Accelerating Partitional Algorithms for Flow Cytometry on GPUs

Jeremy Espenshade, Andrew Pangborn,
Gregor von Laszewski, Douglas Roberts
Service Oriented Architectures Laboratory
Rochester Institute of Technology
Email: laszewski@gmail.com

James S. Cavenaugh
Center for Biodefense Immune Modeling
University of Rochester
Email: james_cavenaugh@urmc.rochester.edu

***Abstract*—Like many modern techniques for scientific analysis, flow cytometry produces massive amounts of data that must be analyzed and clustered intelligently to be useful. Current manual binning techniques are cumbersome and limited in both the quality and quantity of analysis produced. To address the quality of results, a new framework applying two different sets of clustering algorithms and inference methods are implemented. The two methods investigated are fuzzy c-means with minimum description length inference and k-medoids with BIC. These approaches lend themselves to large scale parallel processing. To address the computational demands, the Nvidia CUDA framework and Tesla architecture are utilized. The resulting performance demonstrated 1-2 orders of magnitude improvement over an equivalent sequential version. The quality of results is promising and motivates further research and development in this direction.**

***Keywords*-data clustering, flow cytometry, gpgpu, cuda;**

I. Introduction

Flow cytometry is a technique for elucidating the phenotypes of cells in a suspension. It is a mainstay technology used in immunology, although other fields also use it. The process involves allowing fluorescently dyed antibodies to bind to proteins (antigens) on the surface of the suspended cells. Then using a technique known as hydrodynamic focusing, the stained cells pass through a laser beam one at a time, which excites the fluorophores that are indirectly attached to the antigens of interest. (Modern instruments use sequential lasers with a delay time to attach information from subsequent laser excitations to information from the first laser. Presently 18-color instruments are available, which also have information from light scatter as well.) The resulting fluorescence and light scattering data are measured by a set of sensors and recorded as a vector of d-length. Such a vector is generated for each *event*, which is typically a cell passing through the laser beam. Cells pass through the beam at the rate of thousands each second and a data file for even a single stained sample may typically involve up to one million events or more. This large data set is stored in an FCS file format (for flow cytometry standard). Finding clusters in these large and high-dimensional data sets is an application ideally suited for massively parallel computation.

The Center for Biodefense Immune Modeling at the University of Rochester is engaged in immunology research that includes the use of flow cytometry for cellular analysis. The level and value of such analysis is currently limited by the manual techniques involved, and an opportunity exists to apply novel methodologies leveraging cyberinfrastructure and current parallel computing architectures. Specifically, Nvidia's CUDA framework for scientific computation on parallel streaming processors presents a promising opportunity for low cost, high performance analysis [1].

II. Background

Flow cytometry allows researchers to identify and characterize populations of cells of interest by their co-expression of antigens which serve as markers. Currently, this is done using manual filtering where the researcher draws bins (called gates in the flow cytometry literature) around clusters of data in successive two dimensional histograms. This approach is essentially unchanged from twenty years ago and has a number of disadvantages. Variability between experienced immunologists can be as high as 10-fold for difficult data sets (unpublished research from the University of Rochester's David H. Smith Center for Vaccine Biology and Immunology). As d increases, the number of histograms increases combinatorially, making the data analysis process more difficult and tedious. There are presently no widely used standards in flow cytometry data analysis, and gates are not reported. It is therefore impossible to accurately reproduce other's work from only the raw data, and a sensitivity analysis using slight variations in the bin positions is impractical. The outcome of this time-consuming manual process is a result that is both imprecise and not very accepting of modification. Furthermore, manual sequential bivariate binning does not make full use of the multivariate nature of the data and is not conducive to making theoretically sound statistical inferences from these data sets.

