A Parallel Algorithm for Incremental Stereo Matching on SIMD Machines

Andrew F. Laine and Gruia-Catalin Roman

Abstract—**The goal of stereo vision is the recovery of depth information from the relative lateral displacements in the positions of objects within a pair of images taken from slightly differing viewpoints. While recent stereo matching techniques have yielded improvements in reliability and speed, most of these algorithms fall short of the real-time stereo matching requirements for navigation systems, robot vision, machine inspection, and other areas of computer vision where rapid response is critical. Traditionally, matching algorithms have achieved high speeds through algorithm simplification and/or relied on custom hardware. The objective of our work has been the development of a robust high-speed stereo matcher by exploiting parallel algorithms executing on general-purpose SIMD machines. Our approach is based on several existing techniques dealing with the classification and evaluation of matches, the application of ordering constraints, and relaxation-based matching. The techniques have been integrated and reformulated in terms of parallel execution on a theoretical SIMD machine. An ideal machine topology for executing this parallel algorithm is identified through complexity analysis. Feasibility is demonstrated by implementation on a commercially available SIMD machine, and its performance is compared with that of the idealized machine. Sample results are shown for real and synthetic stereo pairs.**

I. Introduction

THE goal of stereo vision is the recovery of depth information from the relative lateral displacements in the positions of objects within a pair of images taken from slightly differing viewpoints. The recovered depth information may then be used to reconstruct the three-dimensional structure of a scene.

The typical stereo vision technique assumes the availability of only two images, by analogy to human binocular vision (stereopsis). The usual paradigm for stereo vision algorithms include the following steps: 1) Features are identified and extracted from each image view independently. 2) Extracted features are then matched such that they are projections of the same real-world coordinate. 3) The disparity between matched features is used, along with known camera geometry, to compute the depth of the matched features. While all of the above steps are important, the fundamental problem in stereo vision is step two, the matching of corresponding points in the different views. Because of possible occlusions, not all points may have matches and context information is used to infer the depth of the unmatched points [1]. Techniques differ in the strategy they follow with regard to the generation of a unique and consistent set of matches.

Barnard and Fischler [2] surveyed previous stereo vision techniques. Most approaches may be classified as area based or feature based. Area-based techniques rely on the surface continuity assumption and often involve correlation-based matching. Feature-based approaches focus on intensity variations that correspond to physical and geometric properties and intensity anomalies that may not have any physical relevance. Matching is often done at the symbolic level [6]. Much effort has been devoted to the study of feature-based techniques [3], [4], [22], [23] because they provide better localization and exploit more contextual information. In this paper, we have focused on feature-based methods because we are interested in extracting depth information from urban scenes containing mostly man-made features, with many surface discontinuities.

Recent techniques have been able to reduce the search space required to maintain global consistency between matches and have yielded speed improvements without compromising reliability [9], [27], [29]. However, these algorithms still fall short of the real-time stereo matching requirements for navigation systems, robot vision, machine inspection, and other areas of computer vision where rapid response is critical.

Some matching algorithms have achieved high speeds through algorithm simplification [11] and/or relied on custom hardware [10]. Drumheller and Poggio [24] demonstrated the feasibility of mapping a simple stereo matching algorithm onto a commercially available parallel (SIMD) architecture. While our work also focuses on the development of parallel algorithms for real-time stereo matching, we target them for execution on a general class of SIMD machines. Our approach is based on several existing techniques dealing with the classification and evaluation of matches [12], the application of ordering constraints [8], [28], and relaxation-based matching [13]. The techniques have been integrated and reformulated in terms of parallel execution on a theoretical SIMD machine. Complexity analysis is shown for each step that takes into account parameters of the machine model and maximum disparity. The algorithm was then implemented on a commercially available SIMD machine.

Our matching algorithm assumes an epipolar camera model. Processing is divided into two phases. In phase one, unlikely matches are first discarded based on a loose geometric constraint and the ordering of any previous matches. The remaining matches are then evaluated using criteria based on precomputed similarity measures (such as direction and intensity on each side of an edge). Each set of matches, perhaps containing several candidate matches, is classified based on the aggregate of the previously evaluated candidate matches. Finally, each set of matches is sorted and truncated so that it contains no more than three of the most likely candidates.

Phase two computes initial estimates of the probability of each possible match based on the individual evaluation of the match and the classification of its set. These initial estimates are refined during a relaxation process, by a consistency rule that successively increases (decreases) the probability of matches if nearby points have similar (different) disparity. Afterward, accepted matches are identified and the entire algorithm may be repeated. At the start of each iteration, previously accepted matches may

Manuscript received April 5, 1989; revised September 10, 1990. An abridged version of this paper was presented at the IEEE 10th International Conference on Pattern Recognition, Atlantic City, NJ, June 16–21, 1990.

A. F. Laine is with the Computer and Information Sciences Department, Center for Computer Vision Research, University of Florida, Gainesville, FL 32611-2024.

G.-C. Roman is with the Department of Computer Science, Washington University, Saint Louis, MO 63130-4899.

IEEE Log Number 9041095.

1042-296X/91/0200-0123$01.00 ©1991 IEEE

be used to provide a context for the incremental accumulation of new matches.

In contrast to our approach, Drumheller and Poggio use zero crossings [5], [25] and sign of convolution as features for matching. Potential matches are saved if they have the same zero-crossing values. Next, local support for each candidate pair is gathered by applying a constraint on disparity continuity. The support function for computing smoothness is executed in parallel by a three-dimensional convolution operation on a Connection Machine. Finally, matches are selected based on the amount of local support gathered for disparity continuity, uniqueness and an ordering constraint [8].

The machine model on which our parallel algorithm has been formulated assumes a 2-D array of pipelined processors and a set of memory arrays that may be read and/or updated during each machine cycle. Stage processors are capable of performing four kinds of operations: logical, integer arithmetic, max/min, and functions of one variable. Model parameters include the number of stages per pipeline, input and output bandwidth, and stage interconnection bandwidth.

The performance of the parallel algorithm is constant time and is directly related to maximum disparity (δ pixels) on an idealized machine having 2δ pipelines, four stages per pipeline, 16-bit interconnection links, and an output bus no wider than six pipelines, (6×16) bits. The parallel algorithm has been implemented as part of an interactive environment [11] consisting of a stereo workstation [7], driven by dual Gould/DeAnza IP-8500's, and a MicroVax II host. Performance measurements show that, on a typical commercial pipelined SIMD machine, the algorithm accomplishes the match of a $512 \times 512 \times 8$-bit pair of stereo images with a maximum disparity range (δ) of 32 pixels in less than 30 s.

The remaining sections of this paper are organized as follows. Section II presents an informal overview of the matching strategy. Section III provides a formal description of the SIMD machine model. Section IV contains a detailed description of the parallel algorithm and the resources it requires. Section V compares the performance analysis for a theoretically ideal machine and an actual implementation. Sections VI presents sample results obtained with both real and synthetic stereo pairs. Finally, a discussion and conclusions are presented in Section VII.

II. Matching Strategy

This section presents an overview of the matching process including all necessary preprocessing steps. We assume an epipolar camera model (i.e., the horizontal scan lines of both cameras are parallel to the baseline so that all disparities are horizontal) and constrain searches for candidate matches to some predetermined disparity range. Although our method performs bidirectional matching, for the sake of clarity, we will restrict our discussion to matching features from the left image to features in the right image. While the primitive features matched are edgels (edge pixels), the similarity measures used to compare features are based on the properties of edge segments (i.e., chains of edgels). The paragraphs below describe the preprocessing steps, candidate selection criteria, and relaxation method used to produce a consistent set of matched features.

A. Preprocessing

Edgels with magnitude and direction information are produced by a Kirsh edge detector [15]. Edge thinning is accomplished using nonmaxima absorption in the gradient direction [16]. The edge magnitudes are then thresholded and the resulting binary edgels are thinned [14] producing eight connected edge segments one pixel wide. Segments are then converted into chains by deleting junctions (of the forms T, Y, X, +) and linking edgels such that each edgel has at most two neighbors in the same chain. Nonhorizontal chains have at most one pixel per scanline, and horizontal and nearly horizontal chains have at most one pixel per column. Subchains consisting of edgels all lying on the same epipolar line are treated for matching purposes as single points and are represented by the center edgel in the subchain.

