Parallel Algorithms For Color Image Quantization on Hypercubes and Meshes

Extended Abstract

Frank Dehne*
School of Computer Science
Carleton University
Ottawa, Canada K1S 5B6

Andrew Rau-Chaplin*
School of Computer Science
Carleton University
Ottawa, Canada K1S 5B6

We study parallel algorithms for colour image quantization based on the k-mean clustering method. We obtain $O(T \Delta \log N)$, and $O(T \sqrt{N})$ expected time algorithms for hypercubes and meshes respectively, *where* N the size of the original image, K is the number of colors in the new quantized image $K << N$, $\Delta = min\{\log K , \log^2 \log N\}$, and T the number of iterations of the k-mean algorithm. The best sequential algorithm for colour quantization based on k-mean clustering requires O(T logK N) expected time.

1 INTRODUCTION

A fundamental problem in colour image processing is the need to quantize the RGB (Red, Green, Blue) colour space relative to an image. The aim of colour quantization is to produce from a colour image Y a new colour image Y' which "approximates" Y, but uses far fewer colours. Note that the new image Y' is far smaller than the original image and therefore can be manipulated with a reduced computational cost and can be rendered by a simpler rendering device.

The colour of each pixel of an image produced by a colour camera is typically determined by three bytes: a red byte, a blue byte and a green byte. The RGB colour space of such an image therefore contains a total of 2^{24} or $\approx$17 million different colours. For most applications, such a vast colour space is not required, in fact the average colour image of a natural scene contains less than 10,000 different colours. Furthermore, it has be shown that in many cases a quantized image containing as few as 64 or 128 carefully chosen colours is scarcely distinguishable from an original image rendered a 2^{24} RGB colour space [21].

In this paper we address the problem of colour image quantization in the parallel domain. We will develop parallel algorithms for colour quantization on fine-grained SIMD hypercubes and meshes. This problem is of particular interest on these models as more and more image processing systems move from special purpose hardware to these general purpose architectures. Consider, for example, the Connection Machine and its associated high-speed graphics system. This system, based on a fast frame-buffer, makes the Connection Machine potentially very useful in high-speed image processing applications, if efficient algorithms for fundamental image processing operations, such as colour quantization, can be developed for it.

Colour quantization can be viewed as a multidimensional clustering problem (see [11,21,22]). The problem is to create a small number of groups or clusters of data points (pixels) in multidimensional space (RGB colour space), such that some criterion function is minimized. For colour quantization the criterion function most commonly used in clustering data points is the sum-of-squared-errors measure. This well known criterion function produces

F. Dehne and A. Rau-Chaplin, "Parallel Algorithms for color image quantization on hypercubes and meshes," in Proc. Workshop on Algorithms and Parallel VLSI Architectures II, Bonas (France), 1991, North Holland, pp. 91-96.

hyperellipsoidal clusters that have been found, through experimentation, to give the best results when evaluated visually. In this paper we also follow this approach and develop a parallel clustering algorithm based on the well known k-mean clustering method.

We now state our problem more formally. A colour image Y consisting of N pixels, where each pixel is represented by a record storing the coordinates of the pixel (in x and y) and the colour of the pixel in RGB space. Consider the "colours" of the N pixels to be points in 3-dimensional vector space Ω. We will refer to them as $S = s_1, s_2, \ldots, s_N$. The objective of colour image quantization is to find $K << N$ colours $C = c_1, c_2, \ldots, c_K$ such that the average sum-of-squared-errors defined by $E = (1/N) \Sigma_{1 \le i \le N} |s_i - R(s_i)|^2$ is minimized, where $R(s_i) \in \{c_1, c_2, \ldots, c_K\} \subseteq \Omega$ is the colour closest to s_i in RGB space, namely, $|s_i - R(s_i)|^2 = \min_{1 \le i \le N} |s_i - c_j|^2$. For simplicity of exposition we will refer to $S = \{s_1, s_2, \ldots, s_N\}$ as our input data points and $C = \{c_1, c_2, \ldots, c_K\}$ as our cluster centers. The quantized image Y' is formed from the original image Y by substituting for the vector (colour) s_i of each pixel the value of $R(s_i)$.

