[16] B. Liu, *Theory and Practice of Uncertain Programming*. Heidelberg, Germany: Physica-Verlag, 2008.
[17] B. Liu, *Uncertainty Theory: An Introduction to its Axiomatic Foundations*. Berlin, Germany: Springer-Verlag, 2004.
[18] B. Liu, "A survey of credibility theory," *Fuzzy Optim. Decision Making*, vol. 5, pp. 387–408, 2006.
[19] B. Liu, "A survey of entropy of fuzzy variables," *J. Uncertain Syst.*, vol. 1, pp. 4–13, 2007.
[20] B. Liu and Y.-K. Liu, "Expected value of fuzzy variable and fuzzy expected value models," *IEEE Trans. Fuzzy Syst.*, vol. 10, no. 4, pp. 445–450, Aug. 2002.
[21] N. R. Pal and S. K. Pal, "Objective background segmentation using a new definition of entropy," *IEE Proc. E*, vol. 136, pp. 284–295, 1989.
[22] N. R. Pal and J. C. Bezdek, "Measuring fuzzy uncertainty," *IEEE Trans. Fuzzy Syst.*, vol. 2, no. 2, pp. 107–118, May 1994.
[23] M. R. Simonelli, "Indeterminacy in portfolio selection," *Eur. J. Oper. Res.*, vol. 163, pp. 170–176, 2005.
[24] R. R. Yager, "On measures of fuzziness and negotiation, Part I: Membership in the unit interval," *Int. J. General Syst.*, vol. 5, pp. 221–229, 1979.
[25] H. Markowitz, "Portfolio selection," *J. Finance*, vol. 7, pp. 77–91, 1952.
[26] H. Markowitz, *Portfolio Selection: Efficient Diversification of Investments*. New York: Wiley, 1959.
[27] D. N. Nawrocki and W. H. Harding, "State-value weighted entropy as a measure of investment risk," *Appl. Econ.*, vol. 18, pp. 411–419, 1986.
[28] M. A. Parra, A. B. Bilbao, and M. V. R. Uría, "A fuzzy goal programming approach to portfolio selection," *Eur. J. Oper. Res.*, vol. 133, pp. 287–297, 2001.
[29] G. C. Philippatos and C. J. Wilson, "Entropy, market risk, and the selection of efficient portfolios," *Appl. Econ.*, vol. 4, pp. 209–220, 1972.
[30] G. C. Philippatos and N. Gressis, "Conditions of equivalence among E–V, SSD, and E–H portfolio selection criteria: The case for uniform, normal and lognormal distributions," *Manag. Sci.*, vol. 21, pp. 617–625, 1975.
[31] C. E. Shannon, *The Mathematical Theory of Communication*. Urbana, IL: Univ. of Illinois Press, 1949.
[32] K. Smimou, C. R. Bector, and G. Jacoby, "A subjective assessment of approximate probabilities with a portfolio application," *Res. Int. Bus. Finance*, vol. 21, pp. 134–160, 2007.
[33] H. Tanaka, P. Guo, and B. Türksen, "Portfolio selection based on fuzzy probabilities and possibility distributions," *Fuzzy Sets Syst.*, vol. 111, pp. 387–397, 2000.
[34] J. Watada, "Fuzzy portfolio selection and its applications to decision making," *Tatra Mountains Math. Publication*, vol. 13, pp. 219–248, 1997.

Speedup of Fuzzy Clustering Through Stream Processing on Graphics Processing Units

Derek T. Anderson, Robert H. Luke, and James M. Keller

***Abstract*—As the number of data points, feature dimensionality, and number of centers for clustering algorithms increase, computational tractability becomes a problem. The fuzzy c-means has a large degree of inherent algorithmic parallelism that modern CPU architectures do not exploit. Many pattern recognition algorithms can be sped up on a graphics processing unit (GPU) as long as the majority of computation at various stages and the components are not dependent on each other. We present a generalized method for offloading fuzzy clustering to a GPU, while maintaining control over the number of data points, feature dimensionality, and the number of cluster centers. GPU-based clustering is a high-performance low-cost solution that frees up the CPU. Our results show a speed increase of over two orders of magnitude for particular clustering configurations and platforms.**

***Index Terms*—Fuzzy clustering, fuzzy c-means, graphics processing units (GPUs), stream processing.**

I. Introduction

Graphics processing units (GPUs) herald a new way to perform general purpose computing on hardware that is better suited for many image processing and pattern recognition algorithms. However, there is a GPU learning curve related to programming and setting up the environment in order to exploit its advantages. In addition, many pattern recognition algorithms are not designed in a parallel format conducive to GPU processing. There may be too many dependencies at various stages in the algorithm that will slow down GPU processing. In these cases, such as the parameter update step in fuzzy clustering, the algorithm must be altered in order to be computed fast on a GPU.

