Scalable Data Parallel Algorithms for Texture Synthesis and Compression using Gibbs Random Fields

David A. Bader*
dbader@eng.umd.edu

Joseph JáJá†
joseph@src.umd.edu

Rama Chellappa‡
chella@eng.umd.edu

Department of Electrical Engineering, and
Institute for Advanced Computer Studies,
University of Maryland, College Park, MD 20742

October 4, 1993

Abstract

This paper introduces scalable data parallel algorithms for image processing. Focusing on Gibbs and Markov Random Field model representation for textures, we present parallel algorithms for texture synthesis, compression, and maximum likelihood parameter estimation, currently implemented on Thinking Machines CM-2 and CM-5. Use of fine-grained, data parallel processing techniques yields real-time algorithms for texture synthesis and compression that are substantially faster than the previously known sequential implementations. Although current implementations are on Connection Machines, the methodology presented here enables machine independent scalable algorithms for a number of problems in image processing and analysis.

Permission to publish this abstract separately is granted.

Keywords: Gibbs Sampler, Gaussian Markov Random Fields, Image Processing, Texture Synthesis, Texture Compression, Scalable Parallel Processing, Data Parallel Algorithms.

*The support by NASA Graduate Student Researcher Fellowship No. NGT-50951 is gratefully acknowledged.

†Supported in part by NSF Engineering Research Center Program NSFD CDR 8803012 and NSF grant No. CCR-9103135. Also, affiliated with the Institute for Systems Research.

‡Supported in part by Air Force grant No. F49620-92-J0130.

1 Introduction

Random Fields have been successfully used to sample and synthesize textured images ([14], [10], [24], [21], [17], [32], [12], [11], [15], [18], [9], [37], [6], [7], [8]). Texture analysis has applications in image segmentation and classification, biomedical image analysis, and automatic detection of surface defects. Of particular interest are the models that specify the statistical dependence of the gray level at a pixel on those of its neighborhood. There are several well-known algorithms describing the sampling process for generating synthetic textured images, and algorithms that yield an estimate of the parameters of the assumed random process given a textured image. Impressive results related to real-world imagery have appeared in the literature ([14], [17], [12], [18], [37], [6], [7], [8]). However, all these algorithms are quite computationally demanding because they typically require on the order of $G\ n^2$ arithmetic operations per iteration for an image of size $n \times n$ with G gray levels. The implementations known to the authors are slow and operate on images of size 128×128 or smaller. Recently, a parallel implementation has been developed on a DAP 510 computer [21]. However, the DAP requires the structure of the algorithms to match its topology, and hence the corresponding algorithms are not as machine-independent as the algorithms described in this paper. In addition, we show that our algorithms are scalable in machine size and problem size.

In this paper, we develop scalable data parallel algorithms for implementing the most important texture sampling and synthesis algorithms. The data parallel model is an architecture-independent programming model that allows an arbitrary number of virtual processors to operate on large amounts of data in parallel. This model has been shown to be efficiently implementable on both SIMD (Single Instruction stream, Multiple Data stream) or MIMD (Multiple Instruction stream, Multiple Data stream) machines, shared-memory or distributed-memory architectures, and is currently supported by several programming languages including $C^\star$, data-parallel C, Fortran 90, Fortran D, and CM Fortran. All our algorithms are scalable in terms of the number of processors and the size of the problem. All the algorithms described in this paper have been implemented and thoroughly tested on a Connection Machine CM-2 and a Connection Machine CM-5.

The Thinking Machines CM-2 is an SIMD machine with 64K bit-serial processing elements (maximal configuration). This machine has 32 1-bit processors grouped into a *Sprint* node to accelerate floating-point computations, and 2^{11} *Sprint* nodes are configured as an 11-dimensional hypercube. See Figure 1 for the organization of a Sprint node.

The Thinking Machines CM-5 is a massively parallel computer with configurations containing 16 to 16,384 sparc processing nodes, each of which has four vector units, and the

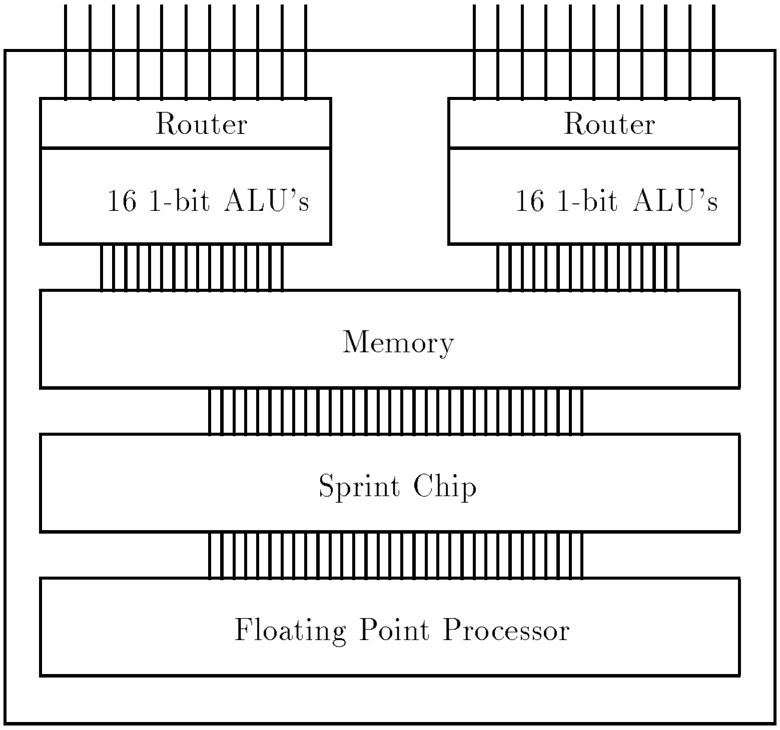

Figure 1: The organization of a CM-2 Sprint node

nodes are connected via a fat-tree communications network. The CM-5 is an MIMD machine which can run in Single Program Multiple Data (SPMD) mode to simulate SIMD operation. An in-depth look at the network architecture of this machine is described in [29]. The nodes operate in parallel and are interconnected by a fat-tree data network. The fat-tree resembles a quad-tree, with each processing node (PN) as a leaf and data routers at all internal connections. In addition, the bandwidth of the fat-tree increases as you move up the levels of the tree towards the root. Leiserson ([28]) discusses the benefits of the fat-tree routing network, and Greenberg and Leiserson ([19]) bound the time for communications by randomized routing on the fat-tree. In this paper, we assume the Single Program Multiple Data (SPMD) model of the CM-5, using the data parallel language $C^\star$. In the SPMD model, each processing node executes a portion of the same program, but local memory and machine state can vary across the processors. The SPMD model efficiently simulates the data parallel SIMD model normally associated with massively parallel programming. References [40] and [34] provide an overview for the CM-5, and both [43] and [45] contain detailed descriptions of the data parallel platform. Note that a CM-5 machine with vector units has four vector units per node, and the analysis given here will remain the same. See Figure 2 for the general organization of the CM-5 with vector units.

This paper addresses a simple image processing problem of texture synthesis and com-

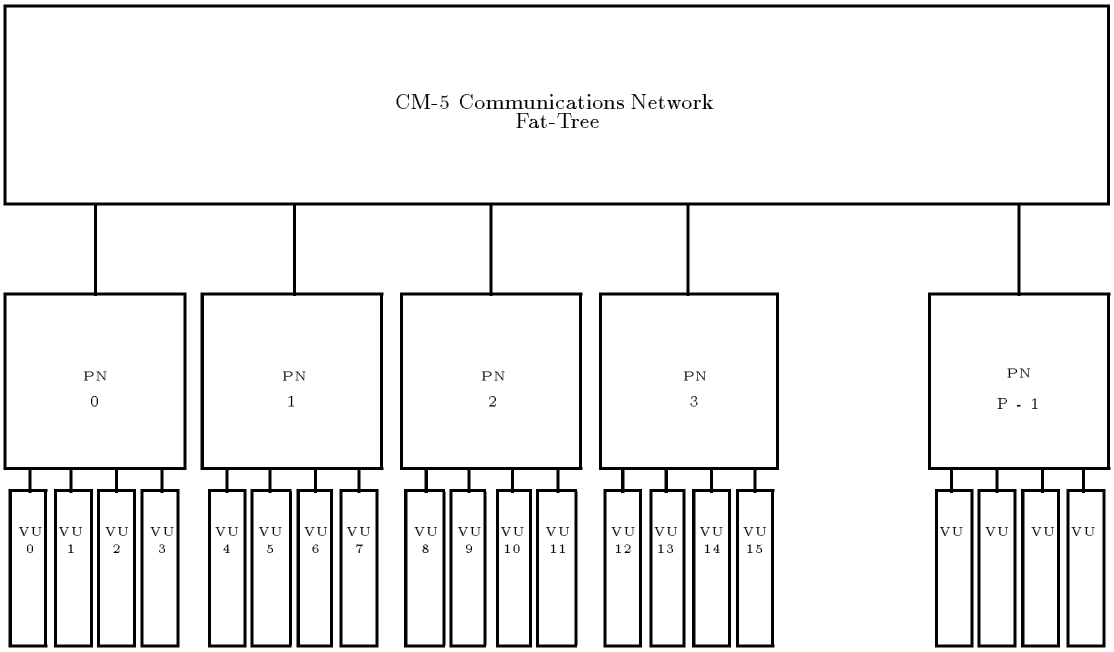

Figure 2: The organization of the CM-5

pression using a Gibbs sampler as an example to show that these algorithms are indeed scalable and fast on parallel machines. Gibbs sampler and its variants are useful primitives for larger applications such as image compression ([8], [27]), image estimation ([18], [21], [37]), and texture segmentation ([32], [15], [12], [16], [13]).

Section 2 presents our model for data parallel algorithm analysis on both the CM-2 and the CM-5. In Section 3, we develop parallel algorithms for texture synthesis using Gibbs and Gaussian Markov Random Fields. Parameter estimation for Gaussian Markov Random Field textures, using least squares, as well as maximum likelihood techniques, are given in Section 4. Section 5 shows fast parallel algorithms for texture compression using the maximum likelihood estimate of parameters. Conclusions are given in Section 6.

2 Parallel Algorithm Analysis

A data parallel algorithm can be viewed as a sequence of parallel synchronous steps in which each parallel step consists of an arbitrary number of concurrent primitive data operations. The complexity of a data parallel algorithm can be expressed in terms of two architecture-independent parameters, the *parallel time*, i.e., the number of parallel steps, and the *work*, i.e., the total number of operations used by the algorithm [20]. However we concentrate here on evaluating the scalability of our algorithms on two distinct architectures, namely the Connection Machines CM-2 and CM-5. In this case we express the complexity of a parallel algorithm with respect to two measures: the *computation complexity* $T_{comp}(n,p)$ which is the time spent by a processor on local computations, and the *communication complexity* $T_{comm}(n,p)$ which is the time spent on interprocessor communication of the overall algorithm. We use the standard sequential model for estimating $T_{comp}(n,p)$. However an estimate of the term $T_{comm}(n,p)$ depends in general on the data layout and the architecture of the machine under consideration. Our goal is to split the overall computation almost equally among the processors in such a way that $T_{comm}(n,p)$ is minimum. We discuss these issues next.

2.1 Parallel Communications Model

The model we present assumes that processors pass messages to each other via a communications network. In this model, a processor sending a message incurs overhead costs for startup such as preparing a message for transmission, alerting the destination processor that it will receive this message, and handling the destination's acknowledgment. We call this time $\tau(p)$.

The sending processor partitions the outgoing message into sections of length β bits, determined by hardware constraints. Each packet, with routing information prepended and an integrity check field using a cyclic redundancy code appended, is sent through the data network. The destination processor then reassembles the complete message from the received packets.

The communications time to send a message with this model is

$$T_{comm}(n,p) \; = \tau(p) + l(n,p)\gamma \tag{1}$$

where $l(n,p)$ is the message length, and γ is the transmission rate of the source into the network measured in seconds per packet.