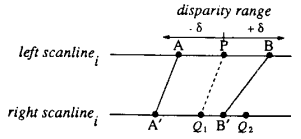

Fig. 1. An intra-scanline ordering constraint.

Next, for each chain in the image we compute its direction and the average intensity along its left and right sides. These properties are used later to discriminate between alternate candidates. The precomputed properties are stored as *characteristic images*. A characteristic image associates with each edgel in a chain a value in the range (1–255) corresponding to some property of the whole chain to which the edgel belongs. Image points that do not belong to a chain are assigned the value zero.

B. Candidate Selection

Input to the matcher consists of a set of selected edgels to be matched, three sets of characteristic images, and (optionally) a set of previously matched edgels. On each scanline, intervals between any previously matched edgels are labeled sequentially to provide a global context in which to embed new matches. Previously matched edgels are then removed from the set of selected edgels to avoid the possibility of rematching. An initial set of possible matches is constructed by pairing each edgel in the left image with every edgel from the right image which is on the same epipolar line and within some distance δ of the location of the edgel in the left image. Let us consider one such edgel P (from the left image) and its associated pool of candidates $\{Q_1, Q_2, \ldots, Q_n\}$. Some of these candidates may be eliminated by the ordering constraint imposed by previous matches. For example, Fig. 1 shows that previously matched edge pairs $\langle A, A' \rangle$ and $\langle B, B' \rangle$ provide a context that makes candidate pair $\langle P, Q_1 \rangle$ unique within disparity range δ. Candidate Q_2 is eliminated from consideration although it lies within δ of P.

Candidates are also eliminated from the pool by applying an orientation constraint. Given the candidate pair $\langle P, Q_1 \rangle$, if the absolute difference in the orientations of corresponding chains C_P and C_{Q_1} is greater than some threshold (e.g., 45 degrees), then candidate Q_1 is eliminated.

The remaining candidates within the pool are sorted by evaluating the degree of similarity between the precomputed properties of corresponding chains. Chains C_P and C_{Q_i}, associated with point P and candidate Q_i, respectively, are considered similar if the absolute difference between their property values is less than some threshold parameter. The evaluation process results in partitioning the pool of candidates into three disjoint sets. Considering again candidate pair $\langle P, Q_i \rangle$, if the corresponding chains C_P, C_{Q_i} are similar with respect to orientation and average intensity along the left and right sides, then candidate Q_i is considered credible and is assigned to the first partition. If the corresponding chains C_P and C_{Q_i} are similar in

orientation, but dissimilar only in left-side or right-side average intensity, then candidate Q_i is seen as less credible and is assigned to the second partition. All other candidates in the pool are assigned to the third partition and are treated as unreliable.

Next, the pool itself is classified based on the aggregate of its previously evaluated candidates. First, the pool may have a single (unique) candidate belonging to the first or second partition. Alternatively, the pool may contain a single candidate assigned to the first or second partition that is discernibly better than all other candidates within the pool. Finally, the pool may contain a single candidate belonging to the third partition or may have multiple candidates belonging to the first or second partitions. This classification information is used later to assign initial estimates of probabilities of candidates for relaxation.

Following the classification process, each pool is truncated so that it contains no more than three of the most promising candidates. Since more than one promising candidate are kept, the opportunity to reevaluate their credibility and to correct some false matches remains. We do this by employing a relaxation method that corrects most local errors [21] by relying on two types of continuity constraints.

C. Relaxation

Initial probabilities are assigned to each candidate in the pool using a simple weighting function that takes into account the candidate's evaluation and the classification of its pool. These initial estimates are iteratively refined by applying a consistency rule to all the candidates within the pool, updating the probability of each candidate, and normalizing the new probability estimates. The relaxation follows the procedure of Bernard [13] except that our consistency rule has been formulated to apply constraints on *figural continuity* as well as *disparity continuity*. Candidates that are continuously connected along the same figural contour are allowed to support each other in a cooperative sense. This is accomplished by constraining the region of support so that only neighbors *connected* above and below a candidate are allowed to contribute. Such connected neighbors with disparity differences of no more than one pixel are considered *consistent*. After a few iterations, consistent candidates *increase* in probability. Conversely, inconsistent candidates having no *connected* neighbors satisfying the disparity constraint, receive little support, and *decrease* in probability.

Although the relaxation procedure may be repeated until the network reaches steady state, in practice, only small changes in normalized probabilities are observed after a few iterations. For this reason, matches with probabilities greater than some empirically selected threshold (e.g., 0.7) are accepted. When used in an incremental paradigm, accepted matches are assigned a probability of 1.0, and saved. In subsequent iterations, the previous matches provide global support for any new locally consistent matches. Threshold values for differences in orientation, property values, and probability were treated as "universal" constants for all the image pairs tested in our investigation.

The same matching process is also applied to matching features in the right image with features in the left image, and a final set of matches is produced by selecting those pairs of edgels that survived both right-to-left and left-to-right matching processes.

III. Machine Model

The method given above has been cast as a parallel algorithm executing on a theoretical pipelined SIMD machine. As shown in Fig. 2, the machine consists of a two-dimensional array of processors and a set of dual ported memories. The machine is organized as a system of q pipelines, each consisting of m stages. Stages are linked to form a rectangular network of horizontal pipelines, with no feedback between stages. Data may flow from processor (i, j), the jth stage of pipeline i, to processors $(i, j+1)$, $(i+1, j+1)$, or $(i-1, j+1)$. Conversely, processor (i, j) may accept inputs from processors $(i, j-1)$, $(i+1, j-1)$, and $(i-1, j-1)$. Input data, supplied from as many as w memories ($w \leq 2q$) is processed and delivered to as many as r memories ($r \leq q$) within a single *machine cycle*. Each memory may be connected to several pipeline inputs and read simultaneously by the first stage of each pipeline. However, each memory may be connected to one pipeline output at most. A pipeline may deliver its data to more than one memory within the same machine cycle.

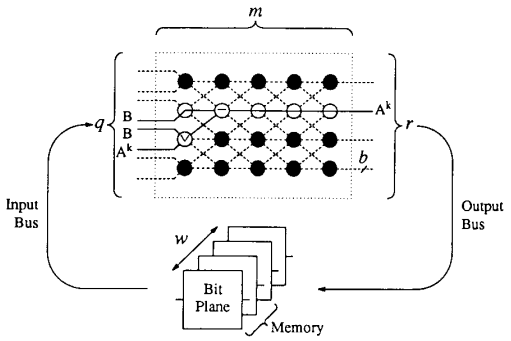

Fig. 2. Machine model of a parallel synchronous pipelined network.

The notion of a machine cycle characterizes the "coarse grain" clock speed of a parallel synchronous pipelined network. In this sense, a machine cycle is the maximum time required to process all the pixels from an image memory through each stage of the longest pipeline. For purposes of analysis, machine cycles are specified in the context of the model parameters image size (i, j), pipeline length (m), and interconnection path width (q). For example, consider a machine operating at 30 machine cycles per second, processing image memories 512 × 512 in size, with four stages per pipeline and 8-bit-wide interconnection links. To process all 242 144 pixels within 1/30 s (1 machine cycle at 30 Hz), a pipeline must process a single pixel within 1.28×10^{-7} s. Thus, for a pipeline consisting of four stages, the clock cycle of each stage can be no longer than 32 ns on the average.

For b-bit processors, all links (input, stage, output) are b bits wide, the input bus has a width less than or equal to $2qb$ bits and the output bus has a width less than or equal to qb bits. Internal processing within each stage includes the following simple operations: logical operations ($\wedge, \vee, \neg, \otimes$), integer arithmetic ($+, -, \times, \min, \max$), accumulation, and functions of one variable. Each processor may have no more than two forward input links[1] and one output link, active during any one machine cycle. We adopt the notation $A^{[1:N]}$ to denote a group of *contiguous* memories 1 through N. Each distinct memory A^n may consist of one or more contiguous bit-planes, depending on the data type stored. Thus, A^k identifies the kth memory of group A. When the name of the memory group A stands alone, it represents the complete set of contiguous memories $[1:N]$.