Improving the computational performance of clustering is not a new concept. Shankar and Pal presented a progressive subsampling method called fast fuzzy c-means (FFCM) [1]. FFCM generates a sequence of extended partitions of the entire dataset by applying the original FCM to a nested sequence of increasing size subsamples. It terminates when the difference between successive extended partitions is below a threshold. Speed is always a concern, but so is the size of the dataset. In [2], Pal and Bezdek developed the extensible FFCM (eFFCM) clustering algorithm for the segmentation of very large digital images. In [3], Hathaway and Bezdek discuss an extension of the eFFCM method, geFFCM, to nonimage object data.

It should be made clear that we are presenting a procedure to transfer fuzzy clustering to specialized hardware, which is different from many previous attempts to find algorithmic or mathematical reformulations. As improvements are made to the clustering algorithms, they can be converted to a GPU implementation for additional performance enhancement. Our research goal is to present a different computational platform for clustering.

Harris and Hanes conducted the only known previous research in the area of offloading clustering to a GPU in [4]. In their paper, a speedup of 1.7–2.1 times over a CPU is reported for a NVIDIA GeForceFX

Manuscript received March 21, 2007; revised May 31, 2007; accepted July 21, 2007. This work was supported by the National Library of Medicine under Grant T15 LM07089.

The authors are with the Department of Electrical and Computer Engineering, University of Missouri, Columbia, MO 65211 USA (e-mail: dtaxtd@mizzou.edu; rhl3db@mizzou.edu; kellerj@missouri.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TFUZZ.2008.924203

5900 Ultra. Their method is designed to handle three linearly separable clusters with a dimensionality of 3. The benefit is that they can handle a large number of data points. A problem is that the proposed formulation is not scalable. As the method is presented, it is not capable of extension with respect to either the feature dimensionality size or number of cluster centers. They acknowledge this and state that it would be a large undertaking. These problems are addressed in this short paper.

We focus on the FCM as an example, but many high-throughput problems, such as a protein sequence structure search, are excellent candidates for GPU enhancement. In the following sections, we: 1) present a brief overview of the FCM; 2) provide an introduction to GPUs; 3) discuss the generalized algorithm for computing fuzzy clustering on a GPU; 4) present the experiments; 5) display the results; and 6) discuss extensions to this research. Source code and a technical report that discusses implementation are available at http://cirl.missouri.edu/gpu/.

II. Clustering

Clustering is an unsupervised learning procedure that can be used to reveal patterns in a collection of data, denoted by $X = \{\vec{x}_1, \ldots, \vec{x}_N\}$. Each sample vector contains K features, represented as $\vec{x}_i = (f_{i1}, \ldots, f_{iK})$. Each cluster can be represented by a set of parameters θ_j $(1 \leq j \leq C)$. In the simplest case, θ_j is a K-dimensional vector representing the jth cluster center. In the standard approach [6], the clustering algorithm alternately estimates the collection of cluster centers, $\theta = \{\theta_1, \ldots, \theta_C\}$, and a membership matrix U, where the membership of the ith sample in the jth cluster is denoted by $u_{(i,j)}$. In the hard c-means (HCM) clustering algorithm, cluster membership values are crisp, i.e., $u_{(i,j)} \in \{0, 1\}$. Fuzzy clustering allows $u_{(i,j)} \in [0, 1]$, i.e., each element can be shared by more than one cluster. Fuzzy clustering follows the principle of least commitment, which is the belief that one should never overcommit or do something that might have to be later undone [5]. The problem, originally proposed by Bezdek [6], is defined as

$$J_{\text{FCM}}(\theta, U) = \sum_{j=1}^{C} \sum_{i=1}^{N} u_{(i,j)}^{q} d\left(\vec{x}_i, \theta_j\right)$$

$$\text{where} \sum_{j=1}^{C} u_{(i,j)} = 1, \quad \text{for } i = 1, \ldots, N$$

