A New Parallel Vision Environment in Heterogeneous Networked Computing

K. N. Kim[a], T. S. Choi[b], and R. S. Ramakrishna[a]

[a]Department of Information and Communications, KJIST, Kwangju, KOREA

[b]Department of Mechatronics, KJIST, Kwangju, KOREA

ABSTRACT

Many vision tasks are very complex and computationally intensive. Real time requirements further aggravate the situation. They usually involve both structured (low-level vision) and unstructured (high-level vision) computations. Parallel approaches offer hope in this context.

Parallel approaches to vision tasks and scheduling schemes for their implementation receive special emphasis in this paper. Architectural issues are also addressed. The aim is to design algorithms which can be implemented on low cost heterogeneous networks running PVM. Issues connected with general purpose architectures also receive attention. The proposed ideas have been illustrated through a practical example (of eye location from an image sequence). Next generation multimedia environments are expected to routinely employ such high performance computing platforms.

Keywords: computer vision, software environment, parallel computing, heterogeneous networked computing, PVM

1. INTRODUCTION

Computer vision finds application in a wide variety of fields such as factory inspection, automatic object tracking, character recognition, surveillance, and medical image processing. These applications impose real-time requirements, in general. In addition, vision tasks are very complex and computationally intensive. Parallel computing is a natural candidate whose potential must be explored in this context.[1] Commercial viability of parallel computers owing to phenomenal growth of the computing power of microprocessors is noteworthy in this regard.

However, straightforward replacement of existing serial platforms with parallel machines is unlikely to be successful. Appropriate parallel programs for performing vision tasks must be developed. Although use of parallelizing compilers which convert serial programs to parallel ones is a solution, the characteristics of vision tasks are basically different from most other structured computations that can exploit compiler technologies. Many parts of vision tasks consist of symbolic computations, and therefore have irregular data dependencies. Further, vision computations draw techniques from signal and image processing, advanced mathematics and artificial intelligence. It is, therefore, necessary to develop by hand suitable parallel algorithms. Each level (viz., low, intermediate, and high) has distinct computational characteristics. Low-level processing is well structured with regular computations. Locality and relaxation are found at this level. But, at higher levels, the operations on groups of features extracted from low-level computations are highly unstructured.[2]

Parallel vision environments have received due attention by the research community.[3–6] Some of them employ specialized high performance parallel computing platforms, while others employ general purpose workstations. Low cost, prior familiarity, single (virtual) machine image favour the latter. High-level programming paradigms help in the development of portable code. Parallel random access machine(PRAM, Figure. 1) model allows parallel algorithm designers to treat processing power as an unlimited resource.[1,7] The message passing paradigm has been widely used on account of its excellent support for process control and data movement between processes. It is believed that this approach to design of high speed parallel solutions to vision application is now most efficient.

Other author information: (Send correspondence to K.N.K.)
K.N.K.: E-mail: knkim@kjist.ac.kr; WWW: http://parallel.kjist.ac.kr/∼knkim
T.S.C.: E-mail: tschoi@pia.kjist.ac.kr
R.S.R.: E-mail: rsr@kjist.ac.kr

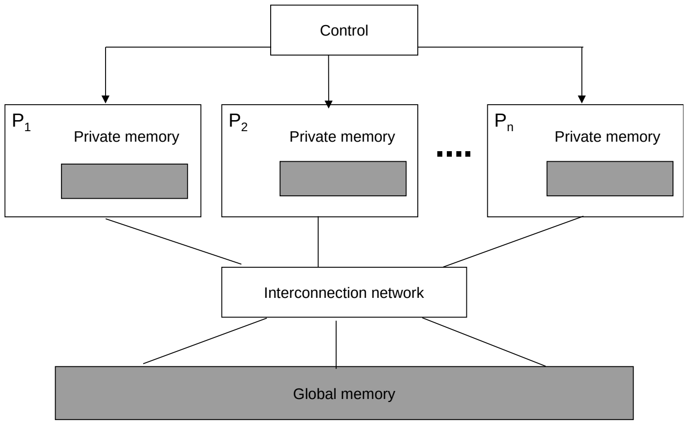

Figure 1. PRAM Model.

Three kinds of vision tasks and their characteristics are discussed in section 2. Section 3 presents a very brief review of environments and heterogeneous networked computing. Parallel algorithms and scheduling schemes are considered in section 4. Section 5 shows experimental results. And section 6 concludes the paper with a summary and directions for future work.

