Scalable Data Parallel Implementations of Object Recognition using Geometric Hashing

Ashfaq Khokhar, Hyoung Kim,
Viktor Prasanna and Cho-Li Wang

CENG Technical Report 93-04

Department of Electrical Engineering - Systems
University of Southern California
Los Angeles, California 90089-2562
(213)740-4483

Scalable Data Parallel Implementations of Object Recognition using Geometric Hashing *

Ashfaq A. Khokhar†, Hyoung J. Kim§, Viktor K. Prasanna†, and Cho-Li Wang†

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

§Department of Control and Instrumentation Engineering
Kangwon National University
Chuncheon, 200-701
Korea

email: {ashfaq + khj + prasanna + wang}*@halcyon.usc.edu*

September 30, 1992
Revised: February 15, 1993

*This research was supported in part by NSF under grant IRI-9145810, in part by DARPA and AFOSR contracts F-49260-89-C-0126 and F-49620-90-C-0078, and in part by the Army Research Office contract number DAAL03-89-C-0038 with the University of Minnesota Army High Performance Computing Research Center.

The experimental results on MasPar MP-1 reported in this work were made possible through access to machines at MasPar Computer Corporation.

Abstract

Object recognition involves identifying known objects in a given scene. It plays a key role in image understanding. Geometric hashing has been recently proposed as a technique for model based object recognition in occluded scenes. However, this technique is computationally demanding; a probe of the recognition phase on a serial machine can take several minutes to complete.

In this paper, we present scalable data parallel algorithms for geometric hashing. We perform implementations of the proposed algorithms on MasPar MP-1, a Single Instruction Multiple Data (SIMD) machine, and on the Connection Machine CM-5 operating in Single Program Multiple Data (SPMD) mode. The implementations are carried out after a careful study of the computation and communication characteristics of the underlying architectures. Based on the results, we compare the merits of the above classes of architectures for vision.

In earlier parallel implementations, the number of processors employed is independent of the size of the scene but depends on the size of the model database which is usually very large. We design new parallel algorithms and map them onto MP-1 and onto CM-5. These techniques significantly improve upon the number of processors employed while achieving much superior time performance. Earlier implementations claim 700 to 1300 *msec* for the recognition phase, assuming 200 feature points in the scene on an 8K processor CM-2. Our implementations run on a P processor machine, such that $1 \leq P \leq S$, where S is the number of feature points in the scene. Our results show that the recognition phase for a scene consisting of 1024 feature points takes less than 50 *msec* on a 1K processor MP-1 and it takes less than 10 *msec* on a 256 processor CM-5. The model database used in these implementations contains 1024 models and each model is represented by 16 feature points. The implementations developed in this paper require number of processors independent of the size of the model database and are scalable with the machine size. Results of concurrent processing of multiple probes of the recognition phase are also reported.

Keywords: Object Recognition, Geometric Hashing, Image Understanding, MasPar MP-1, CM-2, CM-5, Scalable Parallel Processing.

1 Introduction

Object recognition, a high level vision task, is a key step in an integrated vision system. Most model based recognition systems work by hypothesizing matches between scene features and model features, predicting new matches, and verifying or changing the hypotheses through a search process [5, 6, 7, 8, 10, 11, 28]. Geometric hashing [16] offers a different and more parallelizable paradigm. However, parallel techniques are needed to use geometric hashing in real time applications.

In geometric hashing, given a set of models and their features points, for each model, all possible pairs of the feature points are designated as a *basis set.* The coordinates of the features points of a model are computed relative to each member of its basis set. These coordinates are then used as indices into a hash table. The records in the hash table comprise of (*model, basis*) pairs. In the recognition phase, an arbitrary pair of feature points in the scene is chosen as a basis and the coordinates of the feature points in the scene are computed. The new coordinates are used to hash into the hash table and the corresponding entries of the hashed bin are accessed. Votes are accumulated for the (*model, basis*) pairs stored in the hashed locations. The pair winning the maximum number of votes is chosen as a candidate for matching.

There have been two prior efforts in parallelizing the geometric hashing algorithm [1, 27]. Both implementations have been performed on SIMD hypercube based machines. These implementations are among the early efforts [24, 22, 26, 2, 29] in using parallel techniques to solve intermediate/high-level vision problems. One of the major problems in both the implementations is the requirement of large number of processors. In our results, we exploit the fact that the number of votes cast in an iteration of the recognition phase is bounded by S, the number of feature points in the scene. Therefore, no more than S locations of the hash table are accessed during the execution of the recognition algorithm. This allows us to reduce the number of processors employed to at most S. Previous implementations used $O(Mn^3)$ processors, *i.e.*, the number of bins in the hash table, where M is the number of models in the database and n is the number of feature points in each model. Also, the implementation by Bourdon and Medioni [1] suffers due to inefficiency of the routing algorithm. This inefficiency limits the scope of their implementation to a small model database. In [27], Rigoutsos and Hummel suggest to use radix sort to implement histogramming, a technique used in their implementation to count the number of votes for each *(model, basis)* pair. The use of radix sort in histogramming is advantageous only if the number of levels in the histogram is much less than the number of data points [20]. In case of geometric hashing, this is not true.

In this paper, we present various partitioning, mapping and routing techniques to address the above issues and design scalable data parallel algorithms for the recognition phase. This leads to significantly less number of processors to be used, while achieving much superior time performance. Given a scene consisting of S feature points, the proposed parallel algorithms for one probe of the recognition phase take $O(\frac{S}{P}\log S)$ time on a *fat tree* based architecture and is scalable over the range $1 \leq P \leq \sqrt{S}$. We also show that a probe can be performed in $O(\sqrt{S})$ time on a mesh array of size S. We perform implementations of the proposed algorithms on CM-5 and MP-1 after a

careful study of the computation and communication characteristics of the underlying architectures. Earlier implementations [1, 27] claim 700 to 1300 *msec* for the recognition phase, assuming a scene consisting of 200 feature points, on an 8K processor CM-2. We provide techniques to implement the recognition phase on a P processor machine, such that $1 \leq P \leq S$. Our results show that one probe of the recognition phase for a scene consisting of 1024 feature points takes less than 50 *msec* on a 1K processor MP-1 and it takes less than 10 *msec* on a 256 processor CM-5. The model database used in the implementations contains 1024 models and each model is represented using 16 feature points. The implementations developed in this paper require number of processors independent of the size of the model database and are scalable with the machine size. Results of concurrent processing of multiple probes of the recognition phase are also reported. Based on the implementation results, we compare the merits of classes of machines used for vision.

The organization of the paper is as follows. The geometric hashing technique is outlined in Section 2. Section 3 discusses parallelization of geometric hashing. In Section 4, implementation details are shown and experimental results are tabulated and compared. Conclusions are presented in Section 5.

2 Object Recognition Using Geometric Hashing

In a model-based recognition system, a set of objects is given and the task is to find instances of these objects in a given scene. The objects are represented as sets of geometric features, such as points or edges, and their geometric relations are encoded using a minimal set of such features. The task becomes more complex if the objects overlap in the scene and/or other occluded unfamiliar objects exist in the scene.

Many model based recognition systems are based on hypothesizing matches between scene features and model features, predicting new matches, and verifying or changing the hypotheses through a search process. Geometric hashing, introduced by Lamdan and Wolfson [30], offers a different and more parallelizable paradigm. It can be used to recognize flat objects under weak perspective. For the sake of completeness, we briefly outline the geometric hashing technique in Section 2.1. Additional details can be found in [30].

2.1 Geometric Hashing Algorithm

Figure 1 shows a schematic flow of the geometric hashing algorithm. The algorithm consists of two procedures, preprocessing and recognition. These are shown in Figures 2 and 3 respectively.

Preprocessing:

The preprocessing procedure is executed off-line and only once. In this procedure, the model features are encoded and are stored in a hash table data structure. However, the information is stored in a highly redundant multiple-viewpoint way. Assume each model in the database has n feature points. For each ordered pair of feature points in the model chosen as basis, the

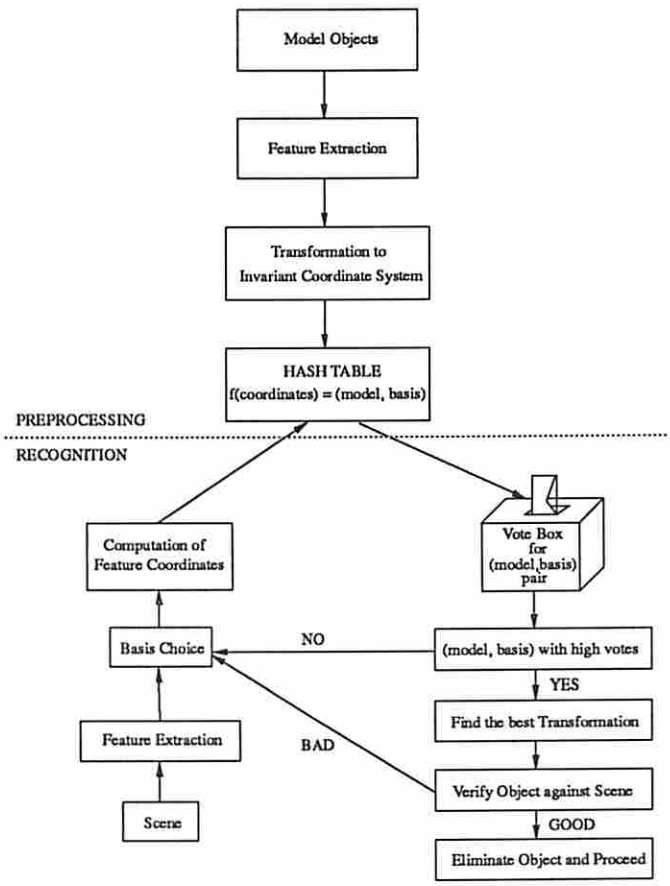

Figure 1: A general scheme for the geometric hashing technique.

coordinates of all other points in the model are computed in the orthogonal coordinate frame defined by the basis pair. Each such coordinate is quantized and is used as an entry to a hash table, where the (*model, basis*) pair, at which the coordinate was obtained, is recorded. The complexity of this preprocessing procedure is $O(n^3)$ for each model, hence $O(Mn^3)$ for M models.