Clearly an automated process for identifying cells of interest would be advantageous. Whether a sequential bivariate approach or a fully multidimensional approach is used, the problem is essentially one of finding a suitable clustering of the data. Clustering can either be hard or soft (fuzzy); hard clustering requires every datum to belong to exactly one cluster. The current practice of sequential bivariate gating is a manual version of hard clustering. Fuzzy clustering allows

each datum to belong to different clusters with different probabilities of membership (or viewed another way, with different mixture amounts from underlying archetypal distributions). By representing an event's cluster membership as a set of probabilities, one for each possible cluster, single events can be included in multiple clusters and also exert influence on the cluster location based on how closely associated they are with the cluster in question. The benefit of this is most obviously realized by the marginalization of outliers. Since cluster centers are essentially a mean of the member events, outliers have a tendency to shift the calculated center away from the logical center without fuzzy clustering. Fuzzy clustering is a much better characterization of the underlying biology than is hard clustering. By varying a cutoff for probability of membership in a cluster, it is trivial to convert fuzzy clustering to hard clustering, which makes it easy to do a sensitivity analysis. It is also possible to go in the reverse direction: one can soften a hard clustering by a function which maps the distance of each datum from each cluster's center to a probability of belonging to that cluster.

A remaining difficulty is determining the number of clusters to use. Center-based clustering algorithms, like the fuzzy c-means or k-medoids, require some initial number of clusters. If too many clusters are chosen, the results may be duplicated or muddled by separating logically singular clusters, and if too few clusters are chosen, meaningful data will be lost due to combination or elimination of distinct clusters. To solve this problem, Selb et al [2] integrated c-means into a Minimum Description Length (MDL) framework. The MDL Principle states that the more similarity that exists in a data set, the more the data can be compressed. Learning is then equated with compression. An MDL-framework then works to identify the ideal set of reference vectors, or cluster centers in this context, that describe most of the data while minimizing the number of such vectors. MDL is less well suited to a hard clustering method like k-medoids, but other inference methods like Bayesian Inference Criterion are potential candidates.

As with any fuzzy clustering algorithm, and extended with the use of MDL, several meaningful parameters governing relations between variables can have a large impact on the final result. Such parameters include the degree of fuzziness, or how much influence cluster members exert on the cluster center, fuzzy membership threshold, and the MDL weighting parameters that govern the relative weight data point description, reference vector description, and error. Such parameters are impossible to optimally set a priori and vary between data sets. Therefore, multiple analysis with differing parameters are potentially useful as well.

A. FCS Data Analysis

Figure 1 illustrates the statistically formulated cluster analysis work flow for FCM data. It is a multi-step process with some optional steps. First the data is read from an FCS file. Header information and metadata are ignored, leaving only raw flow data. The extracted data may be filtered in order to restrict attention to the regions of parameter space known a priori to be interesting and thereby to shorten the data upon which clustering will actually be performed. Ideally this would involve an automated approach as well, probably based on image analysis of the forward versus side scatter plot.

Second, a decision as to follow a sequential bivariate approach or a simultaneous multivariate approach needs to be made. The former is current practice, but could be automated using image processing algorithms such as found in Matlab. It offers the advantages of familiarity and ease of use for finding populations of cells which are known in advance to be of interest. The latter approach is more conducive to exploring the data for unanticipated findings and is more suited for formal statistical inference. It is also the approach followed in this paper. Third, the data may need to be transformed, and compensation may need to be applied to reduce the effect of fluorescence spillover from a fluorophore maximally excited in one channel to other channels. Fourth, the data may need to be transformed, as for example by log, biexponential, or logicle transformation [3]. Compensation and transformation are especially important for image based approaches such as sequential bivariate gating. Fifth, a distance measure needs to be decided upon, Euclidean being the most common. Sixth, the (possibly transformed) data may optionally be standardized and normalized. Seventh, the data are then clustered using any of the many possible clustering algorithms. After clustering the data a statistical summary should be prepared and the results should be visualized graphically [4]. Finally, the process is repeated for other samples in the experiment and then statistical and biological inferences can be made.

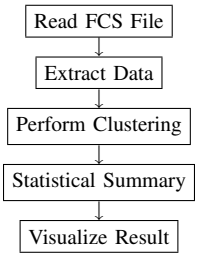

Figure 1. Objective and Automated Cluster Analysis Workflow [4]

III. Parallel Computing for Flow Cytometry

Given the large amount of data, the complexity of the algorithms involved, and the need for many computations on the same data set, a robust set of computing resources are needed. Fortunately, the problem is inherently parallel and its computation easily distributed across a number of resources. Besides the algorithmic design, the complexity

of data allocation, optimal task sizing, and communication all remain difficult problems. As such, the biostatistician is unlikely to have the expertise and uncommitted time necessary to manage the computations at a computational resource level. There is therefore a need for an abstraction of these resources such that the researcher can focus on the conceptual challenges and receive results that are immediately useful.

