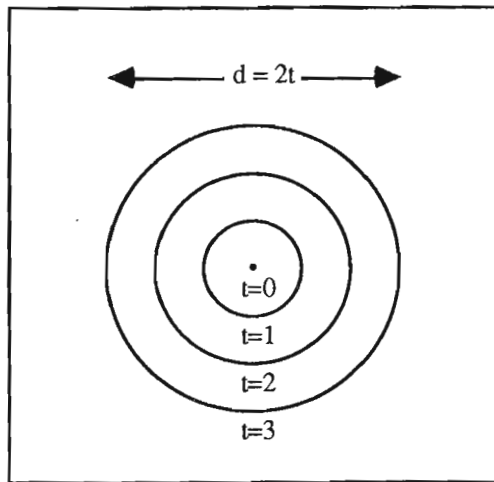


Fig. 1 Diffusion-like model of computation. Information propagates at a constant rate.

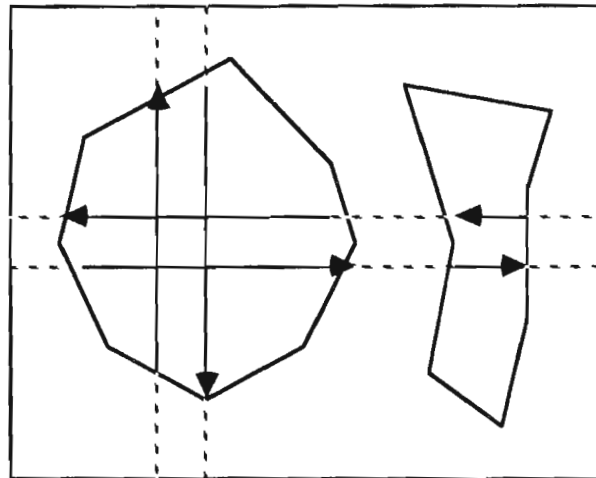


Fig. 2 Scan-based model of computation. Information propagates across regions in constant time.

3.4 CONNECTED COMPONENT LABELING

If q and r are any two pixel locations in an image, p and q are said to be connected if there exists a set of pixels $\{p_0, p_1, \dots, p_{T-1}, p_T\}$ where $q = p_0$ and $r = p_T$ such that p_{t-1} and p_t are neighbors. Connected sets of pixels with the same value in A (the input pvar) are computed by first assigning a different number to each processor in

the output pvar, Λ and then repeating the following operation until the output does not change:

For each neighbor (m',n') of (m,n)
 If $\alpha(m,n) = \alpha(m',n')$ then $\lambda(m,n) = \max \{ \lambda(m,n), \lambda(m',n') \}$.

The algorithm will always stop in an amount of time proportional to the radius of the largest region.

3.5 DISTANCE TRANSFORMATION

Again let q and r be any two pixel locations in an image. If $\{p_1, p_2, \dots, p_{T-1}\}$ are the set of pixels between q and r where $q = p_0$, $r = p_T$, and p_{t-1} and p_t are neighbors, the distance between p and q is defined to be the length of the shortest path (i.e., the one with the smallest T). The distance transform¹¹ of an image is defined to be the minimum distance to any pixel in the set $Q = \{q_k\}$. Initially, all points in Q are marked with zeros in the output Δ ; points not in Q are set to a constant $\geq (M+N)$. Then, the following operation is repeated until Δ does not change:

For each neighbor (m',n') of (m,n)
 If $\delta(m,n) > \delta(m',n') + 1$
 Then $\delta(m,n) = \delta(m',n') + 1$.

The algorithm will terminate after, at most, $(M+N)$ iterations.

3.6 VORONOI DIAGRAMS

By combining the previous two algorithms, the Voronoi diagram of Q can be computed in, at most, $O(M+N)$ time as follows. Let Λ be a label map, where $\lambda(m,n) = \lambda_k$, the label of the k^{th} point in Q . If, at each iteration, the following local operation is performed:

For each neighbor (m',n') of (m,n)
 If $\lambda(m,n) \neq \lambda(m',n')$ and $\delta(m,n) > \delta(m',n') + 1$
 Then $\lambda(m,n) = \lambda(m',n')$ and $\delta(m,n) = \delta(m',n') + 1$

the label map, upon completion of the algorithm will be the Voronoi diagram of Q .

3.7 COMPUTING MINIMUM COST PATHS

Finally it is noted that by using a diffusion model, minimum cost path problems¹² can be solved in $O(M+N)$ time as well. The minimum cost array with respect to some point set Q is computed in $O(M+N)$ time using a modified distance transform:

For each neighbor (m',n') of (m,n)
 If $\delta(m,n) > \delta(m',n') + \chi(m',n')$
 Then $\delta(m,n) = \delta(m',n') + \chi(m',n')$

where $X = \chi(m,n)$ is the cost array. The minimum cost path from any point p to the nearest point in Q is computed serially by moving in the direction of decreasing δ which takes, at most, $(M+N)$ steps.