Fortunately, all the algorithms described in this paper use regular communication patterns whose complexities can be easily estimated on the two architectures of the CM-2 and of the CM-5. Each such communication pattern can be expressed as a constant number of *block permutations*, where given blocks of contiguous data, $B_1, B_2, \ldots, B_i, \ldots, B_p$, residing in processors $P_1, P_2, \ldots, P_i, \ldots, P_p$, and a permutation $\Pi_i\{1, 2, \ldots, p\}$, block B_i has to be sent from processor P_i to $P_{\Pi(i)}$, where each block is of the same size, say $|B_i| = \frac{n}{p}$. As we illustrate next, on both the CM-2 and the CM-5, the communication complexity of a regular block permutation can be expressed as follows:

$$T_{comm}(n,p) \; = \mathrm{O}\Big(\tau(p) + \tfrac{n}{p}\,\gamma\Big) \;\; , \tag{2}$$

where τ is the start-up cost and γ is the packet transmission rate. We next address how to estimate the communication complexity for operations on $\sqrt{n} \times \sqrt{n}$ images and their relationship to block permutations.

2.2 CM-5 Communications Model

As described in Subsection 2.1, we use (1) as the general model for CM-5 communications. As stated above, a common communications pattern on parallel machines is a block permutation, where given blocks of contiguous data $B_1, B_2, \ldots, B_p$, and a permutation $\Pi_i\{1, 2, \ldots, p\}$, and $|B_i| = l(n,p)$, block B_i has to be moved from P_i to $P_{\Pi(i)}$. If each block has length $|B_i| = l(n,p)$, we claim that the CM-5 time complexity of this operation is the following:

$$T_{comm}(n,p) \; = \mathrm{O}\Big(\tau(p) + l(n,p)\gamma\Big). \tag{3}$$

Each node in the CM-5 connects to the fat-tree via a 20 Mb/sec network interface link [34]. The fat-tree network provides sufficient bandwidth to allow every node to perform sustained data transfers at a rate of 20 Mb/sec if all communications is contained in groups of four nodes (i.e. the nodes differ only in the last two bits of their logical address), 10 Mb/sec for transfers within groups of 16 nodes (i.e. the nodes differ in only the last four bits), and 5 Mb/sec for all other communication paths in the system [34].

A regular grid shift is an example of a block permutation pattern of data movement as shown in Subsection 2.2.2. For the corresponding regular block communications, the CM-5 can achieve bandwidths of 15 megabytes/sec per processor to put messages into and take messages out of the data network [29]. The runtime system (RTS) of the CM-5 will choose a data section for a message of between 1 and 5 words in length. If we assume that the data section of a message, β, is four 4-byte words, and the header and trailer are an additional 4 bytes, then $\gamma = \frac{16+4}{15*2^{20}} \approx 1.27\mu s$/packet.

To support these claims, we give the following empirical results. Using the message passing library of the CM-5, CMMD version 3.0 [41], each processing node swaps a packet with another node of fixed distance away. Figure 3 confirms that there is a linear relationship between message length and total transmission time on the CM-5. We find that when the regular block permutation consists of only local communications, i.e. messages are contained in each cluster of four leaf nodes, $\Pi_1(i) = \{0 \leftrightarrow 1, 2 \leftrightarrow 3, 4 \leftrightarrow 5, \ldots, p-2 \leftrightarrow p-1\} = i \oplus 1$, the lower bound on time is obtained. A least squares analysis of this data finds $\tau(p = 32) = 120\mu s$ and $\gamma \approx 199 \frac{\text{ms}}{\text{Mb}} * 20 \frac{\text{bytes}}{\text{packet}} = 3.82\mu s$/packet. This is equivalent to a sustained transfer rate of 5.03 Mb/sec.

Other regular block permutations are shown in Figure 3 such as

$\Pi_2(i) = \{0 \leftrightarrow 4, 1 \leftrightarrow 5, 2 \leftrightarrow 6, 3 \leftrightarrow 7, 8 \leftrightarrow 11, \ldots\} = i \oplus 4,$

$\Pi_3(i) = \{0 \leftrightarrow 8, 1 \leftrightarrow 9, 2 \leftrightarrow 10, 3 \leftrightarrow 11, \ldots\} = i \oplus 8,$ and

$\Pi_4(i) = \{0 \leftrightarrow 16, 1 \leftrightarrow 17, 2 \leftrightarrow 18, 3 \leftrightarrow 19, \ldots\} = i \oplus 16.$

Both Π_2 and Π_3 use two levels of the fat-tree, while Π_4 needs three levels. Our results show that all non-local regular permutations routed through the fat-tree have similar time complexities, with $\tau(p = 32) = 129\mu s$ and $\gamma \approx 220\frac{\text{ms}}{\text{Mb}} * 20\frac{\text{bytes}}{\text{packet}} = 4.20\mu s/\text{packet}$. This corresponds to a sustained transfer rate of 4.55 Mb/sec.

2.2.1 $C^\star$ Layout of parallel arrays

This subsection describes the compiler generated data layout of a $\sqrt{n} \times \sqrt{n}$ parallel array using $C^\star$. A major grid of size $v \times w$, where

$v \times w = p = 2^k$

is placed over the data array, with the constraints that both v and w are powers of two, and the lengths v and w are chosen such that the physical grid is as close to the original aspect ratio of the parallel data array as possible:

$v = 2^{\lfloor \frac{k}{2} \rfloor}$

$w = 2^{\lceil \frac{k}{2} \rceil}$

Axis 0 is divided equally among v nodes, and similarly, Axis 1 among w nodes. Each node thus receives an $\frac{\sqrt{n}}{v} \times \frac{\sqrt{n}}{w}$ subgrid, or tile, of the original data array. In each node, the data is laid out in row-major order form, that is, elements in each row of the subgrid are contiguous, while elements adjacent in a column have a stride in memory by $\frac{\sqrt{n}}{w}$ positions [43].

2.2.2 Simple and Regular Grid Shift

A typical operation in our image processing algorithms requires that each pixel be updated based on the pixel values of a small neighborhood. Such an operation amounts to a regular grid shift of a $\sqrt{n} \times \sqrt{n}$ element parallel variable data array.

Two phases occur in the regular grid shift:

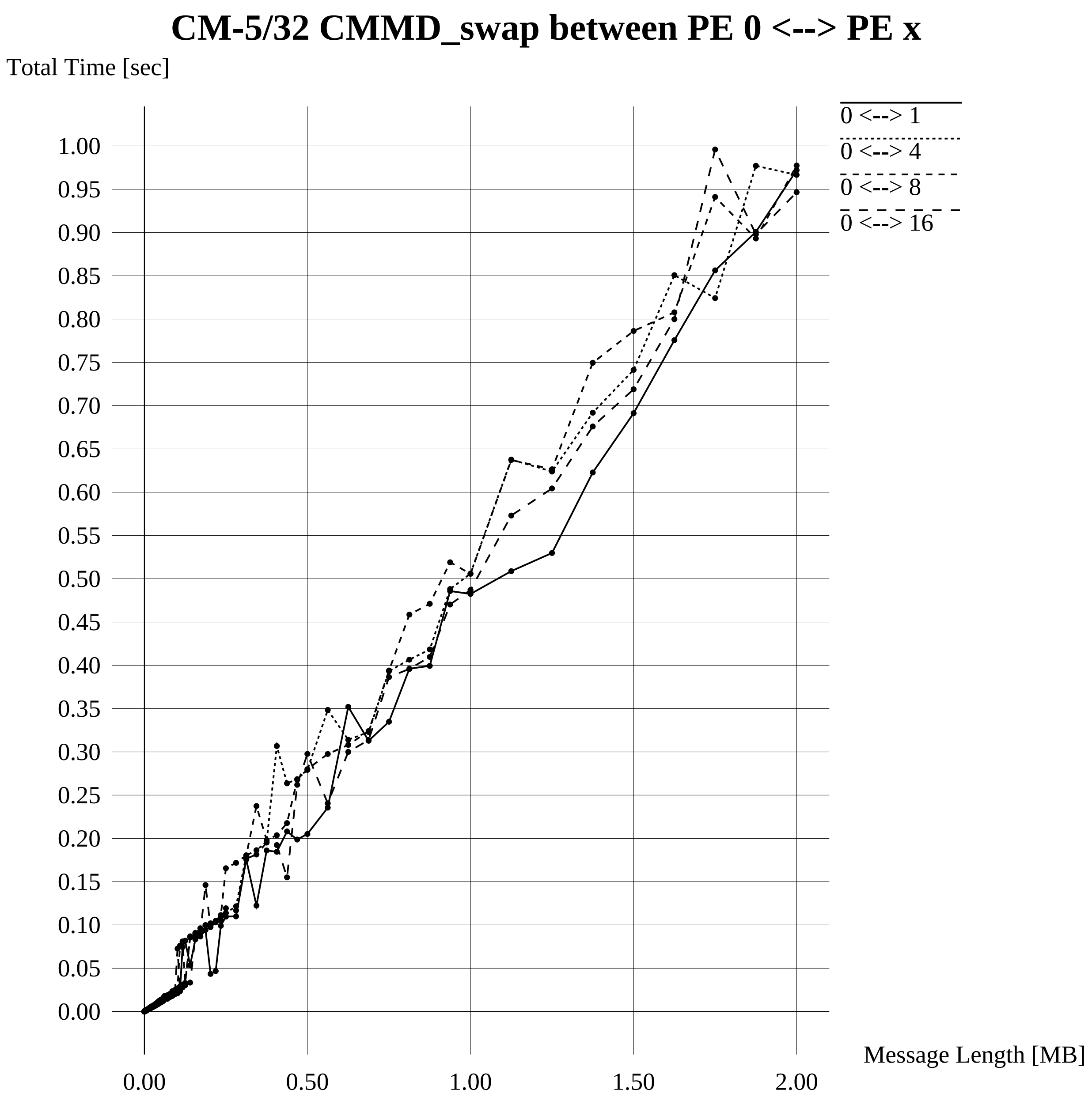

Figure 3: Sending Time of a Regular Block Permutation

Computation Time $T_{comp}(n,p)$ = Each node accesses its own local memory to shift up its own subgrid;

Communication Time $T_{comm}(n,p)$ = Each node sends and receives the elements lying along the shifted subgrid border.

For $p = v * w$ processors, $T_{comp}(n,p)$ takes $\mathrm{O}\left(\frac{\sqrt{n}}{v} * \frac{\sqrt{n}}{w}\right) = \mathrm{O}\left(\frac{n}{p}\right)$ time.

Next we make some observations about the communication phase:

- All communications are regular:
  P_i sends to $P_{(i-w) \bmod p}$,
  P_i receives from $P_{(i+w) \bmod p}$;
- This regular permutation is known a priori;

This communication pattern is a block permutation, and the time complexity for this operation is given in (3). Note that each node must send a constant number of upper rows of its subgrid to its north adjacent node in the major grid. There are $\frac{\sqrt{n}}{w}$ elements along this border. Thus, $l(n,p) = \mathrm{O}\left(\frac{\sqrt{n}}{w}\right)$, and the communications time is as follows:

$$T_{comm}(n,p) \;= \mathrm{O}\left(\tau(p) + \sqrt{\tfrac{n}{p}}\,\gamma\right) \;\text{ time.} \tag{4}$$

Therefore, a regular grid shift of a $\sqrt{n} \times \sqrt{n}$ data array on p nodes connected by a CM-5 fat-tree has the following time and communication complexity:

$$\begin{cases} T_{comp_{[\mathrm{shift}]}}(n,p) &= \mathrm{O}\left(\frac{n}{p}\right); \\ T_{comm_{[\mathrm{shift}]}}(n,p) &= \mathrm{O}\left(\tau(p) + \sqrt{\frac{n}{p}}\,\gamma\right). \end{cases} \tag{5}$$

2.3 CM-2 Communications Model

The CM-2 contains from 2^8 to 2^{11} Sprint nodes interconnected by a binary hypercube network. Each Sprint node consists of two chips of processors, each with 16 bit-serial processing elements. Thus, the CM-2 has configurations ranging between $8K$ and $64K$, inclusive, bit-serial processors. In this analysis, we view each Sprint node as a single 32-bit processing node.

