8-1990

# Mesh and Pyramid Algorithms for Iconic Indexing

Alok Choudhary
*Syracuse University*

Sanjay Ranka
*Syracuse University*

SU-CIS-90-25

# Mesh and Pyramid Algorithms for Iconic Indexing

Alok Choudhary[1] and Sanjay Ranka

August 1990

*School of Computer and Information Science*
*Syracuse University*
*Suite 4-116*
*Center for Science and Technology*
*Syracuse, NY 13244-4100*

*(315) 443-2368*

[1]This work was supported in part by the Engineering Foundation grant no. 3537143.

# Mesh and Pyramid Algorithms for Iconic Indexing

Alok Choudhary [1] and Sanjay Ranka

School of Computer and Information Science

4-116 CST

Syracuse University

Syracuse, NY 13244-4100

August 2, 1990

[1]This work was supported in part by the Engineering Foundation grant no. 3537143.

## Abstract

In this paper parallel algorithms on meshes and pyramids for iconic indexing are presented. Our algorithms are asymptotically superior to previously known parallel algorithms.

# Mesh and Pyramid Algorithms for Iconic Indexing

Alok Choudhary [1] and Sanjay Ranka

School of Computer and Information Science

4-116 CST

Syracuse University

Syracuse, NY 13244-4100

August 2, 1990

[1]This work was supported in part by the Engineering Foundation grant no. 3537143.

# 1 Introduction

Several approaches have been proposed for pictorial information retrieval. They include giving-by-pictorial-example [1], database queries [2], quad-trees [3], and iconic indexing [4]. As proposed in [4], images in a pictorial database can be represented by symbolic pictures. In other words, a symbolic picture contains symbols to denote objects in the source image. The retrieval can be performed by matching the symbolic picture representing the query with the symbolic representation of the pictorial database. For example, a query can be "Find all pictures containing a jeep to the right of a house," or "Find all pictures containing a man next to a dog."

In [4] a method is proposed to represent a symbolic pictures and a picture query by two-dimensional string (2-D string). The problem of pictorial information retrieval then becomes a problem of 2-D subsequence matching. This approach allows an efficient way to construct iconic indexes [21]. Iconic index means a linear index containing symbols from the symbolic picture being indexed and representing the spatial relations in the picture.

In this paper we present parallel algorithms for iconic indexing on massively parallel mesh and pyramid architectures. We present algorithms for Type-2, Type-1, and Type-0 matching problems (as defined in [4] and section 4). Furthermore, we introduce another matching problem (called Type-3) and present algorithms for it. Given a symbolic picture $F$ (dimension $n \times n$) and a symbolic pattern $P$ (dimension $m \times m$), the proposed algorithms have the following asymptotic execution time on a $n \times n$ mesh and an $n \times n$ base pyramid. Type-2 matching can be performed in $O(m^2)$ on a mesh and in $O(m^2)$ on a pyramid. By using hashing techniques the same algorithms can be performed in $O(m)$ on a mesh and in $O(m)$ on a pyramid. Type-2 matching in cases where wild characters exist (a wild character can match with any other character) can be performed in $O(m^2)$ on a mesh and $O(m^2)$ on a pyramid. Type-1 matching takes $O(min(n^2, m^2 n))$ time a mesh as well as a pyramid. Type-0 matching can be performed in $O(n)$ on a mesh and $O(min(n, m^2 + \log n))$ time on a pyramid. Finally, Type-3 matching takes $O(m^3)$ on a mesh and $O(m^3)$ on a pyramid. Our algorithms for pyramid architecture are asymptotically superior to the one presented in [5].

The rest of this paper is organized as follows. In Section 2, a description of mesh and pyramid architectures is provided. Section 3 describes some preliminary operations required for Section 4 and 5. Section 4 defines the matching problems and contains examples. Section 5 presents the parallel algorithms for iconic indexing on massively parallel meshes and pyramids. Concluding remarks are given in Section 6.

# 2 Model of Computation

The block diagram of an SIMD multicomputer is given in Figure 1. The important features of an SIMD multicomputer and the programming notation we use are:

1. There are $P$ processing elements connected together via an interconnection network. Different interconnection networks lead to different SIMD architectures. Each processing element (PE) has a unique index in the range $[0, P-1]$. We shall use brackets ([ ]) to index an array and parentheses ('( )') to index PEs. Thus, $A[i]$ refers to the $i$-th element of array $A$ and $A(i)$ refers to the $A$ register of PE $i$. Also, $A[j](i)$ refers to the $j$-th element of array $A$ in PE $i$. The local memory in each PE holds data only (i.e., no executable instructions). Hence PEs need to be able to perform only the basic arithmetic operations (i.e., no instruction fetch or decode is needed).