For purposes of making more obvious the mapping of the

[1]However, in practice, lateral inputs for bit-sliced arithmetic and carry operations are allowed when necessary.

algorithmic steps to the architecture, operations along the stages of each pipeline are contained between the delimiters "⟨" and "⟩", concurrent processing of more than one pipeline is specified by the symbol ‖, and memory coordinates ranging over an entire image memory are preceded by the symbol □. For example, suppose memories A^k and B contain Boolean data, 1 bit deep; then the statement below specifies the following computation (see Fig. 2):

$$\| k: 1 \le k \le N :: \langle \Box i, j :: A^k(i,j) := \langle A^k(i,j) \vee B(i+k, j) \rangle - B(i+k, j) \rangle .^2$$

Each (distinct) A^k is computed over a subnetwork of two pipelines, each having two stages, executing in parallel and synchronously. The first stage of one pipeline forms the logical union of memories A^k and B, with memory B shifted k pixels along the horizontal axis. The result is passed to the second stage of an adjacent pipeline where B is subtracted from the union, and its output delivered back to memory A^k. We need two pipelines because each stage may process no more than two inputs per operation. One way of linking the processors needed to carry out this computation is shown above in Fig. 2 for the kth network. The complete computation requires N identical pipelined subnetworks executing in parallel. For a real machine with q pipelines, where $q < 2N$, the operation requires $\lceil 2N/q \rceil$ passes. Memory $B(i+k)$ is read by the first stages of each distinct subnetwork simultaneously. All the pixels within a memory $(\Box i, j)$ are processed uniformly within each pipeline in a single machine cycle. Thus, all A^k's are updated in parallel in one machine cycle. In the next section, we use this notation to present a parallel version of the matching algorithm described earlier.

IV. Algorithm

In describing the algorithm, we will use L and R to refer to the sets of edgels belonging to chains selected from the left and right images. M_L and M_R will refer to any edgels previously matched. All these sets are encoded as binary two-dimensional arrays with the index ranging over the image space. $L(i, j)$, for instance, assumes the value "1" if the position (i, j) contains an edgel belonging to a chain selected from the left image and the value "0" otherwise. As described earlier, the first phase of the matching algorithm identifies the three most promising candidates within each match pool. The second phase applies a relaxation method to identify the most consistent of the three candidates.

A. Phase 1

For the sake of clarity, we restrict our discussion to matching features from the left image to features in the right image. Corresponding right-based matching may be performed in parallel and is symmetric with respect to the equations and steps below. The first phase is described in nine steps.

Step 0 – Label Intervals Between Previous Matches: The set of previously matched edgel pairs is a bijection. Therefore, on each epipolar line, the intervals between matched edgels match pairwise. In this step, we label the intervals between already matched edgels; we assign identical labels to all pixels within a given interval. The labels are used later to enforce an ordering constraint. The label sets are stored in memories B_L and B_R.

$$\Box i, j :: B_L(i,j) := \langle + \bmod (2^b - 1) y,\ x: (0 < y \le j) \wedge (0 < x \le i) :: M_L(x,y) \rangle \quad (1a)$$

$$\Box i, j :: B_R(i,j) := \langle + \bmod (2^b - 1) y,\ x: (0 < y \le j) \wedge (0 < x \le i) :: M_R(x,y) \rangle \quad (1b)$$

$B_L(i, j)$ is computed by counting the number of previously matched edgels from $M_L(0, 0)$ to $M_L(i, j)$. The result is that all pixels within matching intervals have the same label. All the labels stored in B_L and B_R are computed in parallel in one machine cycle. The range of label values is reduced to save storage. Memories B_L and B_R are no more than b bits wide, such that $(2^b - 1)$ is at least the length of a scanline. Thus label values remain unique within each scanline.

Step 1 – Eliminate Previous Matches: Any previous matches (stored in memories M_L and M_R) are eliminated from the set of selected edgels (L and R). Memories M_L and M_R supply input data for both pipelines simultaneously, allowing L and R to be modified in parallel.

$$\Box i, j :: \langle L(i,j) := \langle L(i,j) \vee M_L(i,j) \rangle - M_L(i,j) \rangle \quad (2a)$$

$$\Box i, j :: \langle R(i,j) := \langle R(i,j) \vee M_R(i,j) \rangle - M_R(i,j) \rangle . \quad (2b)$$

Step 2 – Build Disparity Map: For each edgel (i, j) in the left image and disparity k in the range 0 to δ, if an edgel exists in the right image, a bit is set in an image memory CP_L^k. For each match pool associated with an edgel in L, candidates from R having disparity k are identified by computing the conjunction of input edgel sets L and R, where R has been shifted k pixels along the epipolar axis. Shifts in k that index outside the sets L and R are ignored.

$$\| k: 0 < k < \delta :: \langle \Box i, j :: CP_L^k(i,j) := L(i,j) \wedge R((i+k), j) \rangle . \quad (3)$$

All candidates are identified in a single machine cycle by a stack of δ independent pipelines executing in parallel.

Step 3 – Apply Ordering Constraint: In this step, candidates are eliminated from further consideration if they do not fall within matching intervals, i.e., if they do not have the same label number. Candidates not violating the ordering constraint simply pass through and are returned to memory CP_L for further processing.

$$\| k: 0 < k < \delta :: \langle \Box i, j :: CP_L^k(i,j) := \text{if } \langle B_L(i,j) - B_R((i+k), j) = 0 \rangle \text{ then } CP_L^k(i,j) \text{ else } 0 \rangle . \quad (4)$$

[2] The notation serves only to help the reader in understanding the parallel algorithm presented in this paper, and is not intended to be a complete specification language for parallel synchronous computations in general. Each statement consists of a quantified expression of the form ⟨*operator variables*:*range*::*expression*⟩. Allowed operators are ‖, ∀, □, +, −, ×, ∨, ∧, min, and max. An expression may be an assignment, a conditional assignment, or another quantified expression. A statement is evaluated by applying an *operator* to the set of *expressions* obtained by instantiation over the *range* of bound *variables*. If the range is obvious, it is omitted. For example, $\Box i, j \equiv \forall ij$: x min $\le i \le x$ max, y min $\le j \le y$ max.

Candidates for all δ disparities, each stored in a distinct memory CP_L^k, are processed in parallel by δ pipelined networks. Thus, candidates of all disparities are tested in one machine cycle.

Step 4 – Apply Orientation Constraint: L_{DIR} and R_{DIR} are characteristic images containing the precomputed values for orientation of chains belonging to the left and right edgel sets (L and R), respectively. The absolute difference between the orientations of corresponding chains belonging to a candidate are compared to a preselected threshold ϵ. For candidates of disparity k, property values in the right characteristic image are found by shifting memory R_{DIR} k pixels along the epipolar axis. Candidates of all δ disparities are compared in parallel. If for some candidate $CP_L^k(i, j)$, the difference is less than some threshold (e.g., $\epsilon = 30$ degrees), the candidate remains viable and is stored in $CP_L^k(i, j)$. Otherwise, it is discarded.

$$\begin{aligned}\| k: 0 < k < \delta :: \langle \square i, j :: CP_L^k(i, j) \\ := \text{if } \langle | L_{\mathrm{DIR}}(i, j) \\ - R_{\mathrm{DIR}}((i + k), j)| < \epsilon \rangle \\ \text{then } CP_L^k(i, j) \text{ else } 0 \rangle . \end{aligned} \tag{5}$$

The computational structure is similar to the ordering constraint described above and requires only a single machine cycle to process candidates at all δ disparities.

