Yale University
Department of Computer Science

Histogram Computation on Distributed Memory Architectures

Dimitris Gerogiannis, Stelios Orphanoudakis
S. Lennart Johnsson

YALEU/DCS/TR-682
February 1989

HISTOGRAM COMPUTATION ON DISTRIBUTED MEMORY ARCHITECTURES

Dimitris C. Gerogiannis †, *Stelios C. Orphanoudakis* †‡ *and S. Lennart Johnsson*†[1]

Departments of Diagnostic Imaging‡ *and Electrical Engineering*†

Yale University

New Haven, CT 06510

[1]Also affiliated with the Department of Computer Science, Yale University, and Thinking Machines Corp., Cambridge, MA 02142

Abstract

One data-independent and one data-dependent algorithm for the computation of image histograms on parallel computers are presented, analyzed, and implemented on the Connection Machine system CM–2. The data-dependent algorithm has a lower requirement on communication bandwidth by only transferring bins with a non-zero count. Both algorithms perform *all-to-all reduction*, which is implemented through a sequence of exchanges as defined by a butterfly network. The two algorithms are compared based on predicted and actual performance on the Connection Machine CM–2. With few pixels per processor the data-dependent algorithm requires on the order of $\sqrt{B}$ data transfers for B bins compared to B data transfers for the data-independent algorithm. As the number of pixels per processor grows the advantage of the data-dependent algorithm decreases. The advantage of the data-dependent algorithm increases with the number of bins of the histogram.

1 Introduction

Histogram computation is commonly encountered in image analysis and computer vision. Algorithms for the computation of image histograms on parallel machines with the memory distributed among the processing elements are described in [10,11,5], and for shared memory machines in [2]. The algorithm in [5] assumes a tree interconnection structure, while the algorithm in [10] is intended for a machine with a multistage interconnection network for interprocessor communication. A similar algorithm for bucket sorting on a hypercube is presented in [6]. The implementations proposed in [10] and [6] use a butterfly network for communication, and the results of the performance analysis are identical. Techniques that make full use of the communications bandwidth of hypercube networks are described in [7].

In this article, the histogram computation algorithm described in [10] is discussed and an alternative implementation, suggested in [6], analysed and implemented on the Connection Machine. The modified algorithm reduces the communication complexity by only transfering counts for non-empty bins. The architectural model used in the analysis consists of an ensemble of processing elements interconnected through a network that allow for the emulation of butterfly and tree communications without delay. Each processing element has its own local memory. Hypercube networks are examples of networks that support the required communication.

2 A data-independent algorithm for parallel histogram computation

The N pixels of the image are evenly distributed over the memories of M processors. The histogram to be computed is assumed to have B bins. In the parallel algorithms discussed here local histograms are computed in every processor, and the global histogram obtained by combining partial results, in a way that distributes the total count for the bins of the histogram among the processors. If the number of processors is smaller than the number of bins, then the total count for many bins of the histogram will reside in one processor. In the description and analysis of the algorithm that follows, it is assumed initially that $M \geq B$ so that the total count for at most one bin resides in a processor. However, results for $M < B$ are given at the end of this section.

For $M \geq B$ the algorithm consists of three phases:

1. Each processor computes a local histogram for $\frac{N}{M}$ pixels.
2. A histogram is computed for groups of B processors, with one processor holding the count of one bin of the histogram for all pixels in the group.
3. The global histogram is computed from the $\frac{M}{B}$ partial histograms.

In the first phase there is no communication. The second phase involves both communication, and local computation. Processors execute a recursive doubling procedure [9], which brings the count of a bin to one processor [10,6]. The communication is in fact *all-to-all reduction* [7] within each group of B processors.

All reductions within a group can be performed concurrently, and the different groups can also be treated concurrently. The third phase consists of B independent reductions, each over $\frac{M}{B}$ processors storing the count of one bin.

The *all-to-all reduction* can be performed through a sequence of exchange steps, one for each bit required for the encoding of the index of the bins. Such an algorithm is optimal for *one-port* communication, i.e., communication on one port per processor at a time. Other algorithms [7] may be chosen for *n-port* communication, i.e., concurrent communication on all ports of every processor. On the Connection Machine system CM–2, and other computers with shared memory primitives, a send-with-add would accomplish the desired task. The algorithm description and analysis in this paper provides the insight into what such an instruction implies in terms of communication and computational requirements.

The bins for the histogram are assumed to be labeled from 0 to $B-1$. The exchange sequence defines communication in the form of a butterfly network of $\log B$ stages for B bins, and B processors in each group. The recursive splitting of bins among processors can be performed by processors whose addresses differ in bit $\log B - k$ exchanging segments of their local histograms during step $k, k \geq 1$. Specifically, at the beginning of step 1 each processor holds the complete histogram of its subimage. Before exchange step k a processor holds a segment of length $\frac{B}{2^{k-1}}$. Let the index of a bin be $(B_{\log B-1} B_{\log B-2} \ldots B_0)$. The segment a processor with address $w = (w_{m-1} w_{m-2} \ldots w_0)$ holds after exchange step k has a starting index of

$$t_k = \sum_{j=\log B-k}^{\log B-1} w_j 2^j, \qquad 1 \leq k \leq \log B \tag{1}$$

In the k^{th} exchange, processor $w = (w_{m-1} w_{m-2} \ldots w_{\log B-k} \ldots w_0)$ exchanges a segment of length $\frac{B}{2^k}$ with processor $w = (w_{m-1} w_{m-2} \ldots \overline{w_{\log B-k}} \ldots w_0)$. The

starting index of the segment being exchanged is

$$s_k = \sum_{j=\log B-k+1}^{\log B-1} w_j 2^j + \overline{w_{\log B-k}} \frac{B}{2^k} \tag{2}$$

where $\overline{w_i}$ denotes the complement of w_i. The index of the first bin of the segment of the local histogram that is updated is given by Equation (1). The details of this implementation are presented in [10] and [6]. If, for every node, $hist(j)$ denotes the j^{th} bin of the local histogram, $0 \leq j \leq B-1$, and $receive(j)$ the j^{th} data item received by a processor, then the following procedure is executed by a node with address w, during the second phase:

for k:=1 to $\log B$ do

  for i:=0 to $B/2^k - 1$ do

    send $hist(s_k + i)$ to processor $(w_{\log B-1} \ldots \overline{w_{\log B-k}} \ldots w_0)$

  endfor

  for i:=0 to $B/2^k - 1$ do

    $hist(t_k + i) = hist(t_k + i) + receive(i)$

  endfor

endfor

The third phase of the algorithm combines the histograms computed by every group of B processors. The reduction operation in this phase requires $\log \frac{M}{B}$ communication steps. It can be performed such that the first B processors hold the histogram of the whole image. If $M < B$ then at the end of the algorithm every

processor holds $\frac{B}{M}$ bins of the histogram of the whole image. The algorithm terminates after $\log M$ steps of the second phase. This difference is the only difference between the cases $M < B$ and $M \geq B$.

The first phase has a computational complexity of $\frac{N}{M}$ additions. The second phase introduces a communications overhead due to the transfer of segments of the histogram between processors, as well as a computational cost due to the updates of segments of the local histograms. With the *all-to-all reduction* performed through an emulation of the butterfly network, a total of $B-1$ data exchanges distributed over $logB$ communication steps are required for B bins. For *n-port* communication the data transfer time can be improved by a factor of $logB$ [7].