$$\vec{x}_i \in X \text{ and } |X| = N.$$

In the J_{FCM} equation, $d(\vec{x}_i, \theta_j)$ is a distance metric and $q > 1$ is a parameter called the fuzzifier, typically $q = 2$. The update equations found as necessary conditions to minimize J_{FCM} for the membership values $u_{(i,j)}^{\text{FCM}}$ and the cluster center locations are

$$u_{(i,j)}^{\text{FCM}} = \frac{1}{\sum_{k=1}^{C} \left(\frac{d(\vec{x}_i, \theta_j)}{d(\vec{x}_i, \theta_k)} \right)^{1/q-1}} \tag{1}$$

$$\theta_j = \frac{\sum_{i=1}^{N} \left(u_{(i,j)}^{q} \vec{x}_i \right)}{\sum_{i=1}^{N} u_{(i,j)}^{q}}. \tag{2}$$

III. Graphics Processing Units

Until about a decade ago, programmers had to rely on the CPU for mathematical transformations, rasterization, shading, and other graphics operations. There was a big need to offload these operations from a CPU to specialized hardware. GPUs were invented in order to generalize the graphics pipeline and the interface to it. GPUs are significant as they free up the CPU and offer high-end specialized computing at relatively low costs.

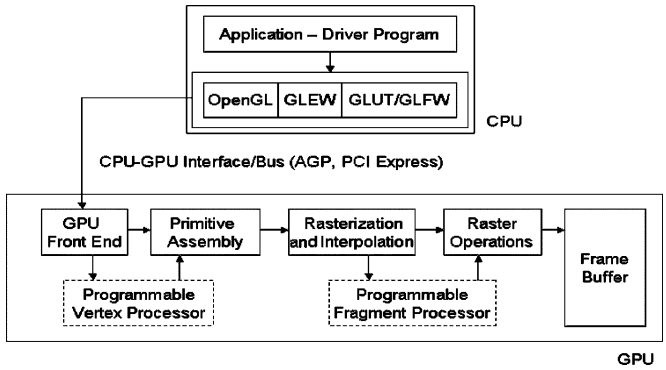

Fig. 1. Stages in the graphics pipeline with programmable GPU components noted by dashes.

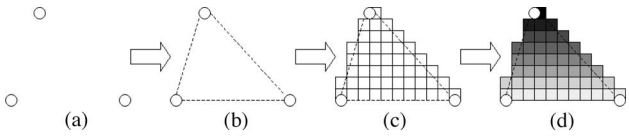

Fig. 2. Primitive rasterization and shading stages. (a) Phase 1 is the specification of vertices. (b) Phase 2 is primitive assembly. (c) Phase 3 is rasterization of the primitive into fragments. (d) Phase 4 is per fragment shading.

The key to understanding how to program a GPU is in the design of the host driver application, the graphics pipeline and GPU processor programming. Fig. 1 shows the stages in the graphics pipeline with programmable GPU components noted by dashed boxes, and Fig. 2 shows the standard operations in the graphics pipeline.

The process of rendering a triangle with respect to the standard graphics pipeline is shown in Fig. 2. Vertices are specified, the primitive is rasterized into fragments, and the fragments are finally shaded. The two current programmable components of a GPU are the vertex and fragment units. Vertex processors are intended for transformations on vertices, such as conversions between object space, world space, eye space, clip space, and texture coordinate generation. Fragment processors, also known as pixel shaders, typically support a greater degree of operational and texture sampling functionality. Because a few vertices are responsible for the generation of many fragments, the workload is typically on the backend, and there are usually more fragment processors than vertex processors. A good GPU introduction and architecture overview can be found in [7] and [8].

GPUs are specialized stream processors. Stream processors are capable of taking large batches of fragments that can be thought of as pixels in image processing applications, and computing similar independent calculations in parallel. The vendor and generation of the adapter determine the number of fragment processors. Each calculation is with respect to a program, often called a kernel, which is an operation applied to every fragment in the stream.

The computational power of GPUs is increasing significantly faster than CPUs. The annual growth in CPU processing power is approximately 1.4, while GPU development is approximately 1.7 for pixel processors and 2.3 for vertex processors [9]. GPUs have instructions to handle many linear algebra operations given their vector processing architecture. GPUs support operations such as the dot product, vector and matrix multiplication, and computing the determinant of a matrix.