The CM-2 programming model of $C^\star$ uses a canonical data layout similar to that on the CM-5 described in Section 2.2.1 ([39], [38], [44], [4]). The only difference on the CM-2 and CM-5 is that the Sprint node, or major grid, portion of each data element address is in reflected binary gray code, insuring that nearest neighbor communications are at most one hop away in the hypercube interconnection network.

We now analyze the complexity for a regular grid shift on the CM-2. Each node holds a contiguous subgrid of $\frac{n}{p}$ data elements. During a grid shift, $\mathrm{O}\left(\frac{n}{p}\right)$ elements remain local to each Sprint node, giving

$$T_{comp_{[\mathrm{shift}]}}(n,p) \ = \mathrm{O}\left(\frac{n}{p}\right). \tag{6}$$

Each Sprint node will send border elements to their destination node. By the canonical layout, we are assured that this destination is an adjacent node in the hypercube. Every Sprint node will send and receive elements along unique paths of the network. In this model, we neglect $\tau(p)$ since we are using direct hardware links which do not incur the overhead associated with packet routing analyses. Thus, each Sprint node will need to send and receive $\mathrm{O}\left(\sqrt{\frac{n}{p}}\right)$ elements, yielding a communications complexity of

$$T_{comm_{[\mathrm{shift}]}}(n,p) \ = \mathrm{O}\left(\sqrt{\frac{n}{p}}\,\gamma\right), \tag{7}$$

where γ is the CM-2 transmission rate into the network.

Therefore, on a CM-2 with p *Sprint* nodes, a regular grid shift of a $\sqrt{n} \times \sqrt{n}$ data array has the following time complexity analyses:

$$\left\{ \begin{array}{lcl} T_{comp_{[\mathrm{shift}]}}(n,p) & = & \mathrm{O}\left(\frac{n}{p}\right); \\ T_{comm_{[\mathrm{shift}]}}(n,p) & = & \mathrm{O}\left(\sqrt{\frac{n}{p}}\,\gamma\right). \end{array} \right. \tag{8}$$

As shown, a regular grid shift on the CM-2 is scalable for the array size and the machine size.

2.4 Complexity of Some Basic Operations

A two-dimensional Fast Fourier Transform (FFT) is a commonly used technique in digital image processing, and several algorithms in this paper make use of it. The FFT is well-suited for parallel applications because it is efficient and inherently parallel ([20], [1], [22], [23], [42]). With an image size of n elements, $\mathrm{O}(n \log n)$ operations are needed for an FFT. On a parallel machine with p processors, $\mathrm{O}\left(\frac{n}{p}\log n\right)$ computational steps are required. The communications needed for an FFT are determined by the FFT algorithm implemented on a particular parallel machine. The CM-2 pipelines computations using successive butterfly stages [42]. Its total time complexity is given by:

$$\left\{ \begin{array}{lcl} T_{comp_{[\mathrm{fft}]}}(n,p) & = & \mathrm{O}\left(\frac{n}{p}\log n\right); \\ T_{comm_{[\mathrm{fft}]}}(n,p) & = & \mathrm{O}\left(\frac{n}{p}\,\gamma\right). \end{array} \right.$$

On the CM-5, however, this algorithm would not be efficient. Instead, a communications efficient algorithm described in [42] is used, and has complexity:

$$\begin{cases} T_{comp_{[\mathrm{fft}]}}(n,p) & = & \mathrm{O}\Big(\frac{n}{p}\log n\Big); \\ T_{comm_{[\mathrm{fft}]}}(n,p) & = & \mathrm{O}\Big(\tau(p) + \frac{n}{p}\,\gamma\Big). \end{cases}$$

for $p \leq \sqrt{n}$.

Another extensively used and highly parallel primitive operation is the *Scan* operation. Given an array A with n elements, $\{A(1), A(2), \ldots, A(n)\}$, its scan will result in array C, where $C(i) = C(1) \star C(2) \star \ldots \star C(i)$ and $\star$ is any associative binary operation on the set, such as addition, multiplication, minimum, and maximum, for real numbers. A scan in the forward direction yields the parallel-prefix operation, while a reverse scan is a parallel-suffix. We can extend this operation to *segmented* scans, where we also input an n element array B of bits, such that for each i, $1 \leq i \leq n$, the element $C(i)$ equals $A(j) \star A(j+1) \star \ldots \star A(i)$, where j is the largest index of segment array B with $j \leq i$ and $B(j) = 1$.

A scan operation on a sequential machine obviously takes $\mathrm{O}(n)$ operations. An efficient parallel algorithm uses a binary tree to compute the scan in $\mathrm{O}(\log n)$ time with $\mathrm{O}(n)$ operations [20]. On the CM-2, the complexity for a scan is given by: [5]

$$\begin{cases} T_{comp_{[\mathrm{scan}]}}(n,p) & = & \mathrm{O}\Big(\frac{n}{p}\Big); \\ T_{comm_{[\mathrm{scan}]}}(n,p) & = & \mathrm{O}\Big(\frac{n}{p}\,\gamma\Big). \end{cases}$$

The CM-5 efficiently supports scans in hardware [29] and has complexity:

$$\begin{cases} T_{comp_{[\mathrm{scan}]}}(n,p) & = & \mathrm{O}\Big(\frac{n}{p}\Big); \\ T_{comm_{[\mathrm{scan}]}}(n,p) & = & \mathrm{O}\Big(\tau(p) + \big(\log p\big)\gamma\Big). \end{cases}$$

As the above complexities show, these algorithms efficiently scale for problem and machine size.

The data parallel programming paradigm is ideally suited for image processing since a typical task consists of updating each pixel value based on the pixel values in a small neighborhood. Assuming the existence of sufficiently many virtual processors, this processing task can be completed in time proportional to the neighborhood size. There are several powerful techniques for developing data parallel algorithms including scan (prefix sums) operations, divide-and-conquer, partitioning (data and function), and pipelining. We use several of these techniques in our implementations of texture synthesis and compression algorithms.

3 Texture Synthesis

3.1 A Parallel Gibbs Sampler

A discrete Gibbs random field (GRF) is specified by a probability mass function of the image as follows:

$$\Pr\left(X = x\right) = e^{-\frac{U(x)}{Z}} , \tag{9}$$

where $U(x)$ is the energy function, and $Z = \sum U(x)$, over all G^n images; G being the number of gray levels, and the image is of size $\sqrt{n} \times \sqrt{n}$. Except in very special circumstances, it is not feasible to compute Z. A relaxation-type algorithm described in [14] simulates a Markov chain through an iterative procedure that re-adjusts the gray levels at pixel locations during each iteration. This algorithm sequentially initializes the value of each pixel using a uniform distribution. Then a single pixel location is selected at random, and using the conditional distribution that describes the Markov chain, the new gray level at that location is selected, dependent only upon the gray levels of the pixels in its local neighborhood. The sequential algorithm terminates after a given number of iterations.

```
                                        x
                                        x
     x x x                          x x x x x
   x x x x x                        x x x x x
   x x o x x                  x x x x o x x x x
   x x x x x                        x x x x x
     x x x                          x x x x x
                                        x
                                        x
```

Figure 4: (A) Fourth Order Neighborhood (B) Higher Order Neighborhood

The sequential algorithm to generate a Gibbs random field described in [14] and [17] are used as a basis for our parallel algorithm. We introduce some terminology before presenting the parallel algorithm.

The neighborhood model N of a pixel is shown in Figure 4. For all the algorithms given in this paper, we use a symmetric neighborhood N_s which is half the size of N. This implies that if the vector $(\imath, \jmath) \in N$, then $(-\imath, -\jmath) \in N$, but only one of $\{(\imath, \jmath), (-\imath, -\jmath)\}$ is in N_s. Each element of array Θ is taken to represent the parameter associated with its corresponding element in N_s. We use the notation y_σ to represent the gray level of the image at pixel location σ.

Our Gibbs random field is generated using a simulated annealing type process. For an image with G gray levels, the probability $\Pr\left(X = k \mid \text{neighbors}\right)$ is binomial with parameter

$\Psi(T,k) = \frac{e^{kT}}{1+e^{T}}$, and number of trials $G-1$. The array {T} is given in the following equation for a first-order model:

$$\mathrm{T} = \alpha + \theta_{(1,0)}(y_{\sigma+(1,0)} + y_{\sigma-(1,0)}) + \theta_{(0,1)}(y_{\sigma+(0,1)} + y_{\sigma-(0,1)}) \tag{10}$$

and is a weighted sum of neighboring pixels at each pixel location. Additional examples of {T} for higher order models may be found in [14].

This algorithm is ideal for parallelization. The calculation of {T} requires uniform communications between local processing elements, and all other operations needed in the algorithm are data independent, uniform at each pixel location, scalable, and simple. The parallel algorithm is as follows:

Algorithm 1 *Gibbs Sampler*

Generate a Gibbs Random Field texture from parameters, assuming toroidal wrap-around for an $I \times J$ rectangular image.

Input:

$\{\ \alpha\ \} \leftarrow$ the parameter used to bias {T} in order to give the sampled texture a non-zero mean gray level.

$\{\ \Theta\ \} \leftarrow$ the array of parameters for each element in the model.

$\{\ G\ \}$ is the number of gray levels.

begin

1. **Initialize** image in parallel to uniformly distributed colors between 0 and G-1, inclusive.
2. **Iterate** for a given number of times, **for all** pixels **in parallel do**:
   - 2.1 **Calculate** T using parameters $\{\ \alpha\ \}$ and array $\{\ \Theta\ \}$.
   - 2.2 **Calculate** $p[0] = \frac{1}{1+e^{T}}$
   - 2.3 **For all** gray levels $\{g\}$ from [1..G-1] **do**:
     - 2.3.1 $\Psi(T,g) = \frac{e^{gT}}{1+e^{T}}$
     - 2.3.2 $p[g] = \binom{G-1}{g} \Psi^{g}(1-\Psi)^{G-1-g}$ (The Binomial Distribution)
   - 2.4 **Generate** a random number in the interval [0,1] at each pixel location and use this to **select** the new gray level $\{g\}$ from p[g].

end

An example of a binary synthetic texture generated by the Gibbs Sampler is given in Figure 5.

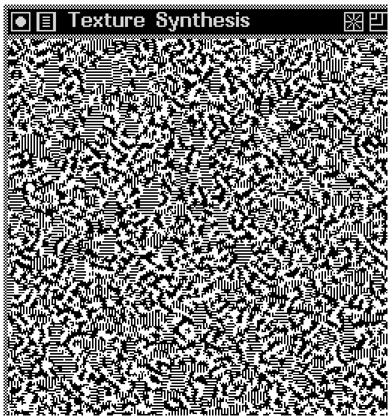

Figure 5: Isotropic Inhibition Texture using Gibbs Sampler (Texture 9b from [14]).

With $p \leq I \times J$ processing elements, and within each iteration, step 2.1 can be executed in $\mathrm{O}\Big(|N_s| T_{comp_{[\text{shift}]}}(n,p) \Big)$ computational steps and $\mathrm{O}\Big(|N_s| T_{comm_{[\text{shift}]}}(n,p) \Big)$ communication complexity, and steps 2.3 and 2.4 in $\mathrm{O}\Big(G\left(\frac{n}{p}\right)\Big)$ computational time, yielding a computation complexity of

$$T_{comp}(n,p) \ = \mathrm{O}\Big(\tfrac{n\ (G + |N_s|)}{p} \Big)$$

and communication complexity of

$$\begin{cases} T_{comm}(n,p) & = & \mathrm{O}\Big(|N_s| \sqrt{\tfrac{n}{p}}\ \gamma \Big), \text{ on the CM-2;} \\ T_{comm}(n,p) & = & \mathrm{O}\Big(|N_s| \big(\tau(p) + \sqrt{\tfrac{n}{p}}\ \gamma \big) \Big), \text{ on the CM-5,} \end{cases}$$

per iteration for a problem size of $n = I \times J$.