Step 5 – Count Candidates within Each Pool: For each candidate pool $CP_L^{1:\delta}(i, j)$, the surviving candidates are counted and the total sum is stored in memory COUNT_L. Candidates at all δ possible disparities are tallied in parallel by one or more function tables (f_{count}) that count the number of bits set within each $CP_L(i, j)$ pool. Since it is only possible to count δ/b bits[3] at a time, if $[\delta/b] \geq 1$, memories $CP^{1:\delta}$ are partitioned into $[\delta/b]$ disjoint groups of contiguous bitplanes. The kth distinct group is addressed below as memory $CP^{[k]}$. The sums for all candidate pools are computed in a single machine cycle.

$$\begin{aligned}\langle \square i, j :: \mathrm{COUNT}_L(i, j) \\ := \left(+ k: 1 \leq k \leq \left\lceil \frac{\delta}{b} \right\rceil :: f_{\mathrm{count}}(CP_L^{[k]}(i, j)) \right) \rangle .^4 \end{aligned} \tag{6}$$

Step 6 – Identify Candidates Satisfying at Least One Similarity Constraint: The three parallel computations below inject the candidates of CP_L into three *similarity groups*, S_{DIR}, S_{LS}, and S_{RS}. L_{LS}, R_{LS} and L_{RS}, R_{RS} are pairs of characteristic images for the average intensity along the left side and right side of chains within edge images L and R, respectively. After the completion of one machine cycle, memory S_{DIR} contains all the candidates whose corresponding chains are similar in orientation [see (7a)]. For example, consider a candidate identified by the bit set in memory $CP_L^k(i, j)$; the candidate (bit) will be copied into memory $S_{\mathrm{DIR}}^k(i, j)$ if its corresponding property values $L_{\mathrm{DIR}}(i, j)$ and $R_{\mathrm{DIR}}(i + k, j)$ differ (absolutely) by less than max.dir.

$$\begin{aligned}\| k: 0 < k < \delta :: \langle \square i, j :: S_{\mathrm{DIR}}^k(i, j) \\ := \text{if } \langle | L_{\mathrm{DIR}}(i, j) \\ - R_{\mathrm{DIR}}((i + k), j)| < \max.\mathrm{dir} \rangle \\ \text{then } CP_L^k(i, j) \text{ else } 0 \rangle . \end{aligned} \tag{7a}$$

Likewise, memories S_{LS} and S_{RS} contain candidates whose corresponding chains are similar in left-sided and right-sided intensity, respectively. Candidates may be included in more than one S group.

$$\begin{aligned}\| k: 0 < k < \delta :: \langle \square i, j :: S_{LS}^k(i, j) \\ := \text{if } \langle | L_{LS}(i, j) \\ - R_{LS}((i + k), j)| < \max.ls \rangle \\ \text{then } CP_L^k(i, j) \text{ else } 0 \rangle \end{aligned} \tag{7b}$$

$$\begin{aligned}\| k: 0 < k < \delta :: \langle i, j :: S_{RS}^k(i, j) \\ := \text{if } \langle | L_{RS}(i, j) \\ - R_{RS}((i + k), j)| < \max.rs \rangle \\ \text{then } CP_L^k(i, j) \text{ else } 0 \rangle . \end{aligned} \tag{7c}$$

All three parallel computations above are similar to Step 4 in structure. Candidates at all δ disparities are identified in parallel in one machine cycle. The three similarity groups S_{DIR}, S_{LS}, and S_{RS} are used in the next step to evaluate all candidates in parallel.

Step 7 – Evaluate Each Candidate within a Pool: Each candidate within a pool (i, j) is assigned to one of three disjoint partitions based on "goodness" of match. The highest quality candidates are assigned to partition T_1 and are identified by the conjunction of candidates belonging to similarity groups S_{DIR}, S_{LS}, and S_{RS} [see (8a)]. The conjunction of candidates of all δ disparities is computed in parallel.

$$\begin{aligned}\langle \square i, j :: T_1^{1:\delta}(i, j) \\ := \langle S_{\mathrm{DIR}}^{1:\delta}(i, j) \wedge S_{LS}^{1:\delta}(i, j) \rangle \wedge S_{RS}^{1:\delta}(i, j) \rangle . \end{aligned} \tag{8a}$$

Candidates are assigned to partition T_2 by first computing the disjunction of candidates in similarity groups S_{RS} and S_{LS}. Candidates of the union are then intersected with candidates of similarity group S_{DIR}. To keep partitions T_2 and T_1 disjoint, candidates must not have already been assigned to partition T_1 [see (8b)].

$$\begin{aligned}\langle \square i, j :: T_2^{1:\delta}(i, j) \\ := \langle S_{\mathrm{DIR}}^{1:\delta}(i, j) \wedge \langle S_{LS}^{1:\delta}(i, j) \vee S_{RS}^{1:\delta}(i, j) \rangle \rangle \\ \wedge \neg T_1^{1:\delta}(i, j) \rangle . \end{aligned} \tag{8b}$$

Candidates assigned to the third partition, T_3, are simply the remaining candidates that have not been assigned to either partition T_1 or T_2.

$$\begin{aligned}\langle \square i, j :: T_3^{1:\delta}(i, j) \\ := CP_L^{1:\delta}(i, j) \otimes \langle S_{LS}^{1:\delta}(i, j) \vee S_{RS}^{1:\delta}(i, j) \rangle \rangle . \end{aligned} \tag{8c}$$

[3]The parameter b is the number of bits per interconnection link in the machine model described earlier.

[4]In this case, our notation fails to explicitly specify the linkage between the stages of the pipes used to accomplish the (binary) addition operation. However, for N inputs, the operation requires at most $\lceil (N/2) + 1/2 \rceil$ stages per pipe.

COUNT_{T_1} and COUNT_{T_2} are computed similarly to Step 5 above and are used in the next step to classify each match pool in parallel.

$$\left\langle \square\, i, j :: \text{COUNT}_{T_1}(i,j) := \left(+k : 1 \le k \le \left\lceil \frac{\delta}{b} \right\rceil :: f_{\text{count}}\left(T_1^{[k]}(i,j)\right) \right) \right\rangle \quad (8d)$$

$$\left\langle \square\, i, j :: \text{COUNT}_{T_2}(i,j) := \left(+k : 1 \le k \le \left\lceil \frac{\delta}{b} \right\rceil :: f_{\text{count}}\left(T_2^{[k]}(i,j)\right) \right) \right\rangle . \quad (8e)$$

Step 8 – Classify Each Pool into One of Four Disjoint Sets: In each classification operation below, all pools are processed in parallel. A pool at position (i, j) having exactly one candidate at some disparity k belonging to partition T_1 is identified in memory CLASS_{U1} (Unique of type 1) at the same position [see (9a)]. Similarly, pools in image memory CLASS_{U2} have a unique candidate belonging to partition T_2. The pools identified in image memory CLASS_{B1} have more than one candidate, but exactly one of them belongs to partition T_1 making it discernibly *better* than the rest [see (9c)]. Similarly, the pools of CLASS_{B2} have more than one candidate, but have exactly one candidate belonging to T_2 and exactly zero candidates belonging to partition T_1 [see (9d)].

$$\langle \square\, i, j :: \text{CLASS}_{U1}(i,j) := \text{if } (\text{COUNT}_{T_1}(i,j) = 1 \wedge \text{COUNT}_L(i,j) = 1) \langle \text{then 1 else 0} \rangle \quad (9a)$$

$$\langle \square\, i, j :: \text{CLASS}_{U2}(i,j) := \text{if } (\text{COUNT}_{T_2}(i,j) = 1 \wedge \text{COUNT}_L(i,j) = 1) \langle \text{then 1 else 0} \rangle \quad (9b)$$

$$\langle \square\, i, j :: \text{CLASS}_{B1}(i,j) := \text{if } (\text{COUNT}_{T_1}(i,j) = 1 \wedge \text{COUNT}_L(i,j) > 1) \langle \text{then 1 else 0} \rangle \quad (9c)$$

$$\langle \square\, i, j :: \text{CLASS}_{B2}(i,j) := \text{if } (\text{COUNT}_{T_1}(i,j) = 0) \wedge (\text{COUNT}_{T_2}(i,j) = 1) \wedge \langle (\text{COUNT}_L(i,j) > 1) \text{ then 1 else 0} \rangle . \quad (9d)$$

The four classes above may be computed in a single machine cycle by four independent pipelines executing in parallel. The four classes are consolidated into a single memory C by the disjunction of all classes. Each distinct function f_c outputs a unique value labeling CLASS_c pools.

$$\langle \square\, i, j :: C(i,j) := (\forall c : c \in \{U1, U2, B1, B2\} :: f_c(\text{CLASS}_c)) \rangle \quad (9e)$$

B. Phase 2

The next phase of the computation consists of a relaxation method that employs constraints on figural continuity [19], [20] and disparity continuity to correct most local errors. Below, phase two is described in six steps.