2. There is a separate program memory and control unit. The control unit performs instruction sequencing, fetching, and decoding. In addition, instructions and masks are broadcast by the control unit to the PEs for execution. An *instruction mask* is a boolean function used to select certain PEs to execute an instruction. For example, in the instruction

$$A(i) := A(i) + 1, \ (i_0 = 1).$$

($i_0 = 1$) is a mask that selects only those PEs whose index has bit 0 equal to 1; i.e., odd indexed PEs increment their $A$ registers by 1. Sometimes we shall omit the PE indexing of registers. So, the above statement is equivalent to the statement:

$$A := A + 1, \ (i_0 = 1).$$

3. We shall consider the following interconnection networks:

   (a) **Mesh:** A $P = n \times n$ mesh connects $n^2$ PEs that are logically arranged as a two-dimensional array (Figure 2) . Each PE has a unique index in the range $(0 \ldots n-1, 0 \ldots n-1)$. PE$(i,j)$ is connected to PE$((i-1) \bmod n, j)$, PE$((i+1) \bmod n, j)$, PE$(i, (j-1) \bmod n)$, and PE$(i, (j+1) \bmod n)$. Sometimes we will use a one-dimensional indexing of the mesh. This is obtained using the standard row major mapping in which $(i,j)$ is mapped $in+j$. A number of mesh-connected computers have been constructed. Examples include the CLIP4 [6,7], the GAPP [8], and the MPP [9,10].

   (b) **Pyramid:** A pyramid with an $n \times n = 2^{2q}$ base connects $P = (4n^2-1)/3$ processors. These PEs form $q+1$ meshes of size $n \times n$, $n/2 \times n/2, \ldots, 1 \times 1$, respectively. These meshes are stacked one on top of the other in decreasing order of size and interconnected as shown in Figure 3. Each PE has a unique index PE$(i,j,k)$, where $0 \leq i \leq q$ and $0 \leq j,k \leq n-1$. Examples of pyramid computers that have been or are being built include the HCL Pyramid [11,12], the MPP Pyramid [13], the SPHINX [14], and PAPIA [15].

   The following relationships can be defined for a pyramid.

      i. The father of the PE $(l,i,j)$ is the PE $(l-1, \lceil i/2 \rceil, \lceil j/2 \rceil)$, where $0 < l \leq \log n$ and $0 \leq i,j \leq 2^l - 1$.

      ii. The sons of the PE $(l,i,j)$ are the PEs $(l+1, 2i-1, 2j-1)$, $(l+1, 2i-1, 2j)$, $(l+1, 2i, 2j-1)$ and $(l+1, 2i, 2j)$, where $0 \leq l < \log n$.

4. Interprocessor assignments are denoted using the symbol $\longleftarrow$, while intraprocessor assignments are denoted using the symbol $:=$. Thus the assignment statement:

$$B(i,j) \longleftarrow B(i, (j+1) \bmod n), (i = 0)$$

   on a mesh is executed only by those processors in the 0-th row. These processors transmit their $B$ register data to the processors on their left.