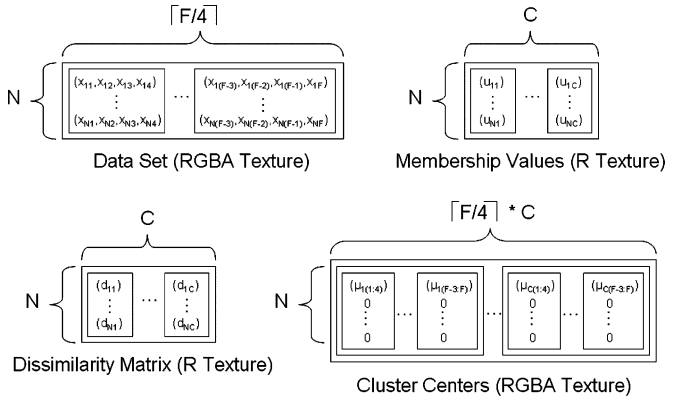

Fig. 3. Clustering parameters packed into texture memory. Texture formats are comprised of combinations of RGBA channels.

GPUs are also capable of executing more floating point operations per second. A 3.0-GHz dual-core Pentium 4 can execute 24.6 GFLOPS, while an NVIDIA GeForceFX 7800 can execute 165 GFLOPS [9]. The new NVIDIA GeForce 8800 GTX has 128 stream processors, a core clock of 575 MHz, a shader clock of 1350 MHz, and is capable of over 350 GFLOPS [10].

IV. Fuzzy Clustering on a GPU

The key concept required for converting a CPU program to a GPU program involves the idea that arrays are equivalent to textures. Data used in a GPU program are passed from the CPU as a texture. Fig. 3 is our proposed texture packing scheme.

The dataset is packed into a texture that holds red, green, blue, and alpha data (RGBA). There are N rows, one for each data point, and $\lceil F/4 \rceil$ columns, where $\lceil * \rceil$ represents the "ceiling" of $*$, i.e., the smallest integer greater than or equal to $*$. F represents the feature dimensionality, which is divided by four because four values can be packed into each pixel. The latest NVIDIA 8800 adapter supports textures of size 8192 $\times$ 8192. A method to overcome the row size limitation, allowing for over a million data points, is discussed later after each packing format is described.

The membership values are scalars and can be encoded into a single-channel texture. Again, only up to 8192 data points can be packed into a single texture, given the texture row size limitation. There are C columns, where C represents the number of clusters, limiting the size to at most 8192 clusters. Again, we discuss a method later to bypass these texture size-related limitations.

In order to avoid computing the same dissimilarity values repeatedly at multiple stages in the fuzzy clustering algorithm, all values are precomputed for each FCM iteration. Just like the membership values texture, we pack these scalar values into a single-channel image that is N rows $\times$ C columns.

The last representation is the cluster centers. The cluster centers share the same feature dimensionality as the data points, so this impacts the column size in a similar fashion. However, instead of packing the vectors in a row format, we pack them in a column format. The number of columns is $\lceil F/4 \rceil \times C$. This way, all cluster centers can be encoded in one row of the texture. The number of rows for this texture is, however, set equal to the number of input vectors. All rows after the first are initialized to zero (see Fig. 3). This is explained shortly when we show the reduction procedure for cluster parameter updating. The six-pass FCM GPU algorithm is shown in Fig. 4.

