A Reconfigurable and Hierarchical Parallel Processing Architecture: Performance Results for Stereo Vision

Alok N. Choudhary

Science and Technology Center
Department of Electrical and Computer Engineering
Syracuse University
Syracuse, NY 13244

Subhodev Das, Narendra Ahuja and *Janak H. Patel*

Coordinated Science Laboratory
and Department of Electrical and Computer Engineering
University of Illinois
1101 W. Springfield
Urbana, IL 61801

Abstract

The degree of exploitable parallelism in computer vision tasks varies with time and image position. Therefore, an architecture for vision must be highly flexible and modular. In this paper we consider one such multiprocessor architecture (NETRA) which is highly reconfigurable and does not involve the use of complex interconnection schemes. The topology of this multiprocessor is recursively defined, and hence, is easily scalable from small to large systems. It has a tree-type hierarchical architecture each of whose leaf nodes consists of a cluster of small but powerful processors connected via programmable crossbar with selective broadcast capability. The architecture is simulated on a hypercube multiprocessor and the performance of one processor cluster is evaluated for stereo vision tasks. The particular stereo algorithm selected for implementation requires computation of two-dimensional Fast Fourier Transform (2D-FFT), template matching, histogram computation and least-squares surface fitting. We use static partitioning of data for the data independent tasks such as 2D-FFT and dynamic scheduling and load balancing for the data dependent tasks of feature matching and disambiguation. The superior performance of our architecture is demonstrated by comparing the results with that of a similar implementation on the hypercube itself.

1. Introduction

Computer vision algorithms perform a variety of mathematical, signal processing, graph theoretic and data manipulation computations. Typically, these tasks are performed repeatedly in some sequence, and thus are characterized by significant amount of temporal and spatial parallelism. The degree of exploitable parallelism varies with time and with image position in general. Therefore, an architecture for vision must be highly flexible and modular.

Several multiprocessor architectures such as, mesh-connected, pyramids, array processors, systolic arrays (CMU Warp), and hypercubes, have been proposed for computer vision. Most of these architectures are SIMD computers (with the exception of WARP), and hence, are mainly suitable for low level vision algorithms. However, low level vision algorithms only constitute the first few steps in a vision system [1, 2]. Therefore, SIMD computers, such as listed above, are not suitable for efficient implementation of complex vision systems. Several hierarchical and partitionable architectures have been proposed that contain both SIMD and MIMD types of computing modules. Examples include PM4[3], PASM [4], REPLICA [5], INSPECTOR [6], and IUA [7]. Design of these architectures has addressed the issues of flexibility, partitionability, and reconfigurability which are needed in an architecture for vision systems.

In this paper we consider one such multiprocessor architecture (called NETRA) which is highly reconfigurable and does not involve the use of complex interconnection schemes[2, 8]. The topology of NETRA is recursively defined, and hence, is easily scalable from small to large systems. It has a tree-type hierarchical architecture each of whose leaf nodes consists of a cluster of small but powerful processors connected via programmable crossbar with selective broadcast capability.

The focus of this paper is on the performance evaluation of clusters of NETRA on a stereo algorithm. The architecture is simulated on a hypercube multiprocessor and the performance of one processor cluster is evaluated. The performance of the clusters for these tasks has been compared with that of a hypercube multiprocessor system. It is observed that the flexibility of the crossbar in the clusters yields significant performance gains over a hypercube multiprocessor with similar computational capabilities.

Stereo vision provides a means to calculate the depth of a point in the three-dimensional (3D) world from two or more two-dimensional (2D) images taken from different viewpoints. To reconstruct 3D surfaces from stereo requires selecting feature points from the two images, matching points belonging to one image with those from the other, using the disparity in image locations of a matched pair of points along with the imaging geometry to compute the position of the corresponding 3D point, eliminating matching ambiguities, and fitting surfaces to the 3D points. The particular stereo algorithm selected for implementation requires computation of two-dimensional Fast Fourier Transform (2D-FFT) to detect edges, template matching to establish feature correspondence, and histogram computation and least-squares surface fitting to eliminate mismatches. We use static partitioning of data for the data independent tasks such as

This research was supported in part by the National Aeronautics and Space Administration under contract NASA NAG-1-613 and the National Science Foundation under grant ECS 8352408.