2. CHARACTERISTICS OF VISION TASKS

Computer vision process consists of *sensing, preprocessing, segmentation, description, recognition, and interpretation.* Sensing is the process of acquiring images. Preprocessing includes image enhancement and noise suppression. The segmentation process partitions the image into a useful set of objects and background. These techniques usually work by extracting certain features from an image by emphasizing the desired properties and suppressing the unfavorable ones and by balancing one desired property against another until favorable results are obtained. Description labels the objects with information such as their texture, size or shape. Recognition then identifies the labeled objects. Finally, interpretation puts the recognized objects into real-world context.[8,9]

2.1. Low-level processing

Low-level processing involves sensing and preprocessing of image data. The raw input image is processed to extract primitive features, such as lines and edges. The input image is represented as a 2-D array of pixels. Most operations in this level are performed using image data in the neighborhood of pixels. The communication pattern is local and processing across the image is uniform. Such algorithms are easily parallelized.

Most of the past work in the area of parallel processing for computer vision has addressed the use of parallelism for problems in low-level and some geometric problems in the interface between image processing and image understanding. Design of parallel techniques for individual problems in low-level processing seems to be reasonably well understood.

Low-level processing is usually centered around:

- *Image Quality* (eg. pointwise mean)
- *Image Enhancement* (eg. histogram modification and transformation)
- *Filtering* : (eg. linear and non-linear, spatial low, high and bandpass filters)
- *Edge Detection* (eg. roberts, prewitt, sobel)

2.2. Intermediate-level processing

As mentioned above, low-level features are grouped in this level into a hierarchy of increasingly complex and progressively abstract descriptions. For example, edges are grouped into curves, curves into surfaces and surfaces into objects. Of the many possible combinations, only some are meaningful. Efficient techniques are needed for forming and evaluating the groups.

This level of processing may involve some iconic-symbolic transformations. In perceptual grouping, lower level features, such as lines and curves, are grouped into successively more complex and abstract features such as surfaces and parts of objects. These methods typically rely on geometric and other symbolic relations between the lower level features. Parallel implementation of such grouping operations is complicated as the computations are mostly symbolic and not homogeneous for all features. The communication pattern is data dependent and irregular.

Intermediate-level processing focuses on:

- *Segmentation* (eg. thresholding, region-based segmentation)
- *Hough Transform* (eg. detecting straight lines in digital images)
- *Corner Detection* (eg. L-shaped corner detection)
- *Thinning* (eg. leaving behind a single-pixel-wide connected frame)
- *Feature and Shape Description* (eg. template matching)

2.3. High-level processing

The computations consist of both integer-based numerical operations and symbolic manipulations. Many of the high-level symbolic computations do not have localized data access patterns. This results in large communication overheads, when the computations are parallelized. In executing these high-level tasks, the flow of control and the load distribution on the processor are often unpredictable at compile time. Parallel algorithms, data partitioning and mapping methods, and load balancing strategies are needed for good speed-ups. Thus, vision tasks are computationally complex mainly because of their irregular dependencies, frequent interprocessor communication, and heterogeneous operations. Larger parallel systems do not automatically yield higher speed ups for vision applications, as they do for many other scientific applications. High processor efficiency for the irregular vision tasks is therefore difficult to achieve.

Object recognition consists of matching observed descriptions with stored models in a database. Often, moving-objects must be visually tracked. Object recognition requires matching of attribute graph structures. Commonly used techniques are those of subgraph isomorphism and of maximal cliques.

Major tasks in high-level processing:

- *Abstract Pattern Matching* (eg. object location, maximal cliques)
- *3D vision* (eg. shape from shading and texture, stereo)
- *Motion* (eg. stereo from motion)
- *Pattern Recognition* (eg. statistical pattern recognition)
- *Knowledge Based System* (eg. using a priori geometrical or topological knowledge) suitable

3. VISION ENVIRONMENTS AND HETEROGENEOUS NETWORKED COMPUTING

Considerable effort has been expended on the development of hardware and software components for experimental and simulation studies of the theory and application of computer vision.[3–6,10,11]

3.1. Vision Environments

3.1.1. Non-Parallel Vision Environment

Many serial vision systems have been built for education, research, and development purposes.[10,11] Some of them have been listed below.

- **Khoros**(*New Mexico Univ., Koral Research*), **Explorer**(*SGI*), **Wit**(*Logical Vision LTD.*), **Ad-Oculos**(*UK*), **Visix**, **IUE**(*ARPA*), **Hello-Vision**(*Kyunghee Univ.*), **KASION-III**, **IPAWB**(*Postech*), **Vision package for X windows**(*Chungang Univ.*), etc
- Strong point of these systems is that they facilitate application of existing functions in designing new algorithms.
- They are, however, inflexible and lack universal standards and intelligent support.
- They are not suitable for real time tasks.