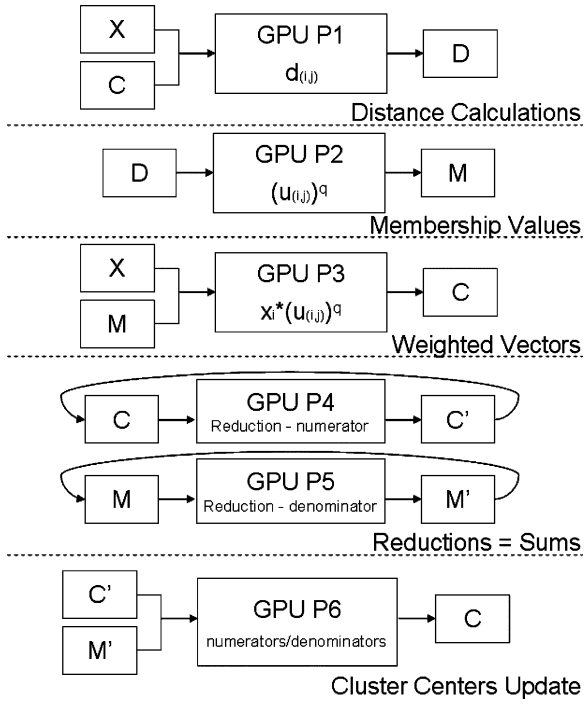

Fig. 4. Generalized GPU algorithm for fuzzy clustering. X is the dataset texture, C is the cluster centers texture, M stores the membership values, and D is the dissimilarity matrix. GPU P1, program 1, calculates the dissimilarity, GPU P2 is the membership update step, GPU P3 calculates the numerator terms in the cluster centers update equation, GPU P4 and GPU P5 are reduction steps that compute the summation terms in the numerator and denominator for cluster centers update, and GPU P6 calculates the final updated centers.

GPU P1, program 1, computes the dissimilarity values for each data point to each cluster center. GPU P2 takes these dissimilarity values and computes the new membership values (1). The membership values, raised to the qth power, are used by the following GPU passes. Thus, we compute the membership values and raise them to the qth power to avoid recomputing these values. Program P3 takes these membership values and creates an array of data points multiplied by their membership values [individual terms in the top part of (2)].

A reduction in this context is the repeated application of an operation to a series of elements to produce a single-scalar result. The reduction procedure here is the addition of a series of elements in texture memory, where operations are running in parallel given the number of fragment processors on a particular GPU. GPU P4 performs a reduction on the numerator of (2). GPU P5 performs membership value reduction, which is the denominator of (2). GPU P6 runs on C elements, dividing the P4 values by those calculated in P5, which results in the updated cluster centers (2).

Reduction takes $\log_2(N)$ number of passes, where each pass processes a fraction of the last iteration's results. The first pass processes $N/2$ elements per column, the next pass processes $N/4$ elements per column, and so forth. Each iteration is responsible for handling $N/2^i$ elements per column, where i is the iteration number. The reduction procedure is shown in Fig. 5 for two columns.

As previously stated, current hardware has a maximum texture size of 8192 $\times$ 8192, which impacts the maximum number of data points in our proposed format. A method to bypass the 8192 limitation involves packing samples into a set of columns in the same texture. We define H as the number of rows in the texture, which corresponds to the number of data points packed into a single-column set. For the NVIDIA 8800,

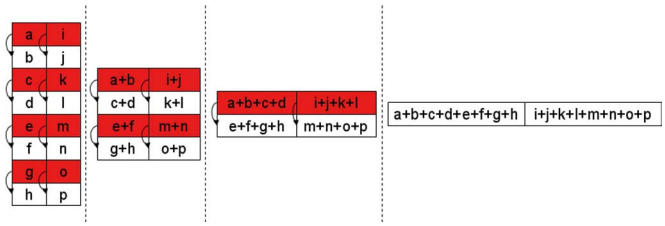

Fig. 5. GPU reduction procedure, summation operation, for a texture with two columns and eight rows. At each step, $\log_2(N)$ total steps, $N/2^i$, where i is the iteration index, and elements in a single column are pairwise summed. Each column results in the summation of values for a respective column.

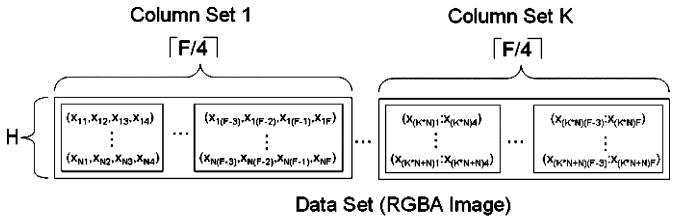

Fig. 6. Packing of input vectors into multiple columns within the dataset texture in order to support a greater number of samples.

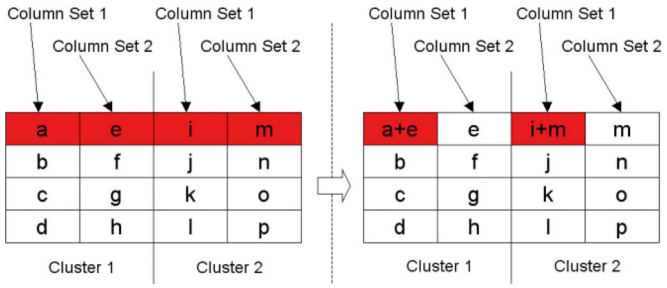

Fig. 7. Column set summation per cluster. Vertical reduction accumulates the vectors in the cluster center matrix, labeled (a) through (p), into the first row, shaded above. Column set summation is the accumulation of vectors in the first row over column sets per cluster.

the maximum H is 8192. The column set width is $\lceil F/4 \rceil$. This is usually possible because the dimension F, and in particular $\lceil F/4 \rceil$, is normally much smaller than 8192. The only limitation is the size of the GPU texture memory. Fig. 6 displays this format.