CH2898-5/90/0000/0389$01.00 © 1990 IEEE

2D-FFT and dynamic scheduling and load balancing for the data dependent tasks of feature matching and disambiguation.

This paper is organized as follows. Section 2 contains a description of the NETRA. Section 3 shows how to map an algorithm onto a processor cluster in NETRA. A brief description of the stereo algorithm appears in Section 4. Section 5 describes the mapping of the algorithm on a cluster. Section 6 presents the performance results of the implementation of the algorithms on the NETRA cluster and a hypercube multiprocessor (Intel iPSC/2). Concluding remarks are presented in Section 7.

2. Architecture of NETRA

Figure 1 shows the architecture of NETRA that consists of the following components :-

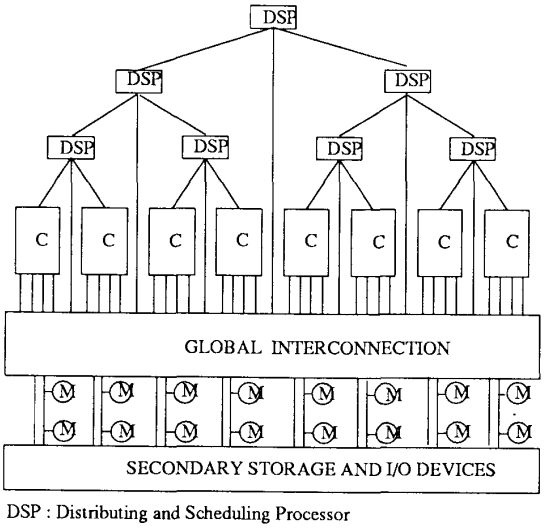
(1) A large number (1000 - 10000) of *Processing Elements (PEs)*, organized into clusters of 16 to 64 PEs each.

(2) A tree of *Distributing-and-Scheduling-Processors (DSPs)* that make up the task distribution and control structure of the multiprocessor.

(3) A parallel pipelined shared *Global Memory* and a *Global Interconnection* that links the PEs and DSPs to the Global Memory.

2.1. Processor Clusters

The clusters consist of 16 to 64 PEs each with its own program and data memory. They form a layer below the DSP-tree, with a leaf DSP associated with each cluster. PEs within a cluster also share a common data memory. The PEs, the DSP associated with the cluster, and the shared memory are connected together with a crossbar switch. The crossbar switch permits point-to-point communication as well as selective broadcast by the DSP or any of the PEs. The crossbar design consists of pass transistors connecting the input and output data lines. The switches are controlled by control bits indicating the connection pattern. If a processor of DSP needs to broadcast then all the control bits in its row are made one. In order to connect processor P_i to processor P_j, control bit (i,j) is set to one and the rest of the control bits in row i and column j are set to zero.



DSP : Distributing and Scheduling Processor
C : Processor Cluster M : Memory Module

Figure 1: Organization of NETRA.

Clusters can operate in an SIMD mode, a systolic mode, or an MIMD mode. Each PE is a general purpose processor with a capability for high speed floating point operations. In an SIMD mode, PEs in a cluster execute identical instruction streams from private memories in a lock-step fashion. In the systolic mode, PEs repetitively execute an instruction or set of instruction on data streams from one or more PEs. In both cases, communication between PEs is synchronous. In the MIMD mode PEs asynchronously execute instruction streams resident in their private memories. The streams may not be identical. In order to synchronize the processors in a cluster, a synchronization bus is provided which is used by processors to indicate to the DSP that a processor(s) has finished its computation or a processor wants to change the communication pattern. The DSP can either poll the processors or the processors can interrupt the DSP using the synchronization bus.

There is no arbitration in the crossbar switch. That is, the interconnection between processors has to be programmed before processors can communicate with each other. Programming a crossbar requires writting a communication pattern into the control memory of the crossbar. A processor can alter the communication pattern by updating the control memory as long as it does not conflict with the existing communication pattern. The DSP associated with the cluster can write into the control memory to alter the communication pattern. The most common communication patterns such as linear arrays, trees, meshes, pyramids, shuffle-exchanges, cubes, broadcast, can be stored in the memory of the crossbar. These patterns need not be supplied externally. Therefore, switching to a different pattern in the crossbar can be fast because switching only requires writing the patterns into the control bits of the crossbar switches from its control memory.