Recognition:

In the recognition procedure, a scene consisting of S feature points is given as input. An arbitrary ordered pair of feature points in the scene is chosen. Taking this pair as a basis, the coordinates of the remaining feature points are computed. Each such coordinate is used as a key to enter the hash table (constructed in the preprocessing phase), and for every recorded (*model, basis*) pair at the corresponding location, a vote is collected for that pair. The pair winning the maximum number of votes is taken as a matching candidate. The execution of the recognition phase corresponding to one basis pair is termed as a *probe*. Finally, edges of the matching candidate model are verified against the scene edges. If no (*model, basis*) pair scores high enough, another basis from the scene feature points is chosen and a probe is performed. Therefore, the worst case time complexity of the recognition procedure is $O(S^3)$. However, if some classification for choosing a basis from the scene is available, the complexity can be reduced to $O(S)$ [16].

Preprocessing()

for each model i such that $1 \leq i \leq M$ **do**
  Extract n feature points from the model;
  for $j = 1$ to n
    for $k = 1$ to n
    - Compute the coordinates of all other features points in the model by taking this pair as basis.
    - Quantize each of the above computed coordinates and use it as a key to enter into a hash table where the pair (*model, basis*), i.e., (i, jk), is recorded.

    next k
  next j
next i
end

Figure 2: A sequential procedure to construct a hash table consisting of (*model, basis*) pairs.

The time taken per probe depends on the hash function employed. The vision community has experimented with various hash functions and hash functions distributing the feature points uniformally into the hash table are known [27]. We will be using these hash functions in our

Recognition()

1. Extract S feature points from the scene.
2. *Selection:*
   Select a pair of feature points as basis.
3. *Probe:*
   a. Compute the coordinates of all other features points in the scene relative to the selected basis.
   b. Quantize each of the above computed coordinates and use it as a key to access the hash table containing the entries of the (*model, basis*) pairs.
   c. Vote for the entries in the hash table.
   d. Select the (*model, basis*) pair with the maximum votes as the matched model in the scene.
4. *Verification:*
   Verify the candidate model edges against the scene edges.
5. If the model wins the verfication process, remove the corresponding feature points from the scene.
6. Repeat steps 2, 3, 4, and 5. (until some specified condition)

end

Figure 3: Outline of the steps in sequential recognition.

implementations. Assuming that S feature points of the input scene leads to $O(S)$ total number of votes, the voting process in a probe of the recognition phase can be implemented in $O(S \log S)$ time using sorting. Other parts of the computation are time consuming, even though they do not contribute to the time complexity; this makes the serial implementations to be time consuming. Note that the total number of (*model, basis*) pairs is $O(Mn^2)$. The voting time can be reduced to $O(S + Mn^2)$ by employing $O(Mn^2)$ boxes to collect the votes. Through out this paper, we assume $S << Mn^2$. For example, $S = 1K$, $M = 1K$, and $n = 16$.

3 Scalable Data Parallel Geometric Hashing

In this section, we present parallel techniques to implement the recognition phase on a P processor machine. Algorithms presented in this section are implemented on Connection Machine CM-5 operating in SPMD mode and on MasPar MP-1, an SIMD array. In an SIMD machine, each processor executes a stream of instructions in a lock-step mode on the data available in its local memory. The instructions are broadcast by the control unit. The SPMD mode of execution combines the characteristics of SIMD and MIMD modes. In this mode, the control processor broadcasts a section of the data parallel program to the processing nodes, rather than broadcasting an instruction at a time (as in a typical SIMD machine). At the start of the execution of a program, the complete program is sent to all the nodes with pseudo synchronization instructions embedded in the code. Each node executes the program independent of others until an embedded synchronization instruction is reached. It resumes the execution of the program only after all the nodes reach the synchronization barrier. The control unit assists in enforcing the synchronization barriers embedded in the program. This operation mode is also referred to as *synchronized* MIMD mode [4].

We will not elaborate on parallelizing the preprocessing phase, since it is a one time process and can be carried out off-line. However, details of that procedure can be found in [13]. The size of the model database (the number of hash bins) is $O(Mn^3)$.

3.1 Partitioned Implementation of the Recognition Procedure

We use P processors such that $1 \leq P \leq S$, where S is the number of feature points in a scene. Each Processing Element (PE) in the array is assumed to have $O(\frac{Mn^3}{P})$ memory. In the recognition phase, possible occurrence of the models (stored in the database) in the scene is checked. The models are available in the hash table created during preprocessing. All the models are allowed to go under rigid and or similarity transformations. An arbitrary ordered pair of feature points in the scene is chosen. Taking this pair as a basis, a probe of the model data base is performed. The main steps of a parallel algorithm to process a single probe of the recognition phase are given in Figure 4.

As we are using less number of processors than the size of the hash table, each PE will have several hash table bins stored in its local memory. Two issues arise during the execution of the

Parallel_Probe(S, P)

/* S is the number of features in the input scene
and P is the number of processors.
Initially each PE is assumed to have S/P distinct
scene feature points stored in a local array FP[]. */

- Choose an arbitrary pair of feature points in the scene as a basis and broadcast it to all the PEs.
- Compute_Keys()
- Vote()
- Compute_Winner()

end

Figure 4: A parallel procedure to process a probe of the recognition phase.

procedure *Parallel_Probe*().

1. More than one feature point in the scene may cast their votes to the same location in the hash table, resulting in a contention for a single memory location in a PE (see Figure 5).

2. More than one feature point in the scene may cast their votes to different bins stored in a PE, resulting in a congestion at a PE (see Figure 6).

The worst case in both the cases will be $O(S)$ contention and congestion. Such a scenario can lead to no speedup at all. On the other hand, other researchers have used large number of processors to avoid congestion and contention problems. However, in their implementations the processor utilization is extremely low. Also, such solutions result in enormous communication overheads in performing global operations, such as global max and histogramming, as evident in the implementations proposed in [1, 27]. In the following, we address these issues and present efficient mapping and routing techniques to resolve the contention and congestion problems arising in performing a probe, while using a small number of processors.

In order to eliminate the memory contention problem in the array, we introduce a *Merge_Key*() procedure shown in Figure 7. This procedure sorts the hash table keys corresponding to the input scene. All keys having the same value reside in a block of PEs and in each block the least indexed PE holds the *leader key*. The leader key has the number of elements in its block. During the voting process, each leader key accesses the PE holding the corresponding location of the hash table and casts a vote on behalf of all the keys in its block, *i.e.* if there are m elements in the block, m votes will be registered for the corresponding location in the hash table. This reduces the number of accesses to the hash table stored in a PE and thus reduces the traffic over the network.

Similarly, in order to address the processor *congestion* problem, we propose to store multiple copies of the hash table in the array. In the worst case, a copy of the hash table can be assigned

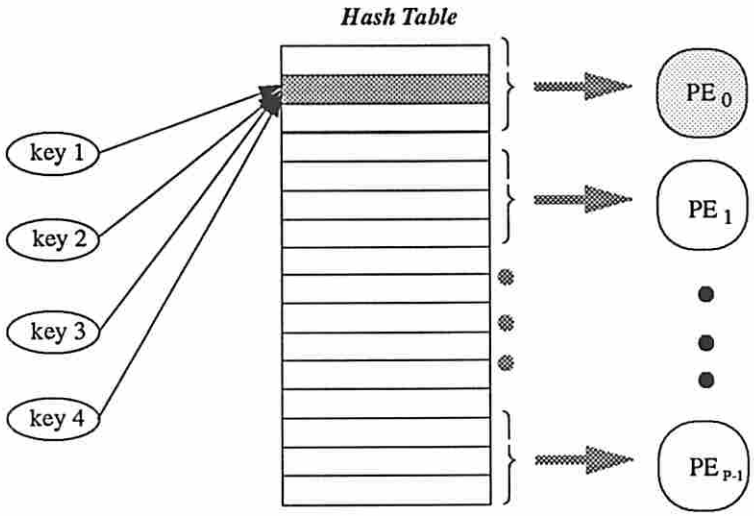

Figure 5: Contention for a single hash bin.