The other textures change accordingly with respect to the new column format. The membership and distance matrices were $N \times C$ and are now $H \times C \times S$, where S is the number of column sets $S = \lceil N/H \rceil$. The cluster center matrix is also increased by a factor of S in the column texture dimension. This packing scheme requires an additional GPU program, a column set summation. The column set summation GPU pass is placed after GPU P5 in Fig. 4. Fig. 7 illustrates the column set summation pass.

The cluster centers matrix in Fig. 3 contains zeros in all texture positions except for the first row. This is shown in order to stress the fact that only the first row stores the cluster centers. The reason for all the extra rows is to perform the reduction process described earlier. GPU P3 populates the cluster center matrix with the numerator terms in (2). It is not necessary to initialize the cluster centers matrix with zeros in all rows after the first. The only data that need to be passed down to the GPU, as well as transferred back at the end of the FCM GPU algorithm, belong to the first row.

Frame buffer objects (FBOs) are an extension to OpenGL that allow for rendering to destinations other than the default buffers used by the windowing system. This is one of the enabling steps for using a GPU for general purpose computing. FBOs allow for rendering to a GPU texture. This is the quintessential step for fast multipass processing of data on a GPU. If a single FBO is used, then all textures must have the same format and dimension. Multiple FBOs can be used in order to avoid wasting texture memory. The four texture formats proposed in Fig. 3 are not all the same size, which would result in a significant waste of memory in the case of the dissimilarity and membership matrices. The simplest approach to solving this problem involves utilizing a separate FBO for each different texture format and size. The dataset matrix would require an FBO, the membership and dissimilarity matrices could share an FBO, and the cluster centers matrix would require an FBO. Hence, three FBOs could be used in order to minimize the amount of texture memory wasted on a GPU for the FCM.

Using an NVIDIA 8800, the format displayed in Fig. 6, and the multiple FBO mechanism mentioned earlier, we can support profiles such as 4 194 304 data points of dimensionality four with four cluster centers, or 131 072 data points of dimensionality 32 with 16 cluster centers. Fig. 4 indicates that the texture used for the dissimilarity matrix is used only in GPU P1 and GPU P2. Therefore, from GPU P3 to GPU P6, this texture can be used for other purposes. It can be used in GPU P5 to help in the membership matrix reduction calculation. Fig. 4, in particular GPU P4 and GPU P6, shows that two textures will be needed for the cluster centers matrix. Using these assumptions, the calculation for the amount of memory needed for a specific clustering profile can be found according to the equation at the bottom of the page, where Q_j denotes the number of floating point values needed for all textures with a specific texture format and size. The inner sum is multiplied by B in order to account for the number of bytes in the floating point representation used. Q_1 represents the number of floats in the dataset texture. The 4 in this calculation is to take into account that the pixel is RGBA. Q_2 is the size of the membership and dissimilarity matrices. It is multiplied by 1 because it only uses a single color channel and the 2 signifies that only two textures of that format and size are necessary. Q_3 is for the cluster center matrix. The 2 is in there because a second copy of this texture is needed, and the 4 represents that RGBA is used. Using the aforementioned equation, one may verify that his or her particular cluster profile will fit into the GPU texture memory size.

V. Experiments

There is a tradeoff in terms of time spent setting up the GPU programs, which were developed using the Cg, i.e., C for graphics language, transferring texture memory from CPU to GPU, and managing the graphics pipeline. This means that there are situations where it is more efficient to implement the FCM on the CPU rather than on the GPU. GPUs are the most efficient when large batches of data are presented to them. We vary the number of data points, feature dimensionality, and the number of cluster centers to show performance

$$((Q_1)+(Q_2)+(Q_3))*B=\left(\left(H*\left\lceil\frac{F}{4}\right\rceil*S*4\right)+(H*C*S*1*2)+\left(H*\left\lceil\frac{F}{4}\right\rceil*C*S*2*4\right)\right)*B$$