2.2. The DSP Hierarchy

The DSP-tree is an n-tree with nodes corresponding to DSPs and edges to bi-directional communication links. Each DSP node is composed of a processor, a buffer memory, and a corresponding controller.

The tree structure has two primary functions. First it represents the control hierarchy for the multiprocessor. A DSP serves as a controller for the subtree structure under it. Each task starts at a node on an appropriate level in the tree, and is recursively distributed at each level of the sub-tree under the node. At the bottom of the tree, the subtasks are executed on a processor cluster in the desired mode (SIMD or MIMD) and under the supervision of the leaf DSP.

The second function is that of distributing the programs to leaf DSPs and the PEs. Under the DSP-hierarchy approach, one copy of the program is fetched by the controlling DSP (the DSP at the root of the task subtree) and then broadcast down the subtree to the selected PEs. Also, DSP hierarchy provides communication paths between clusters to transfer control information or data from one cluster to others. Finally, the DSP-tree is responsible for Global Memory management.

2.3. Global Memory

The multiport global memory is a parallel-pipelined structure as introduced in [9]. Given a memory(chip)-access-time of T processor-cycles, each line has T memory modules. It accepts a request in each cycle and responds after a delay of T cycles. Since an L-port memory has L lines, the memory can support a bandwidth of L words per cycle.

Data and programs are organized in memory in *blocks*. Blocks correspond to "units" of data and programs. The size of a block is variable and is determined by the underlying tasks and their data structures and data requirements. A large number of blocks may together constitute an entire program or an entire image. Memory requests are made for blocks. The PEs and DSPs are connected to the Global Memory with a multistage interconnection network.

Two main functions of the global memory are input-output of data and program to and from the DSPs and processor clusters, and to provide intercluster communication between various tasks as well as within a task if a task is mapped onto more than one cluster.

2.4. Global Interconnection

The PEs and the DSPs are connected to the Global Memory using a multistage circuit-switching interconnection network. Data is transferred through the network in pages. A page is transferred from the global memory to the processors which is given in the header as a destination port address and the header also contains the starting address of the page in the global memory. When the data is written into the global memory, only starting address needs to be stated. In each case, end-of-page may be indicated using an extra flag bit appended to each word. For further details of the NETRA the reader is referred to [2, 8, 10].

3. Mapping Parallel Algorithms on a Cluster

There are two main considerations in the mapping of parallel algorithms onto a cluster. First is selection of a computation mode such as SIMD, MIMD or systolic, and the second is the number of available processors on a cluster and selection of the best way to map the algorithm. For a data dependent algorithm, there may be need for nonuniform data partitioning and local load balancing. The load balancing scheme may be static or dynamic. In the static scheme, the DSP in a cluster allocates tasks to the processors using some *a priori* knowledge about the computation such that each processor receives an average amount of computation. Under the dynamic load balancing scheme the DSP maintains a queue of ready tasks and assigns the tasks to the available processors as they become free to execute the next task.

The methodology we use for mapping parallel algorithms is multidimensional, divide-and-conquer with medium to large grain parallelism. An individual task (in the following discussion task and algorithm are used interchangeably) can be efficiently mapped using spatial parallelism, because most of the vision algorithms are performed on two dimensional data. However, integration of tasks involves exploiting both spatial as well as temporal parallelism by recognizing intertask data dependencies.

Mapping a task on one cluster implies that intratask communication will only involve communication between processors of that cluster. Assume that there are P processors in a cluster. First, program and data are loaded onto the DSP of the cluster. Both in the case of SIMD or MIMD mode, the program is broadcast to the processors. The data division depends on the particular algorithm. If an algorithm is mapped in SIMD or systolic mode, then the compute and communication cycles will be intermixed. If an algorithm is mapped in MIMD mode, then each processor computes its partial results and then communicates with others to exchange or merge data.