A data communication either transfers the count associated with one bin, or transfers the index of the bin. The term *word* will be used to denote a piece of data holding either the address of a bin, or the corresponding count. The term should not be taken strictly as an 8-bit entity, but rather as an elementary data element used to hold the information that needs to be transferred. The computational cost associated with the updates of segments of the local histograms during the second phase is $B-1$ additions. Finally, the third phase of the algorithm requires one data transfer during each of $log\frac{M}{B}$ steps for a total of $log\frac{M}{B}$ data transfers, and an equal number of additions. Thus, the three phases of the algorithm result in $B-1+log\frac{M}{B}$ data transfers and a computational cost of $\frac{N}{M}+B-1+log\frac{M}{B}$ additions.

In the case $M < B$ the $\log M$ communication steps of the second phase require $B-\frac{B}{M}$ data transfers for *one-port* communication. The number of additions performed by every processor during this phase is also $B-\frac{B}{M}$. Thus, the total

communications overhead is equal to $B - \frac{B}{M}$ data transfers (*one-port* communication) and the total computational cost is equal to $\frac{N}{M} + B - \frac{B}{M}$ additions. The local storage needed in this algorithm is an array of size B.

3 A data-dependent algorithm for histogram computation

In the algorithm described above, pairs of nodes exchange histogram segments without checking which bins of the histogram have non-zero counts. This results in a straightforward implementation of the algorithm. But, many of the exchanged bins may be empty, particularly in the first few steps of communication. Communication bandwidth is used up without any effect on the histogram computation. In the data-dependent algorithm, every processor manipulates - sends, receives and updates - only bins with non-zero count. The data dependency for important implications for the computational cost and communications overhead of the algorithm in cases of very few pixels per processor.

In the case of one pixel per processor every processor will have only one bin of its histogram updated. During the first communication step, pairs of processors need to exchange at most one bin. Every processor sends or keeps the the count of a bin depending on which half of the histogram the bin falls in. Since the processors must also send the index of the bin, the worst case number of data transfers is two words. The bin a processor receives may be different from the bin to which the local pixel belongs. Consequently, the data which needs to be transferred during the second communication step (in the worst case) is the counts for two bins, and

their indices. It is clear that in the worst case the number of bins that need to be communicated doubles for every communication step. However, the histogram segment a processor keeps updated is reduced by a factor of two for each step. Hence, the doubling can only take place up to some communication step $r < \log B$. For communication steps $k \geq r$ the data-independent algorithm is invoked. It will be shown in section 4.1 that the data dependent algorithm may yield a reduced communications overhead for p pixels per processor, if $p < B/4$.

Next, two different approaches to organizing the local computation at each node are proposed and analyzed.

3.1 Approach 1

In the data-dependent algorithm, the arithmetic overhead incurred at each node originates mainly from the need to select only the active bins from the segment of the histogram that will be transferred. Every node knows, through the simple arithmetic described above, the segment of its local histogram which is updated and the segment whose active bins are transferred at each step of the butterfly. What the processors do not know in advance, are the specific bins of this segment that will be received, and whether some of these bins correspond to bins in the local histogram which are already active. Therefore, upon receipt of the bins, the processors must check the addresses of the received bins and update the corresponding bins in the local histograms. Since the processors send only the active bins, they must scan the appropriate segment of the histogram to find and send only those bins. The procedure that every node executes at step k, for $1 \leq k \leq r$, is the following:

1. $t_k = \sum_{j=\log B-k}^{\log B-1} w_j 2^j$

2. $s_k = \sum_{j=\log B-k+1}^{\log B-1} w_j 2^j + \overline{w_{\log B-k}}\, \frac{B}{2^k}$

3. for i:=0 to $\frac{B}{2^k} - 1$ do

   if $hist((s_k + i) \neq 0)$ then

   send [$(s_k$+$i)$,$hist(s_k$+$i)$] to processor $w = (w_{m-1} w_{m-2} \ldots \overline{w_{\log B-k}} \ldots w_0)$

   endfor

4. for i:=0 step 2 until all messages received do

   $hist(receive(i)) = hist(receive(i)) + receive(i + 1)$

   endfor

Note: $receive(i)$ holds the address of the bin and $receive(i + 1)$ the corresponding count.

In the first several steps, the processors scan big segments of the histogram looking for very few active bins. In the worst case, when the active bins happen to lie at the end of a segment, they have to scan all the bins of that segment. For example, in the case of a 256 bin histogram and 1 pixel per processor, every processor receives at most 1 bin that may lie at the end of an array of 128 elements. This procedure applies to every communication step, and the processors scan histogram segments of decreasing length until a step is reached where the full segment is sent without searching for active bins.

Since the bins to be sent are selected by searching the list of potential candidates, this approach to organizing the local computation does not fully realize the goal of the data-dependent algorithm, namely the manipulation of active bins only.

A different approach, which overcomes this problem and accesses directly the bins that need to be sent, is discussed next.

3.2 Approach 2

The scanning of bins to find active ones can be avoided if each processor keeps a list of all active bins in the segment of the histogram that is to be transferred during the various communication steps. This list is updated at every communication step. One way to organize the list is as a collection of lists, one for each of the first r steps of communication. Sublist k holds the addresses of the bins that are active and fall in the histogram segment that must be exchanged during communication step k. In our implementation we use a set of r arrays, with a set of counters for the number of active bins entered into each array. At the k^{th} step of communication, the processors check the counter associated with the k^{th} array and fetch from it the indices of as many bins as the value stored in the counter. For each fetched address the count is retrieved from the local histogram and the address and the count sent to the appropriate processor. No scanning of histogram segments is required.

Upon receipt of a bin, the processors must determine the communication step at which the bin will be transferred, i.e. to which array the bin belongs. From Equation (2) it follows that the bin should be inserted into array j, where j is the highest order bit in which the $\log B$ lowest order bits of the processor address and the bin index differs. In addition, the processors must determine whether a received bin appears in the corresponding array, or it is inactive. A bin appears in the list if the count for the bin is non-zero, and the processor only needs to update the count for the bin. Otherwise, the bin index needs to be added to the array, and its

counter updated.

This approach to deciding what bins are to be sent at each communication step results in more bookkeeping per received bin, and increased memory requirements. In addition to the local histogram stored at every node, r one-dimensional arrays $z_i(.)$ are required, one for each of the first r steps of communication, and r registers C_i. The registers hold the number of active entries in each array. The length l_i of array $z_i(.)$ is the number of bins communicated during step i. The length is a function of the granularity. If initially there are p pixels per processors, then $l_1 \leq p$, $l_2 \leq 2p$ and, in general, $l_i \leq 2^{i-1}p$, for $1 \leq i \leq r$. The decision making process at every node is implemented by the following procedure:

1. for j:=0 to $C_k - 1$ do

   send $z_k(j), hist(z_k(j))$ to processor $(w_{m-1}w_{m-2}\ldots\overline{w_{\log B-k}}\ldots w_0)$

   endfor

2. for i step 2 until all messages received do

   if $(hist(receive(i)) = 0)$ then

   compute k' from the processor address and bin index

   $C_{k'} = C_{k'} + 1$

   $z_{k'}(C_{k'}) = receive(i)$

   endif

   $hist(receive(i)) = hist(receive(i)) + receive(i+1)$

   endfor

The procedure above is executed only during the first r communication steps, at which point a switch is made to the data-independent algorithm. Many standard processors have dedicated circuitry for determining the number of leading zeros in a word, making the computation of the array index very fast. Should the above scheme be too slow a look-up table can be used. However, such a table requires extra storage. The step r at which the data-independent algorithm is invoked is predetermined based on a worst case analysis.

