1993

# A Coarse-Grained, Architecture-Independent Approach for Connected Component Labeling

Mikhail J. Atallah
*Purdue University*, mja@cs.purdue.edu

Frank Dehne

Susanne E. Hambrusch
*Purdue University*, seh@cs.purdue.edu

Report Number:
93-008

# A COARSE-GRAINED, ARCHITECTURE-INDEPENDENT APPROACH FOR CONNECTED COMPONENT LABELING

**Mikhail J. Atallah**
**Frank Dehne**
**Susanne E. Hambrusch**

**CSD-TR-93-008**
**January 1993**

# A Coarse-grained, Architecture-independent Approach for Connected Component Labeling

Mikhail J. Atallah[*]
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907, USA
mja@cs.purdue.edu

Frank Dehne[†]
School of Computer Science
Carleton University
Ottawa, Canada K1S 5B6
dehne@scs.carleton.ca

Susanne E. Hambrusch[‡]
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907, USA
seh@cs.purdue.edu

January 19, 1993

**Abstract**

Let $I$ be a binary image of size $\sqrt{n} \times \sqrt{n}$ stored in a coarse-grained $p$-processor parallel machine in which each processor has $O(n/p)$ local memory and stores a $\sqrt{n/p} \times \sqrt{n/p}$ subimage of $I$. We consider the problem of determining the connected components of $I$. Our algorithm makes no assumptions about the architecture of the parallel machine, and it is expressed in terms of local computations and communication rounds. Communication rounds tend to be expensive in any parallel architecture, and thus the objective is to minimize the number of communication rounds. It is easy to label the connected components in $O(1)$ communication rounds, using $O(n/p)$ local computation steps, when one assumes $n \geq p^3$. When no assumptions about the relative sizes of $n$ and $p$ are made, simulating any of the published solutions would result in at least $\log p$ communication rounds. We give a solution for determining the connected components that requires at most $\log\log p$ communication rounds.

[*]Research supported in part by the Air Force Office of Scientific Research under Contract AFOSR-90-0107 and by the National Science Foundation under Grant CCR-9202807.

[†]Research supported in part by the Natural Sciences and Engineering Research Council of Canada.

[‡]Research supported in part by DARPA under contract DABT63-92-C-0022ONR.

## 1 Introduction

Most commercially available parallel machines (e.g., the Intel Paragon, Intel iPSC/860, and CM-5) are *coarse-grained* in the sense that a processor is typically a state-of-the art unit with considerable processing power and local memory. This contrasts sharply with the $O(1)$ local memory traditionally assumed in *fine-grained* models and algorithms. Another feature of commercially available parallel machines is that basic communication primitives (e.g., routing, broadcasting, and sorting) are usually available as system calls or as highly optimized utilities. By using these primitives, an applications programmer can design solutions in an architecture-independent setting without having to be familiar with the specific communication patterns of the problem being solved. Nevertheless, the fact remains that communication primitives are typically much more expensive than local computations. These facts motivate the parallel model we used in this paper, which we describe next.

Our parallel model is the architecture-independent *coarse-grained, communication round model*. In this model $n$ inputs are evenly distributed among $p$ processors (i.e., every processor stores $n/p$ of the $n$ input items). A processor has a memory size of $O(n/p)$ (we assume that the associated constant allows the implementation of our algorithms). The processors communicate via an interconnection network in a *communication round* in which they specify the type of communication to occur. Typical operations to be performed in a communication round are broadcasts, routings, or sorting. Algorithms are designed by specifying the local computation done within each processor between two communication rounds, and by specifying the type of communication performed in a communication round. This model has recently been used to develop new and efficient algorithms for computational geometry problems [3]. Designing algorithms within the coarse-grained, communication round model has the considerable advantage of being easier to program than algorithms that re-implement the communication operations. However, they still place the burden on the algorithm designer of designing schemes that use as few communication rounds as possible.