The problem of finding global minimum solutions for the equations given above is known to be NP-Complete [13]. Therefore efficient algorithms for colour quantization must rely on heuristic clustering methods. There are several approaches to heuristic multidimensional clustering (see [14]). These approaches can be broadly classified as either divisive, agglomerative or iterative. In the sequential domain, colour quantization has typically been addressed using divisive clustering methods (see [11,21,22]), since algorithms realizing these methods tend to be sequentially efficient and produce the required hyperellipsoidal shaped clusters. The best of the sequential (divisive) algorithms for colour image quantization requires O(KN) time. Although the divisive approach tends to be efficient, it has one major drawback, namely that the resulting clusters are farther from optimal than those produced by iterative clustering methods [21].

Our approach to parallel colour quantization will therefore be to design parallel clustering algorithms based on iterative methods. We will base our parallel algorithms on the well studied k-mean clustering method [15] which has been used as the core of such sequential systems as FORGY [10] and ISODATA [2]. The motivation for this approach is twofold. Firstly, as stated earlier, the iterative (k-mean) clustering methods produces better clustering results for colour quantization than divisive methods [21]. Secondly, as we will demonstrate, k-mean iterative clustering has an efficient implementation on fine-grained hypercubes and meshes.

Several other researchers have addressed parallel clustering algorithms. This previous work has tended to be either for special purpose architectures [20] or for systems of coarse-grained multiprocessors [4], and are as such quite different from the work reported on in this paper.

The next section establishes some definitions and recalls some useful previous results. In Sections 3 and 4 we describe an efficent implementation of parallel colour image quantization on hypercubes and meshes, respectively.

2 PRELIMINARIES

2.1. k-d Trees and Nearest Neighbour Search

A (homogeneous) k-d tree [3] is a binary tree in which each node stores a record. Each record contains k keys, some data fields, right and left son pointers and a discriminator (an integer between 1 and k). The defining property of a k-d tree is that for any node x which is a j-discriminator, all nodes in the left subtree of x have k_j values less than x's k_j value, and likewise all nodes in the right subtree of x have k_j values greater than x's k_j value. For balanced k-d trees where the discriminators are chosen cyclically, one for each level of the tree, it has been shown that nearest neighbour search in k-dimensional space can be completed in expected time O(log n) [9].

2.2. MultiSearch on Trees for Hypercubes and Meshes

In the following we will make extensive use of an algorithm called *MultiSearch* [1,7,8] : Given a tree stored on a hypercube or mesh, m search queries on that tree are to be executed

independently and in parallel. At each time step, each query currently visiting a node of the tree decides which adjacent node to visit next, and is then moved to that node. Note that, each node can be concurrently visited by an arbitrary number of queries.

In [7,8] it was shown that the MultiSearch problem can be solved on a hypercube multiprocessor of size O(n) in time O(S + R log n), where S is the time required to sort n items on a hypercube (currently O(log n $\log^2$log n)) and R is the length of the longest search path associated with a search process. Note that for cases where R = O(log n), as in most data structures, O(n) search queries are answered in O($\log^2$ n) time.

In [1] it was shown that the MultiSearch problem can be solved on a mesh connected computer of size O(n) in time O($\sqrt{n}$ + R ($\sqrt{n}$ / log n)), where R is the length of the longest search path associated with a search process. Note that for cases where R = O(log n), as in most data structures, O(n) search queries are answered in optimal O($\sqrt{n}$) time.

2.3. Basic Operations on Hypercubes and Meshes

In addition to the MultiSearch procedure described above we will make use of basic hypercube and mesh operations for sorting, merging, compressing, prefix sum, etc. On the hypercube, all the basic operations we use require O(log n) time (see [12,19]), except for sorting which requires O(log n $\log^2$log n) time [5]. On the mesh all of the basic operations which we use, including sorting, require O($\sqrt{n}$) time (see [16,17,19]).