For this reason the problem at hand is a good candidate for parallel computing infrastructures. The large amount of data that is produced while scanning individual cells, as they pass through the laser beam, should be able to be divided up and distributed across multiple processors or node on a grid. Once the data has been divided and distributed each processor can proceed to perform computations on its own chunk of data. This will reduce the amount of time needed to process the entire data set as compared against the amount of time that is needed to process the data on a single processor machine. Once each of the processors has finished performing computations their results will be combined and a final result will be generated. Rather than using a supercomputer or a cluster of commodity workstations, we chose to use a CUDA-enabled GPU. The parallel architecture of the GPU with 192 cores is essentially a small cluster on a single chip that provides high performance for a very low cost.

IV. CUDA

In recent years, traditionally fixed-function graphics processors have transitioned into massively parallel stream processors capable of general purpose computation. Each Nvidia Tesla C870 has 128 processing units organized into 16 multiprocessors capable of handling thousands of threads, or separate streams of execution, concurrently. To manage thread creation, synchronization, and data allocation, Nvidia has developed and provided a set of APIs, compilers, and supporting libraries collectively referred to as the Compute Unified Device Architecture (CUDA). Applying data-parallel applications to the CUDA framework has been shown to provide performance on the order of hundreds of times faster than a single general purpose processor [5][6][7]. Furthermore, as the cost of a CUDA-enabled device is less than an individual workstation, the cost-performance ratio compared to a traditional cluster can be staggering. Given this potential, combined with the large computation requirements of the flow cytometry application detailed above, CUDA promises to be a valuable platform for investigation. For more details on the Tesla architecture [8] please consult the CUDA Programming Guide from Nvidia [9].

V. Algorithms

This section discusses some of the clustering algorithms applied to the flow cytometry problem. An explanation of each algorithm is given as well as pseudo code for each algorithm.

A. K-Medoids

K-medoids is a clustering algorithm that is related to the k-means algorithm. The k-medoids is a partitioning algorithm that divides the data set up into separate clusters. The algorithm attempts to minimize the squared error, which is the distance between points in the cluster and a point that is designated as the center (medoid) of a cluster. A medoid is considered an object of a cluster whose average dissimilarity to all the objects in a cluster is minimal [10].

The k-medoids algorithm functions by placing data into k clusters. k is a predetermined number that is chosen before the algorithm is executed. The algorithm functions as follows.

1) Randomly select k objects that will serve as the medoids
2) Associate each data point with its most similar medoid using a distance measure and calculate the cost
3) Randomly select a non-medoid object O
4) Replace a current medoid with the chosen non-medoid and calculate the cost again
5) If the new cost is greater than the old cost then stop the algorithm
6) Repeat 2 through 5 until there is no change in the medoid

The cost for the current clustering configuration is calculated using Equation 1, Where x_i is the i^{th} data point in the data set, d is the size of the data set, and *dist* is the distance between the data point and the closest medoid. The distance function can be implemented as any desirable distance measure, however in our implementation we use a Euclidean distance.

$$Cost = \sum_{i=1}^{d} dist(x_i) \tag{1}$$

In order to fuzzify the clusters, a membership value of each data point to every cluster (medoid) is calculated using equation 2.

$$P(x\,|m\,) = 1 - \frac{|x-m|}{\sum_{i=1}^{k} |x-m_j|} \tag{2}$$

Where x is a data point m is the medoid associated with the data point and m_j is the j^{th} medoid.

B. BIC

The Bayesian information criterion (BIC) [11] was integrated into the algorithm in an attempt to determine the best number of cluster for a given data set. The equation for the BIC is shown below.

$$BIC = n * ln\left(\frac{RSS}{n}\right) + k * ln(n) \tag{3}$$

Where *n* is the number of data points and *k* is the number of clusters being considered. RSS is the residual sum of squared errors, as seen in Equation 4, where *n* is the number of data points and x_i is the i^{th} data point and m_j is one of the medoids.

$$RSS = \sum_{i=1}^{n} (x_i - m_j)^2 \tag{4}$$

C. Fuzzy C-Means

K-means is a well known center-based clustering scheme [12] that performs hard clustering on the data by assigning each data point a membership the cluster whose center is closest to the data point. The cluster centers are then recalculated based upon the members of each cluster. Iteration stops once the change in cluster center is less than some epsilon value.