4 Performance analysis of the data-dependent algorithm

In this section the performance of the data-dependent algorithm is analyzed. The communication overhead for *one-port* communication is derived. We will only comment on the possible improvement for *n-port* communication. The analysis of the computational cost, associated with the two proposed approaches to the organization of the local computation follows.

4.1 Communication overhead

The main difference between the data-independent algorithm and the data-dependent algorithm is the selection criterion for transfering bins during a communication step. In the first algorithm, at communication step k, $\frac{B}{2^k}$ bins are transferred. In the second algorithm, every node selects among the $\frac{B}{2^k}$ bins only those whose count is not zero. With p pixels per processor a processor will activate at most p bins in computing its local histogram. In the worst case a processor will

communicate all the p bins in the first communication step, and a total of $2 * p$ words. After a processor receives the bins, it must update the local histogram. The update is done via one of the two approaches discussed in section 3. In the worst case, all p received bins will be different from the already active bins in each node's histogram. Thus, after the first communication the local histogram may have $2p$ active bins. If all $2p$ bins fall in the segment of the histogram which is to be transferred in the second step, then the data to be transferred is $4p$ words ($2p$ bins and the corresponding counts). At step $k \geq 1$, $2^{k-1}p$ bins are transferred in the worst case giving a total amount of transferred data of $2(2^{k-1}p)$ words. Hence, the data dependent algorithm has an advantage with respect to the communications overhead for the first r steps, where r is the largest number for which

$$2(p2^{r-1}) < B/2^r \tag{3}$$

The total communications overhead for the first r steps is equal to:

$$2(p + 2p + 4p + .. + 2^{r-1}p) = 2p(2^r - 1) \tag{4}$$

In the last k steps of phase two, $\log B - r \leq k \leq \log B$, segments of length $\frac{B}{2^k}$ are transferred in step k resulting in a total communications overhead of

$$\frac{B}{2^{r+1}} + \frac{B}{2^{r+2}} + ... + 2 + 1 = \frac{B}{2^r} - 1 \tag{5}$$

words for *one-port* communication. For *n-port* communication the data transfer time is reduced by a factor $\log B - r$ [7]. The communication cost for the second step of the algorithm is the sum of Equations(4) and (5) (*one-port* communication). The largest value of r that satisfies (3) is

$$r = \lfloor \frac{\log \frac{B}{p}}{2} \rfloor \ . \tag{6}$$

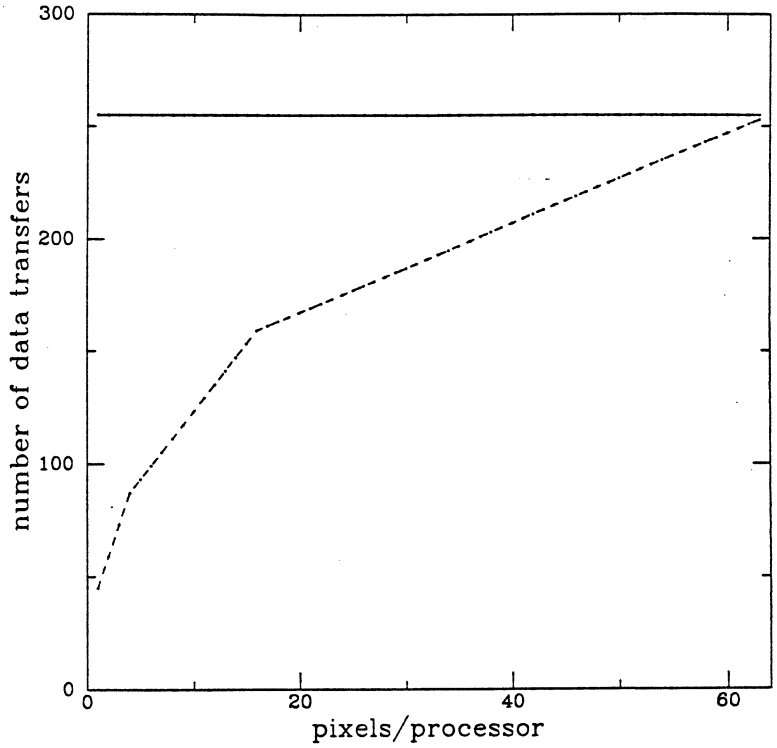

Figure 1: The number of data transfers as a function of the number of pixels per processor. The solid and dashed curves correspond to the data-independent and data-dependent algorithms, respectively.

Since $r \geq 1$, Equation(6) can be used to determine the relationship between B and p for which the data dependent algorithm offers an advantage with respect to data transfer time, in the worst case. The condition is

$$p < \frac{B}{4} . \tag{7}$$

In Figure 1 the communications overhead of the data-dependent algorithm in the worst case and the data-independent algorithm are plotted as a function of the number of pixels per processor p for $B = 256$. If $M < B$, then the algorithm terminates during the second phase, and the complexity expressions for the second phase shows a dependency on M. The analysis is similar to the case $M \geq B$. For very few pixels per processor, the data-dependent algorithm results in considerable savings in the number of data transfers. In the case of one pixel per processor, the

number of data transfers is proportional to $\sqrt{B}$ for the data-dependent algorithm, whereas it is proportional to B for the data-independent algorithm. For commonly used values of B, this results in an order of magnitude difference in the number of data transferred during the second phase of the algorithm. However, a fair comparison between the two algorithms requires a quantitative analysis of computational cost as well.

4.2 Analysis of computational complexities

4.2.1 Computational cost of approach 1

The computational cost of approach 1 for the data-dependent algorithm comes from the need to search the local histogram segments during the first r steps of the second phase of the algorithm, and from the update of the local histograms with the received data. The cost of updating the local histograms during the first r steps is equal to

$$(p + 2p + 4p... + 2^{r-1}p) \tag{8}$$

additions in the worst case, whereas the cost of searching the local histograms is equal to

$$\frac{B}{2} + \frac{B}{2^2} + ... + \frac{B}{2^r} \tag{9}$$

In the last $\log B - r$ steps the data-independent algorithm is used, and the computational cost is due to updating the local histograms. This cost is equal to

$$\frac{B}{2^{r+1}} + \frac{B}{2^{r+2}} + ... + 2 + 1 \tag{10}$$

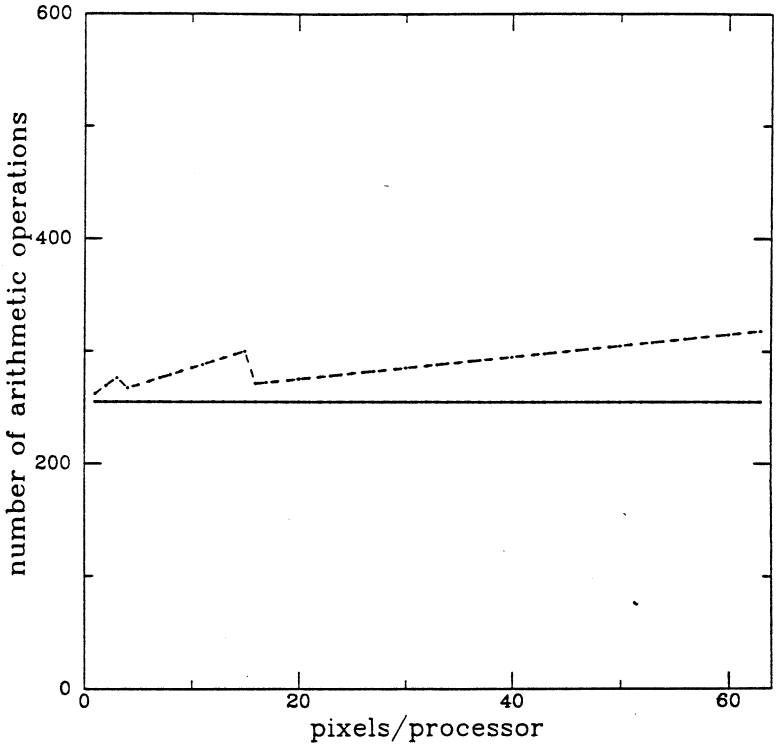

Figure 2: The number of arithmetic operations as a function of the number of pixels per processor. The solid and dashed curves correspond to the data-independent and approach 1 to the data-dependent algorithm respectively.