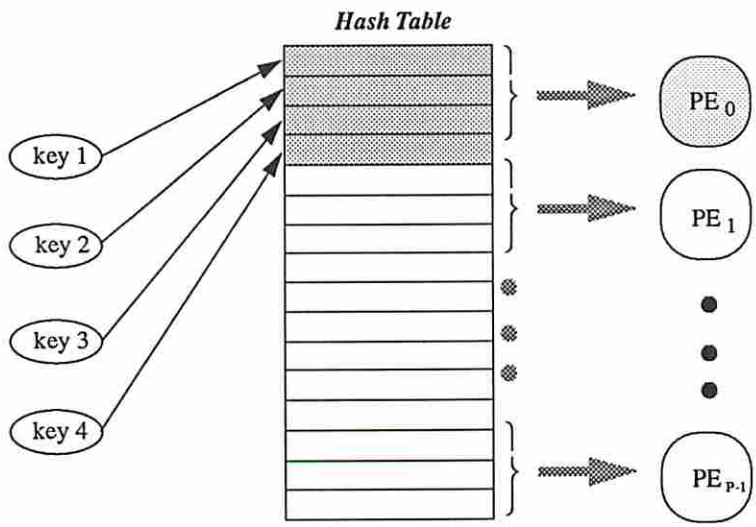

Figure 6: Congestion at a PE while accessing different hash bins stored in a PE.

to each sub-array of suitable size. This increases the size of the memory required within each processor. Each PE restricts its search for the hash table bins within the corresponding subarray. This solution localizes the congestion problem to the sub arrays.

Merge_Keys(INPUT)
- Sort(INPUT)
- **In parallel**, each PE_i, $0 \leq i \leq P-1$
  for each distinct key j, $0 \leq j \leq S/P-1$.
  Identify the *leader key* and mark it in the array
  INPUT[j].
- **In parallel**, each PE_i, $0 \leq i \leq P-1$
  for each *leader key*,
  Count the number of keys same as the *leader key*
  and store it in a seperate output array.

end

Figure 7: A parallel procedure to merge quantized coordinates of feature points of the scene.

3.2 Analysis of the running time of a probe

In the following analysis, we ignore the initialization costs, such as loading the scene points to the processor array, loading the hash table into the processor array, and initialization of memory locations used inside each PE. These assumptions are also made in the previous implementations reported in [1, 27].

A. Analysis of the running time of a probe on CM-5:
For asymptotic time analysis on CM-5, we assume SIMD mode of operation. The *fat tree* [18] is the underlying interconnection network of CM-5. On the *fat tree* model having P leaf nodes, we make the following assumptions:

- A permutation of S items takes $O(\frac{S}{P}\log P)$ time, where $P \leq S$. Initially each PE is assumed to have $\frac{S}{P}$ items and each PE receives $\frac{S}{P}$ items. The permutation is assumed to be known a priori.
- Sort of S items takes $O(\frac{S}{P}\log S)$ time.

  The time bound for sort can be achieved by simulating well known sort algorithms which employ constant number of steps. Each step consists sort on local data and permutation of data. For example, row-column sort procedure described in [17] can be used. The procedure states that if $n = rs, r \bmod s \equiv 0$ and $r \geq 2(s-1)^2$, then n numbers can be sorted by constant

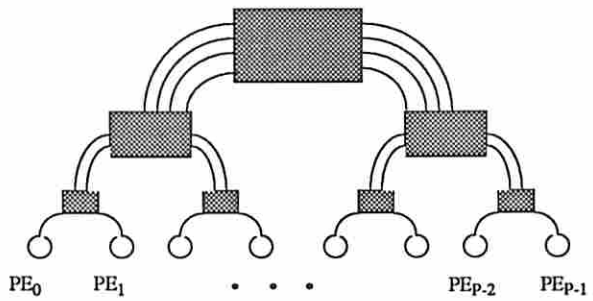

Figure 8: A *fat tree* model.

number of iterations, with each iteration involving one of shifting, shuffling of n numbers or s parallel sort of r numbers.

In our case, $n = S$, and $r = \frac{S}{P}$, i.e. number of elements in each PE. Each PE, in parallel, sorts $\frac{S}{P}$ items in $\frac{S}{P} \log \frac{S}{P}$ time. A shuffle permutation is performed on the sorted elements in all the PEs. This sort-shuffle sequence is applied constant number of times. A shuffle of S items corresponds to a permutation, therefore, total time for sorting S items is:

$$O(\frac{S}{P} \log \frac{S}{P} + \frac{S}{P} \log P)$$

$$= O(\frac{S}{P} \log S)$$

The condition $1 \leq P \leq S^{1/3}$ can be further relaxed to $1 \leq P \leq S^{1/2}$ by using the procedure in [21] for sorting $S^{1/2} \times S^{1/2}$ array.

Based on the above, it is easy to verify the following:

Lemma 1 *Given a fat tree with P processors such that each subarray of size $\log P$ is assigned a copy of the hash table, the voting can be performed in $O(\frac{S}{P} \log^2 P)$ time.*

In the worst case, all the PEs in the subarray of size $\log P$ cast their votes to locations stored in a single PE. Therefore, the total access time for $\frac{S}{P} \log P$ keys in each subarray is $O(\frac{S}{P} \log^2 P)$. This leads to the time bound claimed in the above Lemma.

Lemma 2 *Given a fat tree with P processors, such that each processor has $O(S/P)$ (model, basis) pairs, computing a winner pair takes $O(\frac{S}{P} \log S)$ time.*

We assume that data elements in the hash table are uniformally distributed such that each bin has constant number of elements. The hash function $f()$ in the *Compute_Keys*() procedure ensures this property (see Section 4 for details). We can use a procedure similar to the *Merge_Keys*() procedure to find the total number of votes for each (*model, basis*) pair voted during the voting process. Pairs receiving zero number of votes are not considered. The pair(s) receiving the maximum

number of votes is chosen as a winner. This leads to $O(\frac{S}{P}\log S)$ time performance for computing a winner. Note that even if the voting procedure results in uneven distribution of votes, they can be redistributed such that each PE has $O(\frac{S}{P})$ votes without increasing the asymptotic complexity of computing the winner. The total execution time for one probe of the recognition phase is:

$$
\begin{aligned}
= \quad & \text{time to compute the keys} \quad \{O(S/P)\} \\
& + \text{voting time} \quad \{O(\frac{S}{P}\log^2 P)\} \\
& + \text{time to compute the winner} \quad \{O(\frac{S}{P}\log S)\}
\end{aligned}
$$

Theorem 1 *Given a fat tree architecture consisting of P leaf nodes, one probe of the recognition phase can be processed in $O(\frac{S}{P}\log S)$ time on a scene consisting of S feature points, such that $\log^2 P \leq \log S$.*

Note that the restriction on P can be relaxed if sufficient number of copies of the hash table is available. Based on the above theorem, the algorithm for the recognition phase is processor time optimal and scales linearly with P for $1 \leq P \leq S^{1/2}$.

B. Analysis of the running time of a probe on MP-1:

We use a two dimensional mesh array of P processor as the underlying architecture such that $P = S$. We do not consider the case of $P < S$, as the size of the machines in the MP-1 series is atleast 1024.

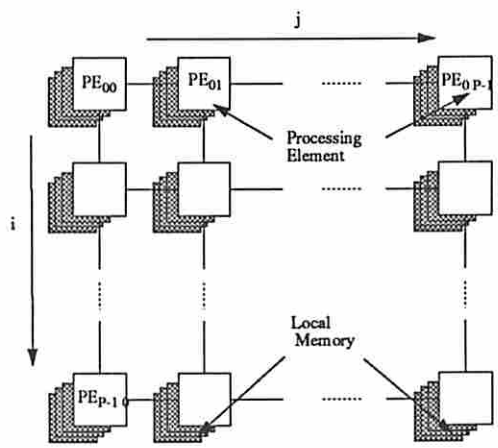

Figure 9: The mesh array model.

Following results on the mesh model are used in our algorithms:

- A permutation of P items can be routed in $O(\sqrt{P})$ time [15].

- Sort of P items takes $O(\sqrt{P})$ time [23].

Lemma 3 *Given a mesh array with S processors such that each subarray of size $\sqrt{s}$ is assigned a copy of the hash table, the voting can be performed in $O(\sqrt{S})$ time.*

Lemma 4 *Given a mesh array with S processors such that each processor has $O(1)$ (model, basis) pairs, the computing of the winner pair can be performed in $O(\sqrt{S})$ time.*

Both, Lemmas 3 and 4 can be proved similar to the proofs of Lemmas 1 and 2 respectively.

Theorem 2 *Given a mesh array of S processors, one probe of the recognition phase can be processed in $O(\sqrt{S})$ time on a scene consisting of S feature points.*

4 Implementation Details and Experimental Results

First, we describe the underlying models of the machines used in our implementations and then present data mapping and partitioning strategies employed on these machines. We also outline algorithms for computing the hash key, for voting, and for computing the winner pair. Based on these experimental results on CM-5 and MP-1 are reported.