Table 1 shows the timings of a binary Gibbs sampler for model orders 1, 2, and 4, on the CM-2, and Table 2 shows the corresponding timings for the CM-5. Table 3 presents the timings on the CM-2 for a Gibbs sampler with fixed model order 4, but varies the number of gray levels, G. Table 4 gives the corresponding timings on the CM-5.

3.2 Gaussian Markov Random Field Sampler

In this section, we consider the class of 2-D non-causal models called the Gaussian Markov random field (GMRF) models described in [6], [12], [21], and [46]. Pixel gray levels have joint Gaussian distributions and correlations controlled by a number of parameters representing the statistical dependence of a pixel value on the pixel values in a symmetric neighborhood. There are two basic schemes for generating a GMRF image model, both of which are discussed in [6].

Image Size	*Order = 1*		*Order = 2*		*Order = 4*	
	8k CM-2	16k CM-2	8k CM-2	16k CM-2	8k CM-2	16k CM-2
8k	0.00507		0.00692		0.01270	
16k	0.00964	0.00507	0.01280	0.00692	0.02293	0.01270
32k	0.01849	0.00962	0.02395	0.01274	0.04214	0.02275
64k	0.03619	0.01846	0.04605	0.02386	0.07836	0.04182
128k	0.07108	0.03615	0.08872	0.04592	0.14520	0.07789
256k	0.14102	0.07108	0.17481	0.08872	0.28131	0.14520
512k		0.14093		0.17455		0.28036

Table 1: Gibbs Sampler timings for a binary ($G = 2$) image (execution time in seconds per iteration on a CM-2 running at 7.00 MHz)

Image Size	*Order = 1*		*Order = 2*		*Order = 4*	
	16/vu CM-5	32/vu CM-5	16/vu CM-5	32/vu CM-5	16/vu CM-5	32/vu CM-5
8k	0.046053	0.024740	0.051566	0.027646	0.068486	0.038239
16k	0.089822	0.046824	0.099175	0.052411	0.130501	0.068630
32k	0.176997	0.089811	0.199399	0.099493	0.252421	0.132646
64k	0.351123	0.178046	0.398430	0.194271	0.560224	0.257647
128k	0.698873	0.351517	0.759017	0.383425	0.943183	0.582303
256k	1.394882	0.700164	1.526422	0.759747	1.874973	0.962165
512k	2.789113	1.394216	3.047335	1.520437	3.744542	1.892460
1M	5.577659	2.782333	6.009608	3.063054	7.428823	3.785890

Table 2: Gibbs Sampler timings for a binary ($G = 2$) image (execution time in seconds per iteration on a CM-5 with vector units)

3.2.1 Iterative Gaussian Markov Random Field Sampler

The Iterative Gaussian Markov Random Field Sampler is similar to the Gibbs Sampler, but instead of the binomial distribution, as shown in step 3.2 of Algorithm 1, we use the continuous Gaussian Distribution as the probability function. For a neighborhood model N, the conditional probability function for a GMRF is:

$$p(y_\sigma | y_{\sigma+r}, r \in N) = \frac{1}{\sqrt{2\pi\nu}} e^{-\frac{1}{2\nu}\left(y_\sigma - \sum_{r\in N} \Theta_r y_{\sigma+r}\right)^2}, \tag{11}$$

where $\{ \Theta_r \}$ is the set of parameters specifying the model, and ν is the variance of a zero mean noise sequence.

An efficient parallel implementation is straightforward and similar to that of the Gibbs Sampler (Algorithm 1). Also, its analysis is identical to that provided for Gibbs Sampler.

Image Size	*G = 16*	*G = 32*	*G = 64*	*G = 128*	*G = 256*
16k	0.03943	0.06976	0.13029	0.25157	0.49415
32k	0.07011	0.12383	0.23101	0.44586	0.87557
64k	0.12966	0.22927	0.42797	0.82639	1.62323
128k	0.24767	0.44017	0.82414	1.59418	
256k	0.47832	0.85602	1.60931		
512k	0.93884	1.68543			

Table 3: Gibbs Sampler timings using the 4th order model and varying G (execution time in seconds per iteration on a 16k CM-2 running at 7.00 MHz)

Image Size	*G = 16*	*G = 32*	*G = 64*	*G = 128*	*G = 256*
8k	0.072073	0.109722	0.186440	0.338833	0.644660
16k	0.123117	0.184224	0.308448	0.554801	1.047374
32k	0.238610	0.340773	0.557644	1.005135	1.883579
64k	0.450731	0.648609	1.093775	1.947461	3.660135
128k	0.845078	1.250694	2.127231	3.754634	7.077714
256k	1.654748	2.462672	4.149417	7.404596	13.958026
512k	3.296162	4.943185	8.190262	14.713778	27.740006
1M	6.566956	9.753557	16.169061	29.217335	

Table 4: Gibbs Sampler timings using the 4th order model and varying G (execution time in seconds per iteration on a 32 node CM-5 with vector units)

3.2.2 Direct Gaussian Markov Random Field Sampler

The previous section outlined an algorithm for sampling GMRF textured images using an iterative method. Unfortunately, this algorithm may have to perform hundreds or even thousands of iterations before a stable texture is realized. Next we present a scheme which makes use of two-dimensional Fourier transforms and does not need to iterate. The Direct GMRF Sampler algorithm is realized from [6] as follows. We use the following scheme to reconstruct a texture from its parameters Θ and a neighborhood N_s:

$$\mathbf{y} = \frac{1}{M^2} \sum_{\sigma \in \Omega} \boldsymbol{f}_\sigma \, \frac{x_\sigma}{\sqrt{\mu_\sigma}} \tag{12}$$

where $\mathbf{y}$ is the resulting M^2 array of the texture image, and

$$x_\sigma = \boldsymbol{f}_\sigma^{*t} \, \boldsymbol{\eta} \, ,$$

$$\mu_\sigma = (1 - 2\Theta^T \Phi_\sigma), \forall \sigma \in \Omega \tag{13}$$

$$\Phi_\sigma = \ \mathrm{Col}[\cos \frac{2\pi}{M} \sigma^t r, r \in N_s]. \tag{14}$$

The sampling process is as follows. We begin with $\boldsymbol{\eta}$, a Gaussian zero mean noise vector with identity covariance matrix. We generate its the Fourier series, via the Fast Fourier Transform from Subsection 2.4, using $\boldsymbol{f}_\sigma$, the Fourier vector defined below:

$$\boldsymbol{f}_\sigma \;=\; \mathrm{Col}[1, \lambda_\imath \,, \lambda_\imath^2 \, \boldsymbol{t}_\jmath \,, \ldots \,, \lambda_\imath^{M-1} \, \boldsymbol{t}_\jmath \,], \text{ is an } M^2 \text{ vector,} \tag{15}$$

$$\boldsymbol{t}_\jmath \;=\; \mathrm{Col}[1, \lambda_\jmath \,, \lambda_\jmath^2 \,, \ldots \,, \lambda_\jmath^{M-1} \,], \text{ is an } M\text{-vector, and} \tag{16}$$

$$\lambda_\imath = \exp\left(\sqrt{-1}\frac{2\pi\imath}{M}\right), \tag{17}$$

and finally apply (12).

Algorithm 2 *Direct Gaussian MRF Sampler*

Reconstruct a GMRF texture from parameters, assuming toroidal wrap-around and an M^2 *image size*

Input:

$\Theta \leftarrow$ the set of parameters for the given model.

$\{ \, G \, \}$ is the number of gray levels.

image $\leftarrow$ a parallel variable for the image. (Complex)

$\{\Phi_r\}$ a parallel variable with serial elements for each parameter in the model.

begin

1. Initialize the real part of the image in parallel to Gaussian noise with mean = 0 and standard deviation = 1.
2. Initialize the imaginary part of the image in parallel to 0.
3. Divide the image by $\sqrt{\nu}$.
4. Perform a parallel, in-place FFT on the noise.
5. **For all** pixels σ **in parallel do**:
   5.1 **For each** $r \in N_s$, $\Phi_r = \cos \frac{2\pi}{M} \sigma^t r$
   5.2 Calculate μ_σ from Equation (13).
   5.3 Divide the image by $\sqrt{\mu_\sigma}$.
6. Perform a parallel, in-place, inverse FFT on the image.
7. Scale the result to gray levels in the interval [0..G-1].

end

Steps 1, 2, 3, 5.2, 5.3, and 7 all run in $\mathrm{O}\left(\frac{n}{p}\right)$ parallel steps, where $n = M^2$ and p is the number of processors available. As stated in Subsection 2.4, an n-point FFT, used in steps 4 and 6, computed on p processors takes $T_{comp_{[\text{fft}]}}(n,p)$ computation time and $T_{comm_{[\text{fft}]}}(n,p)$ communications. Step 5.1 takes $\mathrm{O}\Big(\,|N_s|\,\left(\frac{n}{p}\right)\Big)$ parallel steps.

The Direct Gaussian MRF Sampler algorithm thus has a computation complexity of

$$T_{comp}(n,p) \;= \mathrm{O}\Big(\tfrac{n\,(|N_s|+\log n)}{p}\Big)$$

and communication complexity of

$$\begin{cases} T_{comm}(n,p) & = & \mathrm{O}\Big(\frac{n}{p}\,\gamma\Big), \text{ on the CM-2;} \\ T_{comm}(n,p) & = & \mathrm{O}\Big(\tau(p)+\frac{n}{p}\,\gamma\Big), \text{ on the CM-5,} \end{cases}$$

using $p \leq M$ processors.

Note that the number of gray levels, G, is only used in the last step of the algorithm as a scaling constant. Hence this algorithm scales with image size n and number of processors p, independent of the number of gray levels G used. Notice also that the communication complexity is higher than that of the Gibbs sampler; this is due to the fact that the FFT is a global operation on the image. Our experimental data collected by implementing this algorithm on the CM-2 and the CM-5 confirm our analysis.

4 Parameter Estimation for Gaussian Markov Random Field Textures

Given a real textured image, we wish to determine the parameters of a GMRF model which could be used to reconstruct the original texture through the samplers given in the previous section.

This section develops parallel algorithms for estimating the parameters of a GMRF texture. The methods of least squares (LSE) and of maximum likelihood (MLE), both described in [6], are used. We present efficient parallel algorithms to implement both methods. The MLE performs better than the LSE. This can be seen visually by comparing the textures synthesized from the LSE and MSE parameters, or by noting that the asymptotic variance of the MLE is lower than the LSE ([3], [25]).

4.1 Least Squares Estimate of Parameters

The least squares estimate detailed in [6] assumes that the observations of the GMRF image $\{y_\sigma\}$ obey the model

$$y_\sigma = \sum_{r\in N_s} \Theta_r[y_{\sigma+r}+y_{\sigma-r}] + e_\sigma, \quad \forall \sigma \in \Omega, \tag{18}$$

where $\{e_\sigma\}$ is a zero mean correlated noise sequence with variance ν and correlation with the following structure:

$$
\begin{aligned}
E(e_\sigma e_r) &= -\Theta_{\sigma-r}\nu, \quad (\sigma - r) \in N \\
&= \nu, \quad \sigma = r \qquad (19) \\
&= 0, \quad otherwise.
\end{aligned}
$$

The conditional distribution is given in (11). Then, for $\boldsymbol{g}_\sigma = \text{Col}[y_{\sigma+r'} + y_{\sigma-r'}, r' \in N_s]$, the LSE are:

$$\Theta^\star = \left[\sum_\Omega \boldsymbol{g}_\sigma \ \boldsymbol{g}_\sigma^t\right]^{-1} \left(\sum_\Omega \boldsymbol{g}_\sigma \ y_\sigma\right) \qquad (20)$$

$$\nu^\star = \frac{1}{M^2} \sum_\Omega \left(y_\sigma - \Theta^{\star t} \boldsymbol{g}_\sigma \ \right)^2 \qquad (21)$$

where Ω is the complete set of M^2 pixels, and toroidal wrap-around is assumed.

Algorithm 3 *Least Squares Estimator for GMRF*

Using the method of Least Squares, estimate the parameters of image Y. Assume toroidal wrap-around, an M^2 image size, and a given neighborhood.