4. SCAN-BASED ALGORITHMS

A disadvantage of the diffusion model is that the computational wave spreads at a constant rate across the machine. This is contrasted with scan-based algorithms where the computation spreads across processor sets (e.g.,

runs of pixels in the NEWS directions) in constant time (Fig. 2).

Scans involve the application of a binary associative operator \oplus (e.g., plus, and, or, min, max, or copy) to the set of elements $[a_1, a_2, \dots, a_n]$ to produce the set $[a_1, (a_1 \oplus a_2), \dots, (a_1 \oplus a_2 \oplus \dots \oplus a_n)]$. Scans are unit time operations on the CM, performed on pvars along any dimension in either the forward or reverse direction and can be conditioned on other pvars. See Blleloch (Ref. 4) for a more complete discussion.

a) processor number	0	1	2	3	4	5	6	7
b) value pvar, A	1	2	3	4	5	6	7	8
c) plus-scan (left to right)	1	3	6	10	15	21	28	36
d) segment pvar, B	nil	t	nil	t	nil	t	nil	t
e) segmented copy-scan (right to left)	2	2	4	4	6	6	8	8

Fig. 3 Examples of scan operations

With reference to Fig. 3, assume eight processors (a) with values (b). The result of a plus-scan (left to right) is shown in (c). Given the segment pvar (d), a segmented copy-scan (right to left) shown on the last line. It is implemented in *Lisp as

```
(scan!! A 'copy :segment-pvar B :direction :backward)
```

where A is the value pvar and B is the segment pvar.

4.1 LOCAL MEANS

Local means can be computed in two scans of an image, one for each dimension. In one dimension a plus-scan, say left to right is performed on the image $I = \{t(m,n)\}$:

$$\alpha(m,n) = \sum_{m'=0}^m t(m',n)$$

and shifted versions are subtracted to produce the M by 1 local mean:

$$\{\alpha[m+(M-1)/2,n] - \alpha[m-1-(M-1)/2,n]\}/M = t_{M \times 1}(m,n)$$

for M odd. Repeating this in the other direction produces the M by N local mean

$$\{\beta[m,n+(N-1)/2] - \beta[m,n-1-(N-1)/2]\}/N = t_{M \times N}(m,n)$$

where

$$\beta(m,n) = \sum_{n'=0}^n t_{M \times 1}(m,n')$$

Using scans, local means can thus be computed in constant time (i.e., independent of the window size). Local means can be used to compute the local variance for building image restoration and enhancement functions.

4.2 CONNECTED COMPONENT LABELING

Scans can be used to greatly improve the speed of the connected component labeler discussed earlier. Instead of updating the labels of the nearest neighbors, a scan-based implementation updates labels along horizontal and vertical runs of connected pixels. For example, in the west-to-east direction a segmented max-scan is performed left to right (forward direction in x):

$$\begin{aligned} \text{If } |\alpha(m,n) - \alpha(m+1,n)| < \alpha_0 \\ \alpha(m,n) = \max_{m' < m} \alpha(m',n) \end{aligned}$$

where α_0 is a threshold. For connected component labeling $\alpha_0 = 0$; for region-growing α_0 controls the formation of regions. The scan-based algorithm performs similar operations repeatedly over the four directions until the labels do not change. A comparison between diffusion- and scan-based implementations of a connected component labeler is provided in Table 1. For small, or complex non-compact regions, the results are comparable. The scan-based algorithm shows the greatest benefit for large compact regions.

<u>Data</u>	<u>Diffusion-based</u>	<u>Scan-based</u>
Small regions	0.8s	0.5s
Large regions	17.5s	0.4s
Complex, non-compact regions	9.2s	2.2s

Table 1 Timings for diffusion- and scan-based connected components algorithms for a 512² image on an 8K CM-2

4.3 SHAPE-FROM-SHADING

Shape-from-shading techniques involve the estimation of either the 2-d gradient field of a 3-d surface or the height of the surface itself from shading information in one or more image. Most approaches to single image shape-from-shading use either some form of numerical integration or constrained optimization technique. A particularly simple form of the numerical integration approach that involves summing brightness values along parallel strips in the image is described in Ref. 9. If the image is rotated so that the illuminant is to the left, it can be shown that the elevation map $Z = \zeta(m,n)$ can be determined, up to a scale factor and an offset by,

$$\zeta(m,n) = \zeta(0,n) + (1/\rho \sin\sigma) \sum_{m'=1}^m [I(m',n) - \rho \cos\sigma]$$

where ρ is a scale factor related to the albedo, σ is the slant angle of the illuminant, and $I = I(m,n)$ is the rotated image (the m and n indices correspond to the x and y directions, respectively). The offsets $\zeta(0,n)$ for $n = 0,1,\dots$ and the scale factor ρ are unknown.