The benefit of k-means is in its simplicity and rapid convergence to a reasonable solution. The limitations are that, as a hard clustering algorithm, it is strongly affected by scattered data outside of logical clusters. To address some of the limitations of K-means, fuzzy C-means was proposed by Dunn [13] and later refined by Bezdek [14]. Fuzzy clustering allows each data point to have a membership in every other cluster, with higher membership values being assigned to clusters closest to the data point. This approach has two primary advantages over k-means. It forces outliers to have less effect on the cluster centers by assigning a lower membership value to any particular cluster. It also mitigates the effect of starting with too many clusters for the data. While k-means may split a logical cluster into several distinct sections with cluster centers in each section, fuzzy c-means will converge on the center of logical clusters resulting in nearly duplicate results that are all close to correct. The algorithm is based on the minimization of the following function defining the error associated with a solution [15].

$$E_m = \sum_{i=1}^{N}\sum_{j=1}^{C} u_{ij}^p \|x_i - c_j\|^2,\ 1 \leq m < \infty \tag{5}$$

In Equation 5, *p* is any real number that is greater than one and defines the degree of fuzziness, u_{ij} is the membership level of event x_i in the cluster *j*, and c_j is the center of a cluster. The fuzzy clustering is done through an iterative optimization of Equation 5. Each iteration, the membership u_{ij} is updated using Equation 6 and the cluster centers c_j are updated using Equation 7.

$$u_{ij} = \frac{1}{\sum_{k=1}^{C}\left(\frac{\|x_i - c_j\|}{\|x_i - c_k\|}\right)^{\frac{2}{p-1}}} \tag{6}$$

$$c_j = \frac{\sum_{i=1}^{N} u_{ij}^p * x_i}{\sum_{i=1}^{N} u_{ij}^p} \tag{7}$$

The following is an outline of a fuzzy c-means algorithm.

1) Given the number of clusters, *c*, randomly choose *c* data points as cluster centers.
2) For each cluster, sum the distance to each data point weighted by it's membership in that cluster
3) Recompute each cluster center by dividing by the associated membership value of each event
4) Stop if there is minimal change in the cluster center, otherwise return to 2.
5) Report cluster centers

This procedure exhibits several levels of parallelism which can be exploited via the CUDA framework. Most apparent is the task parallelism between clusters. Since Equations 3 and 4 are completely independent between clusters, each iteration can be performed in *c* parallel tasks, one for each cluster. CUDA supports task level parallelism through the use of multiple thread blocks which, although lacking global synchronization, are effective at computing independent tasks. Within the computation on a given cluster, data parallelism is exhibited by the independent computation of membership values for each event. Since flow cytometry has a minimum of tens of thousands of events, a tremendous degree of parallelism is available. The cluster position calculation defined in Equation 7 does require global synchronization and results collection however. Pseudo code for the parallel implementation is provided in algorithm presented in Figure 2.

D. Minimum Description Length

While the fuzzy c-means algorithm addresses some limitations of k-means, the requirement of choosing the number of clusters a priori continues to be problematic due to overfitting and duplicate clusters. To solve this problem, the Minimum Description Length principle is applied to the final result to identify the optimal number of clusters [16].

The Minimum Description Length (MDL) principle is a formalization of Occam's Razor. The idea behind MDL is that there is a best hypothesis for any set of data that will lead to the largest compression of the data. In other words, the data can be described by using fewer symbols than are needed to describe the data literally. In this problem, this asserts that there is some optimal number of clusters

```
Input:
   Events: array of event vectors
   Clusters: array of current cluster centers
Output:
   newClusters: array of new cluster centers

numerators = denominators = 0;
__syncThreads();
for (j ← 0  to  n_events + = n_threads){
   if (j + threadIdx.y < n_events){
   membershipValueFunc(j + threadIdx.y, blockIdx.x);
   calculateNumerator(memVal);
   incrementLocalDemoninator(memVal);
   }
}
__syncThreads();
sumLocalNumerators();
sumLocalDenominators();
computeNewCenters();
```

Figure 2. Parallel C-means Iteration

than can be used to describe the data while avoiding overfitting. Given the general nature of this assertion, many MDL formulations are possible, however the method proposed by [17] for determining the optimal number of radial basis vectors in RBF networks has been show to be effective in a fuzzy clustering environment in [2].