3 COLOUR IMAGE QUANTIZATION ON A HYPERCUBE

In this section we describe a hypercube algorithm for parallel k-mean clustering in order to solve the parallel colour image quantization problem. It is based on the sequential k-mean method which was first proposed in [15] and then widely used in such systems as FORGY [10] and ISODATA [2]. However, in order to reduce the time complexity of our algorithm we will build not one, but rather N/K balanced k-d trees during each iteration of the basic clustering loop. We assume w.l.o.g. that the number of colours desired in the quantized image Y', is K = 2^j for some j.

Parallel Colour Image Quantization on a Hypercube:
Initial setup: Let S = $s_1,\ldots,s_N$ be the N vectors storing the colours of the N pixels of the original image Y, where s_i is stored on processor P_i.
1) Pick at random a set C = $\{c_1,\ldots,c_K\}$ of K points in RGB space and store them on processors 1..K.
2) Repeat
2a) Distribute to each N/K sub hypercube a copy of the K elements of C.
2b) Within each N/K sub hypercube make three copies of the local set C. Call them C^R, C^G, C^B and sort them by Red, Green, and Blue values, respectively. Using these sorted lists build a balanced k-d tree.
2c) Within each n/k sub hypercube use MultiSearch (see Section 2) to perform for each data point s_j (stored in that subcube) a nearest neighbour search on the local k-d tree. At the end of each search, associate with each s_j a copy of the member of C closest to it.
2d) Using the whole hypercube sort all data points $s_1,\ldots s_N$ by their associated value from C. Note that data points that share an associated C value are now stored in contiguous blocks (ranges) of processors.
2e) Within each such block calculate the centroid of the data points and broadcast this value to all processors in the block. These newly calculated centroids will form the new set C for the next iteration of the algorithm.
2f) Let each processor calculate the square of the Euclidean distance between the data point and centroid it stores. Then calculate the average sum-of-squared-Euclidean distances measure on the whole set of data points.
Until current and previous error measures are "equal" within distance epsilon.

3) Form a new image Y' by copying the image Y and replacing the colour value of each pixel by the associated centroid value.

In Step 1, K points in the RGB space of the image are picked at random to form the set C. This can be executed by the first k processors of the hypercube in O(1) time. In Step 2a, copies of the current set of cluster centers, C, are distributed to N/K sub hypercubes (using the broadcast operation) in O(log N) time. Step 2b requires that in each sub hypercube of size K, three lists are sorted. This can be done in O(log K $\log^2$log K) time [5]. These sorted lists are then used to build a balanced k-d tree in each sub hypercube where the discriminators are cyclically chosen, one per level. Within a sub hypercube, the balanced k-d tree is built level by level by the following recursive algorithm: Pick the median item out of the list sorted by the current level's discriminator, j, and make it the root of the tree. From each list C^x, form two new lists, C^{x-} and C^{x+}, such that C^{x-} contains those elements of C^x whose j-discriminators are less than the root's j-discriminator, and C^{x+} contains the other items. This can be accomplished by three concentrate operations [19], in O(log K) time. Recursively calculate the left and right sons of the root using the C^{x-} and C^{x+} lists, respectively, and the next discriminator. Step 2b requires a total of O($\log^2$ K) time since the process of building the k-d tree requires O(log K) recursive steps each requiring O(log K) time. Note that, the original sort performed in this step saves a O(log K) factor by eliminating the need to sort at each level of the recursion.

In Step 2c, the MultiSearch procedure (see Section 2) is used to perform a nearest neighbour search on the k-d trees created in Step 2b. The MultiSearch procedure requires time O(S + R log K), where S is the time required to sort K items on a hypercube (currently O(log K $\log^2$log K)) and R is the length of the longest search path associated with a search process. From [9] and [6] it follws that, given a k-d tree built as inStep 2b, the nearest neighbour search performed in Step 2c will require O($\log^2$ K) time, in the expected case.

In Step 2d, having calculated for each data point s_i the nearest cluster center, we now sort the entire data set by associated cluster centers. If log K $\leq$ $\log^2$log N we sort using a bitonic merge sort [17] requiring time O(log K log N), otherwise we use Share Sort [5] requiring time O(log N $\log^2$log N).