Step 9 – Identify the Three Most Likely Candidates from Each Match Pool: As described earlier in step seven we partitioned all candidates into one of three disjoint sets, T_1, T_2, and T_3. In the equation below, the variable t ranges over the number Typ of candidate types; so in our method, Typ = 3. In order to identify the most promising candidates first, binding of the variable t proceeds from 1 (the most promising candidates are of type T_1) to 3 (the least promising candidates are of type T_3). The variable n ranges over the number N of (best) candidates saved. In our method, we save the three most likely candidates within each pool; so $N = 3$. For each t, as many as N distinct candidates are identified in parallel. At the completion of this step memories $D^n(i, j)$ and $W^n(i, j)$ will respectively contain disparity and weight values of the nth candidate of pool (i, j). Initially all N disparity memories (D^n) and weight memories (W^n) are cleared. If for some type t there exists a candidate within pool (i, j) of T_t (and $D^n(i, j)$ is empty) then memories $D^n(i, j)$ and $W^n(i, j)$ are assigned disparity and weight values of the candidate, respectively.

$$t = 1 \cdots 3 :: \| n : 1 \le n \le N :: \langle \square\, i, j :: D^n(i,j),\ W^n(i,j) := \text{if } \langle (D^n(i,j) = 0) \wedge \left(f^n_{\text{type}}\left(T_t^{1:\delta}(i,j)\right) \neq 0\right) \rangle \text{ then } f^n_{\text{disparity}}\left(T_t^{1:\delta}(i,j)\right),\ f^t_{\text{weight}}(C(i,j)) \text{ else } D^n(i,j),\ W^n(i,j) \rangle . \quad (10)$$

Let us examine how all N pairs of disparity and weight values are computed in parallel for each candidate type t. Consider a pool (i, j). The function f^n_{type} takes as input all the candidates of pool (i, j) stored in memory T_t and returns the value "1" if there exists at least n candidates in the pool. Similarly, the function $f^n_{\text{disparity}}$ takes as input all the candidates of pool (i, j) and assigns a disparity value to (exactly) the nth candidate of each pool, relative to the (bit-plane) index $[1 : \delta]$ of memory T_t. For example, if the nth candidate of a pool is stored in memory T_t^7, then it (the nth candidate) is assigned a relative disparity value "7". The function f^t_{weight} assigns a weight to each admitted candidate by a table lookup of precomputed values.[5] If during a subsequent iteration t there exists another candidate within the same (i, j) pool and disparity memory D^n contains a previous entry for pool (i, j), the disparity entry and weight value are not effected by the pass. Similarly, if there is no candidate within pool $T_t(i, j)$ and disparity $D^n(i, j)$ is empty, memories D^n and W^n are left unchanged. The best candidates of type t (at any of δ possible disparities) are identified in parallel. After Typ machine cycles, memory groups $D^{1:N}$ and $W^{1:N}$ contain the disparities and

[5] Each value is a measured sum which takes into account the type of candidate and the classification of its pool. Weights range from 0.0 (the least promising candidates) to 1.0 (the most promising candidates). There are Typ distinct f_{weight} functions precomputed as follows:

$$t, c : t \in \{1,2,4\} \wedge c \in \{1,2,8\} :: f^t_{\text{weight}}(c) := 0.6(1/c) + 0.4(1/t).$$

Values of the variable t are associated with candidate types 1, 2, and 3, respectively. Values of the variable c are associated with *unique* classes ($U1$, $U2$), classes containing a discernibly *better* candidate ($B1$, $B2$), and candidates belonging *other* pools (pools with a single type 3 candidate or multiple candidates of the same type), respectively.

associated weights for the N most likely candidates, respectively. The weight of each candidate is used in the next step to compute probability values for all N candidates of each pool in parallel.

Step 10 – Calculate the Initial Probability of Each Candidate in Every Pool: This computation is accomplished in three steps. First, the maximum weight of the N candidates within each pool is identified in parallel. Image memory l^* contains an initial estimate of the probability that every pool (i, j) is matchable.

$$\langle \square i, j :: l^*(i,j) := \langle 1.0 - (\max n: 1 \le n \le N :: W^n(i,j)) \rangle \rangle . \quad (11a)$$

Next, the sum of all N candidates within each pool (i, j) is computed and stored in $S(i, j)$.

$$\langle \square i, j :: S(i,j) := (+n: 1 \le n \le N :: W^n(i,j)) \rangle . \quad (11b)$$

Each candidate weight of pool (i, j) is converted into an initial probability estimate through normalization. The function $f_{\text{reciprocal}}$ is a precomputed table that outputs the inverse of its input value. We use fixed point arithmetic for all analytic computations.[6] Fixed point division is accomplished by computing the reciprocal (at one stage) and then multiplying (at a subsequent stage). We can avoid costly real division without the loss of precision because we can determine the domain of input values for any pool size N. For example, since each candidate within a given pool is assigned a measured weight less than or equal to 1.0, the total sum of all candidates is less than or equal to N, and the range of $S(i, j)$ is constrained to the interval $[0 : N]$.

$$\| n: 1 \le n \le N :: \langle \square i, j :: P^n(i,j) \rangle := \langle W^n(i,j) \times f_{\text{reciprocal}}(S(i,j)) \rangle \times \langle \langle 1 - l^*(i,j) \rangle \rangle . \quad (11c)$$

All N candidates of every pool are normalized in parallel, at a cost of one machine cycle. At the end of the cycle, the memories $P^{1:N}$ contain the normalized probability values for all N candidates of every pool. In the next step, these normalized probabilities are used to determine how consistent each candidate is with its neighbors.

Step 11 – Compute Local Support for Each Candidate: The local support of each candidate is computed by summing the probability values of connected neighbors. Only connected neighbors (above or below) with disparity differences less than one pixel are allowed to contribute support. The total support for the nth candidate of pool (i, j) is stored in memory $Q^n(i, j)$. The variables l and k scroll memories D^m and P^m in phase with respect to pool (i, j) so that the six connected neighbors (three above and three below) of $D^n(i, j)$ are allowed to contribute support for candidate $D^n(i, j)$, provided they are consistent with candidate $D^n(i, j)$.

$$n: 1 \le n \le N :: \langle \square i, j :: Q^n(i,j) := (+k, l, m: k \in \{-1, 0, 1\} \wedge l \in \{-1, 1\} \wedge 1 \le m \le N :: \langle f_{\text{consistent}}(\langle D^n(i,j) - D^m(i+k, j+l) \rangle) \times P^m(i+k, j+l) \rangle) \rangle . \quad (12)$$

[6] In practice, we found 16 bits of precision yielded sufficient accuracy for all numerical computations of this algorithm.

The function $f_{\text{consistent}}$ returns the value "1" if the (absolute) difference between disparity $D^n(i, j)$ and some connected neighbor disparity $D^m(i + k, j + l)$ is less than or equal to one. Otherwise the function returns the value "0." The output of the function is multiplied by the probability value of the same connected neighbor $P^m(i + k, j + l)$. There are two levels of parallelism taking place. The probabilities of all six connected neighbors are allowed to contribute to the sum of $Q^n(i, j)$ in parallel. The support available from all the neighbors of neighborhood m is gathered simultaneously for all N candidates in each pool. Any previous matches (assigned probability 1.0) provide a network of global support for any new locally consistent matches. The operation is repeated N times (over the variable n) to allow each candidate to obtain support from N distinct neighborhoods. In the next step, the quantity of support stored in memory $Q^n(i, j)$ for each candidate is used to modify (increase or decrease) in parallel the probability values stored in memory $P^n(i, j)$.

Step 12 – Update the Probability of Each Candidate: In the computation below, α and β are parameters that influence the convergence characteristics of the updating rule. Briefly, α and β can be interpreted as damping and gain parameters. The value α delays the suppression of unlikely candidates, and β determines the rate of convergence. (A complete discussion of the updating rules in relaxation labeling algorithms is presented in Rosenfeld *et al.* [17].) In our experiments, $\alpha = 0.3$ and $\beta = 3$.

$$\| n: 1 \le n \le N :: \langle \square i, j :: \hat{P}^n(i,j) := \langle \langle P^n(i,j) \times \alpha \rangle + \langle \langle P^n(i,j) \times \beta \rangle \times Q^n(i,j) \rangle \rangle \rangle . \quad (13)$$

For all N candidates, the probabilities stored in each pool of P^n are multiplied by constants α and β in parallel. Image memories $Q^{1:N}$ and $P^{1:N}$, containing the quantity of local support and the previous probability of each candidate, are read simultaneously. The resulting modified probability estimates for all N candidates of each pool are summed in parallel and stored in memories $\hat{P}^{1:N}(i, j)$. The next two steps normalize these updated probability estimates in parallel.