The implementation of the strip integration method by scans involves subtracting the average brightness row by row by performing a plus scan left to right, dividing the result in the rightmost processor by the width of the image, performing a copy scan right to left, and subtracting this result from the image. The actual integration is performed left to right using a plus scan. Row offsets are adjusted by subtracting the average elevation from each row which requires two more scans. Finally, linear smoothing is performed using another scan in the y -direction to reduce striping between rows.

4.4 HIDDEN SURFACE CALCULATIONS

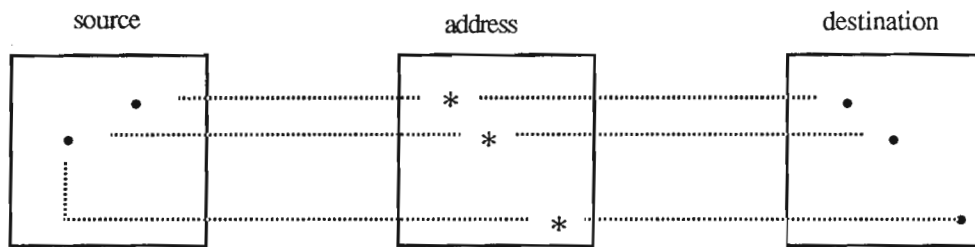
Calculating hidden surfaces is important in computer graphics scene rendering algorithms. Let $t(m,n)$ and $\zeta(m,n)$ be the image and corresponding elevation map rotated so the viewer is below (i.e., along the negative y-axis). If the viewer is at the horizon and is far enough away for a parallel projection to hold, only those portions of the scene whose elevation

$$\zeta(m,n) > \max_{n' < n} \{ \zeta(m,n') \}$$

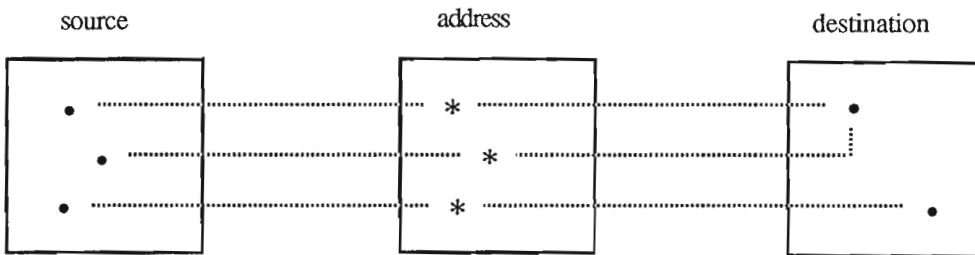
will be visible. This is a trivial hidden surface computation that can be performed with a max scan bottom to top on the elevations, followed by a comparison of this result with the original elevations. The more general situation of a viewer located at an angle ϕ_V with respect to the zenith can be reduced to the above case by rotating the elevation surface:

$$\zeta'(m,n) = n \cos\phi_V + \zeta(m,n) \sin\phi_V.$$

As the viewing angle moves towards the zenith, more of the scene will become visible.



a) Schematic of prefetch operation. The address pvar contains source addresses. Collisions due to multiple reads occur if a destination element reads from more than one source.



b) Schematic of pset operation. The address pvar contains destination addresses. Collisions due to multiple writes occur if a source element writes to more than one destination.

Fig. 4 Interprocessor communication operations

5. GEOMETRIC TRANSFORMATIONS

Thus far, communication between nearby or contiguous processors using news!! and scan!! has been considered. This section addresses the implementation of geometric transformations such as image warping and computing projections in computer graphics. In order to perform these as well as other transformations on pvars (e.g., Hough transforms⁵), more general ways of communicating between processors are needed.

Consider the input/output mapping $O = \mathcal{F} I$ where O and I are pvars. In general, there are two ways to implement the mapping: 1) for each processor in the input, write its value to the output processor specified by the transformation (input-driven), and 2) for each processor in the output read its value from the input processor specified by the inverse transformation (output-driven). The advantage of the input-driven approach is that the input/output mapping can be implemented directly (using *pset as described below). A disadvantage is that interpolation in the output space depends on the data which is usually irregularly spaced. This adds certain complications to the interpolation algorithm. An alternative is to solve for the inverse mapping \mathcal{F}^{-1} and use pref!! (see below). This solves the interpolation problem since interpolation is easier in the input space where the data are regularly-spaced; however, the inverse may be difficult to compute or may not be unique.