TABLE I
CPU/GPU PROCESSING TIME RATIO FOR 4096 POINTS, 64 CLUSTERS, AND 4 DIMENSIONS

	GPU1 (7800)	GPU2 (2500M)	GPU3 (8800)
CPU1 (32 bit)	5.29	9.75	92.81
CPU2 (64 bit)	6.76	12.64	107.46

TABLE II
CPU/GPU PROCESSING TIME RATIO FOR 4096 POINTS, 64 CLUSTERS, AND 32 DIMENSIONS

	GPU1 (7800)	GPU2 (2500M)	GPU3 (8800)
CPU1 (32 bit)	4.65	9.29	83.01
CPU2 (64 bit)	6.35	11.89	88.98

TABLE III
CPU/GPU PROCESSING TIME RATIO FOR 4096 POINTS, 16 CLUSTERS, AND 128 DIMENSIONS

	GPU1 (7800)	GPU2 (2500M)	GPU3 (8800)
CPU1 (32 bit)	3.72	8.26	68.55
CPU2 (64 bit)	4.11	10.36	75.36

TABLE IV
CPU/GPU PROCESSING TIME RATIO TREND FOR THE 32 BIT INTEL AND NVIDIA 8800

Data Points \ Number of Clusters	4	16	64
64	0.26	1.32	9.76
256	0.91	4.32	31.45
512	1.66	7.93	55.95
1024	3.11	14.45	67.14
4096	10.52	49.42	92.81
8192	19.04	90.65	112.48

The dimensionality is fixed at 4 and the number of data points and clusters are varied.

TABLE V
CPU/GPU PROCESSING TIME RATIO TREND FOR THE 32 BIT INTEL AND NVIDIA 8800

Data Points \ Number of Clusters	4	16	64
64	0.55	2.44	14.25
256	1.77	7.95	45.99
512	3.30	14.68	59.74
1024	6.05	26.58	74.72
4096	20.67	63.22	83.01
8192	36.68	109.82	92.03

The dimensionality is fixed at 32 and the number of data points and clusters are varied.

results for various clustering profiles. The tables later show the speedup of clustering on a GPU, but do not reflect the fact that the CPU is freed up to perform additional computing. This means that this computer can be used for other computation at the same time as the GPU is performing the FCM, or it can be used for clustering on the CPU and GPU simultaneously, hence increasing the productivity of the entire platform.

Two unique CPU models and three unique GPU models are benchmarked later. The idea is to show differences as they relate to price, computing power, and manufacturer. We used an Intel Core 2 Duo T7200 and an AMD Athlon 64 FX-55, each of which had 2 GBs of system RAM. The GPUs used were: 1) NVIDIA 7800 BFG GS OC with 256 MB of texture memory, 16 fragment pipelines, and a bus interface of 8X AGP; 2) NVIDIA Quadro FX 2500M with 512 MB of texture memory, 24 fragment pipelines, and a bus interface of PCI Express X16; and 3) NVIDIA 8800 BFG GTX with 768 MB of texture memory, 128 stream processors, and a bus interface of PCI Express X16.

Our operating system is Windows XP with Service Pack 2 and we are using Visual Studio 2005. The GLEW version is 1.3.4, GLUT version 3.7.6, and *Cg* version 1.5. Streaming SIMD Extensions 2 (/arch:SSE2), whole program optimization, and maximize speed (/02) were enabled for the CPU in Visual Studio. The proximity metric used is the Euclidean distance, which gives a fair comparison between the CPU and GPU. More computationally expensive proximity measures can be computed faster on a GPU, which is capable of utilizing built-in instructions such as vector and matrix multiplication, dot products, and other linear algebra operations. In [11], we show how to incorporate non-Euclidean distance metrics, specifically the Mahalanobis distance, into GPU fuzzy clustering.

Because we are not presenting new metrics or clustering algorithms, but rather a speedup method for fuzzy clustering, we use randomly generated clusters. No common datasets from the community were used. We needed many cluster configurations, so we randomly generated spherical clusters of the same size with random means. In order to test the GPU's precision, we compared the GPU results to our CPU FCM implementation, and also to the MATLAB FCM function. In our C implementation, we use the same 32-bit floating-point precision employed in our GPU implementation. MATLAB uses doubles. The C program has the same algorithmic design and final precision as our GPU program.

VI. RESULTS

Tables I–III show CPU over GPU processing time ratios for different feature dimensionalities. We used a single 4096-size row, which is the common maximum row texture size among the various GPUs that were used for benchmarking.