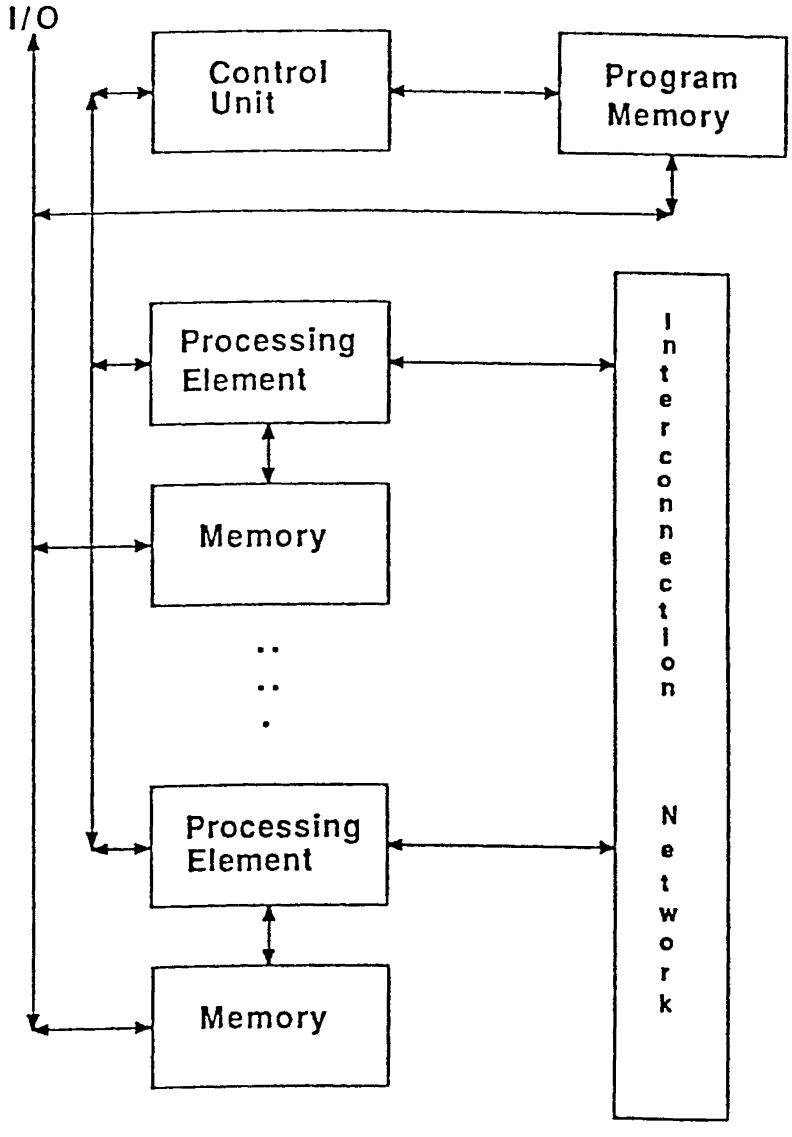


Figure 1: An SIMD Multicomputer

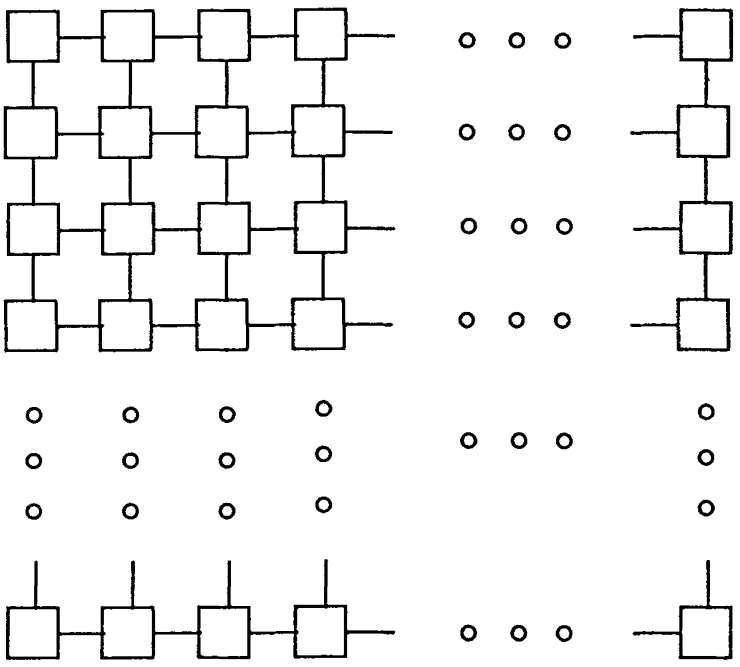

Figure 2: A $4 \times 4$ Mesh (end around connection are not shown)

5. In a *unit route*, data may be transmitted from one processor to another if it is directly connected.

6. Since the asymptotic complexity of all our algorithms is determined by the number of unit routes, our complexity analysis will count only these.

## 2.1 Image Mapping

The image is mapped on the mesh such that PE$(i,j)$ contains I$(i,j)$. For the pyramid, the image is mapped on the base mesh. Thus PE$(\log n, i, j)$ contains I$(i,j)$.

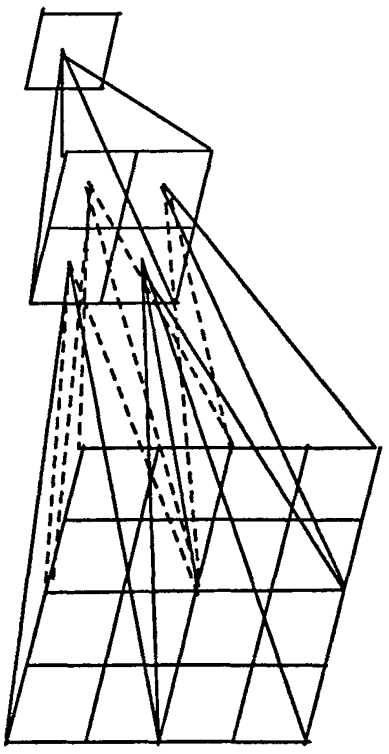

Figure 3: A 21-Node Pyramid

# 3 Preliminaries

## 3.1 Sorting

On a $n \times n$ mesh sorting can be done in $O(n)$ time using the algorithm of [22][10]. The same algorithm can be used to complete sorting on a pyramid in $O(n)$ time.