Let $I$ be a binary image of size $\sqrt{n} \times \sqrt{n}$. Determining the connected components of $I$ is the problem of assigning labels to the entries of value '1' (called the 1-pixels) so that two 1-pixels have the same label if and only if there exists a path of 1-pixels between them. We consider 4-connectivity (i.e., two consecutive pixels on the path are either vertically or horizontally adjacent), but trivial modifications to our scheme handle 8-connectivity. Determining the connected components in images is a fundamental problem in image processing [10, 11], and fine-grained algorithms for different architectures have been developed [1, 2, 4, 6, 7, 8, 9]. In our coarse-grained model we assume that image $I$ is stored in a $p$-processor machine so that each processor contains a subimage of size $\sqrt{n/p} \times \sqrt{n/p}$. The mapping of the subimages to the processors is done in the standard way (i.e., number subimages in row-major fashion, and assign the $i$-th subimage to processor $i$, $1 \leq i \leq p$). In this paper we show that the connected components of $I$ can be determined in $O(\log \log p)$ communication rounds *for all values of $n$ and $p$*. Furthermore, when $n \leq p^{2+\epsilon}$, $\epsilon > 0$, the labeling is determined in a constant number of communication rounds. Simulating known connected component labeling algorithms on the communication round model would result in at least $\geq \log p$ communication rounds. We point out that our result is, in some sense, not comparable to that of [12]. Phrased in terms of communication rounds, the algorithm of [12] would require $\log p$ rounds. Its good performance in the model used comes from its own problem-specific and architecture-specific communication schemes.

Our model and algorithm make a contribution towards the design of scalable algorithms. A *scalable algorithm* is an algorithm that maintains its speedup when parameters of the parallel environment change (in our case these parameters are $p$ and $n$). The design of scalable algorithms is one of the main goals of the recent High Performance Computing and Communication Initiative [5]. Since parallelism has played and will continue to play a crucial role in the area of image processing and computer vison, an architecture-independent framework leading to scalable algorithms is relevant to application researchers in this area. Since the model presented in this paper corresponds closely to the parallel machines currently available, our framework will be useful in solving other image processing problems.

Our objective is to design solutions that minimize the number of communication rounds. We state the running time $T$ of an algorithm executing $k$ communication rounds as

$$T \;\leq\; k \cdot (T_{local}(n/p) + T_{comm}(p, n/p)),$$

where $T_{local}$ is the maximum time spent on local computations within a processor between two consecutive communication rounds. In our algorithm we have $T_{local}(n/p) = O(n/p)$. $T_{comm}(p, n/p)$ is the maximum time spent on executing a communication round on a $p$-processor parallel machine, when every processor has at most $n/p$ data items participating in the communication. In our algorithms a communication round executes either a routing or a broadcasting operation. The routings are such that the destinations are specified at the time the program is written (i.e., they are data-independent). For a number of architectures (e.g., the hypercube) such data-independent routings can be performed more efficiently than arbitrary, data-dependent routings. The broadcast operations executed by our algorithm are of the following type: a processor partitions $n/p$ of its data items into segments so that the $i$-th segment is sent to processor $b_i$.

For certain values of $n$ and $p$, in addition to the above-mentioned $k$ communication rounds, our running times have an additive "fine-grained" term which consists of the time taken by $p$ processors to solve a problem of size $p$ when the $p$ data items are stored with one data item per processor. For a given problem $\Pi$, let $T_{fine}(\Pi, p)$ be the time to solve $\Pi$ by a fine-grained algorithm. Such a fine-grained term is not a serious problem and will typically be much smaller than the above time $T$ associated with the communication rounds. One can argue that such an additive term is unavoidable, since the fine-grained case is just a special case of the coarse-grained case; namely the one where $n = p$. Hence, when $n = O(p)$, specifying an algorithm in terms of communication rounds will resemble a specification of a fine-grained solution. For common interconnection networks there exist efficient fine-grained solutions for almost all relevant problems.