4.1 The Connection Machine CM-5

A Connection Machine Model CM-5 system contains between 32 and 16,348 processing nodes. Each node is a 32 MHz SPARC processor with upto 32 Mbytes of local memory. A 64 bit floating point vector processing unit is optional with each node. Each processing node is a general purpose computer that can fetch and interpret its own instruction stream. System administration tasks and serial user tasks are performed by control processors. Input and output is performed via high-bandwidth I/O interfaces.

The processing nodes, control processors, and I/O interfaces are interconnected by three networks: a data network, a control network, and a diagnostic network. Figure 10 shows a diagram of the CM-5 organization. The data network provides high performance point-to-point data communications between system components. The control network provides cooperative operations, including broadcast, synchronization, and scans (parallel prefix and suffix). The diagnostic network allows back-door access to system hardware to test system integrity and to detect and isolate system errors. The system operates as one or more user partitions. Each partition consists of a control processor, a collection of processing nodes, and dedicated portions of data and control networks. Throughout this paper, size of the processor array refers to the number of PEs in a partition.

From a programmer's perspective, the CM-5 system can be considered as comprising of a control processor, a collection of processing nodes, and facilities for interprocessor communication (see Fig. 11). Each node is a general-purpose SISD processor capable of executing code written in C, Fortran, or in assembly language. Additional details of CM-5 can be found in [3].

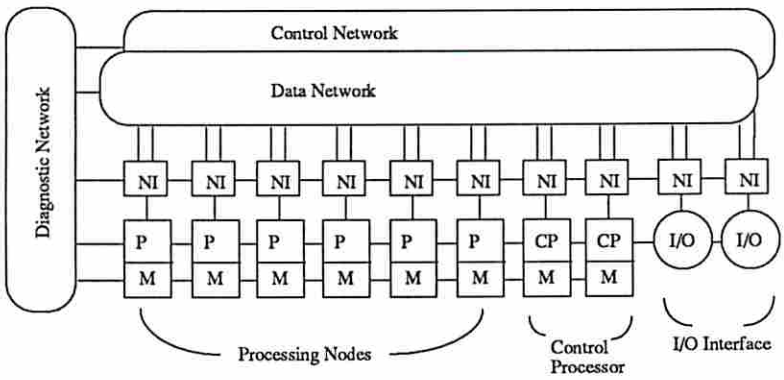

Figure 10: The organization of the Connection Machine CM-5.

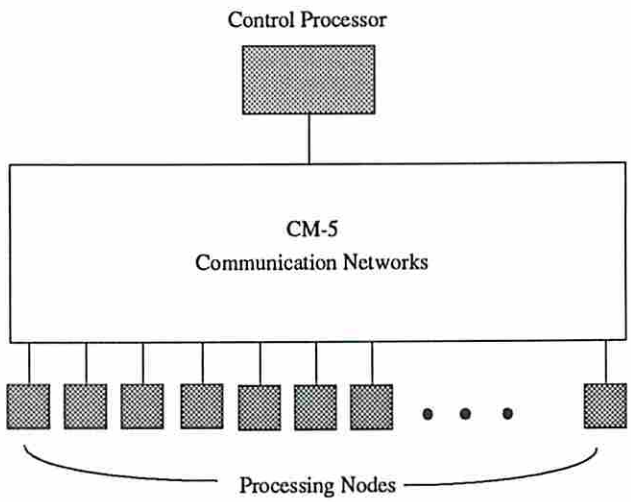

Figure 11: A virtual CM-5 organization from a programmer's perspective.

4.2 The MasPar MP-1

The MP-1 is a massively parallel SIMD computer system with upto 16K Processing Elements. The system consists of a high performance Unix Workstation as Front End (FE) and a Data Parallel Unit (DPU). The DPU consists of PEs, each with upto 64 Kbytes of memory and 192 bytes of register space. All PEs execute instructions broadcast by an Array Control Unit (ACU) in lock step. PEs have indirect addressing capability. and can be selectively disabled.

Each PE is connected to its eight neighbors via the Xnet for local communication. Besides the Xnet, MP-1 has a global router network which provides direct point-to-point global communication. The router network is implemented by a three-stage crossbar switch. It provides an equal distance (constant latency) point-to-point communication between any two PEs. A third network, called *global or-tree* is used to move data from the individual processors to the ACU. This network can be used to perform global operations on data in the entire array such as global maximum, prefix sum, global OR, etc. A block diagram of the MP-1 system is shown in Fig. 12.

MasPar currently supports MasPar Programming Language (MPL) and MasPar Fortran. MPL is MasPar's lowest level programming language and is based on ANSI C. MasPar Fortran is based on Fortran 90. We have used MPL in our implementations.

4.3 Partitioning and Mapping

Three procedures, *Compute_Keys*(), *Vote*(), and *Compute_Winner*() shown in Figures 13, 14, and 15 respectively, correspond to the steps described in the *Parallel_Probe*() procedure shown in Figure 4. The Compute_Keys() procedure computes the transformed coordinates of the scene points and quantizes them according to a hash function $f()$. The transformed and quantized coordinates are stored in NEWFP[]. We use the same hash function as in[27]. This hash function distributes the data uniformly over all the hash bins. The transformed coordinates are then used as *keys* to access the data in the hash table. The *Vote*() procedure routes the keys to their corresponding hash locations stored in $\mathrm{PE}_{g(key)}$. The function $g()$ defines the mapping of the hash table entries onto the processor array. The locations in the hash table accessed during voting are stored in CANDID[] array. This array is used in computing the final winner. The size of this array is much smaller than the size of the hash table stored in each PE.

Next, the *Compute_Winner*() procedure determines the model-basis pair receiving the maximum number of votes. The winning pair is then sent to the control processor to perform the final verification.

Based on the above described subtasks of the *Parallel_Probe*() procedure, several data mapping and partitioning strategies are developed, which affect the overall execution time of the recognition phase. In CM-5, a coarse grain SPMD machine, each processing element (PE) is a powerful SPARC processor. The high computing power of each PE and relatively expensive communication among processors motivates to partition the data such that the algorithms exhibit less communication among PEs at the cost of redundant computation within each PE. However, a balance between these two is needed to attain speed-ups.

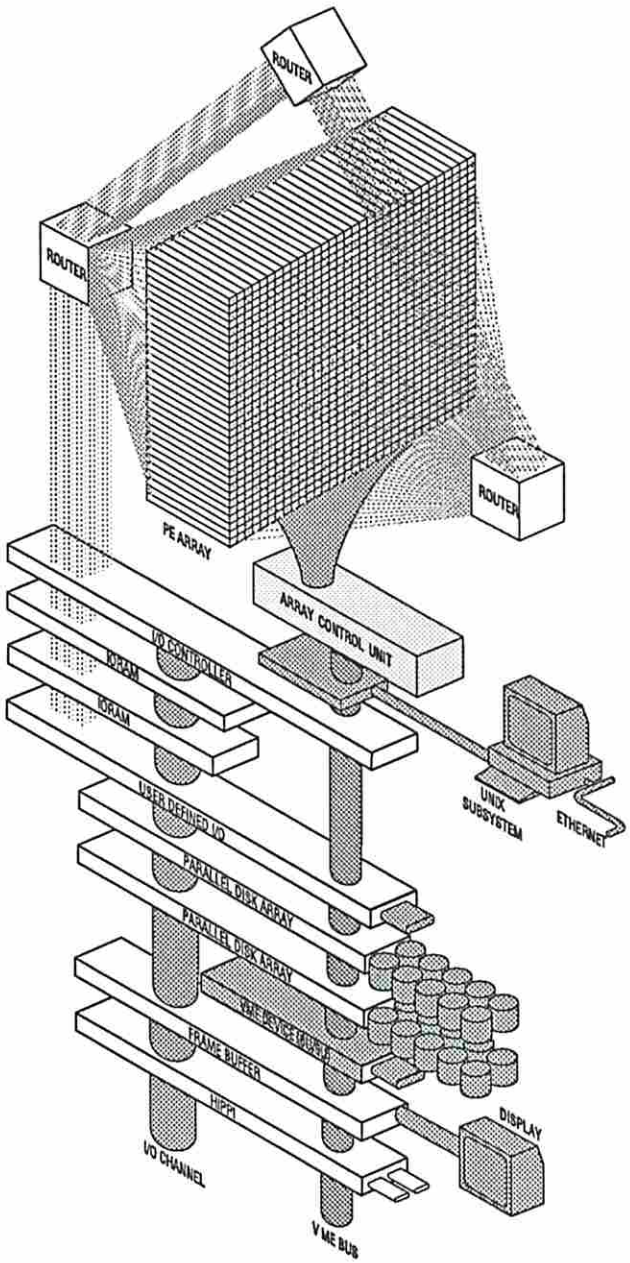

Figure 12: The MasPar MP-1 system block diagram (courtesy MasPar Computer Corporation).

```
Compute_Keys(FP, P)
  In parallel for all PE_i, 0 ≤ i ≤ P − 1, do
    for r = 0 to S/P − 1
    - Compute the transformed coordinate of the
      feature point FP[r] relative to the basis
      and store it in NEWFP[r].
    - Quantize NEWFP[r] using hash function f().
    next r
  Parallel-end
end
```

Figure 13: A parallel procedure to compute the coordinates of feature points of th e scene.