## 3.2 Compress

Assume that each PE has a certain record. Further, the PEs are sorted according to some key on the record. After performing the compress operation duplicate records are combined together and a count field is attached to them which refers to the number of records with that value. For example $(2,2,2,3,4,4,5,6)$ is replaced by $((2,3),(3,1),(4,2),(5,1),(6,1),-,-,-)$, where $-$ represents the null symbol. This can be completed in $O(n)$ time on a mesh and a pyramid by using the algorithms for ranking and concentrating of [17].

## 3.3 Hashing

As suggested in [23] a hash function can be use to reduce the two-dimensional matching problem into a one-dimensional matching. A hash function can be defined as follows : $h(r) = r \bmod q$, where $q$ is a large prime number and $r$ is the symbol. For a pattern of length $m$, each symbol is transformed into an integer and packed into a binary string (integer). This corresponds to writing the symbols as numbers in a $radix - d$ number system, where $d$ is the number of possible symbols. The number $k$ corresponding to a pattern of length $m$ (say, $P(i) \cdots P(i+m-1)$) is :

$$k = ord(P(i) \times d^{m-1}) + ord(P(i+1) \times d^{m-2}) + \cdots + ord(P(i+m-1)$$

$$k = ord(P(i+m-1)) + d(ord(P(i+m-2)) + \cdots + d(ord(P(i))))$$

where $ord(x)$ is the order of the symbol $x$. Shifting the pattern one position to the right (or down) corresponds to replacing $k$ by

$$(k - ord(P(i) \times d^{m-1}) \times d + ord(P(i+m)).$$

Therefore, obtaining a hash of each block of consecutive patterns of size $m$ in a string of length $M$ takes $O(M+m)$ steps if it is performed sequentially.

## 3.4 Shift

SHIFT$(A, i)$ shifts the $A$ register data circularly counter-clockwise by $i$. It can be performed in $|i|$ unit routes.

## 3.5 Data Accumulation

For this operation, PE $j$ has an array $A[0 \ldots m-1]$ of size $m$. The notation $A[i](j)$ refers to $A[i]$ in PE $j$. In addition, each PE has a value in its $I$ register. After the data accumulation the $m$ elements of $A$ in PE $j$ are such that:

$$A[i](j) = I((j+i) \bmod P),\ 0 \leq i < m,\ 0 \leq j < P.$$

This operation can be performed in $(m-1)$ unit routes.

## 3.6 AC Operation

Assume that each PE $i$ has a record $a_i$. An AC operation is an associative and commutative operation on all values of $a_i$ (like $+, \times$, or, and). Let * be an AC operation. The result of the AC operation is $a_1 * a_2 * \cdots a_P$. It can be performed on an $n \times n$ mesh in $O(n)$ time and $O(\log n)$ time on an $n \times n$ base pyramid.

## 3.7 Merge

Assume that each PE has two records $A$ and $B$. Consider the case of 4 PEs. Let $A = (1,3,4,5)$ and $B = (3,4,5,6)$. The merge of $A$ and $B$ is $(1,3,3,4,4,5,5,6)$. Each PE contains

two values. This operation can be completed in $O(n)$ time on a mesh and a pyramid.

### 3.8 Random Access Read (RAR)

In this formulation an index $S(i)$ is contained in PE$(i), 0 \leq i < n$. PE$(i)$ is to receive data from PE$(S(i))$. We shall assume that the data to be transmitted to PE$(i)$ is originally in register $D(S(i))$. ($D(j)$ denotes register or memory cell $D$ in PE$(j)$.) Also, if PE$(i)$ is not to receive data from any other PE, then $S(i) = \infty$. Random Access Read can be completed in $O(n)$ time on a mesh [17] and a pyramid.

## 4 The Matching Problems

Let $V$ be a set of symbols. Each symbol may represent a pictorial object (a named object such as "house," "car," etc.) or a part of an object. A symbolic picture is a mapping $S \times S \longrightarrow W$ where $S = \{1, 2, \ldots, n\}$ and $W$ is the power set of $V$. The null object is denoted by { } [6].

A symbolic pattern is a mapping $S \times S \longrightarrow W \cup \{\#\}$ where "#" is a wild character , i.e., a "don't care symbol" which matches all the elements in $W$.

Given a symbolic picture $F$ and a symbolic pattern $P$ (with dimensions $n \times n$ and $m \times m$, respectively), we want to detect the occurrences of the pattern $P$ in $F$.

In the following we make use of the array-like notation $F(i,j)$ (resp. $P(i,j)$) to denote the $(i,j)$-th element of $F$ (resp. $P$). Also we will use characters ("a," "b," etc.) as symbols rather than named objects.