additions. The total computational cost of approach 1 for the data-dependent algorithm is the sum of expressions (8), (9), and (10). Additions and comparisons are assumed to incur the same cost in terms of performance. Thus, approach 1 results in an increased computational cost compared to the data-independent algorithm given by (8), because the sum of expressions (9) and (10) is equal to the computational complexity of the data-independent algorithm.

Figure 2 shows the computational costs associated with the second phase of the data-independent algorithm and approach 1 for the implementation of the data-dependent algorithm, as functions of the number of pixels per processor p. This plot indicates that the computational cost of approach 1 is a piecewise monotonically increasing function of p. The discontinuities occur for $p = 4$, $p = 16$ and $p = 64$

because, for these values of p, the value of the critical step r decreases by one. For the values of p for which r remains the same, expression (8) shows that the incremental cost increases linearly with p. The magnitude of the step-wise decrements in the communication cost can be obtained from expressions (8) and (6). Let p' be the value of p for which the critical step changes from $r'+1$ to r'. Then, the incremental cost in increasing the number of pixels per processor from $p'-1$ to p' is found to be

$$p'(2^{r'}-1)-(p'-1)(2^{r'+1}-1)=2^{r'}(2-p)-1 \tag{11}$$

Expression 11 is always negative for $p > 2$. Thus, every time the value of r changes, as the number of pixels per processor increases, the computational cost of the data-dependent part of phase two decreases.

Note that the memory requirements of approach 1 for implementing the data-dependent algorithm are the same as in the data-independent algorithm. Approach 1 leads to the desired reduction of the number of transferred bins, but the reduction is achieved at the expense of additional at each node.

4.2.2 Computational cost of approach 2

For every received message, the processors must perform one comparison and one addition, and depending on the result of the comparison compute an array index (or look up a table), increment a counter, and append a bin index to an array. Before the first "send" operation, each processor using the values of its own pixels updates the arrays that hold the index of active bins to be transferred during each of the first r steps. The procedure generates the initial local histogram. In subsequent steps up to step r, the same procedure updates the arrays containing the indices of

active bins, and the local histogram with the counts of the received bins. After the r^{th} "send" operation, each processor updates the local histogram with the received counts, but does not need to update the arrays that hold the indices of active bins. Hence, during the first $r-1$ steps two – five operations are performed per received bin. During the last $\log B - r$ steps only one operation per bin is required. The computational complexity associated with the initial updates of the arrays is three operations per pixel. The operations for the update of the local histogram, is not part of the second step of the algorithm, and is not taken into account. Thus, the total computational complexity associated with the first r steps is at most

$$3 * p + 5 * (p + 2p + 4p + ... + 2^{r-2}p) + 2^{r-1}p \tag{12}$$

arithmetic operations. In the remaining $\log B - r$ steps the computational cost is that of the data-independent algorithm:

$$\frac{B}{2^{r+1}} + \frac{B}{2^{r+2}} + ... + 2 + 1. \tag{13}$$

The total computational cost associated with the second phase of approach 2 to the data-dependent algorithm is the sum of Equations (12) and (13).

In Figure 3 the computational cost of this approach is plotted as a function of the number of pixels p per processor for $B = 256$. The behavior of the cost function is similar to the one discussed in the last section. The form of the curve can be explained by Equations (6), (12) and (13). The memory requirement of approach 2 is higher than for approach 1 due to the use of arrays to record the indices of active bins to be communicated in each of the first r steps. These arrays require a storage of

$$p + 2p + ... + 2^{r-1}p = p(2^r - 1) \tag{14}$$

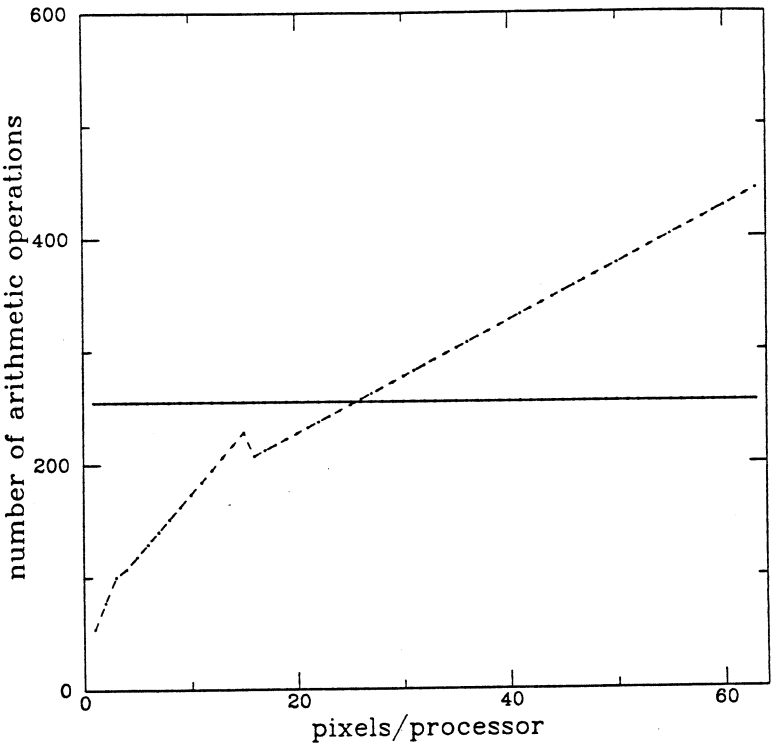

Figure 3: The number of arithmetic operations as a function of the number of pixels per processor. The solid and dashed curves correspond to the data-independent algorithm and approach 2 to the data-dependent algorithm respectively.

words. Although this approach to organizing the local computation at each node results in more complicated bookkeeping, the actual computational cost is reduced compared to the data-independent algortihm for few pixels per processor, i.e. $p < 10$.

5 Implementation issues

The performance of the algorithms depends on the characteristic parameters of the machine, and the problem size. The machine parameters we consider are t_c, the time it takes to transfer one piece of data between adjacent nodes, t_a, the time for an arithmetic operation, or a comparison, and τ, the start-up time for a communication. If the length of a message is larger than maximum packet size as defined by the communications buffers, then the message is split into pieces equal to the maximum packet size. Note that if the maximum packet size is sufficiently large, then the number of communication start-ups is the same for both the data-independent and the data-dependent algorithms. The number of communication steps are the same, but the size of the data set differs. The performance data is derived using Equations (4) and (5) for the communication cost, Equations (8), (9) and (10) for the computational cost of approach 1 for the data-dependent algorithm, and Equations (12) and (13) for approach 2.