This paper is organized as follows. In Section 2 discuss our overall approach, describe some of the data structures used, and give a crucial "compression theorem". In Section 3 we describe the algorithm used when $n = p^2 f(p)$, $f(p) \geq 1$, and in Section 4 we describe the one used when $p \leq n < p^2$.

## 2 Data Structures and Compression Schemas

As already stated in the introduction, our algorithm can handle all values of $n$ and $p$. Our algorithm combines, as is often done in coarse-grained algorithms, parallel and sequential problem-solving approaches. The parallel approach that we use is that of data reduction. (In data reduction a problem is solved by extracting from the data a problem of smaller size whose solution leads to the solution of the original problem.) It is well known that connected component labeling can be solved by data reduction and many efficient parallel algorithms, including the one we present in this paper, exploit this property. Depending on the relative size of $n$ and $p$, we use a somewhat different approach. Our algorithm differs from previous solutions in the way the size of the smaller problems is identified and how this size is made dependent on the relative size of $n$ and $p$. W.l.o.g. we assume throughout that $n$ is a multiple of $p$. (This avoids unnecessarily cluttering the exposition. The algorithms can easily be modified for the general case).

Let $R$ be a $w \times l$ subimage of image $I$. Our data reduction technique relies on the fact that in order to generate the final labeling of the 1-pixels in $R$, only a representation of the connected components lying in $R$ and having 1-pixels on the boundary of $R$ is needed in subsequent computation steps. Such a representation requires significantly less space than the entire subimage $R$. In our algorithm we choose the boundary graph for this representation. The *boundary graph* of $R$ is a planar graph consisting of at most $2w + 2(l-2)$ vertices and $2w + 2(l-2) - 1$ edges, and it is obtained from the connected components of $R$ as follows. Let $C$ be a connected component of image $R$ that has $k$ of its 1-pixels, say $x_1, x_2, \ldots, x_k$, on the boundary of $R$. Assume these 1-pixels are ordered so that $x_i$ is the $i$-th 1-pixel of $C$ encountered when traversing the boundary of $R$ in clockwise order, starting at the top left corner of $R$. Then, the boundary graph of $R$ contains vertices $x_i$, $1 \leq i \leq k$, and edges $(x_i, x_{i+1})$, $1 \leq i \leq k-1$.

A basic sequential operation of our algorithm is the *merging* of boundary graphs. Let $R_1$ and $R_2$ be two horizontally adjacent subimages of image $I$, each of size $w \times l$. W.l.o.g. assume $R_1$ is to the left of $R_2$. We sketch how to generate the boundary graph of the image $R_1 \cup R_2$ in $O(w+l)$ sequential time. Assume the vertices of boundary graph $R_i$, $i = 1, 2$, are represented so that the $k$ vertices belonging to a common connected component can be identified, in clockwise order, in $O(k)$ time, and that all of $R_i$'s vertices can be identified, in clockwise order, in $O(w+l)$ time. This can easily be accomplished by using linked list structures. In addition, assume that every vertex knows its row and column in image $I$.

We first form an intermediate graph $G'$ that consists of the vertices and edges in $R_1$ and $R_2$, plus edges between $R_1$ and $R_2$ induced by adjacent 1-pixels. Let $x_i$ (resp. $x_j$) be a vertex in the boundary graph for $R_1$ (resp. $R_2$) that is in row $r_i$ (resp. $r_j$) and column $c_i$ (resp. $c_j$) in image $I$. If $r_i = r_j$ and $c_i = c_j - 1$, we add the edge $(x_i, x_j)$ to the intermediate graph $G'$. (If we were to use 8-connectivity instead of 4-connectivity, we would change the way edges between $R_1$ and $R_2$ are formed.) We then determine, in $O(w + l)$ sequential time, the connected components of $G'$ and record for every vertex in the data structure used for $R_1$ and $R_2$, respectively, the new component number of this vertex. The boundary graph for $R_1 \cup R_2$ is then generated, in $O(w + l)$ sequential time, by using these updated data structures.