An occurrence of a pattern $P$ can be searched for inside the picture, depending on three different criteria. Given a symbolic pattern $P(m \times m)$ and a symbolic picture $F(n \times n)$, $P$ has a type-$t$ ($t = 0, 1, 2$ and 3) occurrence in $F$ if:

($t = 2$) there exists a pair $(k,l)$ of indices with $1 \leq k \leq n - m + 1$ and $1 \leq l \leq n - m + 1$ such that $F(k,l) = P(1,1)$ and

$$F(k + i - 1, l + j - 1) = P(i,j)$$

for $1 \leq i \leq m$ and $1 \leq j \leq m$.

In other words, $P$ has a type-2 occurrence in $F$ if it occurs somewhere in $F$ in its native configuration; this can be viewed as a 2-D pattern-matching problem. Note that the right member of the equality above determines the occurrence of $P$ in $F$.

($t = 1$) there exists a pair $(k, l)$ of indices and two ascending sequences of positive integers $(x_1, x_2, \ldots, x_{m-1}), (y_1, y_2, \ldots, y_{m-1})$ with $1 \leq k \leq n - x_{m-1}$ and $1 \leq l \leq n - y_{m-1}$ such that $F(k, l) = P(1, 1)$ and

$$F(k + x_{i-1}, l + y_{j-1}) = P(i, j)$$

for $1 \leq i \leq m$, $1 \leq j \leq m$ and $x_0 = y_0 = 0$.

In other words, the elements of $F$, obtained by intersecting the $m$ rows of $F$ in positions $(k, k + x_1, \ldots, k + x_{m-1})$ with the $m$ columns in positions $(l, l + y_1, \ldots, l + y_{m-1})$ determine the elements which form the occurrence of $P$ in $F$. This can potentially lead to an exponential number of matches in a particular row or column, e.g., matching $a_1 a_2 \ldots a_n$ with $a_1 a_1 a_2 a_2 a_3 a_3 \ldots a_n a_n$. Thus any algorithm to recognize this matching may potentially have to consider exponential possibilities and may take exponential time. We restrict this matching by requiring that $F(k + x, l + y) \neq P(i, j)$ for $x_{i-1} < x < x_i, y = 0$ (first row); and $y_{j-1} < y < y_j, x = 0$ (first column). By a similar argument this matching may have to consider exponential number of matches in the presence of wild cards; and hence may take exponential execution time. We restrict this matching to be without any wild cards.

($t = 0$) each symbolic item $i$ in $P$ occurs in $F$, and whenever $i$ occurs $k$ times in $P$ it occurs at least $k$ times in $F$.

($t = 3$) each symbolic item $i$ in $P$ occurs in a size $m \times m$ sub-block of $F$, and whenever $i$ occurs $k$ times in $P$ it occurs at least $k$ times in that sub-block.

We say that $P$ is a type-$t$ subpicture of $F$ if it has at least one type-$t$ occurrence in $F (t = 0, 1, 2, 3)$. For example, in Figure 4 the pictures $f2$, $f1$, and $f0$ are type-0 subpictures of $F$; $f2$ and $f1$ are type-1 subpictures of $f$; only $f2$ is a type-2 subpicture of $F$.

$$F = \begin{bmatrix} b & d & \\ c & a & \\ & & e \end{bmatrix} \quad f2 = \begin{bmatrix} b & \\ c & a \end{bmatrix} \quad f1 = \begin{bmatrix} c & \\ & e \end{bmatrix} \quad f0 = \begin{bmatrix} & b \\ d & \end{bmatrix}$$

Figure 4: Examples of Matching

$$p = \begin{bmatrix} t & \# \\ h & l \end{bmatrix}$$

Figure 5: An Example Query (Type-2)

### Examples

**Type-2 Query.** Figure 5 shows an example of type-2 query. The query is "Find all images containing a house with a lake on the east and a tree on the north." # represents a wild character that can match with any symbol.

**Type-1 Query.** In type-1 query we are interested in finding all images in which the symbols in the pattern maintain alignments between each other but in which the distance between any two symbols is different than in the original pattern. Figure 6 shows an example of type-1 query.

**Type-0 Query.** In type-0 query we are interested in finding all the pictures containing specified symbols (objects). For example, the query shown in Figure 7 is "Find all pictures containing at least one house, two trees, and a lake." In such a query, relative positions of symbols do not matter.