The data points s_i are now arranged in blocks where each block is defined as a group of contiguous processors that store those s_i's sharing a single associated cluster center from C. In Step 2e the centroid of the set of points in each block is calculated and in Step 2f these values are used to calculate the error measure. Both of these steps can be implemented by a constant number of partial sum and broadcast operations in O(log N) time. As a final step of each k-mean pass (Step 2) the set of cluster centers C is updated to be the centroids calculated in this iteration. In Step 3 the new image Y' is constructed from the original image Y by replacing the colours used by those in C.

Summarizing we obtain

Theorem 1: *The colour image quantization problem can be solved on a hypercube multiprocessor of size N in expected time O(T log N Δ), where Δ = min{log K , $\log^2$log N}, N the size of the original image, K is the number of colors in the new quantized image, and T the number of iterations of the parallel k-mean clustering algorithm.*

4 COLOUR IMAGE QUANTIZATION ON THE MESH

We now describe an algorithm for parallel colour image quantization on a mesh connected computer. The algorithm is very similar to the one for the hypercube multiprocessor, with the exception that only one k-d tree is built in each iteration.

Parallel Colour Image Quantization on a Mesh Connected Computer

Initial setup: Let $S = \{s_1,\ldots,s_N\}$ be the N vectors storing the colours of the N pixels of the original image Y, where $s_1,\ldots,s_N$ is stored in snake-like ordering on the mesh.

1) Pick at random a set $C = \{c1,\ldots,c_k\}$ of K points in RGB space and store them on the first K processors with respect to a snake-like ordering.

2) Repeat

2a) Build a balanced k-d tree out of the current cluster centers $c_1,\ldots,c_K$ on the first K processors.

2b) Use MultiSearch (see Section 2) to perform for each data point s_j a nearest neighbour search on the k-d tree constructed in the previous step. At the end of each search, associate with each s_j a copy of the member of C closest to it.

2c) Sort all data points $s_1,\ldots s_N$ by their associated value from C. Note that data points that share an associated C value are now stored in contiguous blocks (ranges) of processors in the snake-like order.

2d) Within each such block calculate the centroid of the data points and broadcast this value to all processors in the block. These newly calculated centroids will form the new set C for the next iteration of the algorithm.

2e) Let each processor calculate the square of the Euclidean distance between the data point and centroid it stores. Then calculate the average sum-of-squared-Euclidean distances measure on the entire set of data points.

Until current and previous error measures are "equal" within distance epsilon.

3) Form a new image Y' by copying the image Y and replacing the colour value of each pixel by the associated centroid value.

In Step 1, K points in the RGB space of the image are picked at random to form the set C. This can be achieved by the first k processors in O(1) time. In Step 2a, a balanced k-d tree is constructed out of the current cluster centers, where the discriminators are cyclically chosen, one per level. The balanced k-d tree is built level by level by the following recursive algorithm: Given a list of cluster centers, C', and a discriminator for this level, j, sort the list of cluster centers by the current discriminator j. Select the medium of the sorted list C' and make it the root of the tree. From the list C', form two new lists, C'^- and C'^+, such that C'^- contains those elements of C' with their j-discriminators less than the root's j-discriminator, and C'^+ contains the other items. This can be accomplished by a constant number of sort and concentrate operations [17] in $O(\sqrt{|C'|})$ time. Now recursively calculate the left and right sons of the root using the C'^- and C'^+ lists and the next discriminator. Solving the recurence relation for Step 2a we obtain a total $O(\sqrt{K})$ time.

In Step 2b, the MultiSearch procedure (see Section 2) is used to perform a nearest neighbour search on the k-d tree created in Step 2a. The MultiSearch procedure on the mesh requires time $O(\sqrt{N} + R(\sqrt{N} / \log N))$, where R is the length of the longest search path associated with a search processes. From [9] and [6] it follws that, given a k-d tree built in the manner of Step 2a, the nearest neighbour search performed in Step 2b will have, in the expected case an R = log N and therefore will require $O(\sqrt{N})$ (expected) time.

The remaining steps are implemented, in $O(\sqrt{N})$, analogously the hypercube algorithm.