In MP-1, a fine grain massively parallel SIMD machine, each PE is a 4 bit processor. Interprocessor communication is supported through two communication networks, 1) Xnet for regular communication and 2) router for random communication. As in the geometric hashing algorithm the communication pattern is irregular, the router network provides superior performance over the Xnet [25]. It is also experienced that the ratio of unit-floating-point-computation time over unit-communication time (thru Xnet or thru contention free router) is approximately 1 [14]. It suggests to carefully partition the data such that both the computation and communication capabilities of the architecture are fully utilized.

In the following we present four algorithms, which we have experimented with. These algorithms differ with respect to partitioning and mapping of the hash table onto the processor array. Various strategies are employed to take into account practical considerations, such as available memory in each PE, processor speed, and I/O speed of the machines. In Algorithm A, and Algorithm B, we assume that each processor is assigned $\frac{Mn^3}{P}$ distinct hash table locations. In Algorithm C, each sub-array of processors is assigned a complete copy of the hash table. Each processor in a sub-array of size, s^2, where $1 \leq s \leq \sqrt{P}$, has $\frac{Mn^3}{s^2}$ distinct entries of the hash table.

The case of large number of processors is considered next. Algorithm D performs concurrent processing of multiple probes of the the recognition phase. The array is divided into disjoint sets of S processors. Each set of PEs processes a probe using a basis (a different basis for each set).

Algorithm A:

- Execute the *Comp_Keys*() procedure serially in the control processor and broadcast the encoded points (keys) to each processor in the processor array.
- Each processor scans through all the keys and accumulates votes for the keys

Vote(FPNEW, P)
In parallel, each PE_i, $0 \leq i \leq P-1$, **do**
$k := 0$;
for $j = 0$ to $S/P - 1$
- Send a vote (additive write) to processor and location specified by $g(NEWFP[j])$.
- If a vote is received, copy the contents of the corresponding hash-table entry in the array CANDID[k++].

next j
Parllel-end
end

Figure 14: A parallel procedure to vote for the possible presence of a model in the scene.

Compute_Winner()
/* Each PE is assigned $\frac{Mn^2}{P}$ distinct model-basis pairs to compute the number of votes cast to them.*/

In parallel, in each PE_i, $0 \leq i \leq P-1$
Send every element of the CANDID[] array to the PE assigned for computing the total number of votes for that element.
In parallel, in each PE_i, $0 \leq i \leq P-1$
Count the total number of votes for each distinct (*model, basis*) pair received and store it in VCOUNT[*model,basis*].
In parallel, in each PE_i, $0 \leq i \leq P-1$,
Compute the local maximum of VCOUNT[] array and store it in *local_max*.
Compute the maximum of *local_max* over the entire processor array.

/* The (*model, basis*) pair with maximum number of votes is the matched model in the scene. */
end

Figure 15: A parallel procedure to compute the winning (*model, basis*) pair.

which correspond to hash table locations stored in its local memory.

- Executes the *Compute_Winner*() procedure.

Algorithm B:

- The control processor broadcasts S/P scene points to each processor along with a basis pair.
- Execute the *Comp_Keys*() procedure
- Execute the *Merge_Keys*() procedure.
- Execute the *Vote*() procedure
- Execute the *Compute_Winner*() procedure.

Algorithm C:

In this algorithm, we assume multiple copies of the hash table stored in the processor array.

- The control processor broadcasts S/P scene points to each processor along with a basis pair.
- Execute the *Comp_Keys*() procedure
- Execute the *Merge_Keys*() procedure.
- Execute the *Vote*() procedure such that the data search is bounded with in its sub-array
- Execute the *Compute_Winner*() procedure.

Algorithm D:

In this algorithm, we assume that the number of PEs is larger than the number of feature points in the scene.

- The control processor broadcasts S/P scene points to each PE.
- The control processor broadcasts a basis pair to all PEs in each subarray of size S.
- Execute the *Comp_Keys*() procedure.
- Execute the *Merge_Keys*() procedure.
- Execute the *Vote*() procedure
- Execute the *Compute_Winner*() procedure.

4.4 Experiments and Summary of Performance Results

We have used a synthesized model database, containing 1024 models, each model consisting of 16 randomly generated points. These points are generated according to a Gaussian distribution with zero mean and unit standard deviation. The models are allowed to undergo only rigid transformation. However, results from other transformations do not affect the performance of the parallel algorithm. Similarly, scene points are synthesized using normal distribution. We apply the equalization techniques, given in [27], to the transformed coordinates, *i.e.*, for each of the transformed point (u, v), following hash function is applied.

$$f(u, v) = (1 - e^{-\frac{u^2+v^2}{3\sigma^2}}, \text{atan2}(v, u))$$

The above hash function uniformally distributes the data over the hash space such that the average hash bin length is constant. We assume a data base of 1024 models with 16 points in each model. This gives a hash table size of 4M entries. Each entry may consist of several (model, basis) pairs. We have experimented on various data granularities in the hash table comprising of average bin lengths of 1, 4, 8, 16 and 32. These granularities can be chosen according to the local memory available within each PE. We have executed these algorithms on various sizes of CM-5 and MP-1, both in terms of number of PEs in the array and local memory available within each PE. Contrary to the results reported in [27], we claim that for a single probe of the recognition phase, machine sizes larger than S would deteriorate the time performance. This is due to the fact that interconnection networks with larger diameter takes more time to perform global operations. We also show, in algorithms D, larger size machines can be used for concurrent processing of multiple probes of the recognition phase.

In the following, we tabulate our results for partitioning algorithms A, B, C, and D. Raw timing data are included in the Appendix A. Table 1 presents execution times of various subtasks using partitioning algorithms described in the previous section. The Algorithm A addresses the congestion and contention problem by computing the keys in the control processor/array control unit at the cost of redundant processing in each processor in the array. As shown in Table 1, for Algorithm A, the computation time in the control processor becomes the dominating factor in the overall execution time. We did not execute Algorithm A on MP-1 because of the large computation time and insufficient memory available within the control unit. We are unable to execute Algorithm B on various size of MP-1 as the smallest size MP-1 consists of 1K processors. The performance of Algorithms A, B, and C on various sizes of CM-5 and MP-1 is shown in Table 2.

Larger MP-1s have been used for concurrent processing of multiple probes (Algorithm D) and results are reported in Table 3. Several interesting observations on the interplay between various components of the MP-1 architecture can be made from this table. As the number of PEs increases with the number of concurrent probes, it affects various components of the execution time. For example, larger size machines mean larger diameter implying more time for global operations. On the other hand, larger machine size reduces the load on each processor, hence less time is spent on local operations.

Partitioning Algorithm	Machine Type	Encoding Scene Points	Hash Bin Access	Voting	Computing Local Maximum of Votes	Computing Global Maximum of Votes	Total Time
A	CM-5	33.3	5.45	1.04	1.83	0.29	41.45
B	CM-5	0.33	1.96	2.27	1.83	0.29	6.68
C	CM-5	0.33	1.33	1.96	1.83	0.29	5.74
B	MP-1	2.51	18.53	24.10	3.36	0.055	48.76
C	MP-1	2.51	6.78	20.02	3.36	0.055	32.72

Table 1: Execution times (in *msec*) of various subtasks in a probe using different partitioning algorithms for a scene consisting of 1024 feature points on a 256 CM-5 and 1K MP-1.

Machine Size/Type	Algorithm A (in *msec*)	Algorithm B (in *msec*)	Algorithm C (in *msec*)
32/CM-5	78.07	35.38	25.48
64/CM-5	61.04	19.75	16.02
128/CM-5	50.02	10.77	8.96
256/CM-5	41.45	6.68	5.74
512/CM-5	51.74	4.41	3.8
1K/MP-1	XX	48.76	32.72

Table 2: Execution times (in *msec*) of various algorithms on a scene consisting of 1024 feature points.

Figures 16 and 18 show the performance of Algorithms B and C. The bin access time and voting time reduces linearly as the number of copies of the hash table in the processor array increases. The hash bin access time refers to the time taken to access hash bins corresponding to feature points in the scene. The voting time corresponds to routing the information within each voted bin, (*model,basis*) pairs, to compute local maximum for each pair. In Figs. 17 and 19, we simulate worst-case and semiworst-case scenario. For the worst-case, we assume that all the scene points hash to locations stored in a single PE and in the semiworst-case, all the keys hash to locations stored in a small subset of PEs. The results show the performance of various partitioning strategies adopted in algorithms B and C. In the case of hash bin access, the access time decreases linearly with the increase in the number of hash table copies resident in the processor array. On the other hand, beyond a certain number of copies of the hash table, the voting time starts increasing (see Fig. 19). This is due to the increased network traffic generated by larger number of copies.

In Table 4, we compare our results with those reported in [1, 27]. We assume no hash table folding, symmetries, and or partial histogramming on the hash table data. Our serial implementation shows that one probe of the recognition phase takes about 13.4 *seconds* on a SUN SPARC2

Number of Probes	Machine Size	Encoding Scene Points	Hash Bin Access	Voting	Computing Local Max of Votes	Computing Global Max of Votes	Total Time
1	1K	2.51	18.53	24.10	3.36	0.055	48.76
2	2K	2.51	23.16	30.60	8.16	0.165	64.59
4	4K	2.51	40.36	49.86	8.16	0.314	101.20
8	8K	2.51	54.62	49.72	8.16	0.608	115.62

Table 3: Execution times (in *msec*) of Algorithm C on a scene consisting of 1024 feature points with concurrent processing of multiple probes on various sizes of MP-1. Average bin size is 8.