In general, our algorithms will not merge two, but $y$ boundary graphs whose associated subimages are arranged in a $\sqrt{y} \times \sqrt{y}$ super-grid (grid of grids) and which are stored in one processor. By generalizing the procedure sketched above, the connected components of the corresponding intermediate graph can be determined in $O(y(w + l))$ sequential time. Then, by traversing the updated data structures of the boundary graphs lying on the boundary of the $\sqrt{y} \times \sqrt{y}$ super-grid, the new boundary graph can be generated in an additional $O(\sqrt{y}(w + l))$ sequential time.

Our connected component labeling algorithm consist of a *forward* phase, followed by a *backward* phase. The final objective of the forward phase is to determine the boundary graph for image $I$. Using this information, the backward phase then updates the connected component information of the vertices in all boundary graphs generated during the forward phase. Once all boundary graphs have been updated, the 1-pixels of every subimage stored at a processor are labeled.

Throughout, it will be convenient to view the processors as being *logically* arranged as a two-dimensional grid. Such an arrangement in no way implies that the processors are *physically* arranged as a grid (hence we are *not* assuming a mesh architecture). This should be kept in mind in what follows, where we repeatedly refer to (logical) "grids" and "subgrids" of processors. The forward phase of our algorithm determines the sizes and numbers of subimages whose boundary graphs are to be merged. The following compression theorem is crucial to the performance of our algorithm.

**Theorem 1** *Let $S$ be a $\sqrt{\hat{p}}\sqrt{\frac{n}{p}} \times \sqrt{\hat{p}}\sqrt{\frac{n}{p}}$ subimage of image $I$. When $\hat{p} < \frac{n}{p}$, the boundary graph of $S$ can be stored in one processor and it can be determined in $k \leq -\log\left(1 - \frac{\log \hat{p}}{\log \frac{n}{p}}\right)$ communication rounds.*

**Proof:** The boundary graph of $S$ contains at most $4(\sqrt{\hat{p}}\sqrt{\frac{n}{p}} - 1)$ vertices and can obviously be stored in one processor when $\hat{p} < \frac{n}{p}$. The remainder of the proof describes how to determine the boundary graph of $S$ in $k$ communication rounds.

Initially image $S$ is partitioned among $\hat{p}$ processors. Every such processor sequentially determines the boundary graph of its $\sqrt{\frac{n}{p}} \times \sqrt{\frac{n}{p}}$ image. This is done in $O(n/p)$ time by finding the connected components of the image and creating from it the boundary graph. After this computation, the first communication round takes place. In general, right before the $i$-th communication round, a subset of the $\hat{p}$ processors contain boundary graphs that were generated either after the $(i-1)$-st communication round (if $i > 1$) or from the stored subimages (if $i = 1$). Assume every such boundary graph contains at most $z_{i-1}$ vertices. These boundary graphs are now compressed by having $y_i$ processors send their boundary graph to the same processor. After the $i$-th communication round has been completed, every processor containing $y_i$ boundary graphs determines the boundary graph for the new, larger subimage. This is done in $O(y_i z_{i-1})$ sequential time by combining the $y_i$ graphs as described above.