**Type-3 Query** is a variant of type-0 query. In this variant we want to find out an $m \times m$ sub-block of the picture which has at least as many symbols of each kind as an $m \times m$

$$(a)\begin{bmatrix} t & \# & a \\ h & \# & l \end{bmatrix} \quad (b)\begin{bmatrix} t & \# & \# & a \\ h & \# & \# & l \end{bmatrix} \quad (c)\begin{bmatrix} t & a \\ \# & \# \\ h & l \end{bmatrix}$$

Figure 6: Examples of Type-1 Query

$[h \quad l \quad t \quad t]$

Figure 7: Example of Type-0 Query

pattern. In this query the relative positions of symbols do not matter. However, the locality of the appearance should be limited to $m \times m$ block.

# 5 Iconic Indexing Algorithms

In this section we describe our results for performing iconic indexing on meshes and pyramids.

## 5.1 Type-2 Matching

We will classify the type-2 matching into two kinds: the first variation in which no wild cards are allowed (exact matching), and the second variation in which wild cards are allowed.

By using a suitable hashing function a two-dimensional matching can be reduced to a one-dimensional matching. In this case each symbol in the symbolic query array is replaced by the hash of all the $m-1$ symbols below it and itself. This computation can be performed on a mesh in parallel in $O(m)$ time. The same hashing function is used to convert the pattern into a one-dimensional pattern.

Assume that the symbol is stored in register B of each processor and the hashed bit pattern of the $m$ symbols (of the symbol with the processor and $m-1$ symbols of the processors below it) is to be stored in register A of each processor. In the following algorithm each processor computers $m$ hash values and sends $m-1$ values to the processors above it. All processors execute in parallel. The following algorithm is given for processor $P(i,j)$.

$T[i,j]$:=$B[i,j]$
$A[i,j]$:=$(ord(B[i,j]))$ mod $q$
**for** $k = 1$ to $m-1$ **do**
    $tmp[i,j] \leftarrow tmp[i+1,j]$
    $A[i,j]$:=$(d \times A[i,j] + tmp[i,j])$ mod $q$

After the above algorithm is executed, each processor's register $A$ contains the hash value of the $m$ symbols below it including its own symbol. The algorithm takes $O(m)$ steps to execute.

Once the hashing is performed, the problem is reduced to performing a one-dimensional matching and can be completed in another $O(m)$ steps. Thus the total time required is $O(m)$ on a mesh. The same can be performed in $O(m)$ time on a pyramid.

**Theorem 1** Exact matching can be performed on a mesh in $O(m^2)$ time on a mesh and $O(m^2)$ on a pyramid by using a variant of template matching algorithm [18]. The amount of memory required per PE is $O(1)$.

**Theorem 2** With hashing, type-2 exact matching can be performed in $O(m)$ on a mesh and $O(m)$ time on a pyramid. The amount of memory required per node is $O(1)$.

**Theorem 3** With wild cards, the type-2 matching can be performed in $O(m^2)$ on a mesh and $O(m^2)$ time on a pyramid. The amount of memory required per PE is $O(1)$. This can be done by a variant of template matching algorithm [18]

## 5.2 Type-1 Matching

A high-level description of the algorithm for computing the type-1 matching is given in Figure 8. Step 1 and Step 2 can be completed in $O(m)$ time. Steps 3 and 4 can be completed in $O(n)$ time. Step 5 can be completed in $O(m^2n)$ time. Step 5 can also be completed in $O(n^2)$ by passing the whole image through every PE. This requires storing the whole pattern in every PE. The amount of memory required is $O(m^2)$.

**Theorem 4** Type-1 matching can be completed in $O(m^2n)$ time on a mesh and a pyramid using $O(m)$ memory per PE. It can also be completed in $O(n^2)$ time using $O(m^2)$ amount of memory.

**Step 1:** Broadcast $PAT[0 \ldots m-1, 0]$ to all PEs. Each PE stores them in $C[0 \ldots m-1]$.

**Step 2:** Broadcast $PAT[0, 0 \ldots m-1]$ to all PEs. Each PE stores them in $R[0 \ldots m-1]$.

**Step 3:**

CMATCH: = FALSE
$I1(i,j) := I(i,j)$
$b := 0$
*for* $a = 0$ to $n-1$ do
  *if not* CMATCH *then*
    *if* $C[b] = I1$ *then*
      $b := b+1$
      CMATCHARRAY $[b] := a$
      *if* $(b = m)$ *then* CMATCH: = TRUE.
  $I1[i,j] \longleftarrow I1[i,(j+1) \bmod n]$
end;
CMATCH := CMATCH *and* (I(i,j) = C[0])

**Step 4:** Same as Step 3 but along rows. The result is stored in RMATCH and RMATCHARRAY $[1 \ldots m]$.

**Step 5:**

MATCH:=RMATCH *and* CMATCH
*for* $a := 1$ to $m$ do
  *for* $b := 1$ to $m$ do
    BROADCAST PAT$(a,b)$ to all PEs
    $S(i,j) := (i + RMATCHARRAY(a), j + CMATCHARRAY(b))$
    $D(i,j) := I(i,j)$
    *RAR*
    MATCH :=MATCH *and* $(D(i,j) = PAT(a,b))$
  end
end

Figure 8: Type-1 Matching Algorithm

## 5.3 Type-0 Matching

In type-0 matching we are supposed to find out whether the symbols appearing in the pattern also appear in the picture. This can be performed by the algorithm in Figure 9.