4. The Stereo Algorithm

The stereo algorithm that has been used to evaluate the performance of our proposed architecture is briefly described in this section. The features used for stereo matching are the zero-crossings in the convolution of the Laplacian of Gaussian ($\nabla^2 G$) with the original image [11]. The orientations of the zero-crossings and the their locations are used as matching constraints. The location of a match in the other image corresponding to a feature in one image is predicted from the available surface information. Such a surface is obtained by analyzing the stereo pair at a lower resolution level. The stereo algorithm therefore works in a coarse-to-fine mode and integrates matching and surface interpolation [12]. Each matched pair of feature points is backprojected to a 3D point. The 3D points are separated into homogeneous clusters such that each cluster supports a smooth 3D patch. The points that fail to satisfy the geometrical constraints imposed by smooth surfaces are deleted. These are mostly caused by mismatching of feature points. Smooth surfaces are finally fit to the 3D points.

5. Mapping the Stereo Algorithm

The stereo algorithm is mapped onto the architecture of NETRA simulated on an Intel iPSC/2 hypercube multiprocessor in a way so as to achieve a high degree of temporal and spatial parallelism. The data independent local operations in different parts of an image have the scope for spatial parallelism. The temporal characteristics of the stereo algorithm - convolution, zero-crossings detection, matching of features, clustering of 3D points and surface fitting - suggest the use of a pipeline environment. The granularity of tasks in our mapping scheme is medium to large.

5.1. Feature detection

Detection of feature points or zero-crossings involves three subtasks: i) convolution, ii) subsampling and iii) template matching. The convolution is performed in the 2D-DFT domain. The DFTs of the image and $\nabla^2 G$ are then multiplied and inverse DFT of the multiplication produces the convolved image. Subsampling is performed to obtain images of reduced resolution. Finally, template matching is done to extract the zero-crossings from the convolved image.

A nice property of the 2D-FFT is that it can be performed in two decomposable steps : a one dimensional N point FFT along the rows followed by a one dimensional N point FFT of the intermediate results along the columns, or vice versa. The algorithm consists of three phases : 1D-FFT computation along rows, transposing the intermediate results and, 1D-FFT along the columns. At the end of the convolution of an image with $\nabla^2 G$, each processor contains a part of the convolved image. Template matching is performed in order to compute the local zero-crossings in this image. Each processor communicates with two adjacent processors to exchange the boundary rows. Since a hypercube contains a linear array, this communication is done

without conflicts. Finally, each processor computes the zero-crossings in its partition using a 3×3 template. Each processor sends the results to the DSP. In this mapping both the computation and communication times reduce as the number of processors increases. In other words, both computation and communication are decomposable for parallel processing. Since the communication is done without conflicts, we obtain linear speedups.

5.2. Feature matching

In this step the image is divided into a two dimensional rectangular grid and the features located inside a rectangle are used for matching. The computation performed within each rectangle proceeds independently of the others. However, unlike the feature detection step, the computation in this step is data dependent. In other words, the computation depends on the number of features present in the corresponding window as well as the occurrences of mismatches. Therefore, correctly partitioning and distributing the data to the processors becomes important in order to obtain significant performance improvement.

One approach is to assign a fixed size window to each processor. If the features are uniformly distributed across the image then balanced utilization of the processors can be expected. Normally that is not the case. Partitioning the data uniformly among the processors does not divide the computation equally among the processors, resulting in imbalance of load, poor utilization and low speedups. Therefore, dynamic scheduling and computational load balancing become important. The gains can be significant if the overhead of dynamic scheduling is low. The feature data is divided into small granules, causing the number of tasks to be much more than the number of processors. The granularity in our case was one grid row. The tasks are kept in a *task_queue* at the DSP. The data is broadcast to all the processors. Each processor is assigned one task in the beginning. Then as processors finish their assigned tasks, they send a *work_request* message to the DSP. If there are more tasks left in the *task_queue* then the requesting processor is sent another task. The tasks are assigned in a first-come first-served order. When there are no more tasks left then the DSP sends a *terminate* message to all the processors. Upon receiving a *terminate* message, each processor terminates computation after finishing its current task.

6. Performance Analysis

The performance results of NETRA on the stereo algorithm are presented and analyzed in this section. Only one cluster of processors on NETRA has been used for the implementation. As a comparison we have also included the results of implementing the same algorithm on the hypercube itself. The processor that we have used for NETRA cluster simulation was the same as that on a node of Intel iPSC/2 (80386) so that fair comparison can be made of their performances. The communication bandwidth of the NETRA crossbar was assumed to be 20 Mbytes/sec for each connection.