In order for the $i$-th communication round to work correctly we need $z_{i-1} y_i \leq \frac{n}{p}$ (because a processor has only $O(n/p)$ local memory). From the $(i+1)$-st round we get $(\sqrt{y_i} z_{i-1}) y_{i+1} \leq \frac{n}{p}$, giving $y_{i+1} = \sqrt{y_i}$. Since $y_1 = \sqrt{\frac{n}{p}}$, we have $y_i = \left(\frac{n}{p}\right)^{\frac{1}{2^i}}$. After the $i$-th communication round, $\frac{\hat{p}}{y_1 y_2 \cdots y_i}$ processors contain boundary graphs of subimages of image $S$. The compression procedure terminates when only one processor contains a boundary graph and this boundary graph is that of $S$. The number of communication rounds, $k$, must thus satisfy:

$$\prod_{i=1}^{k} y_i \geq \hat{p}$$

which is equivalent to saying:

$$\left(\frac{n}{p}\right)^{\sum_{i=1}^{k} \frac{1}{2^i}} = \left(\frac{n}{p}\right)^{1-\frac{1}{2^k}} \geq \hat{p}.$$

This implies

$$\left(1 - \frac{1}{2^k}\right) \log \frac{n}{p} \geq \log \hat{p}$$

which is equivalent to:

$$k \leq -\log \left(1 - \frac{\log \hat{p}}{\log \frac{n}{p}}\right).$$

□

We point out that the assumption $\hat{p} < \frac{n}{p}$ is crucial for the termination of the procedure. Assume $n = p^2$ and $\hat{p} = p$. For these parameters the boundary graph of every square subimage can be stored in a processor, but the number of communication rounds is unbounded. This follows because $p^{1-\frac{1}{2^k}} \leq p$ holds for $k$. Our algorithms will thus carefully choose the subimages to which the compression theorem is applied to.

When our algorithms use the compression theorem, the processor assignment is of course crucial. Boundary graphs generated before the $i$-th communication round need to be available for the backward phase. The following processor assingment guarantees that every processor stores at most two boundary graphs, the first one being the initial one generated from the image and a possible second one generated by combining boundary graphs. For the $i$-th communication round we logically partition the $\hat{p}$ processors into subgrids of size $\sqrt{y_1 y_2 \cdots y_i} \times \sqrt{y_1 y_2 \cdots y_i}$. The $i$-th processor in the first row of each subgrid receives the $y_i$ boundary graphs present in the subgrid, $1 \leq i < k$. These boundary graphs were either generated in the previous iteration (if $i > 1$) or from the input image (if $i = 1$).

## 3 The Case of $n = p^2 f(p)$

We start with the description of a straightforward algorithm for $n \geq p^3$ that requires only two communication rounds to label the connected components of image $I$. In the first step of this algorithm every processor determines, in $O(n/p)$ sequential time, the boundary graph of the $\sqrt{\frac{n}{p}} \times \sqrt{\frac{n}{p}}$ subimage of $I$ assigned to the processor. The $p$ boundary graphs are then sent to one processor, say processor 1. Processor 1 receives at most $4\sqrt{\frac{n}{p}}p = 4\sqrt{pn}$ edges. For $n \geq p^3$, we have $\sqrt{pn} \leq \frac{n}{p}$ and thus processor 1 can store all the edges it receives. Using the $p$ boundary graphs, processor 1 determines the boundary graph of image $I$ and then updates the connected component information in each of the $p$ boundary graphs. In a second communication round the updated boundary graphs are sent back to their original processor, where the final labeling is generated in $O(n/p)$ sequential time. The algorithm uses only two communication rounds and the total running time is $2 \cdot (O(n/p) + T_{route}(p, 4\sqrt{\frac{n}{p}}))$.

Assume now that $n = p^2 f(p)$ with $1 \leq f(p) < p$. Visualize the $p$ processors as being logically arranged in a $\sqrt{p} \times \sqrt{p}$ grid and partition this grid into subgrids of size $p^{\frac{1}{2}-\frac{1}{\alpha}} \times p^{\frac{1}{2}-\frac{1}{\alpha}}$, for some constant $\alpha > 2$. This generates a total of $p^{\frac{2}{\alpha}}$ subgrids. Since $p^{1-\frac{2}{\alpha}} < \frac{n}{p}$, we can apply Theorem 1 to each subgrid with $\hat{p} = p^{1-\frac{2}{\alpha}}$. After