The data-independent algorithm and both approaches 1 and 2 to the data-dependent algorithm were implemented on the Connection Machine system, model CM-2. For an accurate estimation of the communication time for the CM–2 it is necessary to understand some of its architectural features [4,3]. There are 16 Connection Machine processors to a chip, and 4k chips interconnected as a Boolean

cube, for a total of 64k processors. On-chip communication is effectively local memory moves that can take place concurrently, while inter-chip communication is bit-serial. Communication can take place concurrently on all interconnections. The on-chip communication time is in the range $20\mu s - 30\mu s$ for 4 bytes depending on communication pattern. For off-chip communication the time to transfer 4 bytes across a wire is about $35\mu s$. The peak computational performance of a Connection Machine system CM–2 with 64k processors and hardware support for floating-point operations is in the range 1.5-2.3 Gflops/s using *Lisp [1], a parallel extension to Common Lisp. Up to an order of magnitude higher floating-point performance currently can be achieved only in lower level programming languages. The programming language *Lisp was used for the implementation of the algorithms on the CM–2. For *Lisp $t_a \approx 30\mu s$.

With 256 bins and 256 processors per group a group requires 16 processor chips, or 4 off-chip steps. For a 64k processor configuration there are 256 such groups, which implies that phase three has eight steps. Since on-chip communication is faster it is beneficial in the data-independent algorithm to perform the first four steps of phase two on-chip, and the last four steps with off-chip communication. For the first off-chip step each processor holds 16 bins. All 16 processors on a chip exchanges 8 bins each with a "buddy" on another chip, i.e., values for 128 bins are sent, and values for another 128 bins are received over the same inter-chip connection. In the next step, half as many bins are exchanged. The off-chip communication requires approximately $16 \times 15 \times 35\mu s = 8.4\ ms$ for the data-independent algorithm. By pipelining the off-chip communication as described in [8] the communication time can be reduced to about 4.6 ms. The on-chip communication time for the first four steps is about $7 - 8\ ms$ for the data-independent algorithm.

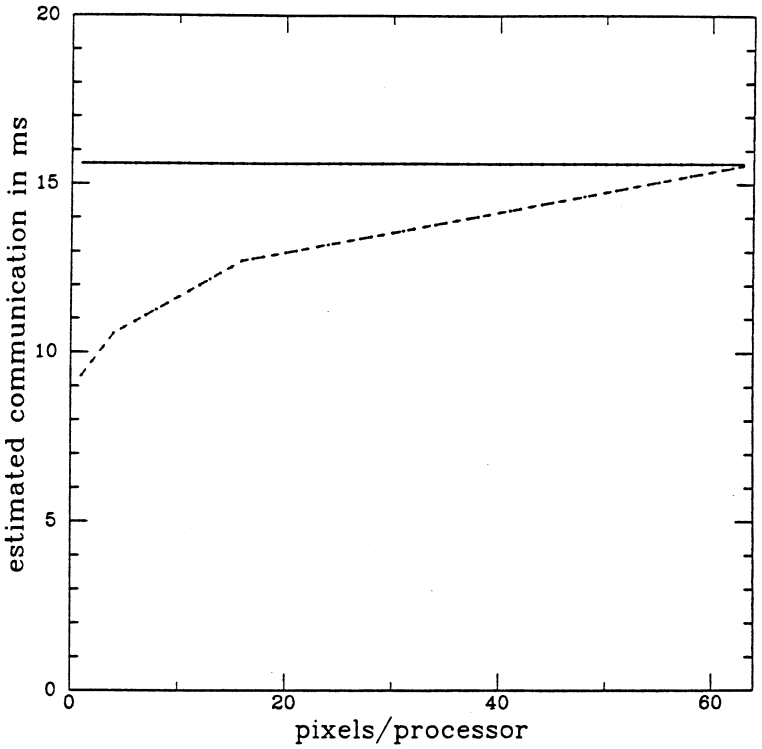

Figure 4: The estimated communication time for the Connection Machine machine as a function of the number of pixels per processor. The solid and dashed curves correspond to the data-independent and data-dependent (the communication time is the same for both approaches) algorithms respectively.

The communication time is the sum of the times for off-chip and on-chip communication. The estimated times of communication and computation, as well as the total execution time for computing a 256 bin histogram on the CM–2 are shown in Figures 4, 5, and 6 respectively.

The data-independent algorithm and approach 1 to the data-dependent algorithm were implemented on the CM–2. For each algorithm two versions were implemented. In the first, the first four communication steps were on-chip and the remaining steps off-chip; in the second, the first four steps are off-chip and the remaining steps on-chip. The performance of each implementation was evaluated with 1, 2, 4, 16 and 32 pixels per processor. The histogram was computed for cor-

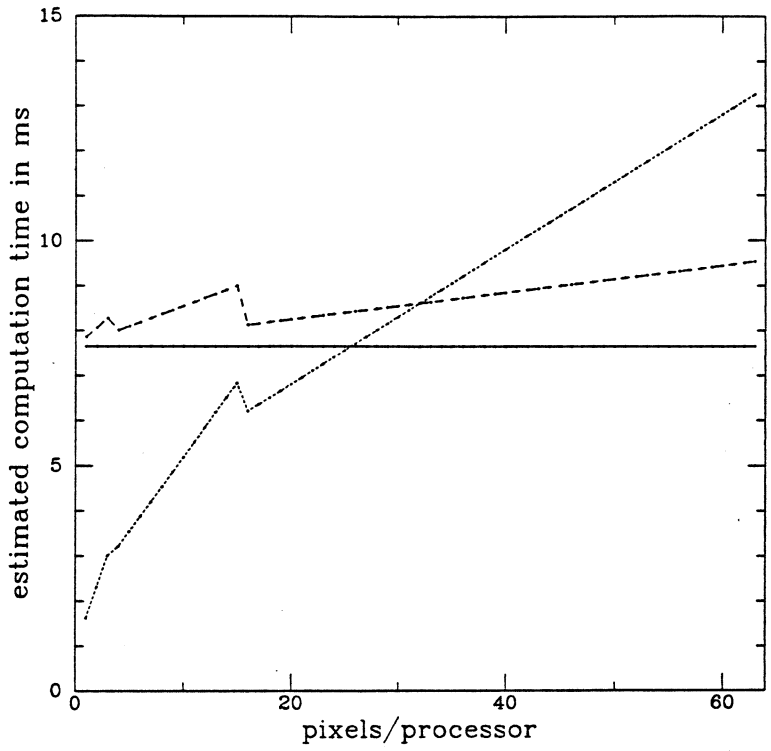

Figure 5: The estimated computation time for the Connection Machine machine as a function of pixels per processor. The solid, dashed, and dotted curves correspond to the data-independent algorithm, approach 1, and approach 2 to the data-dependent algorithm respectively.