While some specifics of the formulation will be abstracted in this description (see[2] for full details), the essential function is to find which of the clusters produced by c-means should be included and which should be removed when determining the final cluster configuration to describe the data. The intrinsic worth of each cluster is related to the number of member events and the error introduced by describing each of those events by the single cluster center. With that must be balanced the number of member events that are also members of other clusters. This balance can be formalized as a symmetric cost/benefit matrix, Q, where the diagonal terms q_{ii} represent the tradeoff for the ith cluster and off-diagonal terms, q_{ij}, represent the crossover between clusters. The values are determined as follows:

$$q_{ii} = K_1 n_i - K_2 \xi_i - K_3 N_i \tag{8}$$

$$q_{ij} = \frac{-K_1 n_{ij} + K_2 \xi_{ij}}{2}, i \neq j \tag{9}$$

Where K_1, K_2, and K_3 are parameters that affect the costs of describing data, explaining error, and describing clusters respectively. The relative values of these parameters effects the scores in Q, and [17] explains how to set them. n_i is the number of events whose membership in cluster *i* exceed the threshold and n_{ij} is the number of events meeting this criteria for both clusters *i* and *j*. N_i is the dimensionality of the data and clusters. ξ_i and ξ_{ij} represent the error in one cluster and the overlap of two clusters respectively and are calculated by Equations 10 and 11. Let δ denote the distance function.

$$\xi_i = \sum_{x \in R_i} \delta\left(x, c_i\right) u_{xi}^p \tag{10}$$

$$\xi_{ij} = \max\left[\sum_{x \in R_i \cap R_j} \delta\left(x, c_i\right) u_{xi}^p, \sum_{x \in R_i \cap R_j} \delta\left(x, c_j\right) u_{xj}^p\right] \tag{11}$$

Here R_i is defined as the region of cluster *i*, or the set of all events that meet the membership criteria for that cluster. u_{xi} is the membership value, which is calculated using Equation 12.

$$u_{xi} = \frac{1}{\sum_{j=1}^{c} \left[\frac{\delta\left(x, c_i\right)}{\delta\left(x, c_j\right)}\right]^{\frac{2}{p-1}}} \tag{12}$$

The construction of the Q matrix is quite computationally intensive and therefore another good candidate for GPGPU acceleration. Each of the elements is completely independent and can be assigned to different thread blocks. Within a thread block, similar methods are used to build temporary results and concatenate them together when computing ξ and n_i as were used for computing the new cluster centers. The pseudo-code is shown in the algorithm presented in Figure 3.

Once the Q matrix has been constructed, a global Tabu Search method is then applied to solve for the optimal configuration of clusters to include. This is done by solving Equation 13 where *h* is a binary array with length equal to the number of clusters. By evaluating varying configurations defined by turning on and off clusters with h, a maximum score can be found.

$$Score = h^T Q h \tag{13}$$

VI. Tool Implementation

A computing portal can help flow cytometry, as it is fundamentally about collaboration among scientists and infrastructure experts and a large part of that is making computational resources available to scientists in a manner than they can use without concern for the underlying implementation. In flow cytometry data analysis, the scientist would like to simply supply an FCS file, possibly specify some parameters, and retrieve the results. To accomplish this, a tool chain was created that allows the scientist to set any of the internal parameters in the C-means/MDL/Tabu-Search implementation detailed above, specify some running conditions, and

Input:
  Events: array of event vectors
  Clusters: array of current cluster centers
  Cluster Index (i): cluster to examine
Output:
  ξ_i: error associate with cluster i
  n_i: membership count in cluster i

```
localError = 0;
localMemberCount = 0;
for (j=0 to n_events+ = n_threads){
 if (j + threadIdx.y < n_events){
   membershipValueFunc(i, j + threadIdx.x);
   incLocalError(memVal^2 * distance^2);
   incLocalMemberCount();
 }
}
__syncThreads();
sumLocalErrors();
sumLocalMemCounts();
```

Figure 3. Parallel ξ_i and n_i Calculation

import either an FCS binary or a previously converted tabular text file. The user interface was implemented as a Java GUI and the selected configuration launches a FCS conversion script through the statistical software package, R, if required, invokes a bash shell script to populate a header file with the selected parameters, compiles the application, and manages execution with varying numbers of clusters if requested. Figure 4 shows the interface as it is presented to the user. The design is modular and supports the inclusion of additional clustering algorithms, distance measures, and other parameters. Since the research effort towards automated intelligent clustering of flow cytometry data is only just beginning, having an extendible framework for comparison of results is very useful.