Input:

$\{\mathbf{Y}\} \leftarrow$ the image.

$\Theta \leftarrow$ the scalar array of parameter estimates for each neighborhood element.

begin

1. **For all** pixels **in parallel do**:
   1.1 **For each** $r \in N_s$ **do**
      1.1.1 $\mathbf{g}_\sigma[r] = y_{\sigma+r} + y_{\sigma-r}$
   1.2 **For** $\imath$ from 1 to $|N_s|$ **do**
      1.2.1 **For** $\jmath$ from 1 to $|N_s|$ **do**
         1.2.1.1 **Calculate** $gcross_\sigma[\imath, \jmath] = \mathbf{g}_\sigma[\imath] \times \mathbf{g}_\sigma[\jmath]$.
2. **For** $\imath$ from 1 to $|N_s|$ **do**
   2.1 **For** $\jmath$ from 1 to $|N_s|$ **do**
      2.1.1 Compute **in parallel** the sum $gmatrix[\imath, \jmath] = \sum_{\sigma \in \Omega} gcross_\sigma[\imath, \jmath]$.
3. **For all** pixels σ **in parallel do**:
   3.1 **For each** $r \in N_s$ **do**

3.1.1 **Calculate** $gv_\sigma[r] = \mathbf{g}_\sigma[r] \times y_\sigma$

4. **For each** $r \in N_s$ **do**

   4.1 Compute **in parallel** the sum $gvec[r] = \sum_{\sigma\in\Omega} gv_\sigma[r]$

5. Solve the $|N_s| \times |N_s|$ linear system of equations:

   $[gmatrix]_{|N_s|\times|N_s|} \times [\Theta^\star]_{|N_s|\times 1} = [gvec]_{|N_s|\times 1}$

6. **Calculate** $\nu^\star = \frac{1}{M^2} \sum_{\sigma\in\Omega} (y_\sigma - \Theta^{\star t}\mathbf{g}_\sigma)^2$

end

For an image of size $n = M^2$, step 1.1 has a computational complexity of $\mathrm{O}\big(\, |N_s| T_{comp_{[\mathrm{shift}]}}(n,p)\,\big)$ parallel steps and a communication complexity of $\mathrm{O}\big(\, |N_s| T_{comm_{[\mathrm{shift}]}}(n,p)\,\big)$. Step 1.2 runs in $\mathrm{O}\big(\,(|N_s|)^2 \left(\frac{n}{p}\right)\big)$ parallel steps. Step 3 takes $\mathrm{O}\big(\,|N_s| \left(\frac{n}{p}\right)\big)$ parallel steps. Steps 2 and 4 contain a reduction over the entire array, specifically, finding the sum of the elements in a given parallel variable. As this is a scan operation, we refer the reader to Subsection 2.4 for algorithm analysis. Thus, step 2 runs is $\mathrm{O}\Big(\,(|N_s|)^2 T_{comp_{[\mathrm{scan}]}}(n,p)\,\Big)$ computational parallel steps with $\mathrm{O}\big(\,(|N_s|)^2 T_{comm_{[\mathrm{scan}]}}(n,p)\,\big)$ communications, and steps 4 and 6 run in $\mathrm{O}\big(\,|N_s| T_{comp_{[\mathrm{scan}]}}(n,p)\,\big)$ parallel steps with $\mathrm{O}\big(\,|N_s| T_{comm_{[\mathrm{scan}]}}(n,p)\,\big)$ communications. Solving the linear system of equations in step 5 takes $\mathrm{O}\big(\,(|N_s|)^3\big)$ computational steps.

The computational complexity of the Least Squares Estimator for an image of size $n = M^2$ is

$$T_{comp}(n,p) \;= \mathrm{O}\Big(\,\frac{n\,|N_s|^2}{p} + (|N_s|)^3\Big)$$

and the communication complexity is

$$\begin{cases} T_{comm}(n,p) & = & \mathrm{O}\Big(\,\frac{n\,|N_s|^2}{p}\,\gamma\Big), \text{ on the CM-2;} \\ T_{comm}(n,p) & = & \mathrm{O}\Big(\,|N_s|^2\,\tau(p) + \big(\,|N_s|\sqrt{\frac{n}{p}} + |N_s|^2\log p\big)\gamma\Big), \text{ on the CM-5,} \end{cases}$$

using $p \le M$ processors.

4.2 Maximum Likelihood Estimate of Parameters

We introduce the following approach as an improved method for estimating GMRF parameters of textured images. The method of maximum likelihood gives a better estimate of the texture parameters, since the asymptotic variance of the MLE is lower than that of the LSE. We also show a much faster algorithm for optimizing the joint probability density function which is an extension of the Newton-Raphson method and is also highly parallelizable.

Assuming a toroidal lattice representation for the image $\{y_\sigma\}$ and Gaussian structure for noise sequence $\{e_\sigma\}$, the joint probability density function is the following:

$$p(y|\Theta,\nu) = \frac{1}{(2\pi\nu)^{\frac{M^2}{2}}} \sqrt{\prod_{\sigma\in\Omega}\left(1-2\sum_{\tau_\imath\in N_s}\left(\Theta_{\tau_\imath}\Phi_{\tau_\imath}(\sigma)\right)\right)}\; e^{-\frac{1}{2\nu}\left[C(0) - \sum_{\tau_\imath\in N}\left(\Theta_{\tau_\imath} C(\tau_\imath)\right)\right]} \tag{22}$$

In (22), $C(\tau_\imath)$ is the sample correlation estimate at lag $\tau_\imath$. As described in [3] and [6], the log-likelihood function can be maximized: (Note that $F(\Theta,\nu) = \log p(y|\Theta,\nu)$).

$$\begin{aligned} F(\Theta,\nu) \quad = \quad - \quad & \frac{M^2}{2}\log 2\pi\nu \quad + \quad \frac{1}{2}\sum_{\sigma\in\Omega}\left(\log\left(1-2\sum_{\tau_\imath\in N_s}\left(\Theta_{\tau_\imath}\Phi_{\tau_\imath}(\sigma)\right)\right)\right) \\ & - \frac{1}{2\nu}\sum_{\sigma\in\Omega}\left(y(\sigma)^2 - y(\sigma)\sum_{\tau_\imath\in N_s}\left(\Theta_{\tau_\imath}\left(y(\sigma+r_\imath)+y(\sigma-r_\imath)\right)\right)\right) \end{aligned} \tag{23}$$

For a square image, $\Phi_{\tau_\imath}$ is given as follows:

$$\Phi_{\tau_\imath}(\sigma) \quad = \quad \cos\left(\frac{2\pi}{M}\sigma^T r_\imath\right) \tag{24}$$

This non-linear function $\mathbf{F}$ is maximized by using an extension of the Newton-Raphson method. This new method first generates a *search direction* ϑ^k by solving the system

$$[\nabla^2 F(\Theta_k)]_{(r+1)\times(r+1)}\ [\vartheta^k]_{(r+1)\times 1} = -[\nabla F(\Theta_k)]_{(r+1)\times 1}. \tag{25}$$

Note that this method works well when $\nabla^2 F(\Theta_k)$ is a symmetric, positive-definite Hessian matrix. We then *maximize* the step in the search direction, yielding an approximation to λ_k which attains the local maximum of $F(\Theta_k+\lambda\vartheta)$ and also satisfies the constraints that each of the M^2 values in the logarithm term for $\mathbf{F}$ is positive. Finally, an *optimality test* is performed. We set $\Theta_{k+1} = \Theta_k + \lambda\vartheta$, and if Θ_{k+1} is sufficiently close to Θ_k, the procedure terminates. We give the first and second derivatives of $\mathbf{F}$ with respect to Θ_k and ν in Appendix B.

For a rapid convergence of the Newton-Raphson method, it must be initialized with a good estimate of parameters close to the global maximum. We use the least squares estimate given in Subsection 4.1 as Θ_0, the starting value of the parameters.

Algorithm 4 *Maximum Likelihood Estimate*

Note that $\Theta_k \equiv \langle\theta_1,\theta_2,\ldots,\theta_r,\nu\rangle$.

Also, this algorithm assumes toroidal wrap-around of the image.

Note that in Step 5, $\beta < 1.0$ *, and we use* $\beta = 0.8$ *.*

Input:

$\{\mathbf{Y}\} \leftarrow$ the image.

begin

1. **Find** Initial Guess Θ_0 using LSE Algorithm 3.
2. **Compute** $\nabla F(\Theta_k) \equiv \langle \frac{\partial F}{\partial \theta_1}, \frac{\partial F}{\partial \theta_2}, \ldots, \frac{\partial F}{\partial \theta_r}, \frac{\partial F}{\partial \nu} \rangle$.
3. **Compute** $\nabla^2 F(\Theta_k) \equiv \begin{bmatrix} \frac{\partial^2 F}{\partial {\theta_1}^2} & \frac{\partial^2 F}{\partial \theta_1 \, \partial \theta_2} & \cdots & \frac{\partial^2 F}{\partial \theta_1 \, \partial \theta_r} & \frac{\partial^2 F}{\partial \theta_1 \partial \nu} \\ \frac{\partial^2 F}{\partial \theta_2 \, \partial \theta_1} & \frac{\partial^2 F}{\partial {\theta_2}^2} & \cdots & \frac{\partial^2 F}{\partial \theta_2 \, \partial \theta_r} & \frac{\partial^2 F}{\partial \theta_2 \partial \nu} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \frac{\partial^2 F}{\partial \theta_r \, \partial \theta_1} & \frac{\partial^2 F}{\partial \theta_r \, \partial \theta_2} & \cdots & \frac{\partial^2 F}{\partial {\theta_r}^2} & \frac{\partial^2 F}{\partial \theta_r \partial \nu} \\ \frac{\partial^2 F}{\partial \theta_v \, \partial \theta_1} & \frac{\partial^2 F}{\partial \theta_v \, \partial \theta_2} & \cdots & \frac{\partial^2 F}{\partial \theta_v \, \partial \theta_r} & \frac{\partial^2 F}{\partial \nu^2} \end{bmatrix}$.
4. **Solve** the following linear system of equations for vector ϑ

   $[\nabla^2 F(\Theta_k)]_{(r+1)\times(r+1)} \; [\vartheta]_{(r+1)\times 1} = -[\nabla F(\Theta_k)]_{(r+1)\times 1}$

5. **Determine** the largest λ *from* $\{1, \beta, \beta^2, \beta^3, \ldots\}$ such that

   (4a.) $1 - 2 \sum_{\tau_\iota \in N_s} \left(\Theta_{\tau_\iota} \; \Phi_{\tau_\iota} \; (\sigma) \right) > 0$; (note that these represent M^2 constraints)

   (4b.) $F(\Theta_k + \lambda\vartheta) > F(\Theta_k)$

6. **Set** $\Theta_{k+1} = \Theta_k + \lambda\vartheta$
7. If $|F(\Theta_{k+1}) - F(\Theta_k)| > \epsilon$ then go to Step 2.

end

The time complexities per iteration of the MLE algorithm are similar to that of the LSE algorithm analysis given in Subsection 4.1.

In Figures 6 - 8, we show the synthesis using least squares and maximum likelihood estimates for wool weave, wood grain, and tree bark, respectively, obtained from standard textures library. Tables 5, 6, and 7 show the respective parameters for both the LSE and MLE and give their log-likelihood function values. Each example shows that the maximum likelihood estimate improves the parameterization. In addition, CM-5 timings for these estimates varying machine size, image size, and neighborhood models can be found in Tables 8, 9, 10, and 11, for a 4th order model on this selection of real world textured images, and in Tables 12, 13, 14, and 15, for a higher order model on the same set of images. Similarly, CM-2 timings for these estimates can be found in [2]. Tables 8 - 15 are given in Appendix C.

5 Texture Compression

We implement an algorithm for compressing an image of a GMRF texture to approximately 1 bit/pixel from the original 8 bits/pixel image. The procedure is to find the MLE of the given image, (e.g. this results in a total of eleven 32-bit floating point numbers for the 4th order model). We then use a Max Quantizer, with characteristics given in [33], to quantize the residual to 1-bit. The quantized structure has a total of M^2 bits. To reconstruct the image from its texture parameters and 1-bit Max quantization, we use an algorithm similar to Algorithm 2. Instead of synthesizing a texture from Gaussian noise, we begin with the 1-bit quantized array. Compressed textures for a 4th order model are shown in Figures 6 and 7. A result using the higher order model is shown in Figure 8.