3.1.2. Parallel Vision Environment

Several systems employing parallel architectures have been proposed with a view to enhance the speed of computation.[3–6] Some of them are listed below.

- **Paragon**(*Cornell Univ.*), **Warp**(*Carnegie Mellon Univ.*), **DISC**(*Purdue Univ.*), **IUA**(*Univ. of Massachusetts*), etc.
- These systems offer help in the development of efficient algorithms for data parallel problems.
- They are, however, very expensive and, by their very nature, architecture-specific.

3.2. Requirement

The requirements that a parallel vision environment should satisfy are outlined below.

- Ideally, parallel vision environments should support multiple architectures. Economic considerations demand that they employ existing hardware. They should also offer assistance under diverse situations in a transparent manner.
- Inclusion of facilities for task management, parallel time optimization, and load balancing would clearly enhance their usefulness.[12]

3.3. Heterogeneous networked Computing : Parallel Virtual Machine

Innovations in VLSI technology have lead to spectacular improvements in price/performance indecis of microprocessors and memory chips. The ready availability of these low cost building blocks has led to their use in parallel computing systems as well. Several commercial parallel machines such as the Cray T3E, AP-1000, and Intel Paragon have been designed. Workstation based heterogeneous networked parallel computing system - the focus of this paper - are also widespread.

The *Parallel Virtual Machine*[13](PVM) is a byproduct of an on-going research effort. It is very popular among developers of high-performance scientific computing software. And it provides a unified framework within which parallel programs can be developed in an efficient and straightforward manner using existing hardware. PVM enables a collection of heterogeneous computer systems to be viewed as a single parallel virtual machine. PVM transparently handles all message routing, data conversion, and task scheduling across a network of incompatible computer architectures.

The virtual computer resources can grow in stages and take advantage of the latest computational and network technologies. Nowadays, many distributed systems utilize workstations interconnected by an appropriate high-speed network such as Ethernet, FDDI, HiPPI, SONET, ATM, although some of them are basically experimental setups. This distributed nature can facilitate collaborative work.

But, there are several limitations too. Heterogeneity may be encountered in the form of architecture, data format, computational speed, machine load, and network load. With PVM, architecture and data format can be treated in a

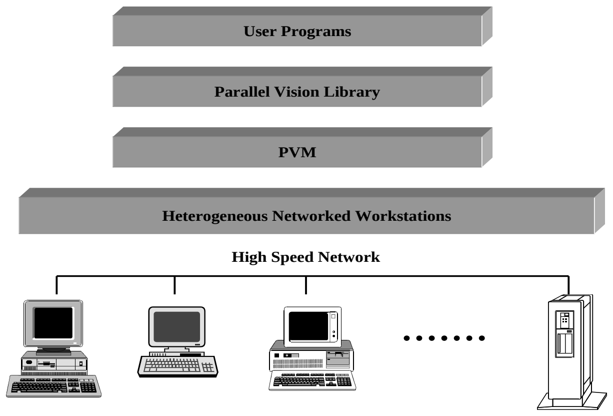

Figure 2. Heterogeneous Networked Computing Environment.

transparent manner. But, unpredictable computational speed, machine load, and network load should be explicitly taken into account. Users or software developers have to construct certain sophisticated strategies for countering these problems. And PVM cannot support perfect distributed systems. The concept of single system image is not guaranteed because certain kinds of transparency, for example location transparency, is absent. Research work in those areas have been quite exciting. As a result, load balancing, task management, and scheduling algorithms have been developed and several distributed systems have been built so far.

Figure. 2 shows the layers of a parallel vision environment. Two kinds of user levels are expected: (1) application development level and (2) algorithm development level. The former allows development of a specific vision application with a variety of functions. The latter allows the integration of the newly developed algorithms into the library.This classification is quite straight-forward. Each basic operator of different levels can be implemented by combining the respective algorithms with PVM primitives such as pvm_spawn, pvm_send, pvm_recv, and etc. If graphical user interface(GUI) level, constructed with the help of a scripting language such as TCL/TK language, is supported, then the highest level user may be considered. Similar comments apply to lower levels.

4. PARALLEL ALGORITHMS AND TASK SCHEDULING

Some aspects of parallel algorithms are illustrated through some representative tasks (Histogram, Edge Detection, Hough Transform, Geometric Hashing). A brief description of the scheduling problem is also provided below.

4.1. A Taxonomy of Vision Operations

This table can be found in [12] and considered enough to illustrate classification of overall tasks.