Pref!! and *pset are two basic functions for reading from, and writing to, pvars (see Fig. 4). The form (*set O (pref!! I Φ)) is used in input-driven transformations. It sets the pvar O to the value of I obtained from the processors addressed by Φ (i.e., \mathcal{F}^{-1}). The form (*pset :overwrite I O Φ) is used in output-driven transformations. It copies the value of I and writes it as the value of O in each processor referenced by Φ (given here by \mathcal{F}). Collisions occur when more than one output processor tries to read from the same input processor, or when more than one input processor tries to write into the same output processor. Collisions are handled in the software at the expense of memory and/or speed. In the above example :overwrite specifies one way of handling multiple collisions; other options include: max, min, and add. This rest of this section discusses implementations of input- and output-driven geometric transformations using *pset and pref!!.

5.1 AFFINE TRANSFORMATIONS

2-d polynomial transformations are often used in image processing for registering two images, and for correcting for relief displacement and sensor distortions. Constant scaling, translation, and rotation can be accomplished with the first order (affine) transformation

$$\begin{aligned} x' &= ax + by + c \\ y' &= dx + ey + f \end{aligned}$$

where (x,y) and (x',y') are input and output pixel coordinates. An output-driven implementation which uses the pref!! operation described earlier is depicted in Fig. 5. The positions of all output pixels are found in the input by solving the inverse transformation for (x,y) in terms of (x',y') . Output pixel values may be computed by finding the nearest input pixel and using its value (nearest neighbor interpolation), or using a weighted combination of nearby input pixels. For example, bilinear interpolation computes each output pixel from the values of its four nearest

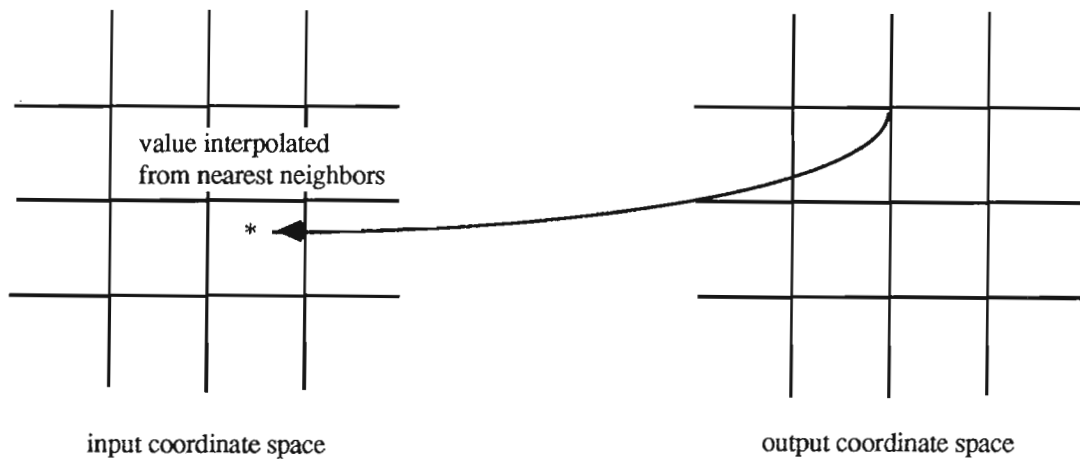


Fig. 5 Output-driven geometric transformation used in 2-d affine warper.

neighbors in the input. This implementation of the affine transformation executes in an amount of time related to the number of collisions (the number of times output processors try to read from the same input processor). Typical run-times are shown in Table 2.

<u>Operation</u>	<u>Reads/Processor (average)</u>	<u>Time (sec.)</u>
translation	1	1.4
rotation (45°)	1	2.1
scale by 1/2	1	1.4
scale by 1	1	1.3
scale by 2	4	4
scale by 4	16	18

Table 2 Example timings for 2-d affine transformation on 512² image with 8192 processor CM-2

5.2 OBLIQUE PARALLEL PROJECTIONS

Oblique parallel projections simulate what an observer sees through a telephoto lens. Assume the image and elevation surface are both rotated so that the observer is along the y axis. Then the visible pixels can be determined using the scan-based algorithm described in Section 4.4. Visible pixels are then projected onto the viewing plane by the transformation

$$\begin{aligned} x' &= x \\ y' &= y \cos\phi_v + z \sin\phi_v. \end{aligned}$$

This transformation is input-driven and is implemented using *pset as shown in Fig. 6. Since the hidden surface algorithm removes those pixels that would otherwise write over the visible pixels in the output plane, no collisions in writing occur; however, not every output processor will be set so intermediate pixels must be interpolated in the y direction. A simple way to do this is to perform two copy scans up-down and down-up, and take the minimum. (This insures that the foreground and background pixels are black.)