The noise sequence $\boldsymbol{\eta}$ is generated as follows:

$$\boldsymbol{\eta} \;=\frac{1}{M^2}\sum_{\sigma\in\Omega}\boldsymbol{f}_\sigma\; x_\sigma\sqrt{\mu_\sigma} \tag{26}$$

where

$$x_\sigma = \boldsymbol{f}_\sigma^{*t}\ \mathbf{y} \tag{27}$$

and the μ_σ is given in (13). We estimate the residual as:

$$\boldsymbol{\eta}^\star \;=\frac{1}{\sqrt{\nu^\star}}\,\boldsymbol{\eta} \tag{28}$$

and $\boldsymbol{\eta}^\star$ is the sequence which is Max quantized.

The image reconstruction from parameters and quantization $\boldsymbol{\eta}^\star$ is as follows:

$$\mathbf{y} = \frac{1}{M^2}\sum_{\sigma\in\Omega}\boldsymbol{f}_\sigma\;\frac{x_\sigma}{\sqrt{\mu_\sigma}} \tag{29}$$

where

$$x_\sigma = \boldsymbol{f}_\sigma^{*t}\;\;\boldsymbol{\eta}^\star \tag{30}$$

and μ_σ is given in (13); Φ_σ is given in (14).

The texture compression algorithm has the same time complexity and scalability characteristics as Algorithm 4. The image reconstruction algorithm has the same complexities as Algorithm 2. Hence these algorithms scale with image size n and number of processors p.

This algorithm could be used to compress the textures regions in natural images as part of segmentation based compression schemes discussed in [27]. Compression factors of 35 have been obtained for the standard Lena and F-16 images, with no visible degradations. Compression factors of 80 have been shown to be feasible when small degradations are permitted in image reconstruction.

6 Conclusions

We have presented efficient data parallel algorithms for texture analysis and synthesis based on Gibbs or Markov random field models. A complete software package running on the Connection Machine model CM-2 and the Connection Machine model CM-5 implementing these algorithms is available for distribution to interested parties. The experimental data strongly support the analysis concerning the scalability of our algorithms. The same type of algorithms can be used to handle other image processing algorithms such as image estimation ([18], [21], [37]), texture segmentation ([12], [17], [32]), and integration of early vision modules ([35]). We are currently examining several of these extensions.

A Example Texture Figures

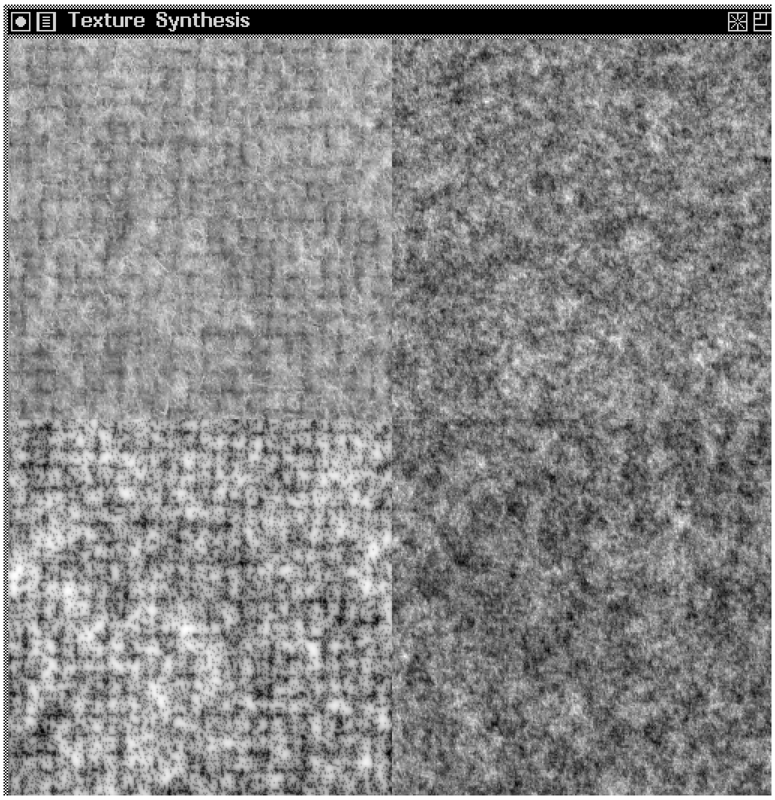

Figure 6: Wool Weave Texture: (clockwise from top left) original image, reconstructed from the LSE, MLE, and Compressed image. A fourth order model was used.

Parameter	LSE	MLE
(1,0)	0.428761	0.416797
(0,1)	0.203167	0.203608
(1,1)	0.021416	0.024372
(-1,1)	-0.080882	-0.082881
(0,2)	0.037685	0.050928
(2,0)	-0.080724	-0.061254
(-2,1)	0.027723	0.026702
(1,-2)	-0.016667	-0.026285
(1,2)	-0.033902	-0.042835
(2,1)	-0.008665	-0.010334
ν	23397.04	128.41
$F(\Theta)$	-264609.19	-264538.63

Table 5: Θ Parameters for Wool Weave

The parameters for the 256×256 image of wool weave in Figure 6 are given in Table 5.

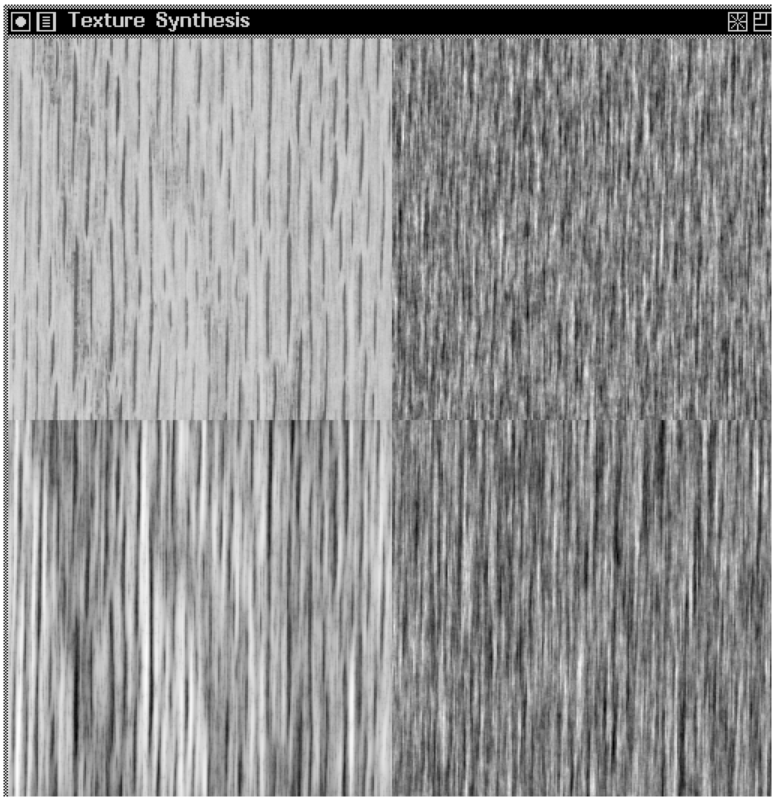

Figure 7: Wood Texture: (clockwise from top left) original image, reconstructed from the LSE, MLE, and Compressed image. A fourth order model was used.

Parameter	LSE	MLE
(1,0)	0.549585	0.526548
(0,1)	0.267898	0.273241
(1,1)	-0.143215	-0.142542
(-1,1)	-0.135686	-0.134676
(0,2)	0.001617	-0.006949
(2,0)	-0.051519	-0.027342
(-2,1)	0.006736	0.003234
(1,-2)	-0.002829	0.000907
(1,2)	0.000248	0.005702
(2,1)	0.006504	0.001766
ν	33337.88	12.84
$F(\Theta)$	-204742.39	-202840.50

Table 6: Θ Parameters for Wood Texture

The parameters for the 256×256 image of wood texture in Figure 7 are given in Table 6.

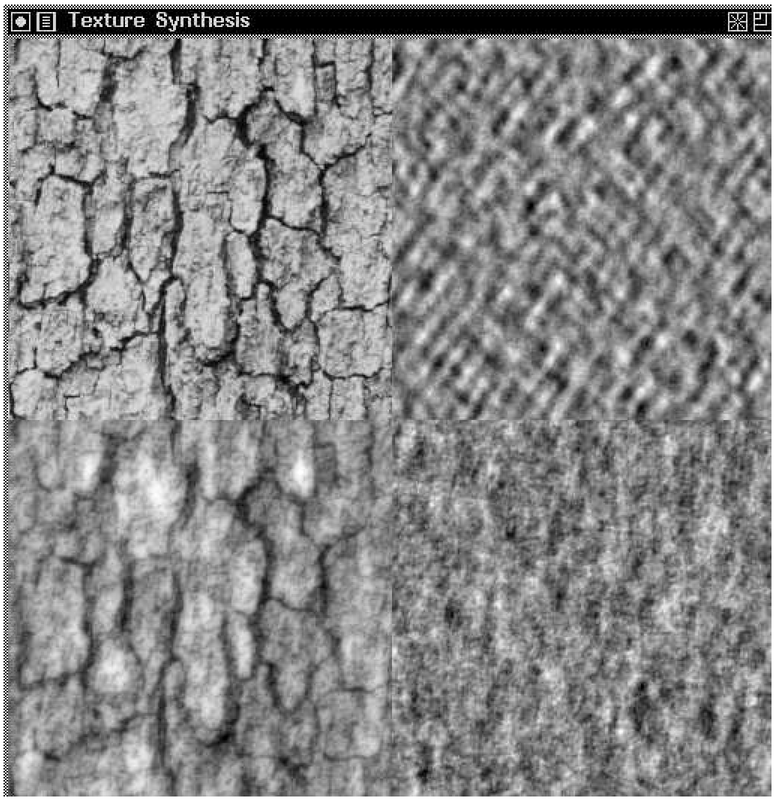

Figure 8: Tree Bark Texture: (clockwise from top left) original image, reconstructed from the LSE, MLE, and Compressed image. A model whose parameters are listed below was used.

Parameter	LSE	MLE
(1,0)	0.590927	0.568643
(0,1)	0.498257	0.497814
(1,1)	-0.281546	-0.272283
(-1,1)	-0.225011	-0.219671
(0,2)	-0.125950	-0.128427
(2,0)	-0.203024	-0.162452
(2,2)	-0.014322	-0.017466
(2,-2)	-0.002711	-0.007541
(3,0)	0.060477	0.034623

Parameter	LSE	MLE
(0,3)	0.024942	0.015561
(4,0)	-0.019122	-0.006186
(0,4)	-0.009040	-0.003748
(-2,1)	0.045105	0.036778
(1,-2)	0.031217	0.040860
(1,2)	0.061537	0.067912
(2,1)	0.067865	0.055445
ν	22205.84	65.45
$F(\Theta)$	-266147.34	-264245.13

Table 7: Θ Parameters for Tree Bark Texture

The parameters for the 256×256 image of tree bark texture in Figure 8 are given in Table 7.