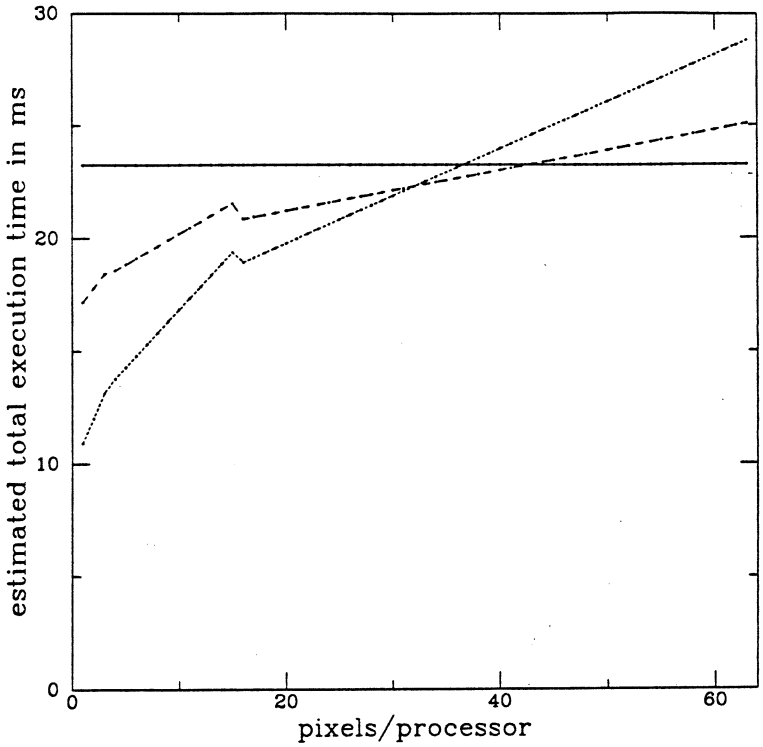

Figure 6: The estimated total execution time for the Connection Machine machine as a function of pixels per processor. The solid, dashed, and dotted curves correspond to the data-independent algorithm, approach 1, and approach 2 to the data-dependent algorithm, respectively.

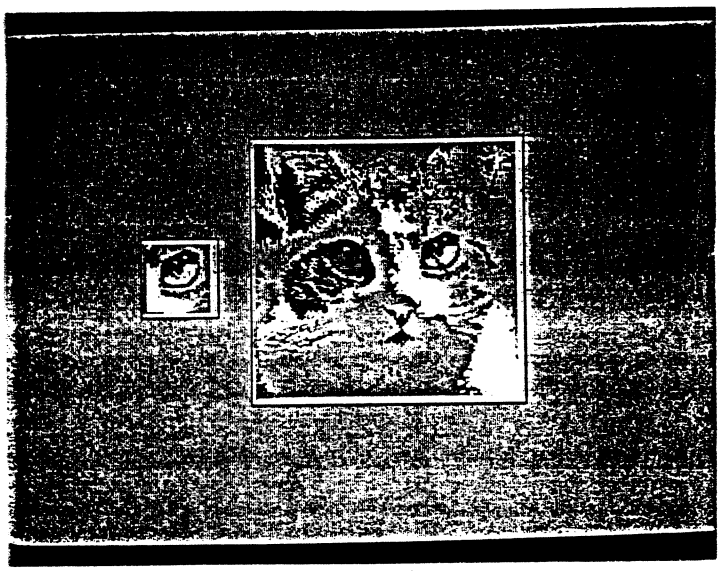

Figure 7: Test images used in the implementations on the CM. The small window on the left was used for 1 pixel/processor and the larger window on the right was used for 16 pixels/processor.

respondingly sized windows of the 64k test image shown in Figure 7. The measured computation and communication time, and the calculated total execution times are shown in Figures 8, 9 and 10 for the first version of the algorithms and in Figures 11, 12, and 13 for the second version.

With the first communication steps being on-chip the data-independent algorithm performs better than approach 1 to the data-dependent algorithm. The reason is that the algorithms are computation bound due to the fast on-chip communication, which is used for the bulk of the data transfers. The on-chip transfer of a 32 bit word is performed as fast as one arithmetic operation, thus making the local computation dominate the total execution time. Most of the time for local processing in approach 1 to the data-dependent algorithm is spent in searching for

the active bins. The search time in approach 1 incurs a cost which cancels out the savings in communication time.

However, in version two of the algorithms with the first four communication communication steps being off-chip communications the data-independent algorithm becomes communication bound. Approach 1 to the data-dependent algorithm performs better than due to the reduced communication time. Figures 4 and 8 show that, in general, there is good agreement between predicted and measured communication time. However, Figures 5 and 9 show a discrepancy between predicted and measured computation time. This is due to the fact that in our calculation of predicted values the time for evaluating and executing the conditional statements is underestimated.

The indirect addressing feature of the Connection Machine system, model CM–2, is used in the search procedure, but incurs a certain overhead. This feature is used in the search of the histogram segments (Section 3.2.1., steps 3 and 4).

The performance analysis of the data-independent algorithm and approach 1 to the data-dependent algorithm demonstrates that the latter algorithm performs better when the histogram computation is communication bound. This is also supported by the results of the implementation on the Connection Machine. To fully realize the potential of the data-dependent histogram computation, approach 2 to the data-dependent algorithm should be used.

6 Discussion

A data-independent and two versions of a data-dependent algorithm for computing image histograms on parallel architectures have been analyzed, and the data-independent algorithm and one of the data-dependent algorithms implemented on the Connection Machine system, model CM–2. The data-independent algorithm has also been analyzed in [10] and [6]. All algorithms consist of two phases: local histogram computation and global accumulation of the counts for the different bins. For convenience in the implementation we perform the second phase in two stages, if the number of processors exceeds the number of bins, effectively making the algorithms consist of three phases. The data-dependent algorithms only differ from the data-independent algorithm for part of the second of the three phases. The purpose of the data-dependent algorithm is to reduce the need for information transfer by only communicating information about bins with a non-zero count. The potential benefits of such an implementation was pointed out in [6]. Transferring only the counts of bins with a non-zero count introduces a need for bookkeeping of what bins are active and when to communicate the active bins. The addresses of bins to be transferred are determined during the execution of the algorithm. The communication cost of the implementations and the computational cost due to local processing have been analyzed. In fact, for the second implementation, two approaches to the organization of the local computation have been considered leading to different costs.

The global accumulation is made using an exchange algorithm. The number of exchange steps in phase two is $\min(\log B, \log M)$, where M is the number of processors and B the number of bins. The third phase consists of B independent

reduction operations (one for each bin) on sets of $\lceil \frac{M}{B} \rceil$ elements each. For an image with 10^6 pixels, 256 bins, and 10^6 processors, the third phase results in 12 data transfers and 12 startups. The second phase results in 255 data transfers with the data independent algorithm versus 45 with the data-dependent algorithm, and 8 startups for both. In general, for a realistic number of bins, pixels, and processors, the third phase involves $8 - 15$ startups and an equal number of data transfers, whereas the second phase involves $8 - 12$ startups and a much larger number of data transfers. The exact contribution of the second and third phase to the total execution time depends on the number of pixels per processor, and the number of bins in the histogram, assuming that there are at least as many processors as bins. For machines with short startup time, the second phase dominates the execution time of the complete algorithm.

An interesting consideration is the effect of the start-up time on the performance of the two algorithms. The improved performance of the data-dependent algorithm deteriorates in machines with high startup times. However, the tendency in current machines is towards lower start-up times and, therefore, the data-dependent algorithm is a reasonable choice for few pixels per processor. The start-up time on the Connection Machine system, model CM–2, is approximately equivalent to the transfer of 4 bytes.

Note that as the number of bins in the histogram increases, the performance of the data-dependent algorithm becomes increasingly better compared to the performance of the data-independent algorithm.