$$k_1 \leq -\log\left(1 - \frac{\left(1-\frac{2}{\alpha}\right)\log p}{\log(pf(p))}\right)$$

communication rounds one processor in each subgrid contains the boundary graph of the subimage stored in the $p^{1-\frac{2}{\alpha}}$ processors of the subgrid. Observe that this boundary graph consists of at most $4p^{\frac{1}{2}-\frac{1}{\alpha}}\sqrt{\frac{n}{p}} = 4p^{1-\frac{1}{\alpha}}\sqrt{f(p)}$ vertices. Since $1 \leq f(p) < p$ and $\alpha > 2$, we have

$$\log\frac{\alpha}{2} \leq k_1 < \log\left(\frac{1}{2} + \frac{1}{\alpha}\right) < \log\alpha = O(1).$$

Next we logically partition the $\sqrt{p} \times \sqrt{p}$ grid of processors into $p^{\frac{1}{\alpha}}$ vertical slabs. Every slab contains $p^{\frac{1}{2}-\frac{1}{\alpha}}$ consecutive columns of the $\sqrt{p} \times \sqrt{p}$ grid. In terms of the subgrids used in the previous step, every slab contains $p^{\frac{1}{\alpha}}$ subgrids. Since $p^{1-\frac{1}{\alpha}}\sqrt{f(p)} \cdot p^{\frac{1}{\alpha}} \leq \frac{n}{p} = pf(p)$, the edges of all the boundary graphs associated with the subgrids of a slab can be stored in one processor. We send each boundary graph to a designated processor within the same slab and determine the boundary graph for each slab. The last step of the forward phase determines the boundary graph of image $I$ by combining the boundary graphs of the slabs using a $\sqrt{f(p)}$-ary merge. In one communication round $\sqrt{f(p)}$ boundary graphs are sent to one processor and the boundary graph of the union of the subimages associated with these slabs is determined. After

$$k_2 \leq \log_{\sqrt{f(p)}} p^{\frac{1}{\alpha}} = \frac{2\log p}{\alpha \log f(p)}$$

communication rounds the boundary graph of image $I$ is stored in one processor and the forward phase is completed. In total, the forward phase uses

$$k_1 + k_2 + 1 \leq \log\alpha + \frac{2\log p}{\alpha\log f(p)} + 1$$

communication rounds. Choosing $\alpha = \frac{2\log p}{\log f(p)}$, gives a bound of $\log\left(\frac{2\log p}{\log f(p)}\right) + 2$ on the number of communication rounds. The forward phase is thus completed in time

$$T \le (\log\left(\frac{2\log p}{\log f(p)}\right) + 2) \cdot (O(n/p) + T_{route}(p, n/p)).$$

We briefly discuss the processor allocation of the forward phase. As already stated, every processor stores at most one additional boundary graph (besides the one generated from the input subimage). Applying Theorem 1 to the subgrids gives the processor allocation described in Section 2. Within each slab we then use the second processor in the first column to receive the boundary graphs of the subgrids. We then use the subsequent processors in the first column of each slab to store the boundary graphs generated during the $\sqrt{f(p)}$-ary merge. The backward phase generates the final labeling by running the computation backwards and making the necessary updates.

We conclude this section by explicitly stating the number of communication rounds for two values of $n$. For $n = p^2$, we use $\log\log p + 2$ communication rounds, while for $n = p^{2+\epsilon}$, $0 < \epsilon < 1$, the algorithm terminates in a constant number of communication rounds ($\log\frac{1}{\epsilon} + 3$, to be precise).

## 4 The Case of $p \le n < p^2$

For $n < p^2$, we have $\sqrt{n} > \frac{n}{p}$, and thus the boundary graph of image $I$ can no longer be stored in one processor. Hence, the technique of using a single processor to merge boundary graphs of subimages has to modified. Our algorithm now determines the largest subimage whose boundary graph can be stored in one processor and then combines these boundary graphs by using a fine-grained algorithm. We next describe the details of this algorithm.