Step 13—Compute the Sum of Probability Estimates for Candidates within Each Pool: The sum of the updated probabilities for all N candidates within each pool is computed in parallel and stored in $S(i, j)$.

$$\langle \square i, j :: S(i,j) := \langle (+n: 1 \le n \le N :: \hat{P}^n(i,j)) \rangle \rangle . \quad (14)$$

Step 14 – Normalize Probability Estimates $\hat{P}$ and l^*: Finally, the probability estimates are normalized in parallel for all N candidates within each pool (i, j). This step is similar to Step 1 except that memory l^* is updated in parallel through an independent pipeline. The final normalized probabilities are stored in memories $P^{1:N}$.

$$\| n: 1 \le n \le N :: \langle \square i, j :: P^n(i,j), l^*(i,j) := \hat{P}^n(i,j) \times f_{\text{reciprocal}}(S(i,j)), l^*(i,j) \times f_{\text{reciprocal}}(S(i,j)) \rangle . \quad (15)$$

In the next section, we present generalized expressions for the time complexity of each step described above, and use these

TABLE I
PHASE 1—IDENTIFICATION
COMPLEXITY ANALYSIS FOR DISPARITY δ

Step	Number of Machine Cycles	Minimum Number of Cycles	
		Value	Machine Model Requirements (b, m, q, r)
0) Label	$\left\lceil \frac{1}{q} \right\rceil \cdot \left\lceil \frac{2}{m} \right\rceil$	1	$(b, 2, 1, 1)$
1) Eliminate	$\left\lceil \frac{2}{q} \right\rceil \cdot \left\lceil \frac{2}{m} \right\rceil$	1	$(1, 2, 2, 1)$
2) Disparity	$\left\lceil \frac{\delta}{q} \right\rceil \cdot \left\lceil \frac{1}{m} \right\rceil$	1	$(1, 1, \delta, \delta)$
3) Order	$\left\lceil \frac{2\delta}{q} \right\rceil \cdot \left\lceil \frac{3}{m} \right\rceil$	1	$(b, 3, 2, \delta, \delta)$
4) Orientation	$\left\lceil \frac{2\delta}{q} \right\rceil \cdot \left\lceil \frac{3}{m} \right\rceil$	1	$(b, 3, 2, \delta, \delta)$
5) Global Count	$\left\lceil \frac{\left\lceil \frac{(\delta/b)+1}{2} \right\rceil}{m} \right\rceil$	1	$(b, 1, 1, 1)$
6) Similarity	$\left\lceil \frac{2\delta}{q} \right\rceil \cdot \left\lceil \frac{3}{m} \right\rceil$	1	$(b, 3, 2\delta, \delta)$
7) Evaluate	$\left\lceil \frac{3\left\lceil \frac{\delta}{b} \right\rceil}{q} \right\rceil \cdot \left\lceil \frac{3}{m} \right\rceil$	1	$\left(b, 3, 3\left\lceil \frac{\delta}{b} \right\rceil, \delta\right)$
Count Types	$\left\lceil \frac{\left\lceil \frac{(\delta/b)+1}{2} \right\rceil}{m} \right\rceil$	1	$(b, 1, 1, 1)$
8) Classify	$\left\lceil \frac{9}{q} \right\rceil \cdot \left\lceil \frac{5}{m} \right\rceil$	1	$(b, 5, 9, 1)$

expressions to compare the optimal performance with that of a real machine.

V. PERFORMANCE ANALYSIS

In this section we show the cost (number of machine cycles) for each step by formulas that take into account parameters of the machine model and maximum disparity. Each formula was derived by analyzing the network of pipelines needed to accomplish each step in minimum time. Column three of Tables I and II shows the optimal performance (minimum time) of each step obtained when ideal machine model requirements are satisfied.

All of the steps of Phase 1 can be executed in constant time for any disparity range δ, provided the network has at least 2δ pipelines (parameter q). Similarly, most of the steps of Phase 2 can be executed in constant time with the exception of Steps 8 (Select) and 10 (Support). The run time of Step 8 is directly related to the number of distinct candidate types, Typ, while the run time of Step 10 is exactly N, the number of *best* candidates saved from each pool. Both parameters are typically small numbers. Assuming Typ = 3 and $N = 3$, the minimum computation time is achievable by an idealized machine having the following configuration:

Number of pipelines $(q) = 2\delta$

Number of stages $(m) = 21$

Number of bits per link $(b) = 16$

Number of outputs $(r) = 6$

The cost of the parallel algorithm is a function of the disparity parameter δ alone. Therefore, parallel architectures that can accommodate a large input bandwidth (q) are best suited for high-speed stereo matching applications. On the average, the algorithm required only four stages to achieve optimal time.[7] This suggests that the topology of such a pipelined network should consist of a *large number of pipelines* with *few stages*.

An ideal machine, operating at 60 Hz, can accomplish stereo matching in 1.5 s, using 88 machine cycles (see Table III). However, typical commercial (general-purpose) image processing machines[8] often have fewer resources and run at slower

[7]Only Steps 8 and 11 required more stages (5 and 21, respectively).

[8]Machines in current production by Pixar, Vicon, Comptel, I^2S, and Sun/TACC.

TABLE II
PHASE 2—RELAXATION[1]
COMPLEXITY ANALYSIS FOR DISPARITY δ

Step	Number of Machine Cycles	Minimum Number of Cycles	
		Value	Machine Model Requirements (b, m, q, r)
9) Select	$\text{Typ} \cdot \left\lceil \frac{2N\left(\left\lceil \frac{\delta}{b} \right\rceil + 2\right)}{q} \right\rceil \cdot \left\lceil \frac{4}{m} \right\rceil$	Typ	$\left(\max(p, d), 4, 2N\left(\left\lceil \frac{\delta}{b} \right\rceil + 2\right), 2N\right)$
10) Initialize	$\left\lceil \frac{(N-1)+1}{q} \right\rceil \cdot \left\lceil \frac{\left\lceil \frac{q+1}{2} \right\rceil + 1}{m} \right\rceil$	1	$\left(p, \left\lceil \frac{q+1}{2} \right\rceil + 1, N, 1\right)$
(Sum)	$\left\lceil \frac{N-1}{q} \right\rceil$	1	$\cdot \left\lceil \frac{\left\lceil \frac{q+1}{2} \right\rceil}{m} \right\rceil \left(p, \left\lceil \frac{q+1}{2} \right\rceil, N, 1\right)$
(Probability)	$\left\lceil \frac{3N}{q} \right\rceil \cdot \left\lceil \frac{4}{m} \right\rceil$	1	$(p, 4, 3N, N)$
11) Support	$N \cdot \left\lceil \frac{12N}{q} \right\rceil \cdot \left\lceil \frac{2 + \left\lceil \frac{q+1}{2} \right\rceil}{m} \right\rceil$	N	$\left(\max(p, d), 2 + \left\lceil \frac{q+1}{2} \right\rceil, 12N, N\right)$
12) Update	$\left\lceil \frac{3N}{q} \right\rceil \cdot \left\lceil \frac{4}{m} \right\rceil$	1	$(p, 4, 3N, N)$
13) Sum	$\left\lceil \frac{(N-1)+1}{q} \right\rceil \cdot \left\lceil \frac{\left\lceil \frac{q+1}{2} \right\rceil}{m} \right\rceil$	1	$\left(p, \left\lceil \frac{q+1}{2} \right\rceil, N, 1\right)$
14) Normalize	$\left\lceil \frac{2N+2}{q} \right\rceil \cdot \left\lceil \frac{2}{m} \right\rceil$	1	$(p, 2, 2N+2, N+1)$

[1] N is the number of candidate pairs, Typ is the number of distinct candidate types, p is number of bits used to store probability values, and d is number of bits used to store disparity offset values.