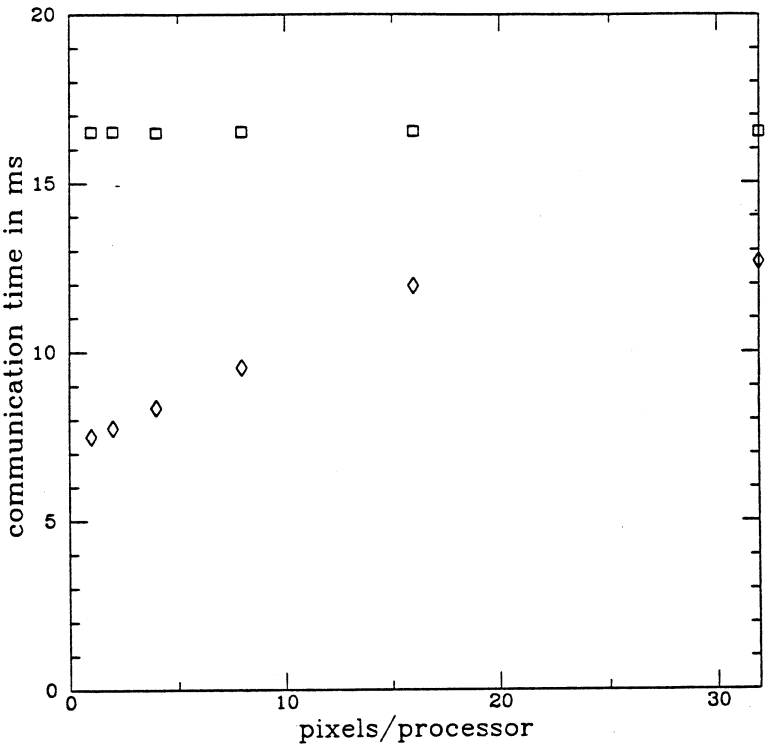

Figure 8: Measured communication time as a function of the number of pixels per processor. The data was obtained from a CM implementation in which the first four communication steps were on-chip and the remaining steps off-chip. ($\square$ data-independent algorithm, $\diamond$ data-dependent algorithm).

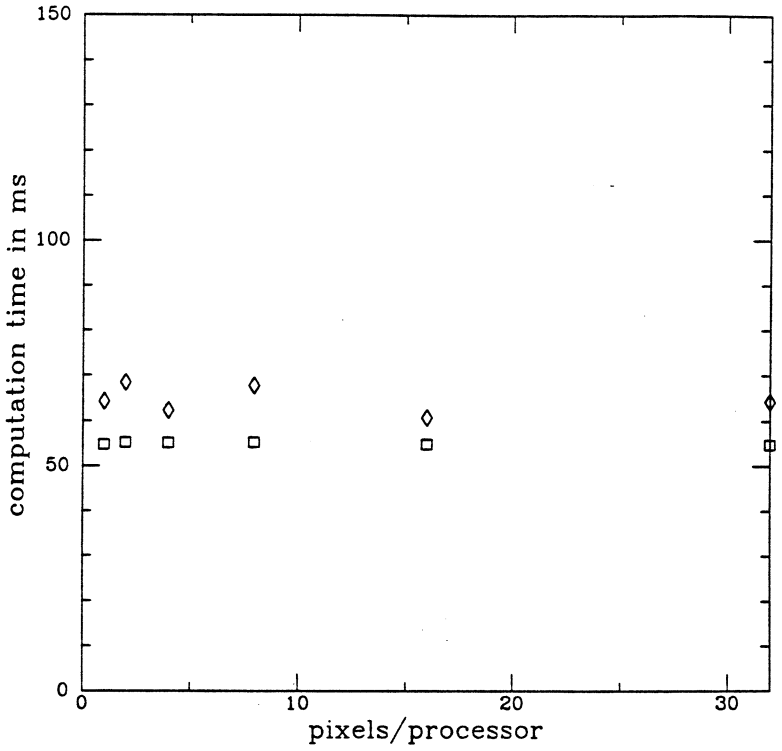

Figure 9: Measured computation time as a function of the number of pixels per processor. The data was obtained from a CM implementation in which the first four communication steps were on-chip and the remaining steps off-chip. (□ data-independent algorithm, ◇ data-dependent algorithm).

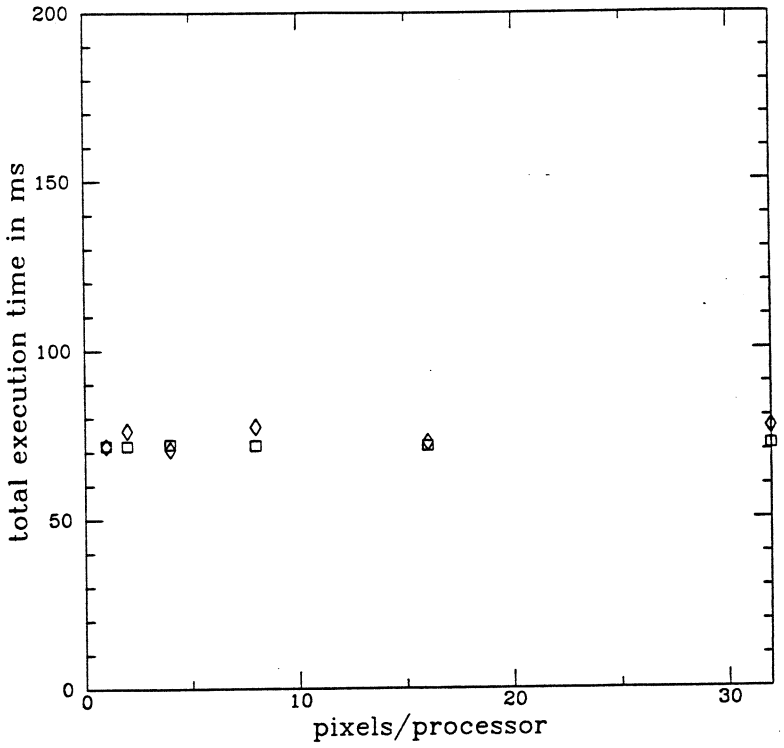

Figure 10: Measured total execution time as a function of the number of pixels per processor. The data was obtained from a CM implementation in which the first four communication steps were on-chip and the remaining steps off-chip. (□ data-independent algorithm, ◇ data-dependent algorithm).

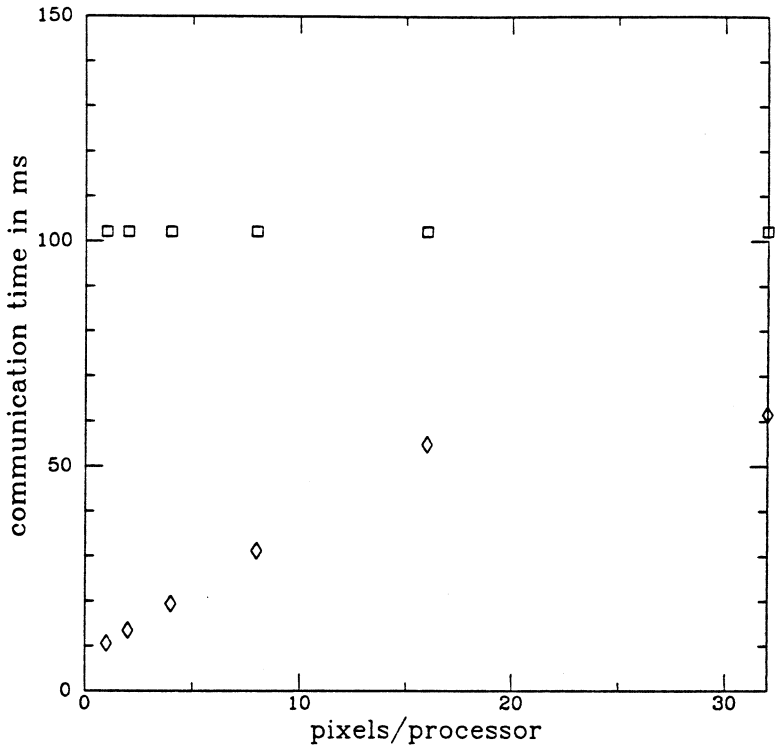

Figure 11: Measured communication time as a function of the number of pixels per processor. The data was obtained from a CM implementation in which the first four communication steps were off-chip and the remaining steps on-chip. (□ data-independent algorithm, ◇ data-dependent algorithm).