	Local operators		Global operators
	pixel	mask	
Data-independent	Arithmetic multiple operand single operand complex arithmetic nonlinear functions trigonometry bitwise operators comparison, logic Data conversion Format conversion Classification	2D convolution	FFT,DCT,Hadamard Statistics Histogram Shape analysis Fractal analysis
Data-dependent	Feature analysis Edge-linking Eigen values/Eigen vectors		

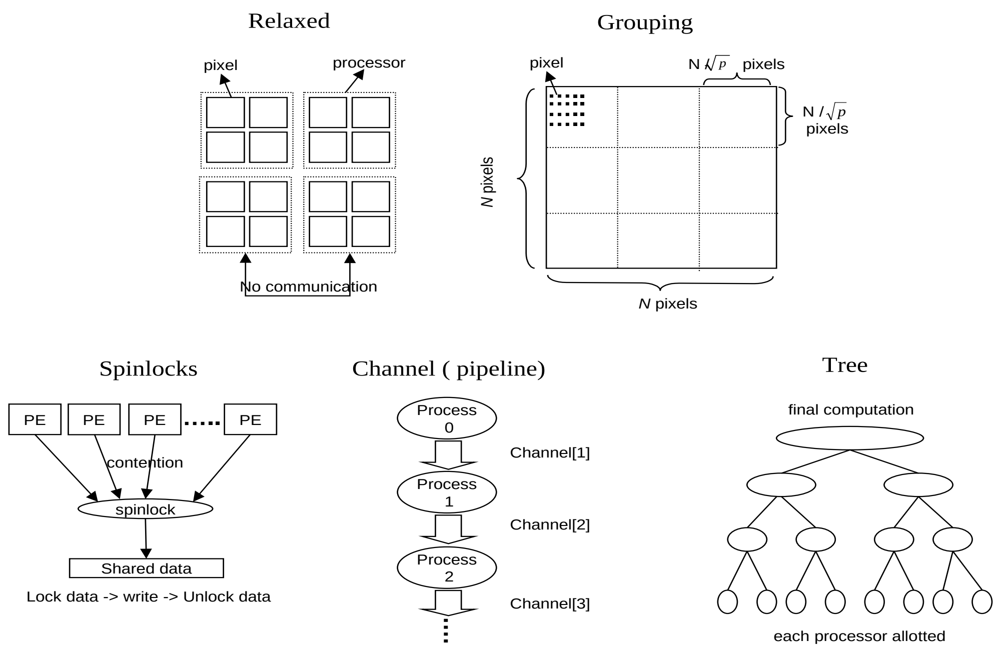

Figure 3. Scheduling Scheme 1.

4.2. Scheduling Schemes

These schemes are depicted in Figure. 3 and Figure. 4. Two scheduling schemes, viz, *local* and *global*, are well known. In local scheduling, each operator can be optimized through appropriate schemes for best performance. Also, a global job consisting of several tasks is arranged by task graph and Gantt chart. At run time, the workload is distributed dynamically among the processors. Dynamic load balancing can be classified as centralized, fully distributed, or semi-distributed. In centralized load balancing, global state information can allow the algorithm to do a good job balancing the work among the processors, but this approach does not scale well. Fully distributed load balancing allows processors to exchange information with neighboring processors. It has the advantage of lower scheduling overhead, but the workload may not be balanced as well as it would be by centralized algorithms. Semi-distributed approach is hybrid of the above two schemes. Efficient load balancing and intelligent task management must be included for better performance.[12,14,15]

Each operator requires a specific scheme in accordance with its computational characteristics. According to the taxonomy of vision operations in subsection 4.1, appropriate local scheduling schemes for each level of operation are indicated below. It is quite hard to build parallel algorithms for data-dependent operations.

- Data-independent
  - Local operators (Pixel) : *Relaxed, Grouping*
  - Local operators (Mask) : *Partially relaxed, Grouping*
  - Global operators : *Spinlock, Channel, Grouping, Tree*
- Data-dependent
  - *Partitioning, Grouping, Tree*
  - *AI-approach* (quite difficult)

Process communication, data sharing, synchronization, data partitioning, and message passing should be combined and coordinated to carry out each operation and, needless to say, user's complex applications.