The results for feature detection are analyzed first followed by those for feature matching. Figure 2 shows the computation time and speed up on the cluster (of NETRA) of varying number of processors and on the hypercube. The computation is 2D-FFT for the convolution of the image with $\nabla^2 G$. The performance numbers are taken for an input image of resolution 256×256. The solid lines are for the speed up and the dashed lines indicate the actual elapsed time when initial program and data loading and processor communication time are taken into account along with the FFT computation. 2D-FFT is a completely decomposable problem for parallel processing, hence, as the number of processors increases the computation time per processor decreases. However, as the number of processors increases, the communication can become a bottleneck because the intermediate step in 2D-FFT involves transposing the 1D-FFT results. In NETRA, since the crossbar connections are available and they can be set-up without conflicts, the transpose time *decreases* as the number of processors increases. The speedup on NETRA is therefore linear as evident from Figure 2. The performance of NETRA is superior to that of the hypercube for larger processors, for example, the speedup gain for a 16 processor cluster is nearly 33%.

Detection of the zero-crossings involves template matching and thresholding in addition to the 2D convolution. However, since FFT for convolution is the most computationally intensive task among the three, the performance of FFT is reflected in the results for zero-crossings shown in Figure 3. We observe that the speedup using NETRA cluster is almost linear. On the other hand, the speedup gain using a hypercube reduces as the number of processors increases when the total overhead (including the program loading time, data loading time and other intermediate communication time) is comparable to the computation time.

Stereo matching algorithm is implemented using dynamic scheduling because the computation is very data dependent. The implementation involves broadcasting the data (features) to processors which is efficiently done using the broadcast capability of the cluster. However, in the hypercube, although broadcast is available in software, there is no broadcast capability supported by hardware (software broadcasting is slower). Furthermore, since the DSP in NETRA can selectively communicate (directly) with each processor in the cluster, dynamic assignment of task is more efficient.

Figure 4 show the speedups for the stereo matching algorithms for the resolution level 128×128. The above mentioned advantages are reflected in the better performance obtained on a cluster of NETRA as compared to that on a hypercube multiprocessor. However, as the resolution level increases, the amount of computation increases in every granule much faster than the communication requirements. Consequently, the performance of the two architectures is more comparable.

7. Summary

In this paper we have studied the performance of a multiprocessor architecture called NETRA that is highly reconfigurable and does not involve the use of complex interconnection schemes. Its processing power is concentrated in clusters of powerful processors connected through flexible and programmable crossbar with selective broadcast capability. We have simulated the architecture in an independent environment on an Intel iPSC/2 hypercube multiprocessor and evaluated the performance of one processor cluster of the NETRA. In particular, we have implemented a stereo algorithm for extracting 3D surface information that requires computation of 2D-FFT to detect features, template matching to establish feature correspondences, histogram computation and least-squares surface fitting to eliminate mismatches. We have used static partitioning of data for the data independent tasks and dynamic scheduling and

load balancing for the data dependent tasks. The results demonstrate the superior performance of NETRA over the hypercube for the same implementation. This performance advantage is attributed to the flexible crossbars and selective broadcast capabilities of NETRA.

REFERENCES

[1] C. Weems, A. Hanson, E. Riseman, and A. Rosenfeld, "An integrated image understanding benchmark: recognition of a 2 1/2 D mobile," in *IEEE Conference on Computer Vision and Pattern Recognition*, Ann Arbor, MI, June 1988.

[2] Alok N. Choudhary, "Parallel architectures and parallel algorithms for integrated vision systems," in *Ph.D. Thesis*, University of Illinois, Urbana-Champaign, August 1989.

[3] F. A. Briggs, K. S. Fu, J. H. Patel, and K. H. Huang, "PM4 - A reconfigurable multiprocessor system for pattern recognition and image processing," *1979 National Computer Conference*, pp. 255-266.

[4] H. J. Siegel et al., "PASM - a partitionable SIMD/MIMD system for image processing and pattern recognition," *IEEE Transactions on Computers*, vol. C-30, pp. 934-947, December 1981.

[5] Y. W. Ma and R. Krishnamurti, "The architecture of REPLICA - a special-purpose computer system for active multi-sensory perception of 3_dimensional objects," *Proceedings International Conference on Parallel Processing*, pp. 30-37, 1984.