Steps 1, 2, 3, and 4 require $O(n), O(n), O(m)$ and $O(m)$ time, respectively. Step 5 requires $O(n)$ time. Steps 6 and 7 require $O(1)$ time. Step 8 requires $O(n)$ time on a mesh and $O(\log n)$ time on a pyramid.

**Theorem 5** If the pattern is already stored in the PEs, type-0 matching can be completed in $O(n)$ time of a mesh and a pyramid. The amount of memory required is $O(1)$. If the pattern is in the controller the total time is $O(n + m^2)$ on a mesh and on a pyramid.

There is another possible algorithm for pyramids. The pattern is first sorted. After sorting, the pattern values are broadcast to all the PEs in a pipelined fashion. At every step the PEs find a match with the current pattern value.The output of each PE is 1, if there is a match. These values are added up in a pyramid and sent back to the corresponding PE. There is a pipeline of result summed up and results sent down. This can be completed in $O(m^2 + \log n)$ time. At the end of this stage, each PE compares the (symbol, count) with the corresponding (pat-symbol, count) and outputs a 1 appropriately. An *and* of all the PEs having pat-symbol gives the desired result. The total time required is $O(m^2 + \log n)$. The complexity of the algorithm is independent of the fact whether the pattern is already loaded on the pyramid or not.

## 5.4 Type-3 Matching

Type-3 matching is a variant of type-0 matching. In this variant we want to find out an $m \times m$ sub-block of the picture which has at least as many symbols of each kind as an $m \times m$ pattern. There is a potential match at every sub-block of size $m \times m$ starting at $0 \leq i, j \leq n - m + 1$. The algorithm consists of $m^2$ stages. Each stage is similar to the one given in Figure 9. Each of the steps of Figure 9 are performed in an $m \times m$ sub-block. We assume that each $m \times m$ sub-block contains a copy of the pattern (This can be loaded in

**Step 1:** [SORT] Sort the elements of the picture $I$ and store the result in $S$ (in a snakelike order).

**Step 2:** [COMPRESS] Perform the compress operation on $S$. Each processor stores the result as

$$A(\text{SYMBOL, COUNT, 1}).$$

**Step 3:** [SORT] Sort the elements of the pattern $P$ and store the result in $T$ (in a snakelike order).

**Step 4:** [COMPRESS] Perform the compress operation on $T$. Each processor stores the result as

$$B(\text{SYMBOL, COUNT, 0}).$$

**Step 5:** [MERGE] $C$ =MERGE $(A, B)$. Each processor has at most two tuples (the last few may have null tuples). Let these tuples be represented by $C1$ and $C2$, respectively. The third field of $C$ represents whether the tuple is from pattern or picture (0 or 1).

**Step 6:** $X \longleftarrow C2$ (LEFT) where LEFT represents the left PE in the snakelike order.

**Step 7:**

MATCH:=TRUE
*if* $C1(3) = 0$ *then*
  *if* not$((X\cdot\text{SYMBOL}=C1\cdot\text{SYMBOL})$ *and* $(X\cdot\text{SYMBOL}\geq C1\cdot\text{SYMBOL}))$
  *then* MATCH:=FALSE
*if* $C2(3) = 0$ *then*
  *if* not $((C2\cdot\text{SYMBOL}= C1\cdot\text{SYMBOL})$*and* $(C1\cdot\text{SYMBOL}\geq C2\cdot\text{SYMBOL}))$
    *then* MATCH:=FALSE

**Step 8:** Perform an AC operation (*and*) on MATCH.

Figure 9: Type-0 Matching Algorithm

$O(m^2)$ time if required). Thus one stage can be completed in $O(m)$ amount of time. This gives possible matches for the top left hand corner for each sub-block. This algorithm has to be applied $m^2$ number of times by choosing suitable sub-block boundaries at every stage so that the match is found for all possible $n^2$ points. The complexity of the resultant algorithm is $O(m^3)$) on a mesh (including the initial loading). It can be completed in the same amount of time on the pyramid.

**Theorem 6** Type-3 matching can be completed in $O(m^3)$ on a mesh and $O(m^3)$ on a pyramid. The amount of memory required per PE is $O(1)$.