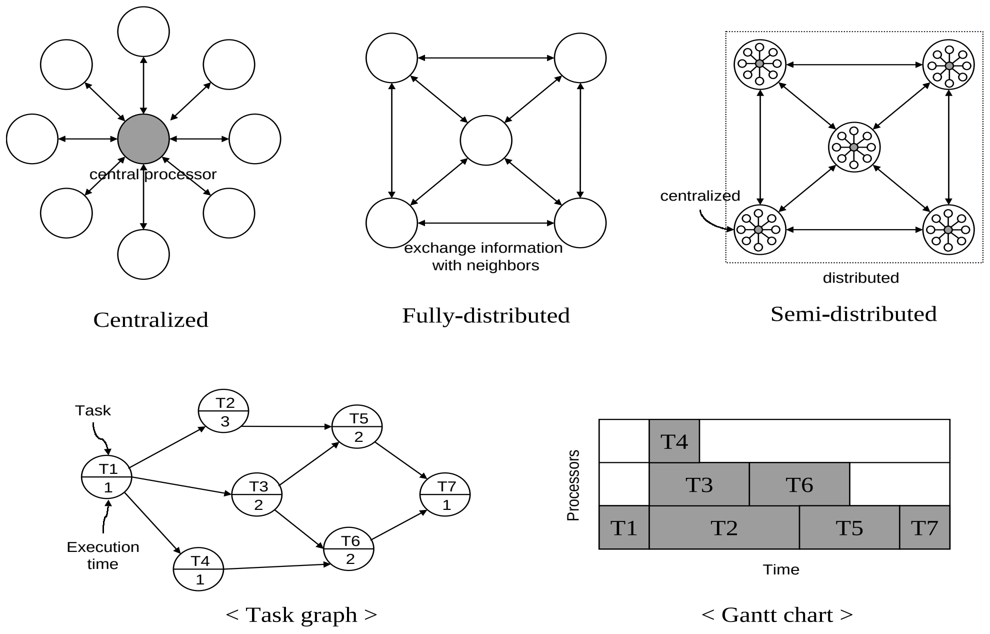

Figure 4. Scheduling Scheme 2.

4.3. Histogram

A histogram of an image is used to display the distribution of gray-values in the image. An 8-bit image has 256 gray-levels ranging from absolute black to absolute white. Generating the profile of the image is important in measuring image quality. Computing the histogram is a very common technique used in image processing as part of some larger overall process, such as identifying objects in the image.

For simplicity, the image will be represented as a two-dimensional array of integral intensity values, whose range is from 0 to some know maximum intensity. Algorithm 1 is the pascal-like sequential algorithm for computing the histogram of an array image.

Algorithm 1 Sequential algorithm for histogram

```
(* SeqHistogram *)
(* N x N image, 256 gray levels *)
BEGIN
Read pixels and store them into imgbuf (image buffer);
FOR i := 0 TO 255 DO  (* initialize histogram *)
    histogram[i] := 0;
FOR i := 1 TO N DO    (* accumulate histogram *)
    FOR j := 1 TO N do
            histogram[imgbuf[i,j]] := histogram[imgbuf[i,j]] + 1;
END.
```

This histogram algorithm can be parallelized by changing the outer sequential *FOR* loop into a *FORALL* statement(FORALL means that whole statements in the FORALL performed in parallel grouped into specified number of partitions). This will create one parallel process for each row of the image. This is shown in Algorithm 2.

Now the histogram array is being accessed and updated by all the processes in parallel. When more than two processes try to increment the value of a variable, an error can sometimes be produced. This problem is solved by making each update an *atomic* operation. One technique is to use spinlocks as given in Algorithm 2.

Algorithm 2 Parallel algorithm for histogram

```
(* ParaHistogram *)
(* N x N image, 256 gray levels, L is spinlock *)
BEGIN
Read pixels and store them into imgbuf (image buffer);
FOR i := 0 TO 255 DO  (* initialize histogram *)
    histogram[i] := 0;
FORALL i := 1 TO N DO
    VAR graylevel,j : INTEGER; (* These should be declared locally *)
    FOR j := 1 TO N DO
        BEGIN
            graylevel := imgbuf[i,j];
            LOCK(L[graylevel]); (* LOCK the histogram of this graylevel *)
                histogram[graylevel] := histogram[graylevel] + 1;
            UNLOCK(L[graylevel]); (* UNLOCK the histogram of this graylevel *)
        END;
END.
```

The following notations are used in the analysis of these algorithms.

- n : the size of problem
- p : Number of processes
- $T(n,p)$: Elapsed time (parallel time complexity)
- $T(n,1)$: Time complexity of the best known serial algorithm for solving same problem
- $S(p) = \frac{T(n,1)}{T(n,p)}$: Speedup
- $E(n,p) = \frac{S(p)}{p}$: Efficiency

In this histogram algorithm case, T(n,1), the serial time complexity is $O(n^2)$ and T(n,n), the parallel time complexity is $O(n)$.(Algorithm 2). Hence, the speedup is $O(n^2)/O(n) = O(n)$ and efficiency $O(n)/n = O(1)$, which may not be bad.