Methods	# of Models (16 points/model)	Machine[a] Size/Type	# of Scene Points	Total Time
Our Method (A)	1024	256/CM5	1024	42.76 *msec*
Our Method (B)	1024	256/CM-5	1024	6.68 *msec*
Our Method (C)	1024	256/CM-5	1024	5.74 *msec*
Our Method (B)	1024	1K/MP-1	1024	53.37 *msec*
Our Method (C)	1024	1K/MP-1	1024	38.89 *msec*
Our Method (B)	1024	1K/MP-1	200	49.50*msec*
Our Method (B)	1024	256/CM-5	200	4.50 *msec*
Hummel et. al.[27]	1024	8K/CM-2	200	800 *msec*
Medioni et. al. [1]	x	8K/CM-2	x	2.0-3.0 *sec*

Table 4: Comparison with previous implementations.

[a]Each processor in CM-5, CM-2, and MP-1 operates at 32MHz, 7MHz, and 12.5MHz respectively.

operating at 25MHz and 32 Mbytes of on board RAM.

4.4.1 Performance Comparison: MP-1 vs CM-5

During the implementation of geometric hashing algorithm, we experimented with various architectural and programming aspects of MP-1 and CM-5. Usually, SIMD machines employ fine grained massive parallelism and computationally less powerful processing elements. In MP-1, we could access machines with upto 16K processors. However, each processor has a 4-bit ALU. It takes 2.51 *msec* to encode a scene point. The encoding process comprises of approximately 7 floating point operations and 5 integer arithmetic operations. On the other hand, SPMD (synchronized MIMD) machines employ coarse grain parallelism with powerful processors as processing nodes. It takes 0.0825 *msec* to encode a scene point. However, communication intensive subtasks perform poorly on CM-5. For example, computing maximum of data elements, one element per processor, takes

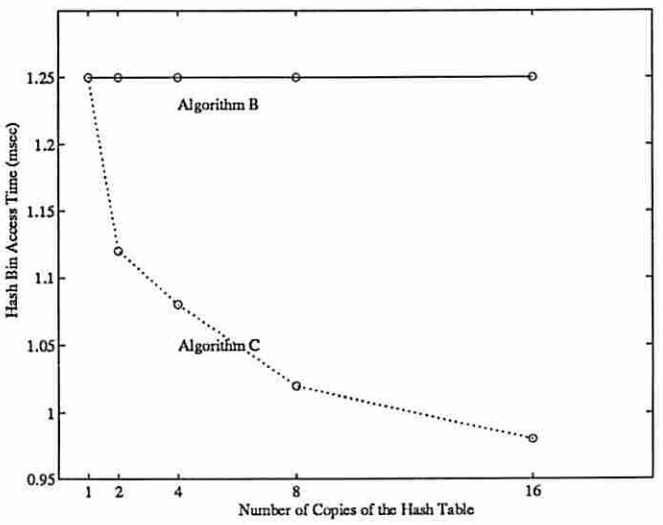

Figure 16: Hash bin access time vs Number of hash table copies for Algorithm B and Algorithm C on a 512 PE CM-5.

0.32 *msec* on a 512 processor CM-5, and it takes 0.055 *msec* on a 1024 processor MP-1. If the communication pattern is irregular, such as in the voting process, performance of SIMD machines degrades drastically. Such computation and communication characteristics suggest the use of SIMD and SPMD machines in applications with varied characteristics. Traditionally, SIMD machines are known to be well-suited for low level vision operations. However, MP-1, with an additional router network motivates the use of MP-1 for applications with moderate computational needs and regular global communication patterns. Several heuristic techniques in high level vision fall in this category. SPMD machines, such as CM-5, are suitable for applications with high computational needs and moderate global communication requirements. In addition, in the absence of efficient data partitioning and routing techniques, the performance of such machines degrades for applications with local neighborhood communication requirements. The memory available with each processor also affects the usage of the underlying architecture. Traditionally, due to limitations of VLSI and cost considerations, memory available within each processor is relatively less in SIMD machines, compared with SPMD machines. Limited memory can affect the performance of applications which require storage of large volume of on-line data.

4.4.2 Scalability of Implementations

Currently, several notions of scalability are available in the literature to analyze the performance of parallel algorithms [9, 19, 12]. However, most of these approaches do not discuss scalability in terms of implementation. We define an implementation to be scalable if the same code, can be used

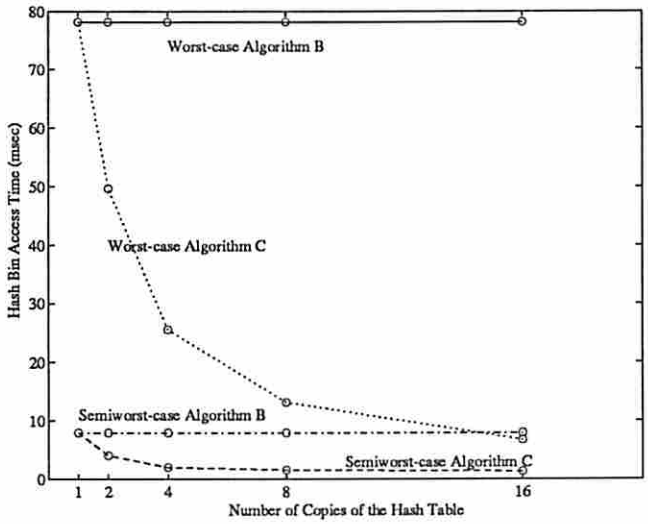

Figure 17: Hash bin access time vs Number of hash table copies for Algorithm B and Algorithm C simulating worst-case scenario on a 512 PE CM-5.

on various machine sizes and achieves proportional speed-ups. In this paper, we have developed a library of scalable programs for object recognition. The various modules that have been developed can be executed on various sizes of MP-1 and CM-5 without modification. From an implementation point of view, the time for one probe of the recognition phase on a P processor machine can be expressed as:

$$\text{Computation time} = (\frac{S}{P} \times t_c) + t_{lmax}$$

$$\text{Communication time} = (\frac{S}{P} \times t_r) + (\frac{S}{P} \times l_b \times t_r) + t_{gmax}$$

$$\text{Total time} = \text{Computation time} + \text{Communication time}$$

where,

t_c = time to compute a transformed coordinate and a hash table key for a given feature point in each PE.

t_r = time to route and vote P hash keys to their corresponding locations in the hash table.

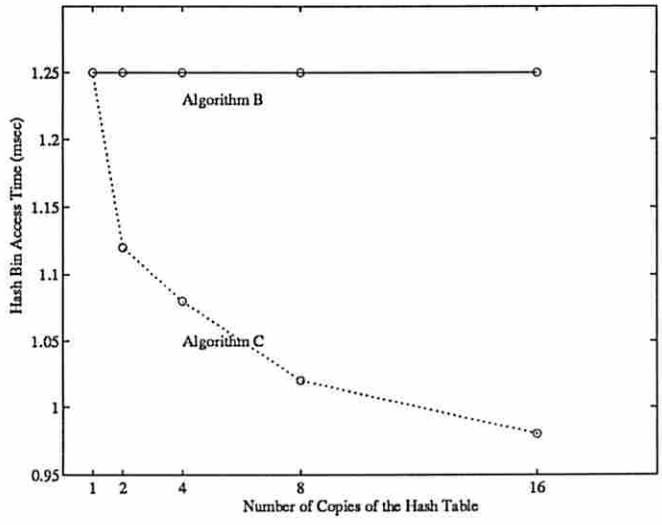

Figure 18: Voting time vs Number of hash table copies for Algorithm B and Algorithm C on a 512 PE CM-5.

t_{lmax} = time to find the local maximum of data within each PE.

t_{gmax} = data routing time to find the global maximum over the entire processor array.

l_b = maximum length of a hash table bin in any PE.

The components of the above described metrics scale linearly with the machine size. The terms t_c, t_r, and t_{lmax} depend upon the load on each processor. We have devised techniques which ensure uniform distribution of data among all the processors available in a machine. Similarly, the time to perform global maximum t_{gmax}, depends upon the machine size. Also, for machines larger than the problem size, we have used several instantiations of the same code for concurrent processing of multiple probes. As shown in the results presented in this paper, each module of our implementation scales linearly with the machine size.