B Equations used for the Computation of the Maximum Likelihood Estimate

$$\frac{\partial F}{d\theta_\imath} = -\sum_{\sigma\in\Omega}\left(\frac{\Phi_{\tau_\imath}(\sigma)}{1-2\sum_{\tau_\imath\in N_s}\left(\Theta_{\tau_\imath}\Phi_{\tau_\imath}(\sigma)\right)}\right) + \frac{1}{2\nu}\sum_{\sigma\in\Omega}\Big(y(\sigma)\big(y(\sigma+r_\imath)+y(\sigma-r_\imath)\big)\Big) \tag{31}$$

$$\frac{\partial F}{d\nu} = -\frac{M^2}{2\nu} + \frac{1}{2\nu^2}\sum_{\sigma\in\Omega}\Big(y(\sigma)^2 - y(\sigma)\sum_{\tau_\imath\in N_s}\big(\Theta_{\tau_\imath}\left(y(\sigma+r_\imath)+y(\sigma-r_\imath)\right)\big)\Big) \tag{32}$$

$$\frac{\partial^2 F}{d\theta_\imath\, d\theta_\jmath} = -2\sum_{\sigma\in\Omega}\left(\frac{\Phi_{\tau_\imath}(\sigma)\Phi_{\tau_\jmath}(\sigma)}{\left(1-2\sum_{\tau_\imath\in N_s}\left(\Theta_{\tau_\imath}\Phi_{\tau_\imath}(\sigma)\right)\right)^2}\right) \tag{33}$$

$$\frac{\partial^2 F}{d\theta_\imath\, d\nu} = -\frac{1}{2\nu^2}\sum_{\sigma\in\Omega}\Big(y(\sigma)\big(y(\sigma+r_\imath)+y(\sigma-r_\imath)\big)\Big) \tag{34}$$

$$\frac{\partial^2 F}{d\nu^2} = \frac{M^2}{2\nu^2} - \frac{1}{\nu^3}\sum_{\sigma\in\Omega}\Big(y(\sigma)^2 - y(\sigma)\sum_{\tau_\imath\in N_s}\big(\Theta_{\tau_\imath}\left(y(\sigma+r_\imath)+y(\sigma-r_\imath)\right)\big)\Big) \tag{35}$$

For an initial value for ν, we use the value for which $\frac{\partial F}{d\nu} = 0$. Thus,

$$\nu = \frac{1}{M^2}\sum_{\sigma\in\Omega}\Big(y(\sigma)^2 - y(\sigma)\sum_{\tau_\imath\in N_s}\big(\Theta_{\tau_\imath}\left(y(\sigma+r_\imath)+y(\sigma-r_\imath)\right)\big)\Big). \tag{36}$$

C Timings for parallel image processing techniques

C.1 Tables for CM-5 with 4th order model

Image	*LSE*		*MLE*		*Max Quant.*		*Reconstruction*	
	CM	Real	CM	Real	CM	Real	CM	Real
Grass	0.18871	0.21552	0.64907	0.68889	0.15110	0.15649	0.15320	0.15889
Tree Bark	0.18902	0.21701	0.77848	0.79900	0.15102	0.15676	0.15298	0.15826
Hay	0.18919	0.21593	0.78369	0.79954	0.15120	0.15681	0.15245	0.15806
Material Weave	0.18977	0.21694	0.44053	0.46859	0.15148	0.15703	0.15311	0.15915
Wool	0.18888	0.21550	0.35270	0.37245	0.15172	0.15734	0.15252	0.15804
Calf Leather	0.18942	0.21629	0.52948	0.56461	0.15102	0.15675	0.15359	0.15906
Sand	0.18965	0.21640	0.53487	0.56807	0.15122	0.15689	0.15292	0.15846
Water	0.18878	0.21687	0.74409	0.75874	0.15075	0.15608	0.15262	0.15834
Wood	0.18888	0.21597	1.12768	1.18931	0.15112	0.15639	0.15332	0.15874
Raffia	0.18905	0.21600	0.66650	0.70604	0.15257	0.15810	0.15331	0.15884
Pigskin	0.19014	0.21731	0.53278	0.56823	0.15284	0.15840	0.15493	0.16046
Brick	0.18942	0.21598	0.50737	0.54052	0.15355	0.15905	0.15537	0.16080
Plastic Bubbles	0.19000	0.21671	0.78290	0.80600	0.15407	0.15934	0.15566	0.16356

Table 8: Timings for 32 node CM-5 with vector units and a 256×256 image using 4th order model (in seconds)

Image	*LSE*		*MLE*		*Max Quant.*		*Reconstruction*	
	CM	Real	CM	Real	CM	Real	CM	Real
Grass	0.35025	0.40559	1.09579	1.73474	0.42910	0.60252	0.41336	0.50837
Tree Bark	0.35098	0.41215	2.12047	3.09966	0.38376	0.58190	0.47184	0.57506
Hay	0.35028	0.48535	1.80150	2.90682	0.32341	0.51846	0.45188	0.53853
Material Weave	0.35056	0.39201	0.79223	1.49892	0.36427	0.46717	0.47648	0.58006
Wool	0.35086	0.58025	0.59332	1.18285	0.39188	0.42692	0.50216	0.60554
Calf Leather	0.35067	0.43579	1.06463	1.81233	0.30748	0.48003	0.29210	0.53882
Sand	0.35042	0.47085	1.13418	1.97015	0.42911	0.57422	0.52275	0.52931
Water	0.35074	0.44033	1.90515	2.81341	0.42568	0.60710	0.32805	0.47829
Wood	0.35024	0.42933	2.27675	3.58605	0.44118	0.53959	0.40582	0.50857
Raffia	0.35034	0.67003	1.22650	2.29592	0.45805	0.58671	0.39621	0.49888
Pigskin	0.35502	0.58772	1.09873	1.76496	0.33998	0.44263	0.36714	0.53695
Brick	0.35868	0.65987	1.15683	1.46099	0.85711	0.99022	0.96830	1.31532
Plastic Bubbles	0.49915	0.78976	1.74521	2.50664	1.67132	1.84861	0.72001	0.99533

Table 9: Timings for 16 node CM-5 with vector units and a 256×256 image using 4th order model (in seconds)

Image	*LSE*		*MLE*		*Max Quant.*		*Reconstruction*	
	CM	Real	CM	Real	CM	Real	CM	Real
Grass	0.66347	0.70880	2.81060	2.85450	0.53827	0.57128	0.54347	0.58556
Tree Bark	0.66375	0.70965	2.76704	2.80584	0.54770	0.59848	0.59314	0.66343
Hay	0.66224	0.70761	2.55092	2.58401	0.53867	0.58000	0.54400	0.58604
Material Weave	0.66283	0.70950	1.48790	1.53534	0.53892	0.58143	0.54315	0.58493
Wool	0.66304	0.70949	2.05468	2.11505	0.55202	0.67596	0.55666	0.59950
Calf Leather	0.66837	0.71377	1.98576	2.03600	0.55320	0.59496	0.55737	0.59935
Sand	0.67016	0.71544	2.05744	2.10879	0.55260	0.58614	0.55671	0.59901
Water	0.66997	0.71546	2.91214	2.95684	0.55021	0.59293	0.55578	0.59912
Wood	0.67039	0.71839	2.93610	2.98055	0.55189	0.58427	0.55838	0.59988
Raffia	0.66970	0.71477	2.95711	2.98426	0.55262	0.58515	0.55707	0.59886
Pigskin	0.66819	0.71437	2.93399	2.97085	0.55329	0.58616	0.55650	0.59834
Brick	0.66917	0.71468	1.82636	1.87932	0.55172	0.59385	0.55780	0.60021
Plastic Bubbles	0.66994	0.72098	2.80696	2.85616	0.53656	0.56927	0.54462	0.58690

Table 10: Timings for 32 node CM-5 with vector units and a 512×512 image using 4th order model (in seconds)

Image	*LSE*		*MLE*		*Max Quant.*		*Reconstruction*	
	CM	Real	CM	Real	CM	Real	CM	Real
Grass	1.36603	1.90141	6.64328	11.46509	1.03455	1.89646	1.28558	1.94048
Tree Bark	1.26896	1.66841	6.60849	11.06427	1.15230	1.88849	1.32597	1.94607
Hay	1.28100	1.89287	6.02011	10.46863	1.03698	1.93140	1.14502	1.80366
Material Weave	1.29521	1.80298	2.74713	6.20137	1.11419	1.82659	1.04486	1.92548
Wool	1.30032	1.78571	3.90378	8.48179	1.04720	1.76455	1.31172	1.94375
Calf Leather	1.26903	1.73373	3.72970	8.43098	1.20791	1.87151	1.24595	2.03756
Sand	1.40796	1.93989	3.83804	8.78400	1.20834	1.85583	1.16566	1.90384
Water	1.33629	1.79228	6.74821	11.16951	1.25288	2.05764	1.21235	1.75456
Wood	1.46343	2.19338	6.55124	11.45757	1.20506	1.92589	1.08904	1.82530
Raffia	1.27830	1.66010	6.46234	11.43468	1.20810	1.85213	1.32857	1.76140
Pigskin	1.27817	1.80461	6.77205	11.45332	1.09704	1.80778	1.04815	2.00044
Brick	1.27798	1.77087	3.83952	7.91782	1.16428	2.18315	1.14691	1.83908
Plastic Bubbles	1.27235	1.66958	6.69361	11.55942	1.14468	1.85555	1.08797	1.90316

Table 11: Timings for 16 node CM-5 with vector units and a 512×512 image using 4th order model (in seconds)

C.2 Tables for CM-5 with higher order model

Image	*LSE*		*MLE*		*Max Quant.*		*Reconstruction*	
	CM	Real	CM	Real	CM	Real	CM	Real
Grass	0.40322	0.44213	0.94195	1.00667	0.17641	0.18229	0.17872	0.19094
Tree Bark	0.40361	0.44326	1.22845	1.31758	0.17751	0.18381	0.18091	0.18858
Hay	0.40472	0.44332	1.92117	2.04819	0.17725	0.18302	0.17958	0.19137
Material Weave	0.40492	0.44825	1.12377	1.20635	0.17674	0.18247	0.17868	0.19115
Wool	0.40493	0.44367	0.75545	0.80905	0.17639	0.18282	0.17899	0.19111
Calf Leather	0.40621	0.44538	0.93570	1.00897	0.17675	0.18344	0.17901	0.18472
Sand	0.40669	0.45053	0.94703	1.00784	0.17720	0.18295	0.18141	0.18772
Water	0.40630	0.44502	1.35860	1.45406	0.17695	0.18348	0.17911	0.19145
Wood	0.40698	0.44554	1.11477	1.15080	0.17640	0.18218	0.18030	0.18918
Raffia	0.40710	0.45160	1.16616	1.24794	0.17733	0.18322	0.18057	0.18744
Pigskin	0.40718	0.44659	0.77520	0.82944	0.17683	0.18264	0.18502	0.19097
Brick	0.40558	0.44437	1.22903	1.31398	0.17247	0.17809	0.17442	0.18630
Plastic Bubbles	0.40208	0.44171	0.96939	1.04141	0.17213	0.17792	0.17583	0.18346

Table 12: Timings for 32 node CM-5 with vector units and a 256×256 image using higher order model (in seconds)

Image	*LSE*		*MLE*		*Max Quant.*		*Reconstruction*	
	CM	Real	CM	Real	CM	Real	CM	Real
Grass	0.75510	0.94495	1.64171	2.95562	0.49920	0.69683	0.43896	0.62601
Tree Bark	0.75529	0.91401	2.43371	4.09217	0.45063	0.73213	0.39573	0.56178
Hay	0.75485	0.83668	3.54220	6.08994	0.42932	0.66296	0.54377	0.64749
Material Weave	0.75359	0.83831	1.87242	3.11225	0.37618	0.47948	0.52253	0.60530
Wool	0.75227	0.92176	1.33899	2.53465	0.43754	0.60825	0.43503	0.63146
Calf Leather	0.75169	0.82723	1.87598	3.45899	0.38286	0.57002	0.50191	0.60552
Sand	0.75243	0.82902	1.53868	3.10369	0.37218	0.88758	0.53837	0.64540
Water	0.75445	1.01632	2.57627	4.19290	0.40781	0.59671	0.52076	0.61641
Wood	0.75831	0.91263	2.46256	4.07550	0.49943	0.60110	0.46189	0.66130
Raffia	0.75254	0.92399	1.95853	3.80243	0.36174	0.55886	0.69533	0.70182
Pigskin	0.75442	0.94364	1.26406	2.47400	0.52605	0.62853	0.50666	0.61040
Brick	0.75446	0.94785	2.59574	4.15548	0.46582	0.60754	0.50111	0.60480
Plastic Bubbles	0.75327	0.88082	1.80579	3.51673	0.41855	0.51143	0.50508	0.70250