# 6 Conclusions

In this paper we have presented algorithms for iconic indexing on pyramids and meshes. Our algorithms are asymptotically superior to the algorithms presented in [16]. In many cases our algorithms are optimal. We are currently implementing these algorithms on the Connection Machine.

# 7 Acknowledgment

The authors would like to thank Ms. Elaine Weinman for converting the handwritten text of this paper into LaTeX.

# References

[1] N. S. Chang and K. S. Fu. "Query-by-pictorial-example." *Proc. COMPSAC 79,* IEEE Comput. Soc. (1979), 325–330.

[2] S. K. Chang and T. Kunii. "Pictorial database systems." *Computer* (special issue on pictorial information systems). S. K. Chang, Ed. (Nov. 1981), 13–21.

[3] H. Samet. "The quadtree and related data structures." *ACM Comput. Survey*, vol. 16, no. 2 (June 1984), 187–260.

[4] S. K. Chang, Q. Y. Shi, C. W. Yan. "Iconic indexing by 2-D strings." *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-9 (May 1987), 413–428.

[5] G. Tortora, Y. Costagliola, T. Arndt, S. K. Chang. "Pyramidal Algorithms for Iconic Indexing." *Manuscript.*

[6] M. J. Duff. "CLIP 4: A Large Scale Integrated Circuit Array Parallel Processor." IEEE Intl. Joint Conf. on Pattern Recognition (Nov. 1976), 728–733.

[7] M. J. Duff. "Review of the CLIP Image Processing System." National Computer Conference, Anaheim, California (1978).

[8] NCR Microelectronics Division, Product Description ncr45cg72, NCR Corporation, Dayton, Ohio (1984).

[9] K. Batcher. "Design of a Massively Parallel Processor." *IEEE Trans. on Computers*, vol. 29, no. 9 (1980), 836–840.

[10] C. P. Schnorr and A. Shamir, "An optimal sorting algorithm for mesh-connected computers" *Proceeding of the 18th ACM Symposium on Theory of Computing*, May 1986, 255-261.

[11] J. E. Devaney. "The MPP—A Totally Different Approach to Programming." IEEE Workshop on Computer Architecture for Pattern Analysis and Image Database Management (Nov. 1985), 420–427.

[12] S. L. Tanimoto, T. J. Ligocki, R. Ling. "A Prototype Pyramid Machine for Hierarchical Cellular Logic." *Parallel Hierarchical Computer Vision*, L. Uhr, Ed. Academic Press, London (1987).

[13] S. L. Tanimoto. "A Hierarchical Cellular Logic for Pyramid Computers." *Journal of Parallel and Distributed Computing*, vol. 1 (1984), 105–132.

[14] D. H. Schaefer, D. H. Wilcox, G. C. Harris. "A Pyramid of MPP Processing Elements—Experience and Plans." Hawaii Intl. Conf. on System Sciences (1985), 178–184.

[15] A. Merigot, B. Zavidovique, F. Devos. "SPHINX, A Pyramidal Approach to Parallel Image Processing." IEEE Workshop on Computer Architecture for Pattern Analysis and Image Database Management (Nov. 1985), 107–111.

[16] V. Cantoni, M. Ferretti, S. Levialdi, F. Maloberti. "A Pyramid Project Using Integrated Technology." *Integrated Technology for Parallel Image Processing.* Academic Press, London (1985), 121–132.

[17] D. Nassimi and S. Sahni. "Data Broadcasting in SIMD Computers." *IEEE Transactions on Computers*, vol. C-30 (May 1981), 342–346.

[18] S. Y. Lee and J. K. Agarwal. "Parallel 2-D convolution on a mesh connected computer." *IEEE Transactions on Pattern Analysis and Machine Intelligence.* (July 1987), 590–594.

[19] N. Ahuja and S. Swamy, "Multiprocessor Pyramid Architectures for Bottom-up Image Analysis," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* Vol. PAMI-6, (July 1984), 463–475.

[20] P. J. Burt, T. H. Hong, and A. Rosenfeld, "Segmentation and Estimation of Image Region Properties through Cooperative Hierarchical Computation," *IEEE Transactions on Systems, Man and Cybernatics,* Vol. SMC-11, (1981), 802–809.

[21] S. L. Tanimoto, "An iconic/symbol data structuring scheme," *in Pattern Recognition and Artificial Intelligence,* C. H. Kohen, Ed., New York, Academic, 1976.

[22] D. Nassimi and S. Sahni, "Bitonic Sort on a mesh connected parallel computer," *IEEE Transactions on Computers*, Vol. C-28, (January 1979), pp. 2-7.

[23] R. F. Zhu and T. Takaoda, "A technique for two-dimensional pattern matching," *Communication of the ACM*, Vol. 32, (September 1989), pp. 1110-1120.