[6] W. A. Perkins, "INSPECTOR - A computer vision system that learns to inspect parts," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-5 , pp. 584-593, November, 1983.

[7] C. C. Weems, S. P. Levitan, A. R. Hanson, E. M. Riseman, J. G. Nash, and D. B. Shu, "The image understanding architecture," *COINS Tech. Rep. 87-76,*.

[8] M. Sharma, J. H. Patel, and N. Ahuja, "NETRA: An architecture for a large scale multiprocessor vision system," in *Workshop on Computer Architecture for Pattern Analysis ans Image Database Management*, Miami Beach, FL, pp. 92-98, November 1985.

[9] F. A. Briggs and E. S. Davidson, "Organization of semiconductor memories for parallel-pipelined processors," *IEEE Transactions on Computers*, pp. 162-169, February 1977.

[10] Alok Choudhary, Janak Patel, and Narendra Ahuja, "NETRA - A parallel architecture for integrated vision systems II: algorithms and performance evaluation," *IEEE Transactions on Parallel and Distributed Processing (submitted)*, August 1989.

[11] Marr D. and Hildreth E., "Theory of edge detection," *Proceedings of Royal Society London*, vol. B 207, pp. 187-217, 1980.

[12] Hoff W. and Ahuja N., "Extracting surfaces from stereo images : an integrated approach," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-11, pp. 121-136, February 1989.

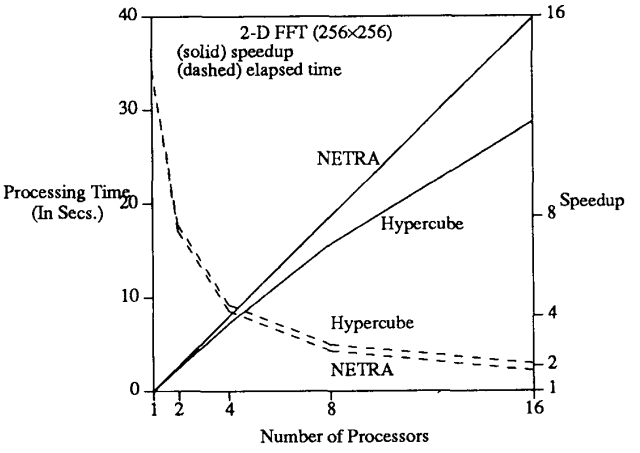


Figure 2: Performance results for 2D-FFT (256×256) computation.

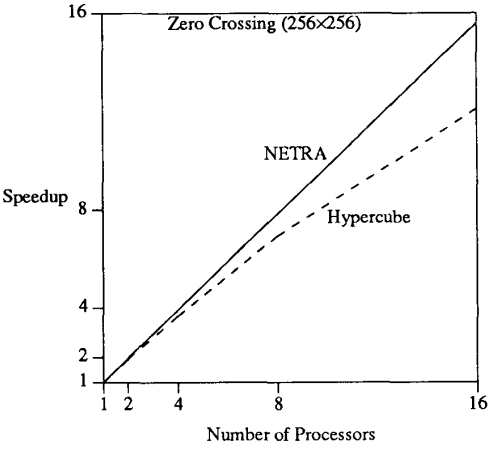


Figure 3: Performance results for zero-crossings computation.

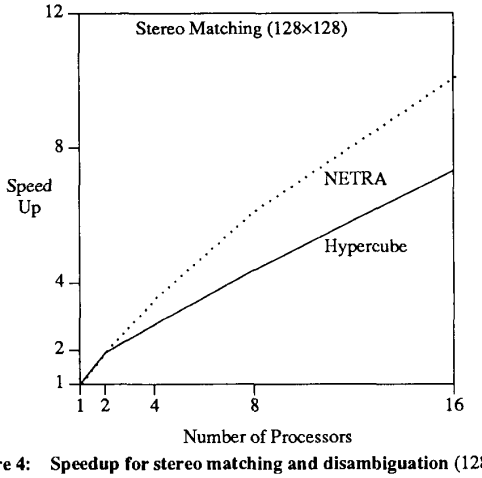


Figure 4: Speedup for stereo matching and disambiguation (128×128).