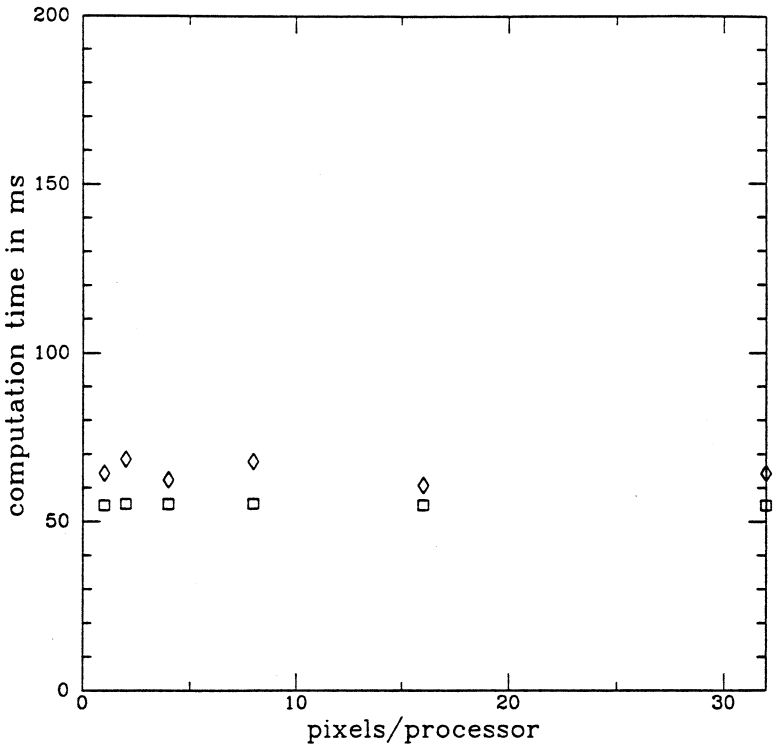

Figure 12: Measured computation time as a function of the number of pixels per processor. The data was obtained from a CM implementation in which the first four communication steps were off-chip and the remaining steps on-chip. (□ data-independent algorithm, ◇ data-dependent algorithm).

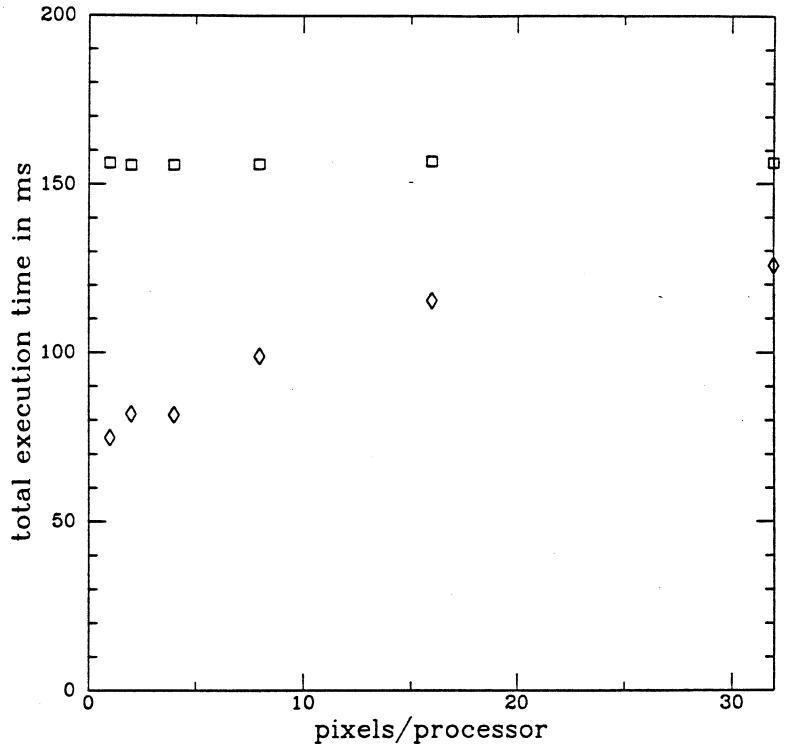

Figure 13: Measured total execution time as a function of the number of pixels per processor. The data was obtained from a CM implementation in which the first four communication steps were off-chip and the remaining steps on-chip. (□ data-independent algorithm, ◇ data-dependent algorithm).

Acknowledgements

The authors wish to thank Luis F. Ortiz for valuable help with the implementation on the Connection Machine. This work was supported in part by the Office of Naval Research under Contract No. N00014-86-K-0310 to the third author.

References

[1] *Lisp release notes. Thinking Machines Corp., 1987.

[2] C.M. Brown. Low level image analysis on an MIMD architecture. In *International Conference on Computer Vision*, pages 468–475, 1987.

[3] Thinking Machines Corp. *Connection Machine Model CM–2 Technical Summary.* Technical Report HA87-4, Thinking Machines Corp., 1987.

[4] W. Daniel Hillis. *The Connection Machine.* MIT Press, Cambridge, MA, 1985.

[5] Hussein A.H. Ibrahim, John R. Kennedy, and David E. Shaw. Low-level image analysis tasks on fine-grained tree-structured simd machines. *Journal of Parallel and Distributed Computing*, 4(6):546–574, 1987.

[6] S. Lennart Johnsson. Combining parallel and sequential sorting on a Boolean n-cube. In *1984 International Conference on Parallel Processing*, pages 444–448, IEEE Computer Society, 1984. Presented at the 1984 Conf. on Vector and Parallel Processors in Computational Science II.

[7] S. Lennart Johnsson and Ching-Tien Ho. *Spanning graphs for optimum broadcasting and personalized communication in hypercubes.* Technical Report YALEU/DCS/RR-500, Dept. of Computer Science, Yale Univ., New Haven, CT, November 1986. Revised November 1987, YALEU/DCS/RR-610. To appear in IEEE Trans. Computers.

[8] S. Lennart Johnsson, Ching-Tien Ho, Michel Jacquemin, and Alan Ruttenberg. Computing fast Fourier transforms on Boolean cubes and related networks. In *Advanced Algorithms and Architectures for Signal Processing II,*

pages 223–231, Society of Photo-Optical Instrumentation Engineers, 1987. Report YALEU/DCS/RR-598, October 1987.

[9] P.M. Kogge and Harold S. Stone. A parallel algorithm for the efficient solution of a general class of recurrence equations. *IEEE Trans. Computers*, 22(8):786–792, 1973.

[10] Leah J. Siegel, Howard J. Siegel, and P.H. Swain. Performance measures for evaluating algorithms for SIMD machines. *IEEE Trans. Softw. Eng*, 8(4):319–330, 1982.

[11] S. Yalamanchili and J.K. Agrawal. Analysis of a model for parallel image processing. *Pattern Recognition*, 18(1):1–16, 1985.