4.4. Edge Detection

An edge is a sharp discontinuity in gray-level profile. However, the situation is complicated by noise and image resolution. An edge is specified by its magnitude and its direction. A number of linked edge points may be better approximated by a linear segment called an edgel. Edges and 'edgels' form an important step in image segmentation and object recognition.

The simplest edge detector is a difference operator. Edge detection is often carried out by spatial differentiation and thresholding:

$$g_x = \frac{\partial G(x,y)}{\partial x}$$

$$g_y = \frac{\partial G(x,y)}{\partial y}$$

And the edge magnitude is given by

$$|\nabla G(x,y)| = \sqrt{{g_x}^2 + {g_y}^2}$$

Algorithm 3 Parallel algorithm for Edge Detection

```
PROGRAM Parallel Edge_Detection
VAR image[1..M,1..N] : INTEGER;    (* M : width, N : height *)
    k : INTEGER;
BEGIN
    FORALL k := 1 TO GROUPING 10
        VAR j : INTEGER;           (* j : local iteration *)
        BEGIN
            FOR j := 1 TO N
                BEGIN
                    G[k,j] := Gradient_Magnitude(k,j);
                    IF G[k,j] > Threshold THEN
                        (* set this pixel to edge point *)
                END
        END

FUNCTION Gradient_Magnitude(x,y : INTEGER) : FLOAT;
BEGIN
    (* Prewitt,Sobel,or Robinson, etc *)
    Gx = 2D_Convolution (x,y,MASK_X1);
    Gy = 2D_Convolution (x,y,MASK_Y1);
    Gradient_Magnitude := SQRT(Gx^2 + Gy^2);
END;
END.
```

The Laplacian is given by

$$L(G(x,y)) = g_{xx} + g_{yy} = \frac{\partial^2 G}{\partial x^2} + \frac{\partial^2 G}{\partial y^2}$$

The difference operators are local neighborhood operations and they are efficiently implemented on parallel array computers because of local neighborhood connectivity. The pseudo code of parallel implementation of the edge detection is given in Algorithm 3.

The directive *GROUPING* n means that each child process has n iterations as its load. Thus, totally M/n processes are created for parallel processing. In this case $M/10$ processes perform computations with 10 columns of array of image pixels. T(n,1), serial time complexity is $O(MN)$ and T(n,p), parallel time complexity is $O(10N)$. The speedup is measured as $O(MN/10N) = O(M/10)$, and efficiency as $O(\frac{M/10}{M/10}) = O(1)$ because these operations are loosely coupled and there is complete data relaxation.

4.5. Hough Transform

The algorithms derived from the formulation of the Hough transform(HT) have been shown to be efficient tools for achieving the objective of detecting lines, circles, ellipses, or generalized shapes for image recognition. So far, many different algorithms for HT have been designed. Especially, using the Fast HT(FHT), the detection is carried out through a hierarchical approximation to the solution. The way in which this process is carried out permits dispensing with vote accumulation in the parameter space. The focalization to the solution is applied by means of a division and deepening process in the parameter space. To this end, a square parameter space is generated and divided into four quadrants. A given quadrant is again divided if a noticeable number of lines traverse it (greater than a threshold value).[16,17]

FHT can be parallelized by assigning a cluster of lines in parameter space to each processor or distributing a set of nodes of quad tree into each processor. This requires synchronization, extra memory space, and intelligent load balancing strategies for achieving high performance.

4.6. 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 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 hypothesis through a search process.*Geometric hashing* offers a different and more parallelizeable paradigm. It can be used to recognize flat objects under weak perspective.

5. EXPERIMENTAL RESULTS

5.1. Example Vision Tasks

Eye location detection from a sequence of facial images has been chosen for experimental study, because it is very computationally intensive and requires real-time processing. This topic is actively being investigated in our laboratory as part of a project on intelligent user interface and facial communication. Details of the algorithm are not important at this stage. Only its computational aspects will be explained for better understanding. A key technique to find eye location is *adaptive histogram templates matching.*[18] Although it consists of several steps such as constructing templates, setting search range, computing histograms, and matching decision, the task of computing histograms is the bottle-neck which delays the entire set of operations. This part must be accelerated by some efficient high performance computing technique. Moreover, it is highly parallelizeable within search range. The histograms are then compared by employing the minimum distance criterion.

The major blocks of computation in this example are: (1) converting RGB values (RGB $\Rightarrow$ customized Intensity values), (2) histogram computation of the area in search range, and (3) comparing entire histograms computed in the previous block with template histogram.