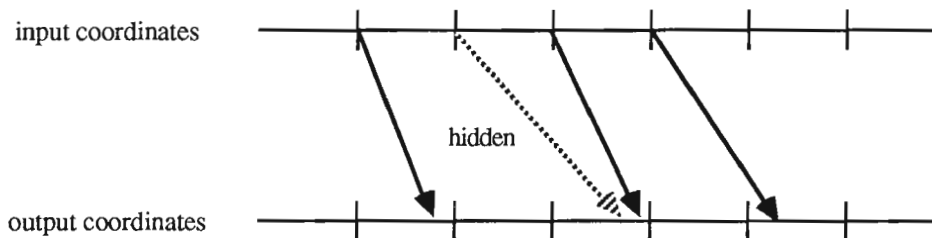


Fig. 6 Input-driven geometric transformation used in oblique parallel projection algorithm.

6. SPATIAL OPERATIONS

Computing properties of and between regions (points, lines, and areas) is necessary in image understanding and geographic information systems. Fig. 7 depicts a region-based model of computation where label maps control the flow of information between pvars in different vp-sets. Reduction and spreading within a vp-set was discussed briefly in Section 2. Within a vp-set information from multiple processors within the currently selected set (i.e., a region) can be reduced to a scalar (e.g., *sum), or conversely a scalar can be spread to one or more processors within the css (using !!). By stepping through each region, i.e., within a (*when (=!! Λ (!! λ_k)) ...) form, where Λ is the label map and λ_k is the k-th label, operations over regions can be performed in time proportional to the number of

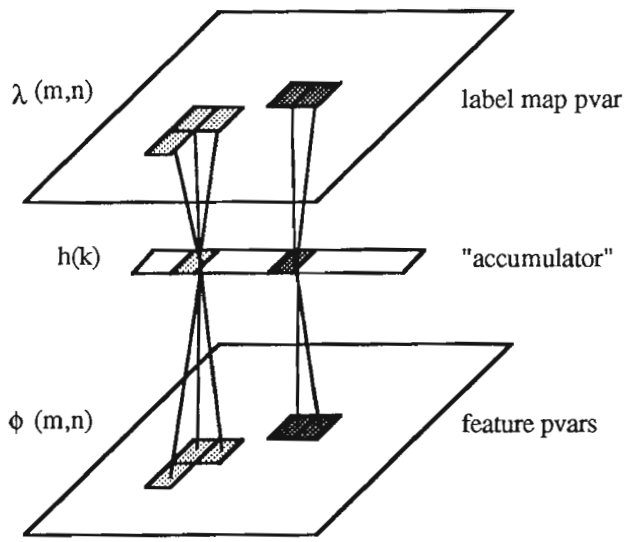


Fig. 7 Reduction and spreading between pvars in different vp-sets controlled by a label map.

regions, K . Through the use of interprocessor communications operations between vp-sets, information from multiple regions can be reduced to vectors and information from vectors can be spread over regions much more quickly, at rates that depend on the number of collisions that occur and the method used for handling collisions.

Let H be a pvar that has been instantiated in a one-dimensional vp-set, and let Φ and Λ be pvars that have been instantiated in a two-dimensional vp-set. Consider the form

```
(*with-vp-set (pvar-vp-set  $\Lambda$ )
  (*pset :add  $\Phi$   $\Lambda$   $H$ )
  (*set  $\Phi$  (pref!!  $H$   $\Lambda$ ))).
```

The form `(*with-vp-set (pvar-vp-set Λ) ...)` specifies the context within which subsequent expressions are evaluated. Here, `*pset` is evaluated within the 2-d vp-set corresponding to Λ . If $\Phi = (! 1)$, all the processors in Λ that are equal to k send a one to the processor in H whose address is equal to k . The contents of H is the histogram of Λ . Again, the k -th region is defined by all processors whose label $\lambda(m,n) = k$. Computing the histogram is an example of a reduction operation across vp-sets. The second expression within the above form spreads the histogram across vp-sets. In particular, it spreads $h(k)$ to all processors in Φ where $\lambda(m,n) = k$ for all k . Φ is thus the frequency of occurrence image of Λ .