5 Conclusion

We have presented scalable data parallel algorithms for geometric hashing. Based on these algorithms, we have obtained fast parallel implementations. The implementations achieve much superior time performance than those known in the literature. These implementations are developed after carefully studying the characteristics of the underlying architectures of CM-5 and MP-1, i.e. *fat tree* and mesh array, respectively. Various experiments were conducted to fine tune the partitioning and the mapping strategies to suit the communication and the computation capabilities of

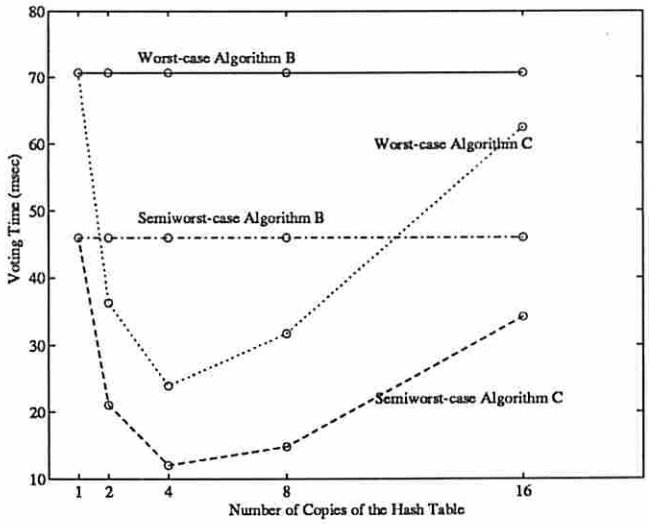

Figure 19: Voting time vs Number of hash table copies for Algorithm B and Algorithm C simulating worst-case scenario on a 512 PE CM-5.

these machines. Based on these experiments, data parallel algorithms were designed to efficiently utilize the architectural and programming features. This experimentation has assisted in achieving uniform distribution of work load in the machines during the execution of algorithms leading to fast and scalable implementations. Our early work in using CM-5 and MP-1 for high level vision is very encouraging and brings a promising future to using parallel processing in realizing real time integrated vision systems.

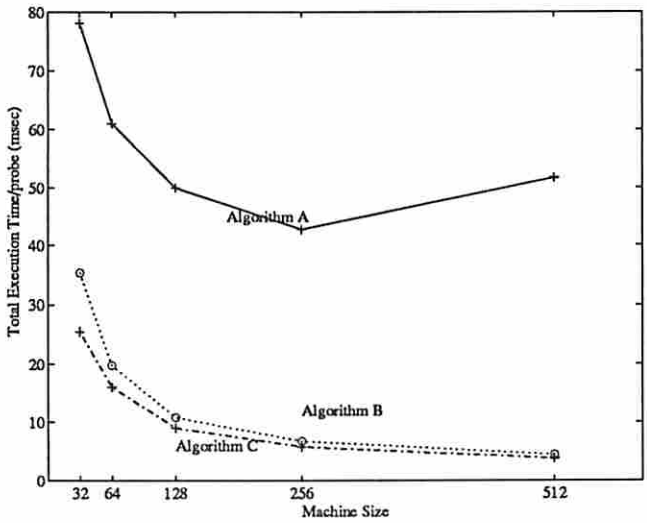

Figure 20: Total execution time vs CM-5 Machine size for various algorithms.

References

[1] O. Bourdon and G. Medioni. Object recognition using geometric hashing on the Connection Machine. In *International Conference on Pattern Recognition*, pages 596–600, 1988.

[2] A. N. Choudhary and J. H. Patel. *Parallel Architectures and Algorithms for Integrated Vision Systems.* Kluwer Academic Publishers, Norwell, MA, 1990.

[3] Thinking Machines Corporation. CM-5: Technical summary. Technical report, Thinking Machines Corporation, 1991.

[4] Charles E. Leiserson et.al. The network architecture of the Connection Machine CM-5. Technical report, Thinking Machines Corporation, 1992.

[5] W. E. L. Grimson. On the recognition of parameterized 2d objects. *International Journal of Computer Vision*, pages 2(4):353–372, 1989.

[6] W. E. L. Grimson and D. P. Huttenlocher. On the sensitivity of geometric hashing. In *International Conference on Computer Vision*, 1990.

[7] W. E. L. Grimson and D. P. Huttenlocher. On the verification of hypothesized matches in model-based recognition. In *First Europe Conference on Computer Vision*, pages 489–498, 1990.

[8] W. E. L. Grimson and T. L. Perez. Localizing overlapping parts by searching the interpretation trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(4):469–482, 1987.

[9] Anshul Gupta and Vipin Kumar. Analyzing scalability of parallel algorithms and architectures. Technical report, Department of Computer Science, Uiversity of Minnesota, Minneapolis, MN 55455, 1991.

[10] D. P. Huttenlocher and S. Ullman. Object recognition using alignment. In *IEEE First International Conference on Computer Vision*, pages 102–111, 1987.

[11] D. P. Huttenlocher and S. Ullman. Recognizing solid objects by alignment. In *DARPA Image Understanding Workshop*, pages 1114–1124, 1988.

[12] K. Hwang. *Advanced Computer Architecture with Parallel Programming.* McGraw Hill, NJ, preliminay edition edition, 1993.

[13] A. Khokhar. *Scalable Data Parallel Algorithms and Implementations for Object Recognition.* PhD thesis, Department of EE-Systems, University of Southern California, Los Angeles, January 1993.

[14] A. Khokhar, H-J. Kim, and V. Prasanna. Scalable geometric hashing on MasPar MP-1. In *Proc. International Conference on Computer Vision and Pattern Recognition*, New York, NY, 1993.

[15] V. K. Prasanna Kumar and C. S. Raghavendra. Array processor with multiple buses. *Journal of Parallel and Distributed Computing*, 4:173–190, 1987.

[16] Y. Lamdan and H. J. Wolfson. Geometric hashing: A general and efficient model based recognition scheme. In *International Conference on Computer Vision*, pages 218–249, Tempa, FL, December 1988.

[17] F. T. Leighton. Tight bounds on the complexity of parallel sorting. *IEEE Transactions on Computers*, C-34:344–354, 1985.

[18] Charles E. Leiserson. FAT-TREES:universal networks for hardware efficient superomputing. In *International Conference on Parallel Processing*, pages 393–402, 1985.

[19] Cho-Chin Lin and Viktor K. Prasanna. Using extensity as scalability measure. Technical report, Department of EE-Systems, University of Southern California, Los Anegeles, CA 90089-2562, 1993.

[20] W. Lin and V. K. Prasanna Kumar. Efficient histogramming on hypercub SIMD machines. *Computer Vision, Graphics, and Image Processing*, 49(1):104–120, January 1990.

[21] J. M. Marberg and E. Gafni. Sorting in constant number of row and column phases on a mesh. *Algorithmica*, pages 603–612, 1987.

[22] P. J. Narayana, Ling T. Chen, and Larry S. Davis. Effective use of SIMD parallelism in low- and intermediate-level vision. *COMPUTER*, v25(n2):86–92, February 1992.

[23] D. Nassimi and S. Sahni. Data broadcasting in SIMD computers. *IEEE Transactions on Computers*, C-30:101–107, 1981.

[24] P. N. Pani, A. Khokhar, and V. K. Prasanna. Parallel algorithms for stereo based on discrete relaxation techniques. In *International Conference on Pattern Recognition*, Amsterdam, Holland, August 1992.

[25] Lutz Prechelt. Measurements of MasPar MP-1261A communication operations. Technical Report DXX/93, Institut für Programmstrukturen und Datenorganisation, Universität Karlsruhe, Postfach 6980, D-7500 Karlsruhe, Germany, 1993.

[26] C. Reinhart and R. Nevatia. Efficient parallel processing in high level vision. In *DARPA Image Understanding Workshop*, pages 829–839, September 1990.

[27] I. Rogoutsos and R. Hummel. Massively parallel model matching: Geometric hashing on the Connection Machine. *Computer*, pages 33–42, February 1992.

[28] F. Stein and G. Medioni. Efficient two dimensional object recognition. In *International Conference on Pattern Recognition*, pages 13–17, Ann Arbor, MI, June 1990.

[29] C. Weems, A. Hanson, E. Riseman, and A. Rosenfeld. An integrated image understanding benchmark: Recognition of a 2 1/2 d mobile. In *IEEE International Conference on Computer Vision and Pattern Recognition*, pages 871–878, Ann Arbor, MI, June 1988.

[30] H. J. Wolfson. Model based object recognition by geometric hashing. In *First Europe Conference on Computer Vision*, pages 526–536, 1990.

Appendix A

A Additional Tables

For readers interested in additional details of implementations results, we are appending several tables comprising of data on execution times of various partitioning algorithms on various machine sizes. We have experimented on various data granularities comprising of average bin lengths of 1, 4, 8, 16 and 32. These granularities can be chosen according to the local memory available within each PE. The first column in these tables depicts the data granularity chosen for the hash table. The second column in these tables represents the average bin length corresponding to its data granularity. We have used a synthesized model database, containing 1024 models, each model consisting of 16 randomly generated points. These points are generated according to a Gaussian distribution of zero mean and unit standard deviation. The models are allowed to undergo only rigid transformation. However, results from other transformations do not affect the performance of the parallel algorithm. Similarly, scene points are synthesized using normal distribution. Unless specified, assume a scene consisting of 1024 feature points.