Block (2) forms the heaviest workload, taking about 70$\sim$80% of the entire load. The first block is fully data parallel without any interprocess communication. Synchronization and granularity considerations are imperative in the second. And the last is performed with the help of tree and grouping schemes.

5.2. Result Graphs and Images

Figure 5 depicts the experimental results pertaining to execution time, speedup, efficiency, and evaluation factor. The graphs show acceptable results in this specific experiment (eye location detection). Figure 6 shows the set of images which result after locating the eyes.

6. SUMMARY AND FUTURE WORK

Vision tasks have different computational characteristics at different levels (low, intermediate, and high). At higher levels, regular computational structures which are found in the lower level disappear and irregular data dependencies appear. Therefore, it is quite difficult to achieve good performance over the entire vision process. But, there is scope for parallel processing here. A method for developing parallel solutions for vision tasks on the heterogeneous networked computing environment was illustrated through practical examples. Economic and practical considerations favour the use of general purpose workstations as computing nodes. Adding or deleting computing points is also easily done (extensibility issue). Neural and optical computing techniques offer possible solutions to problems in computer vision. Load balancing, however, remains a very hard problem. The challenge is to build transparent distributed vision systems which exploit the intrinsic parallelism in an efficient manner.

REFERENCES

1. M. J. Quinn, *Parallel Computing: Theory and Practice 2nd Ed.*, McGRAW-HILL, 1994.
2. C.-L. Wang, P. B. Bhat, and V. K. Prasanna, “High-performance computing for vision,” *Proc. of The IEEE* **84**, pp. 931–946, Jul 1996.
3. L. H. Jamieson, E. J. Delp, and C.-C. Wang, “A software environment for parallel computer vision,” *Computer* **25**(2), pp. 29–45, 1997.

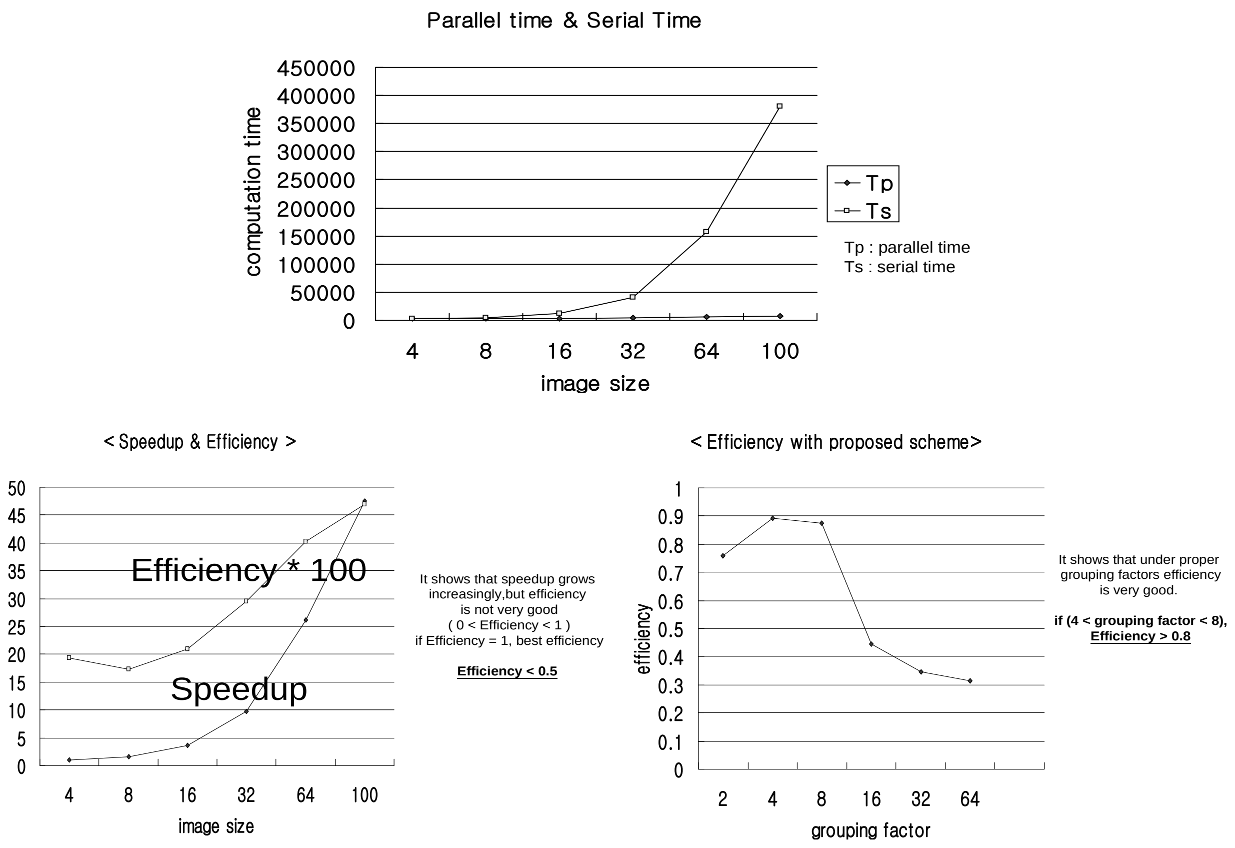

Figure 5. Result Graphs.