Table 13: Timings for 16 node CM-5 with vector units and a 256×256 image using higher order model (in seconds)

Image	*LSE*		*MLE*		*Max Quant.*		*Reconstruction*	
	CM	Real	CM	Real	CM	Real	CM	Real
Grass	1.43952	1.51679	3.24504	3.35994	0.61829	0.67558	0.62485	0.69582
Tree Bark	1.45144	1.53228	3.68973	3.83764	0.63826	0.70764	0.64226	0.69931
Hay	1.44959	1.52739	4.84103	5.06278	0.63773	0.70791	0.64222	0.71177
Material Weave	1.44452	1.52227	2.78452	2.88677	0.63801	0.70782	0.64310	0.71258
Wool	1.44829	1.52647	3.98331	4.12007	0.63837	0.70870	0.64326	0.71323
Calf Leather	1.44067	1.51911	3.32404	3.47150	0.63729	0.69418	0.64260	0.71259
Sand	1.44085	1.52947	2.84737	2.96683	0.63784	0.80315	0.64293	0.71302
Water	1.44850	1.52666	4.29537	4.38623	0.63576	0.69311	0.64040	0.69771
Wood	1.48000	1.61215	4.68701	5.22864	0.64783	0.75573	0.64589	0.76560
Raffia	1.44559	1.55839	4.29394	4.38290	0.63736	0.69405	0.64188	0.69950
Pigskin	1.44618	1.53312	4.25180	4.33038	0.63918	0.70899	0.64235	0.71475
Brick	1.46964	1.69157	4.77538	5.42371	0.63805	0.72941	0.64650	0.74760
Plastic Bubbles	1.44442	1.52329	5.07613	5.29358	0.63747	0.70774	0.64290	0.71350

Table 14: Timings for 32 node CM-5 with vector units and a 512×512 image using higher order model (in seconds)

Image	*LSE*		*MLE*		*Max Quant.*		*Reconstruction*	
	CM	Real	CM	Real	CM	Real	CM	Real
Grass	3.12318	4.72546	6.99671	17.82032	1.49828	2.40623	1.31636	2.26554
Tree Bark	3.32219	4.63255	7.29536	17.69756	1.20850	2.28685	1.21490	2.23169
Hay	2.96341	4.49642	9.58966	23.73069	1.33654	2.23659	1.32477	2.25426
Material Weave	2.98736	4.79122	5.38949	13.77429	1.37614	2.37668	1.21506	2.20862
Wool	3.01502	4.70060	7.63265	19.54811	1.29214	2.39451	1.30015	2.37301
Calf Leather	3.01284	4.85013	7.68454	18.58360	1.19652	2.20188	1.20436	2.05411
Sand	3.00917	4.64956	5.31515	13.95840	1.29127	2.27837	1.41723	2.45459
Water	3.15670	4.72244	9.62219	17.74029	1.30545	2.20439	1.56962	2.38058
Wood	3.30382	4.83820	10.27261	17.35976	1.40333	2.12686	1.32172	2.32923
Raffia	3.17638	4.61360	10.02692	17.54273	1.37101	2.40834	1.21189	2.59668
Pigskin	3.14083	4.68527	11.85554	17.45039	1.22261	2.34884	1.33543	2.23909
Brick	3.26359	4.63380	9.81605	16.98878	1.24787	2.20357	1.49057	2.37116
Plastic Bubbles	2.94002	4.73853	10.88062	26.16783	1.39433	2.32359	1.24875	2.18659

Table 15: Timings for 16 node CM-5 with vector units and a 512×512 image using higher order model (in seconds)

References

[1] S. G. Akl. *The Design and Analysis of Parallel Algorithms.* Prentice-Hall, Englewood Cliffs, NJ, 1989.

[2] D. A. Bader, J. JáJá, and R. Chellappa. Scalable Data Parallel Algorithms for Texture Synthesis and Compression Using Gibbs Random Fields. Technical Report CS-TR-3123 and UMIACS-TR-93-80, UMIACS and Electrical Engineering, University of Maryland, College Park, MD, August 1993.

[3] J. E. Besag and P. A. P. Moran. On the Estimation and Testing of Spacial Interaction in Gaussian Lattice Processes. *Biometrika*, 62:555–562, 1975.

[4] G. E. Blelloch. $C^\star$ data layout for the CM-2. Personal Communications, August 17, 1993.

[5] G. E. Blelloch, C. E. Leiserson, B. M. Maggs, C. G. Plaxton, S. J. Smith, and M. Zagha. A Comparison of Sorting Algorithms for the Connection Machine CM-2. In *Proceedings of the ACM Symposium on Parallel Algorithms and Architectures*, pages 3–16, July 1991.

[6] R. Chellappa. Two-Dimensional Discrete Gaussian Markov Random Field Models for Image Processing. In L. N. Kanal and A. Rosenfeld, editors, *Progress in Pattern Recognition*, volume 2, pages 79–112. Elsevier Science Publishers B. V., 1985.

[7] R. Chellappa and S. Chatterjee. Classification of Textures Using Gaussian Markov Random Fields. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 33:959–963, August 1985.

[8] R. Chellappa, S. Chatterjee, and R. Bagdazian. Texture Synthesis and Compression Using Gaussian-Markov Random Field Models. *IEEE Transactions on Systems, Man, and Cybernetics*, 15:298–303, March 1985.

[9] R. Chellappa, Y. H. Hu, and S. Y. Kung. On Two-Dimensional Markov Spectral Estimation. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, ASSP-31:836–841, August 1983.

[10] R. Chellappa and R. L. Kashyap. Synthetic Generation and Estimation in Random Field Models of Images. In *IEEE Comp. Soc. Comf. on Pattern Recog. and Image Processing*, pages 577–582, Dallas, TX, August 1981.

[11] R. Chellappa and R. L. Kashyap. Texture Synthesis Using 2-D Noncausal Autoregressive Models. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, ASSP-33:194–203, February 1985.

[12] F. S. Cohen. Markov Random Fields for Image Modelling & Analysis. In U. Desai, editor, *Modelling and Applications of Stochastic Processes*, chapter 10, pages 243–272. Kluwer Academic Press, Boston, MA, 1986.

[13] F. S. Cohen and D. B. Cooper. Simple Parallel hierarchical and relaxation algorithms for segmenting noncausal Markovian fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-9:195–219, March 1987.

[14] G. R. Cross and A. K. Jain. Markov Random Field Texture Models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-5:25–39, January 1983.

[15] H. Derin. The Use of Gibbs Distributions In Image Processing. In Blake and H. V. Poor, editors, *Communications and Networks*, chapter 11, pages 266–298. Springer-Verlag, New York, 1986.

[16] H. Derin and H. Elliott. Modeling and segmentation of noisy and textured images using Gibbs random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-9:39–55, January 1987.

[17] R. C. Dubes and A. K. Jain. Random Field Models in Image Analysis. *Journal of Applied Statistics*, 16:131–164, 1989.

[18] S. Geman and D. Geman. Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6:721–741, November 1984.

[19] R. I. Greenberg and C. E. Leiserson. Randomized Routing on Fat-Trees. *Advances in Computing Research*, 5:345–374, 1989.

[20] J. JáJá. *An Introduction to Parallel Algorithms.* Addison-Wesley Publishing Company, New York, 1992.

[21] F. C. Jeng, J. W. Woods, and S. Rastogi. Compound Gauss-Markov Random Fields for Parallel Image Processing. In R. Chellappa and A. K. Jain, editors, *Markov Random Fields: Theory and Application*, chapter 2, pages 11–38. Academic Press, Boston, MA, 1993. Bell Communications Research and ECSE Department, Renssalaer Polytechnic Institute.

[22] S. L. Johnsson, M. Jacquemin, and R. L. Krawitz. Communications Efficient Multi-Processor FFT. *Journal of Computational Physics*, 102:381–397, 1992.

[23] S. L. Johnsson and R. L. Krawitz. Cooley - Tukey FFT on the Connection Machine. *Parallel Computing*, 18:1201–1221, 1992.

[24] R. L. Kashyap. Univariate and Multivariate Random Field Models for Images. *Computer Graphics and Image Processing*, 12:257–270, 1980.

[25] R. L. Kashyap and R. Chellappa. Estimation and Choice of Neighbors in Spacial Interaction Models of Images. *IEEE Transactions on Information Theory*, IT-29:60–72, January 1983.

[26] H. Künsch. Thermodynamics and Statistical Analysis of Gaussian Random Fields. *Zeitschrift für Wahrscheinlichkeitstheorie verwandte Gebiete*, 58:407–421, November 1981.

[27] O. J. Kwon and R. Chellappa. Segmentation-based image compression. *Optical Engineering*, 32:1581–1587, July 1993. (Invited Paper).

[28] C. E. Leiserson. Fat-Trees: Universal Networks for Hardware-Efficient Supercomputing. *IEEE Transactions on Computers*, C-34:892–901, October 1985.

[29] C. E. Leiserson, Z. S. Abuhamdeh, D. C. Douglas, C. R. Feynman, M. N. Ganmukhi, J. V. Hill, W. D. Hillis, B. C. Kuszmaul, M. A. St. Pierre, D. S. Wells, M. C. Wong, S. W. Yang, and R. Zak. The Network Architecture of the Connection Machine CM-5. (Extended Abstract), July 28, 1992.

[30] M. Lin, R. Tsang, D. H. C. Du, A. E. Klietz, and S. Saroff. Performance Evaluation of the CM-5 Interconnection Network. Technical Report AHPCRC Preprint 92-111, University of Minnesota AHPCRC, October 1992.

[31] F. A. Lootsma and K. M. Ragsdell. State-of-the-art in Parallel Nonlinear Optimization. *Parallel Computing*, 6:133–155, 1988.

[32] B. S. Manjunath, T. Simchony, and R. Chellappa. Stochastic and Deterministic Networks for Texture Segmentation. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, ASSP-38:1039–1049, June 1990.

[33] J. Max. Quantizing for Minimum Distortion. *IRE Transactions on Information Theory*, IT-16:7–12, March 1960.

[34] J. Palmer and G. L. Steele Jr. Connection Machine Model CM-5 System Overview. In *The Fourth Symposium on the Frontiers of Massively Parallel Computation*, pages 474–483, Los Alamitos, CA, October 1992. IEEE Computer Society Press.

[35] T. Poggio and D. Weinschall. The MIT Vision Machine: Progress in the Integration of Vision Models. In R. Chellappa and A. K. Jain, editors, *Markov Random Fields: Theory and Application*. Academic Press, Boston, MA, 1993.

[36] B. T. Polyak. *Introduction to Optimization*. Optimization Software, Inc., New York, 1987.

[37] T. Simchony, R. Chellappa, and Z. Lichtenstein. Relaxation Algorithms for MAP Estimation of Gray-Level Images with Multiplicative Noise. *IEEE Transactions on Information Theory*, IT-36:608–613, May 1990.

[38] Thinking Machines Corporation, Cambridge, MA. *C* Programming Guide*, Version 6.0.2 edition, June 1991.

[39] Thinking Machines Corporation, Cambridge, MA. *Paris Reference Manual*, Version 6.0 edition, June 1991.

[40] Thinking Machines Corporation, Cambridge, MA. *The Connection Machine CM-5 Technical Summary*, January 1992.

[41] Thinking Machines Corporation, Cambridge, MA. *CMMD Reference Manual*, 3.0 edition, May 1993.

[42] Thinking Machines Corporation, Cambridge, MA. *CMSSL for CM Fortran*, CM-5 Edition, Version 3.1 edition, June 1993.

[43] Thinking Machines Corporation, Cambridge, MA. *$C^\star$ Programming Guide*, May 1993.

[44] R. Whaley. $C^\star$ data layout for the CM-2. Personal Communications, August 17, 1993.

[45] R. Whaley. $C^\star$ data layout for the CM-5. Personal Communications, June 8, 1993.

[46] J. W. Woods. Two-Dimensional Discrete Markovian Random Fields. *IEEE Transactions on Information Theory*, IT-18:232–240, March 1972.