TABLE III
RUN TIME PERFORMANCE—COMPARATIVE SUMMARY
(Results below are computed for $\delta = 8$, $N = 3$ and Typ = 3)

Phase	Ideal Machine		Typical Machine		Implementation	
	Machine Cycles for (16, 21, 36, 8)	Total	Machine Cycles for (8, 4, 4, 4)	Total	Machine Cycles	Total
Identification	1+1+1+1+1+1+3+5+1	16	1+1+2+4+4+1+12+5+6	36	1+1+3+8+8+1+24+5+8	59
Relaxation	3+3+9+1+1+1	18	14+5+54+3+1+3	80	9+8+54+9+1+5	86
Match						
Total Cycles	16 + (r · 18)		37 + (r · 80)		60 + (r · 86)	
[1]Time (seconds)	88/60 = 1.5 s		357/30 = 11.9 s		404/30 = 13.5 s	

[1]Calculations are based on typical commercial machines operating at 30 Hz (1/30 s machine cycle), and a high-performance machine operating at 60 Hz. In our experiments, the relaxation network required at most four iterations ($r = 4$) to stabilize.

clock speeds. Suppose a typical general-purpose machine has four pipelines, four stages per pipeline, and 8-bit interconnection links. We could expect to accomplish stereo matching on such a machine in about 11.9 s, using 357 cycles, operating at 30 Hz. As shown in Table III, there exists a fourfold difference between the number of cycles needed on a theoretical machine and the number of cycles needed for most practical machines. Depending on the requirements of the application, this difference may be tolerable for high-speed stereo matching. If it is determined that the performance of this algorithm running on some general-purpose machine is not fast enough, this analysis can help evaluate and identify alternative machines that may better exploit the parallelism of this algorithm. As advances in hardware technology continue to reduce machine cycle times, and highly interconnected multicomputers with flexible topologies are produced (at lower costs), real-time performance of this parallel algorithm will almost certainly be achievable.

We implemented each step of the algorithm on a Gould/

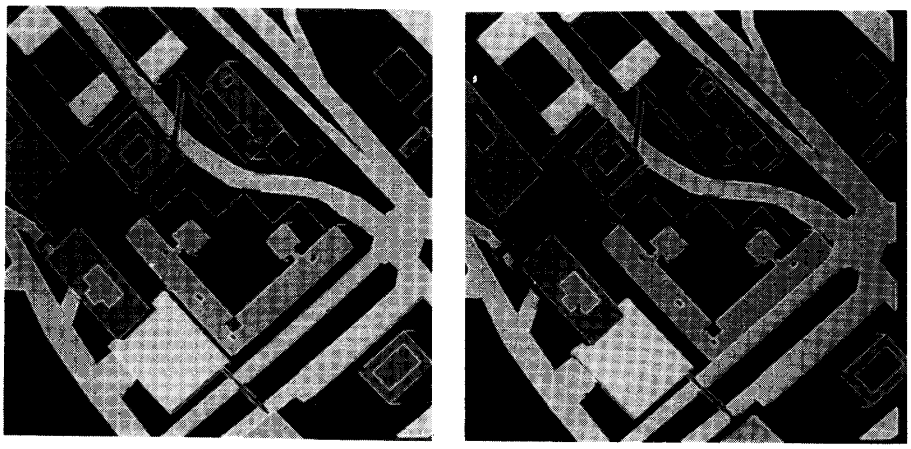

Fig. 3. Stereo views rendered from a synthetic model of an urban scene. (All matched edgels are highlighted.)

TABLE IV
SAMPLE RESULTS OF A PARALLEL IMPLEMENTATION
(For maximum $\delta = 32$ pixels, 1/30 s machine cycle.)

Image	Number of Steps	Number of Relaxation Steps	Percent Matched	Total Cost (Cycles)	Machine Time (Seconds)
Synthetic	2	4	81	1616	53.9 s
Real	4	2	79	1856	61.9 s

DeAnza IP-8500 image processor, equipped with a Digital Video Processor (DVP). The architecture of the DVP consists of a network of four pipelines, with limited interconnection links. The input bus width of the network is 8×10 bits ($b \times 2q$). The output bus width is 4×8 bits ($r \times b$). There are seven distinct stages within each pipeline. The stages of the DVP are not uniform and could not perform the complete set of operations described for the stages of our idealized machine. On account of this mismatch, it was often necessary to adjust the algorithm to accommodate differences in functionality and other considerations (idiosyncrasies) of the DVP architecture. For example, in updating the probability of each candidate in Step 11, it was not possible to delay the necessary multiplication operations beyond the first stage of each pipeline. Therefore, each multiplication required a separate pass (machine cycle). (Sixteen-bit precision was accomplished by bit-slicing and cascading four 8-bit stages to form two 16-bit pipelines.) Stereo matching was achieved in 13.5 s using 404 DVP cycles. The actual execution time (wall clock) is about 23 s due to the hidden cost of down loading at each step pipeline configuration instructions and stage opcodes from a Micro Vax II host machine. In general, the disparity in performance between the ideal machine and the implementation machine can principally be attributed to the limited (specialized) functionality of the stages, lack of flexible (dynamic) interconnection links, and limited (insufficient) input bandwidth.

With respect to memory dynamics and utilization, Phase 1, Step 7, the evaluation of candidate pairs, required the most amount of memory overall: 388 bit-planes. On the DeAnza, this mapped onto 49 8-bit image tiles or 13 Mbytes of image memory. In Phase 2, Step 11 needed 306 bit-planes to compute the local support for each candidate pair. This step utilized 39 image tiles or 10 Mbytes of image memory.

VI. SAMPLE RESULTS

The parallel implementation described earlier was applied to both synthetic and real imagery. The results shown in this section are representative of the sample of imagery used throughout out experiments and development. The real imagery was obtained by extracting selected regions from digitized stereo aerial photographs[9] of the Washington, DC, area. The selected scene contains typical urban features such as complex building structures, roadways, and natural features. A synthetic model of the same aerial scene, consisting of similar buildings and roadways, was built using the MOVIE.BYU geometric modeler [18]. Stereo views of the model were rendered to approximate the perspectives of the stereo aerial photographs. For purposes of matching, the stereo pairs for both synthetic and real imagery consisted of 512×512, 8-bit grey-scaled pixels. All the pixels within each 512×512 image memory were processed in parallel during each machine cycle.

Matched edgels for the synthetic stereo pair are shown in Fig. 3. The maximum search range of disparities was 32 pixels while the actual range of disparities was less than 24 pixels. Matching was accomplished in two iterations. The first iteration accurately matched 5535 edgels, while the next pass recovered an additional 802 matches. Experimentally we have observed that, for most complex scenes, only four iterations are needed for the relaxation network to settle to a reasonably stable state. The algorithm matched 6337 edgels, i.e., 81% of the edge pixels in the left image, at a cost of 808 DVP machine cycles (1/30 s machine cycle).

For the stereo photographs shown in Fig. 4, the maximum search range of disparities was 32 pixels while the actual range of disparities was less than 28 pixels. Matching was accomplished in four iterations. At each iteration, only edgels within a distinct range of orientation were considered for matching. In this case, the parallel matcher was embedded in an interactive stereo matching paradigm called *Complexity Control*. This methodology was described previously in Roman *et al.* [11]. In

[9]The image was provided by Engineering Topographic Laboratories (ETL), Fort Belvoir, VA. The original black and white stereo photographs were digitized at a resolution of 0.6 m pixel. However, the stereo pair used in our experiments was reduced to 300×300 pixels to limit the search space within a range of 32 pixels. This was necessary due to the limited memory on our implementation machine.

Fig. 4. Stereo aerial photographs over an urban area of Washington, DC (All matched edgels are highlighted.)

this experiment, edges were selected interactively at each step based on chain orientation and presented to the parallel matcher to accomplish incremental matching. Edgels were partitioned into four distinct ranges of chain orientation such that edgels of chains running along a mostly horizontal direction were presented to the matcher in one step, while edgels oriented along the vertical, left diagonal, and right diagonal were presented in distinct subsequent steps. At each step, previously matched edgels provided a context in which new matches were embedded. Here, each (controlled) step required only two relaxation iterations to stabilize. A total of 3015 edgels were matched, representing 79% of the edgels in the left image. However, due to the occlusion and misalignment of the stereo pair (the pair was not rectified), not all of the edges in the left image were matchable. Matching required about 62 s of DVP cycles on the Gould/DeAnza IP-8500 image processor, or about 16 s (selection) per step.

The actual elapsed time (wall clock) was about three times longer than the machine times reported. This was principally due to the limited amount of memory within our DeAnza (11 Mbytes), making it necessary to reload DeAnza memory from the Micro Vax II host at each iteration step. In addition, the hidden cost of downloading new instructions to reconfigure the pipeline and load stage instruction opcodes at each step (of the algorithm) contributed to the overhead.