6.1 UNARY OPERATORS

Unary operators compute properties of regions such as area, perimeter, average-over-region, and moments. The area Φ of each connected region in Λ is just the histogram of the label map spread to all processors in Φ with that label as discussed above. Computing the average Φ' of some property Φ over a region is similar. With reference to Fig. 7, two "accumulators" are required: one to tally the number of elements in each region (the area), and the other to sum the property over each region. The result of dividing the contents of the two accumulators is spread over Φ' . If Φ is equal to the processor's grid address in x or y , the result Φ' is the centroid in the x or y direction. Higher order moments can be computed the same way and used to build orientation, length/width, and elongatedness operators¹³.

The perimeter of regions can be computed by marking the border pixels for each region, counting the

number of border pixels, and spreading the count to all pixels in the region as above. From area and perimeter, region compactness ($\text{area}/\text{perimeter}^2$) may be computed.

6.2 N-ARY OPERATORS

N-ary operations compute properties between regions. The minimum distance Φ between regions in pvar Λ_1 to regions in pvar Λ_2 can be computed by marking the regions in Λ_1 with zeros and computing the distance transform (Section 3.5). Then, if Δ is the resultant distance image, the minimum distance Φ can be computed by

```
(*with-vp-set (pvar-vp-set  $\Delta$ )
  (*pset :min  $\Delta$  H  $\Lambda_2$ )
  (*set  $\Phi$  (pref!! H  $\Lambda_2$ )).
```

Regions in pvar Λ_1 that are adjacent to regions in pvar Λ_2 can be extracted by marking the border pixels of the regions in Λ_2 , accumulating in H the labels of their neighbors in Λ_1 , and "turning on" only those regions (labels) in Λ_1 that are in H.

To determine which regions in Λ_1 are contained in Λ_2 (where Λ_1 and Λ_2 are disjoint), a binary pvar is created that is set to zero where Λ_2 is non-zero and one elsewhere. The non-zero connected components of the binary pvar are computed and the label(s) of pixels on the border found. If the label of a pixel belonging to Λ_1 is equal to the label of a border pixel, then it is not contained in Λ_2 .

7. SUMMARY

This paper discussed data-parallel algorithms for image computing on the CM. Algorithms for connected components labeling, distance transformation, Voronoi diagrams, geometric interpolation, and finding minimum cost paths based on a local (diffusion-like) model of computation were described. A disadvantage of the diffusion model is that the computational wave spreads slowly across the machine with typical run-times on the order of the size of the largest region in the image. The use of scanning to speed the spread of computation across the machine was then discussed. Examples of scan-based algorithms for local means, connected components labeling, shape-from-shading, and hidden surface calculations were presented. Geometric transformations for image warping and oblique parallel projection implemented with interprocessor communications operations were shown to run in an amount of time proportional to the number of collisions. Finally, algorithms for computing spatial properties of, and between regions were presented.

APPENDIX A GEOMETRIC INTERPOLATION USING THE VORONOI DIAGRAM

With reference to Section 3.6, if the labels $\{\lambda_k\}$ of the point set Q are treated as numerical quantities, interpolation can be performed geometrically via Voronoi diagrams. It is conjectured that the method yields a good approximation to the steady state solution of the heat equation. To illustrate the technique, consider a one-dimensional example (Fig. A-1). The Voronoi diagram of the initial point set (a) is shown in (b). Next for those points along the borders, set their value to be the average of their neighbors, and add these points to the original point set (c). This creates a new point set half-way between the original point sets both in distance and in value. The Voronoi diagram of this second point set is shown in (d). If this process is repeated until the largest region is split, the final result (e) is obtained. Fig. A-2 shows a two-dimensional example of a hemispherical surface (a) and results of interpolating a random sampling of the surface by diffusion (b) and geometrically via the Voronoi diagram (c and d). In contrast with standard diffusion, the Voronoi technique produces good results after a small number of iterations as shown in the figure.