Hash Table		Time (in *msec*) for					
Hash Table Size	Average Bin Length	Scene points Encoding inside CP	Distribute Scene points from CP to Array	Hash Bin Access	Voting	Computing Maximum of Votes	Total Time
4M/1	1	33.3	3.16	1.85	1.04	2.10	41.45
4M/2	2	33.3	3.16	1.85	1.62	2.11	42.04
4M/4	4	33.3	3.16	1.85	2.78	2.11	43.20
4M/8	8	33.3	3.16	1.85	5.72	2.11	46.14
4M/16	16	33.3	3.16	1.85	9.37	2.11	49.79
4M/32	32	33.3	3.16	1.85	17.39	2.11	57.81

Table 5: Execution times of Algorithm A on a 256 processor CM-5.

Hash Table		Time (in msec) for					
Hash Table Size	Average Bin Length	Encoding Scene Points	Hash Bin Access	Voting	Computing Local Maximum of Votes	Computing Global Maximum of Votes	Total Time
4M/1	1	1.56	7.90	9.79	15.84	0.29	35.38
4M/2	2	1.60	7.42	13.26	15.84	0.29	38.41
4M/4	4	1.59	7.07	22.26	15.85	0.29	47.06
4M/8	8	1.63	6.85	45.71	15.83	0.29	70.31
4M/16	16	1.66	6.42	83.19	15.82	0.29	107.38
4M/32	32	1.66	6.07	155.93	15.75	0.29	179.70

Table 6: Execution times of Algorithm B on a 32 processor CM-5.

Hash Table		Time (in msec) for					
Hash Table Size	Average Bin Length	Encoding Scene Points	Hash Bin Access	Voting	Computing Local Maximum of Votes	Computing Global Maximum of Votes	Total Time
4M/1	1	0.82	4.57	6.09	7.98	0.29	19.75
4M/2	2	0.86	4.76	7.67	7.98	0.29	21.56
4M/4	4	0.87	4.14	12.55	7.97	0.29	25.82
4M/8	8	0.85	4.24	27.68	7.98	0.29	41.04
4M/16	16	0.86	3.78	49.27	7.97	0.29	62.17
4M/32	32	0.86	3.49	86.07	7.96	0.29	98.67

Table 7: Execution times of Algorithm B on a 64 processor CM-5.

Hash Table		Time (in *msec*) for					
Hash Table Size	Average Bin Length	Encoding Scene points	Hash Bin Access	Voting	Computing Local Maximum of Votes	Computing Global Maximum of Votes	Total Time
4M/1	1	0.50	3.02	3.32	3.64	0.29	10.77
4M/2	2	0.52	2.67	4.68	3.64	0.29	11.80
4M/4	4	0.52	2.57	8.45	3.65	0.29	15.48
4M/8	8	0.52	2.50	16.02	3.64	0.29	22.97
4M/16	16	0.53	2.31	26.24	3.63	0.29	33.00

Table 8: Execution times of Algorithm B on a 128 processor CM-5.

Hash Table		Time (in *msec*) for					
Hash Table Size	Average Bin Length	Encoding Scene Points	Hash Bin Access	Voting	Computing Local Maximum of Votes	Computing Global Maximum of Votes	Total Time
4M/1	1	0.33	1.96	2.27	1.83	0.29	6.68
4M/2	2	0.35	1.78	2.98	1.82	0.29	7.22
4M/4	4	0.34	1.62	5.14	1.83	0.29	9.22
4M/8	8	0.34	1.78	8.47	1.83	0.29	12.71
4M/16	16	0.34	1.78	15.88	1.83	0.29	20.12
4M/32	32	0.34	1.32	29.00	1.83	0.29	32.78

Table 9: Execution times of Algorithm B on a 256 processor CM-5.

Hash Table		Time (in msec) for					
Hash Table Size	Average Bin Length	Encoding Scene Points	Hash Bin Access	Voting	Computing Local Maximum of Votes	Computing Global Maximum of Votes	Total Time
4M/1	1	0.25	1.25	1.69	0.93	0.29	4.41
4M/2	2	0.25	1.14	1.87	0.94	0.29	4.49
4M/4	4	0.25	1.10	3.37	0.93	0.29	5.94
4M/8	8	0.25	1.06	5.52	0.93	0.29	8.05
4M/16	16	0.25	0.98	10.74	0.94	0.30	13.21
4M/32	32	0.25	1.02	20.70	0.93	0.27	23.17

Table 10: Execution times of Algorithm B on a 512 processor CM-5.

Hash Table		Time (in *msec*) for					
Hash Table Size	Average Bin Length	Encoding Scene Points	Hash Bin Access	Voting	Computing Local Maximum of Votes	Computing Global Maximum of Votes	Total Time
4M/1	1	0.22	1.14	1.04	1.82	0.28	4.50
4M/2	2	0.22	0.92	1.62	1.82	0.29	4.87
4M/4	4	0.22	0.85	2.78	1.82	0.29	5.96
4M/8	8	0.22	0.90	5.72	1.82	0.29	8.98
4M/16	16	0.22	0.82	9.37	1.82	0.29	12.52
4M/32	32	0.22	0.86	17.39	1.82	0.29	20.58

Table 11: Execution times of Algorithm B corresponding to various data granularities of the hash table on a scene consisting of 256 feature points, on a 256 processor CM-5.

Hash Table		Time (in *msec*) for					
Hash Table Size	Average Bin Length	Encoding Scene Points	Hash Bin Access	Voting	Computing Local Maximum of Votes	Computing Global Maximum of Votes	Total Time
4M/1	1	2.51	18.40	3.14	8.17	0.055	32.3
4M/2	2	2.51	18.45	6.10	8.17	0.055	35.30
4M/4	4	2.51	27.65	12.18	8.17	0.055	50.58
4M/8	8	2.52	18.53	24.10	8.17	0.055	53.37
4M/16	16	2.51	18.57	47.92	8.17	0.055	77.23
4M/32	32	2.51	18.50	95.89	8.17	0.055	125.13

Table 12: Execution times of Algorithm A corresponding to various data granularities of the hash table on a scene consisting of 1024 feature points, on a 1K processor MP-1.

Hash Table		Time (in *msec*) for					
Hash Table Size	Average Bin Length	Encoding Scene Points	Hash Bin Access	Voting	Computing Local Maximum of Votes	Computing Global Maximum of Votes	Total Time
4M/1	1	2.54	20.84	3.12	2.072	0.054	28.62
4M/2	2	2.54	20.84	6.08	2.072	0.054	31.58
4M/4	4	2.54	20.84	12.04	2.072	0.054	37.54
4M/8	8	2.54	20.84	23.99	2.072	0.054	49.50
4M/16	16	2.54	20.84	47.63	2.072	0.054	73.36
4M/32	32	2.54	20.84	95.41	2.072	0.054	120.91

Table 13: Execution times of Algorithm A corresponding to various data granularities of the hash table on a scene consisting of 256 feature points, on a 1K processor MP-1.

Hash Table		Time (in msec) for					
Hash Table Size	Average Bin Length	Encoding Scene Points	Hash Bin Access	Voting	Computing Local Maximum of Votes	Computing Global Maximum of Votes	Total Time
4M/1	1	0.25	6.75	68.84	0.93	0.29	70.42
4M/2	2	0.25	7.15	82.20	0.94	0.29	90.83
4M/4	4	0.25	6.66	84.24	0.93	0.29	92.37
4M/8	8	0.25	6.73	99.81	0.93	0.29	108.01

Table 14: Execution times of Algorithm C on a worst-case data with 16 copies of the hash table on a 512 processor CM-5.

Hash Table		Time (in msec) for					
Hash Table Size	Average Bin Length	Encoding Scene Points	Hash Bin Access	Voting	Computing Local Maximum of Votes	Computing Global Maximum of Votes	Total Time
4M/1	1	0.25	13.13	31.70	0.93	0.29	46.30
4M/2	2	0.25	13.15	34.66	0.94	0.29	49.29
4M/4	4	0.25	12.87	57.30	0.93	0.29	71.64
4M/8	8	0.25	13.06	89.03	0.93	0.29	103.56

Table 15: Execution times of Algorithm C on a worst-case data with 8 copies of the hash table on a 512 processor CM-5.

Hash Table		Time (in msec) for					
Hash Table Size	Average Bin Length	Encoding Scene Points	Hash Bin Access	Voting	Computing Local Maximum of Votes	Computing Global Maximum of Votes	Total Time
4M/1	1	0.25	1.36	34.16	0.93	0.29	36.99
4M/2	2	0.25	1.41	38.82	0.94	0.29	41.71
4M/4	4	0.25	1.31	34.17	0.93	0.29	36.95
4M/8	8	0.25	1.23	38.70	0.93	0.29	41.40

Table 16: Execution times of Algorithm C on a semiworst-case data with 16 copies of the hash table on a 512 processor CM-5.

Hash Table		Time (in msec) for					
Hash Table Size	Average Bin Length	Encoding Scene Points	Hash Bin Access	Voting	Computing Local Maximum of Votes	Computing Global Maximum Votes	Total Time
4M/1	1	0.25	1.58	14.79	0.93	0.29	17.84
4M/2	2	0.25	1.71	15.42	0.94	0.29	18.61
4M/4	4	0.25	1.68	17.68	0.93	0.29	20.83
4M/8	8	0.25	1.74	45.82	0.93	0.29	49.03

Table 17: Execution times of Algorithm C on a semiworst-case data with 8 copies of the hash table on a 512 processor CM-5.