VII. Conclusions

We have presented a parallel algorithm for stereo matching that achieves high speed by exploiting the parallel architectures of typical SIMD processors. The cost of the parallel algorithm was shown to be a function of maximum disparity (δ) alone when executing on an idealized machine having a small number of stages, a reasonable interconnection bandwidth, and modest output bandwidth. Parallel architectures that can accommodate large input bandwidth are best suited for high-speed stereo matching applications. On the average, the algorithm required only four stages per pipeline to achieve optimal time. This suggests that the topology of a pipelined network tuned for stereo matching should consist of a *large number of pipelines* with *few stages*. On the practical side, the complexity analysis presented in this paper can readily determine the performance of our parallel algorithm executing on a variety of fine-grained SIMD machines. In this way, the analysis may be used to evaluate and identify alternative machines that may better exploit the parallelism of this algorithm. Furthermore, the parallel formulations and complexity analysis presented in this paper exhibit a methodology that may be useful to others in the computer vision community who are interested in reformulating existing low-level vision algorithms for high-speed parallel execution.

The feasibility of our parallel algorithm was shown by implementation on a typical commercial SIMD pipelined processor. As advances in hardware technology continue to reduce machine cycle times and highly interconnected multicomputers with flexible topologies are produced at practical costs, real-time performance of this parallel algorithm will be achievable and appealing for applications where rapid response is critical.

Acknowledgment

The authors are very grateful to Dr. J. R. Cox, Jr., for his support and encouragement.

References

[1] D. Arnold and T. O. Binford, "Geometric constraints in stereo vision: Image processing for missile guidance," *Proc. Soc. Photo-Opt. Instrum. Engr.*, vol. 238, pp. 281–292, 1980.

[2] S. T. Barnard and M. A. Fischler, "Computational stereo," *Comp. Surveys*, vol. 14, no. 4, pp. 553–572, 1982.

[3] H. S. Lem and T. O. Binford, "Stereo correspondence: Features and constraints," in *Proc. Computer Vision and Pattern Recognition Conf.* (Miami Beach, FL), June 22, 1986, pp. 373–379.

[4] H. Baker and T. O. Binford, "A system for automated stereo mapping," in *Proc. Image Understanding Workshop* (Palo Alto, CA), 1982, pp. 215–222.

[5] D. Marr and T. Poggio, "Cooperative computation of stereo disparity," *Science*, vol. 194, pp. 283–287, 1976.

[6] M. Herman and T. Kanade, "Incremental reconstruction of 3-D scenes from multiple, complex images," *Artificial Intell.*, vol. 30, pp. 289–341, 1986.

[7] A. Ortony, "A system for stereo viewing," *Computer J.*, vol. 14, no. 2, pp. 140–144, 1971.

[8] A. L. Yullie and T. Poggio, "A generalized ordering constraint for stereo correspondence," MIT AI Lab, Cambridge, MA, Tech. Rep. AIM-777, May 1984.

[9] Y. Ohta, K. Takano, and K. Ikeda, "A highspeed matching system based on dynamic programming," in *Proc. 1st Int. Conf. Computer Vision* (London), June 8, 1987, pp. 335–342.

[10] H. K. Nishihara, "PRISM: A practical realtime imaging stereo matcher, " in *Intelligent Robots: Proc. 3rd Int. Conf. Robot*

Vision and Sensory Controls RoViSeC3 (Cambridge, MA, Nov. 7, 1983), pp. 134-142.
[11] G. C. Roman, A. F. Laine, and K. C. Cox, "Interactive complexity control and high-speed stereo matching," in *Proc. Computer Society Conf. Computer Vision and Pattern Recognition* (Ann Arbor, MI, June 5, 1988), pp. 171-176.
[12] M. Pietikainen and D. Harwood, "Depth from three camera stereo," in *Proc. Computer Vision and Pattern Recognition Conf.* (Miami Beach, FL, June 22, 1986), pp. 2-8.
[13] S. T. Barnard and W. B. Thompson, "Disparity analysis of images," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-2, no. 4, pp. 333-340, July 1980.
[14] T. Pavlidis, *Algorithms for Graphics and Image Processing*. Rockville, MD: Computer Science Press, 1982, pp. 199-201.
[15] R. A. Kirsch, "Computer determination of the constituent structure of biological images," *Computers Biomedical Res.*, vol. 3, pp. 315-328, June 1971.
[16] R. B. Eberline, "An iterative gradient edge detection algorithm," *Computer Graphics Image Processing*, vol. 5, pp. 245-253, 1976.
[17] A. Rosenfeld, R. A. Hummel, and S. W. Zucker, "Science labeling by relaxation operations," *IEEE Trans. Syst. Man Cyber.*, vol. SMC-6, June 1976.
[18] *MOVIE.BYU Reference Manual*, Brigham Young University, Eng. Computer Graphics Lab., Provo, UT, 1987.
[19] J. E. W. Mayhew and J. P. Frisby, "The computation of binocular edges," *Perception*, vol. 9, pp. 69-86, 1980.
[20] ——, "Psychophysical and computational studies towards a theory of human stereopis," *Artificial Intell.*, vol. 17, 1981.
[21] R. Mohan, G. Medioni, and R. Nevatia, "Stereo error detection, correction, and evaluation," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 11, no. 2, Feb. 1989.
[22] Y. Ohta and T. Kanade, "Stereo by two-level dynamic programming," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-7, no. 2, pp. 139-154, Apr. 1985.
[23] G. Medioni and R. Nevatia, "Segment-based stereo matching," *Computer Vision, Graphics, Image Processing*, vol. 31, pp. 2-18, 1985.
[24] M. Drumheller and T. Poggio, "On parallel stereo," in *Proc. IEEE Conf. Robotics Automat.*, 1986, pp. 1439-1448.
[25] W. E. L. Grimson, *From Images to Surfaces*. Cambridge, MA: M.I.T. Press, 1981.
[26] D. Hillis, "The connection machine," Ph.D. dissertation, Dept. Electrical Engineering and Computer Science, M.I.T., Cambridge, MA, 1985.
[27] Y. Ohto, and T. Kanade, "Stereo by intra and inter-scanline search using dynamic programming," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-7, pp. 133-154, 1985.
[28] P. R. Cooper, "Order and structure in stereo correspondence," Dept. Computer Science, Univ. of Rochester, Tech. Rep. 216, June 1987.
[29] G. V. S. Raju, T. Binford, and S. Shekhar, "Stereo matching using Viterbi algorithm," in *Proc. Image Understanding Workshop* (Los Angeles), Feb. 1987, pp. 766-776.

Andrew F. Laine received the B.S. degree from Cornell University, Ithaca, NY, in 1977, the M.S. degree in computer science from Washington University, St. Louis, MO, in 1983, and the D.Sc. degree in computer science from the School of Engineering and Applied Science, Washington University, in 1989.

In 1989, he was a Postdoctoral Fellow with the Center for Intelligent Computer Systems at Washington University. He is now an Assistant Professor with the Computer and Information Sciences Department, University of Florida, Gainesville. His research interests include computer vision, pattern recognition, wavelet theory, parallel algorithms, computer graphics, and image processing.

Dr. Laine is a member of the ACM and the IEEE Computer Society.

Gruia-Catalin Roman received the B.S., M.S., and Ph.D. degrees in computer science from the University of Pennsylvania, Philadelphia, in 1973, 1974, and 1976, respectively.

He has been on the faculty of Washington University, Saint Louis, MO, since 1976 where he is currently a Professor in the Department of Computer Science. His current research deals with models, languages, and visualization methods for concurrent programming. His previous research has been concerned with requirements and design methodologies for distributed systems. Other areas in which he worked include interactive high-speed computer vision algorithms, formal languages, biomedical simulation, computer graphics, and distributed database. He is also an active software engineering consultant. His list of past clients includes the United States Government and several large firms in the United States and Japan. His consulting work involves development of custom software engineering methodologies and training programs.

Dr. Roman was a Fulbright Scholar while at the University of Pennsylvania. He received an IEEE Computer Society Outstanding Paper Award. He is a member of Tau Beta Pi, the ACM, and the IEEE Computer Society.