Summarizing we obtain

Theorem 2: *The colour image quantization problem can be solved on a mesh connected computer of size N in expected time $O(T\sqrt{N})$, where N is the size of the original image, and T is the number of iterations of the parallel k-mean clustering algorithm.*

REFERENCES

[1] M.J. Atallah, F. Dehne, R. Miller, A. Rau-Chaplin, J. Tsay, MultiSearch Techniques For Implementing Data Structures on a Mesh Connected Computer, Dept. of Computer Science, Carleton Univ., Canada, SCS-TR-187, Feb. 1991.

[2] G. F. Ball, Data analysis in the Social Sciences: What about the details? AFIPS Fall Joint Comput. Conf., pp 533-560.

[3] J.L. Bently, Multidimensional bonary search trees used for associative searching , Comm. ACM, 18(9):509-517, 1975.

[4] M. Berry et. al, Parallel algorithms on the CEDAR system, Proc. CONPAR 86, Aachen, West Germany, Sept 17-19, 1986.

[5] R. Cypher and C. G. Plaxton, Deterministic sorting in nearly logarithmic time on a hypercube and related computers, Proc. ACM Symposium on Theory of Computing, 1990.

[6] F. Dehne, A. Ferreira, and A. Rau-Chaplin, "A massively parallel knowledge-base server using a hypercube multiprocessor," in Proc. *IEEE International Conference on Tools for Artificial Intelligence*, Washington, D.C., 1990, pp. 660-666.

[7] F. Dehne, A. Ferreira, and A. Rau-Chaplin, Parallel fractional cascading on a hypercube multiprocessor, Proc. Allerton Conf. on Communication, Control and Computing, Monticello, Ill., 1989.

[8] F.Dehne and A. Rau-Chaplin, Implementing data structures on a hypercube multiprocessor and applications in parallel computational geometry. Journal of Parallel and Distributed Computing, 8(4):367-375, 1990.

[9] Y.V.S. Filho, Optimal choice of discriminators in a balanced k-d binary search tree, Inf. Process. Lett., 13(2):67-70, November 19781.

[10] E.Forgy, Clustering analysis of multivariate data: Efficiency versus interpretability of classification. Biometrics, 21:768, 1965.

[11] P. Heckbert, Colour Image Quantization for frame buffer display, Comput. Gr., 16(3):297-307, July 1982.

[12] W. D. Hillis, The Connection Machine(Ed.), MIT Press, USA, 1985.

[13] L. Hyafil and R.L. Rivest, Construction of optimal binary decision trees is NP-complete, Inf. Process. Lett., 5:15-17, May 1976.

[14] A.K. Jain and R. C. Dubes, Algorithms for Clustering Data, Prentice Hall Advanced Reference series, USA, 1988.

[15] J.B. MacQueen, Some methods for classification and analysis of multivariate observations, Proc. of the 5th Berkley Symposium on Mathematical Statistics and Probability 1,281-297, 1967.

[16] R.Miller and Q.F. Stout, *Mesh computer algorithms for computational geometry.* IEEE Transactions on Computers, January 1989.

[17] R.Miller and Q.F. Stout, *Parallel Algorithms for Regular Architectures.* MIT Press, 1991.

[18] D.Nassimi and S.Sahni, *Data broadcasting in SIMD computers.* IEEE Transactions on Computers, C-30(2):101-107, February 1981.

[19] D.Nassimi and S.Sahni, *Bitonic sort on a mesh-connected parallel computer.* IEEE Transactions on Computers, C-27(1):2-7, January 1979.

[20] L.M. Ni, and A.K. Jain, *A VLSI systolic architecture for pattern clustering*, IEEE Trans. on Pattern Anal. and Mach. Intell. PAMI, 7:80-89, 1985.

[21] S.J. Wan, S.K.M. Wong, and P. Pursinkiewicz, *An algorithm for Multidimensional Data Clustering*, ACM Transcations on Mathematical Software 14(2):153-162, June 1988.

[22] X. Wu. and I. H. Witten, *A fast k-means type clustering algorithm*, Dept. of Computer Science, Univ. of Calgary, Canada, May 1985.