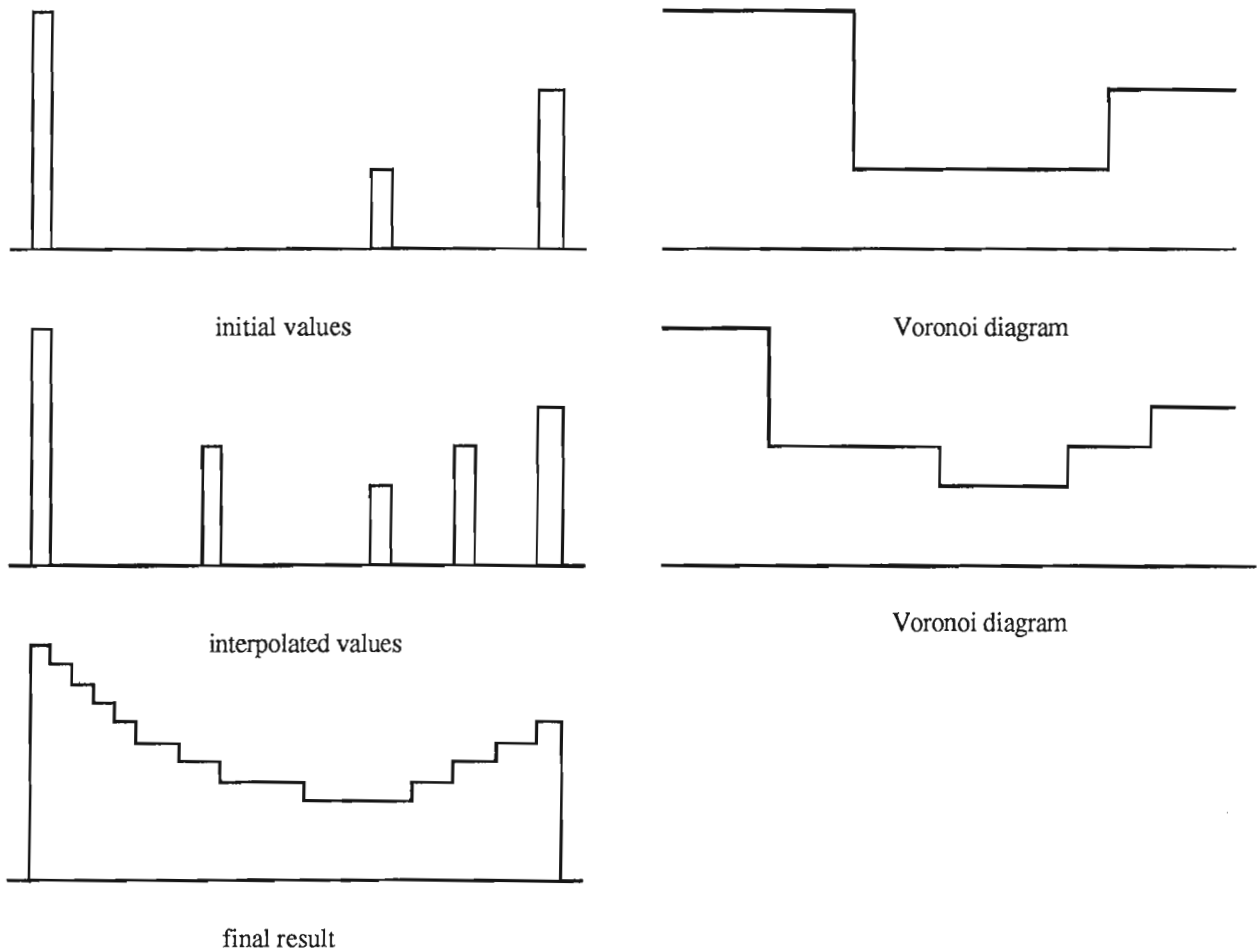


Fig. A-1 One-dimensional example of geometric interpolation via the Voronoi diagram