The forward phase starts by partitioning the $\sqrt{n} \times \sqrt{n}$ image into subimages of size $\frac{n}{p} \times \frac{n}{p}$, resulting in a total of $\frac{p^2}{n}$ subimages. Each subimage contains $n' = \frac{n^2}{p^2}$ pixels and we assign to it $p' = \frac{n}{p}$ processors. We then determine the boundary graph of each subimage by applying the forward phase of the algorithm for $n' = (p')^2$ described in the previous section. After this, $\frac{p^2}{n}$ of the $p$ processors contain a boundary graph on at most $4n/p$ vertices. Let $G'$ be the intermediate graph created when these $\frac{p^2}{n}$ boundary graphs are merged to create the boundary graph of the entire image. Every processor containing a boundary graph of a subimage determines which edges of the intermediate graph $G'$ are incident to its vertices (this can be accomplished through one communication round, followed by a local computation). The total number of edges in $G'$ is $O(p)$, and we can distribute the edges of $G'$ so that every processor receives $O(1)$ edges. We then apply a fine-grained algorithm for determining the connected components of a graph to the edges of $G'$. This completes the forward phase. The backward phase generates the final labeling by running the computation backwards and making the necessary updates. The running time is thus bounded by

$$T \leq (\log\log p + 2) \cdot (O(n/p) + T_{comm}(p, n/p)) \; + \; T_{fine}(GC, p),$$

where $T_{fine}(GC, p)$ is the time to solve graph connectivity when every one of the $p$ processors contains a constant number of edges of an undirected graph.

## References

[1] Cypher, R., Sanz, J., Snyder, L., "Algorithms for Image Component Labeling on SIMD Mesh Connected Computers", *IEEE Trans. on Computers*, 1990, Vol. 39, pp. 276-281.

[2] Cypher, R., Sanz, J., Snyder, L., "Hypercube and shuffle- exchange algorithms for image component labeling", *Journal of Algorithms*, 1989, Vol. 10, pp. 140-150.

[3] Dehne, F., Fabri, A., Rau-Chaplin, A., "Parallel Geometric Algorithms for Coarse-Grained Multiprocessors", to appear in *Proceedings of 9-th ACM Symposium on Computational Geometry*, 1993.

[4] Dyer, C., Rosenfeld, A., "Parallel Image Processing by Memory- Augmented Celluar Automata", *IEEE Pami*, 1981, Vol. 3, pp 29- 41.

[5] *Grand Challenges: High Performance Computing and Communication*, a report by the Committee on Physical, Mathematical, and Engineering Sciences, to supplement the U.S. President's Fiscal Year 1992 Budget.

[6] Hambrusch, S.E., TeWinkel, L., "A Study of Connected Component Algorithms on the MPP", *Proc. of 3rd Internat. Conf. on Supercomputing*, 1988, pp 477-483.

[7] Lim, W., Agrawal, A., Nekludova, L., "A fast parallel algorithm for labeling connected components in image arrays", Techn. Report NA86-2, Thinking Machines Corp., 1986.

[8] Levialdi, S., "On Shrinking Binary Picture Patterns", *CACM*, 1972, Vol. 15, pp. 7-10.

[9] Nassimi, D., Sahni, S., "Finding Connected Components and Connected Ones on a Mesh-Connected Parallel Computer", *SIAM J. on Comp.*, 1980, pp. 744-757.

[10] Pavlidis, T., *Algorithms for Graphics and Image Processing*, CSP, 1982.

[11] Rosenfeld, A., Kak, A., *Digital Picture Processing*, Academic Press, 1982.

[12] Sanz, J., Cypher, R., "Data Reduction and Fast Broadcasting: A Strategy for Efficient Algorithms and Message-Passing Parallel Computing", *Algorithmica*, 1992, Vol. 7, pp. 77-89.