Tables I–III show impressive computational speed improvements for a GPU versus the CPU. Depending on the particular GPU and CPU, one to two orders of magnitude improvement is observed. The performance numbers are entirely dependent on the GPU generation and CPU that it is compared to. As each new generation of GPU emerges,performance numbers are expected to increase, given the popularity of these devices and the need for stream processing. Tables IV and V show the performance behavior when we fix the dimensionality and let the number of clusters and data points vary for the best CPU and GPU.

The results in Tables IV and V indicate that when only a few clusters and data points are pushed down to the GPU, performance gain is minimal. In some cases, the CPU is even faster. However, as the number of data points increases, the GPU becomes faster. The largest speed improvements are noticed as the number of clusters is increased. Table VI shows the actual time, reported in seconds, required to compute a few clustering profiles on the GPU.

TABLE VI
TIME IN SECONDS FOR VARIOUS CLUSTERING PROFILES ON THE NVIDIA 8800

	Time in Seconds
C=4, DP=4096, F=4	0.039
C=4, DP=4096, F=128	0.053
C=64, DP=4096, F=4	0.172
C=64, DP=8192, F=4	0.362
C=16, DP=40960, F=32	0.471
C=4, DP=409600, F=8	0.780

C is the number of clusters, DP is the number of data points, and F is the feature dimensionality.

VII. CONCLUSION AND FUTURE WORK

In this short paper, we presented a novel methodology to perform fuzzy clustering on common devices normally used for graphics applications. While we specifically described the FCM algorithm, the general outline can be adapted to many other heavy computational techniques. This opens up the potential to perform clustering (and other) algorithms on large datasets at low cost and in a somewhat real-time environment, for example, segmenting a continuous video stream or for bioinformatics applications (like the basic local alignment search tool (BLAST) searches).

With respect to the algorithm and its implementation, future extensions include the topic of exploring various reformulations or reductions in the clustering algorithms or mathematics. We computed the FCM equation in its literal form. We expect, but need to verify, that reductions in the algorithm generate relative computational savings if implemented on a GPU. In addition, we plan to see how a cluster of low-end PCs equipped with GPUs performs larger clustering tasks. A final area of extension deals with very large datasets. We intend to take the eFFCM work described by Hathaway and Bezdek in [3] and implement it on a single GPU, a cluster of PCs equipped with GPUs, or multiple 8800 GPUs on a single machine connected together.

REFERENCES

[1] B. U. Shankar and N. R. Pal, "FFCM: An effective approach for large data sets," in *Proc. 3rd Int. Conf. Fuzzy Logic Neural Nets Soft Comput.*, Iizuka, Fukuoka, Japan, 1994, pp. 332–332.
[2] N. R. Pal and J. C. Bezdek, "Complexity reduction for 'large image' processing," *IEEE Trans. Syst., Man, Cybern., Part B: Cybern.*, vol. 32, no. 5, pp. 598–611, Oct. 2002.
[3] R. J. Hathaway and J. C. Bezdek, "Extending fuzzy and probabilistic clustering to very large data sets," *Comput. Statist. Data Anal.*, vol. 51, pp. 215–234, 2006.
[4] C. Harris and K. Haines, "Iterative solutions using programmable graphics processing units," in *Proc. 14th IEEE Int. Conf. Fuzzy Syst. 2005*, May, pp. 12–18.
[5] D. Marr, *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*. San Francisco, CA: W. H. Freeman, 1982.
[6] J. C. Bezdek, *Pattern Recognition With Fuzzy Objective Function Algorithms*. New York: Plenum, 1981.
[7] M. Pharr and R. Fernando, *GPU Gems 2*. Reading, MA: Addison-Wesley, 2005.
[8] GPGPU. (2006, Nov. 18), Available: http://www.gpgpu.org/
[9] J. D. Owens *et al.*, "A survey of general-purpose computation on graphics hardware," in *Proc. Eurograph. 2005, State Art Rep.*, Aug. 2005, pp. 80–113.
[10] Nvidia Corporation. (2006, Nov. 18), "GeForce 8800" [Online]. Available: http://www.nvidia.com/page/geforce_8800.html
[11] D. Anderson, R. Luke, and J. M. Keller, "Incorporation of non-Euclidean distance metrics into fuzzy clustering on graphics processing units," presented at the Int. Fuzzy Syst. Assoc. Conf., Cancun, Mexico, Jun. 2007.