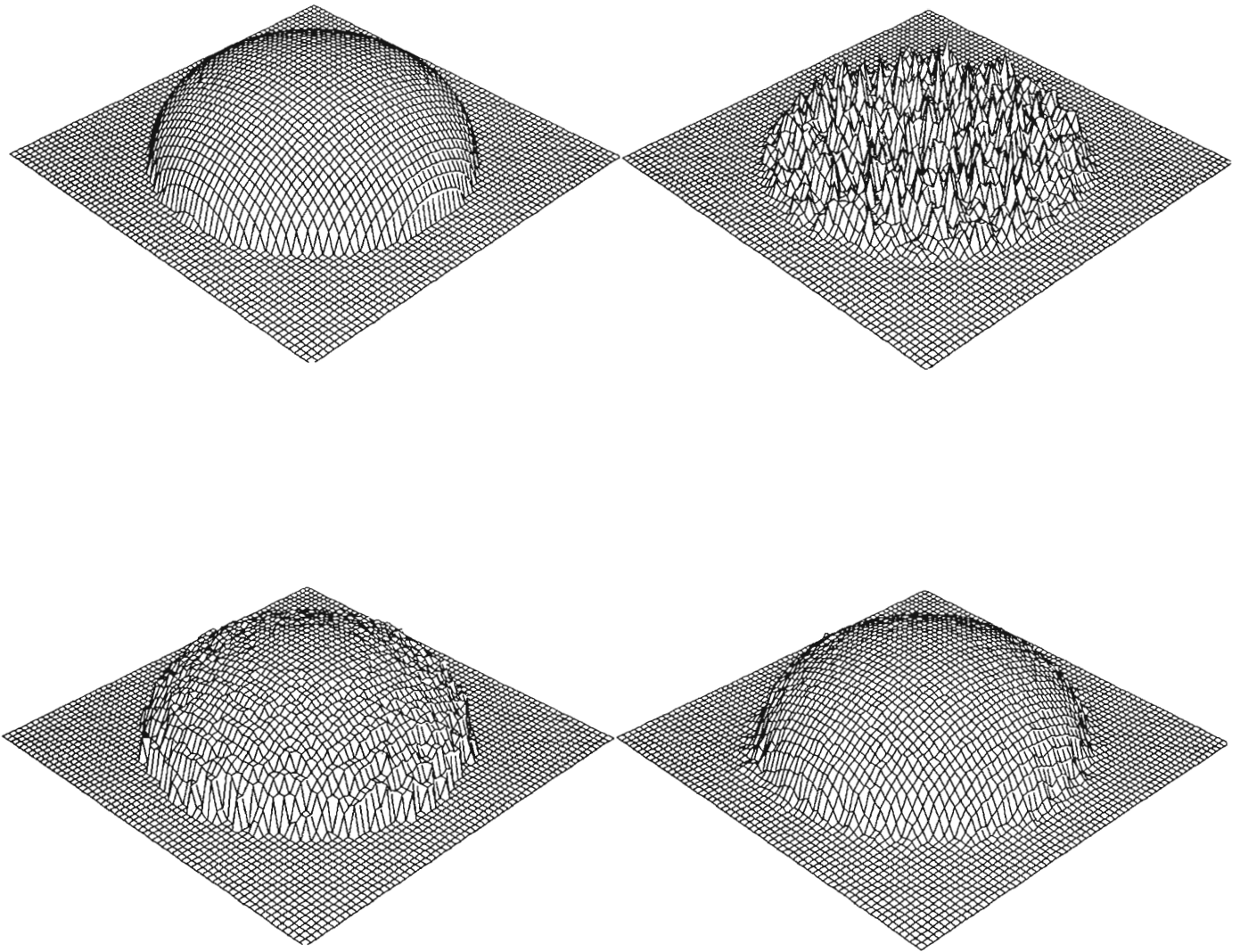


Fig. A-2 Example of geometric interpolation. Hemispherical surface (upper left). Reconstructed surfaces from random sampling (every 100 points): after 100 iterations of standard diffusion algorithm (upper right), and after 1 iteration (lower left) and 10 iterations (lower right) of Voronoi-based interpolator.

REFERENCES

1. W.D.Hillis and G.L.Steele, Jr., "Data-parallel algorithms," Comm. ACM, Vol. 29, No. 12, Dec. 1986.
2. L.W. Tucker and G.G.Robertson, "Architecture and applications of the Connection Machine," Computer, Vol. 21, No. 8, August, 1988.
3. M.Drumheller and T.Poggio, "Parallel stereo," Proc. IEEE Conf. on Robotics and Automation, San Francisco CA, 1986.
4. G.E.Blelloch, "Scans as primitive parallel operations," Proc. Inter. Conf. on Parallel Processing, pp 355-362, August 1986.
5. J. Little, G.E.Blelloch, and T.Cass, "Parallel algorithms for computer vision on the Connection Machine," Proc. Inter. Conf. on Computer Vision, pp 587-591, 1987.
6. L.Tucker, C.Feynman, and D.Fritzsche, "Object recognition on the Connection Machine," IEEE Conf. Computer Vision and Pattern Recognition, Ann Arbor MI, 1988.
7. M.J.Carlotto, "A homogeneous computational model for spatial inference on massively-parallel architectures," Proc. 2nd Symposium on the Frontiers of Massively Parallel Computation, Oct. 1988.
8. D.Izraelevitz, "A fast algorithm for Voronoi diagram calculation based on distance doubling," Proc. 2nd Symposium on the Frontiers of Massively Parallel Computation, Oct. 1988.
9. M.J.Carlotto and K.Hartt, "Connection Machine system for planetary terrain reconstruction and visualization," Proc. SPIE Symposium on Advances in Intelligent Robotics Systems, Vol. 1192, Nov. 1989.
10. *Lisp Reference Manual (Version 5.0), Thinking Machines Corp., Cambridge, MA, Sept. 1988.
11. A. Rosenfeld and J.L.Pfaltz, "Sequential operations in digital picture processing," Journal of the ACM, Vol. 13, pp 471-494, 1966.
12. M.A.Fischler, J.M.Tenenbaum, and H.C.Wolf, "Detection of roads and linear structures in low-resolution aerial imagery using a multisource knowledge integration technique," Computer Graphics and Image Processing, Vol. 15, pp 201-223, 1981.
13. P.Winston and B.K.P.Horn, Lisp, Addison Wesley, Reading MA, 1981.