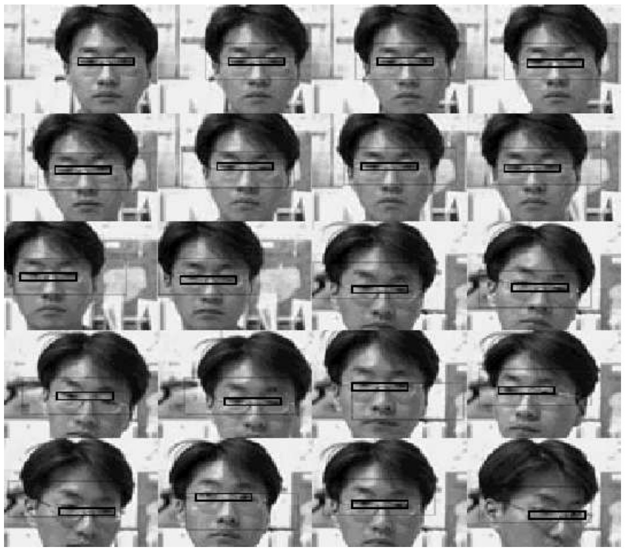

Figure 6. Images after Eye Loaction.

4. C. C. W. Jr., "The second generation image understanding architecture and beyond," in *Proc. 1993 Computer Architecture for Machine Perception*, pp. 276–285, 1993.
5. N. Alexandridis, P. Papaioannou, B. Narahari, and A. Youssef, "A software environment of architecture prototypes for evaluating parallel vision systems and algorithms," in *IEEE Int. Workshop on Tools for Artificial Intelligence, 1989. 'Architecture, Languages and Algorithms'*, pp. 518–525, 1989.
6. V. K. Prasanna, "Parallel architectures and algorithms for image understanding," ch. Software Environments, pp. 453–562, Academic Press, INC, 1991.
7. B. P. Lester, *The Art of Parallel Programming*, Prentice HALL, 1993.
8. Horn, Berthold, Klaus, and Paul, *Robot Vision*, MIT, 1986.
9. G. X. Ritter and J. N. Wilson, *Handbook of Computer Vision Algorithms in Image Algebra*, CRC Press, 1996.
10. H. K. Song, W. S. Jung, and J. S. Choi, "On the development of a software package for vision system," *Journal(B) of Korea Information Science Society* **22**, Sep 1995.
11. J. H. Lee and O. S. Chae, "Integrated development environment for management and reuse of computer vision and image processing algorithms," *Journal of Korea Information Science Society* , 1996.
12. C. Lee, Y.-F. Wang, and T. Yang, "Global optimization for mapping parallel image processing tasks on distributed memory machines," *J. Parallel and Distrib. Computing* **45**, pp. 29–45, 1997.
13. A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sundream, *PVM: Parallel Virtual Machine, A Users' Guide and Tutorial for Networked Parallel Computing*, MIT Press, 1994.
14. A. R. Nolan and B. Everding, "Polynomial time scheduling of low level computer vision algorithms on networks of heterogeneous machines," in *Int. Conf. on Parallel Processing, Stockholm Sweden*, Aug 1995.
15. D. Gerogiannis and S. C. Orphanoudakis, "Load balancing requirements in parallel implementations of image feature extraction tasks," *IEEE Trans. on Parallel and Distributed Systems* **4**, Sep 1993.
16. N. Guil, J. Villalba, and E. L. Zapata, "A fast hough transform for segment detection," *IEEE Trans. on Image Processing* **4**, Nov 1995.
17. N. Guil and E. L. Zapata, "Fast hough transform on multiprocessor: A branch and bound approach," *J. Parallel and Distirb. Computing* **45**, pp. 82–89, 1997.
18. K.-I. Kang, J.-H. Ra, and M.-H. Kim, "Tracking the eye trajectories in dynamic images using adaptive template mathcing," in *Proc. of the 9th Spring Conf. of Korea Information Processing Society*, 1998.