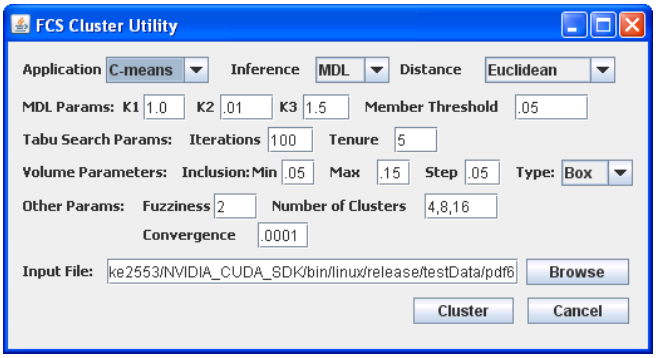

Figure 4. FCSCluster User Interface

VII. Results

Two approaches to the problem of clustering moderately high dimensional data have been presented, one based on a fuzzified K-medoids and BIC and another based on C-means and MDL.

A. K-medoids and BIC

	Sequential (ms)	CUDA (ms)	Speed Up
2	147	73.68	2.00
4	339	110.66	3.06
8	1007	295.66	3.41
16	3439	1014.24	3.39
32	12753	2272.78	5.61
64	48983	6311.81	7.76

Table I
K-Medoids Performance Summary

This section will compare the performance of a sequential k-medoids algorithm against the CUDA version of the algorithm. Each version of the algorithm was tested using a 100000 by 21 FCS file. Each version was run ten times using 2, 4, 8, 16, 32, and 64 clusters and the results were averaged. This was done to see how well they would perform when increasing the number of clusters. Table 1 is a summary of the results.

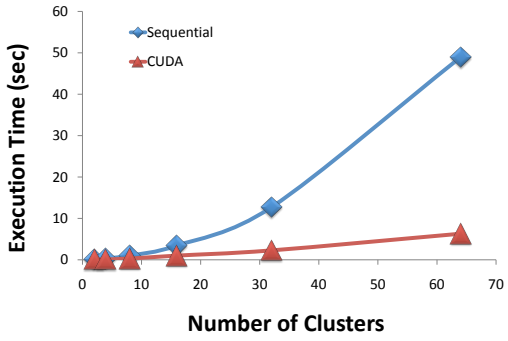

Figure 5. K-Medoids Execution Time

As you can see from Table I the execution times increase rapidly as the number of clusters increases. The CUDA version outperforms the sequential version for all numbers of clusters tested. As the number of clusters increases, more task parallelism is available and the speedup of the CUDA-enabled version increases further. Figure 5 is a graph of the performance of the CUDA and sequential versions of k-medoids as the number of clusters increases. Figure 6 shows the speedup of the CUDA version relative to the sequential CPU-only version.

B. C-means and MDL

The object of this approach is two-fold. First, the results must show functionality and demonstrate promise for FCS

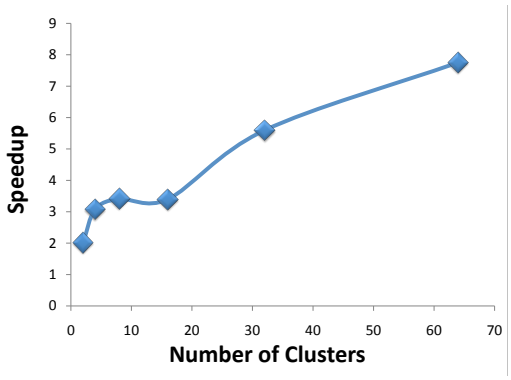

Figure 6. K-Medoids Speedup

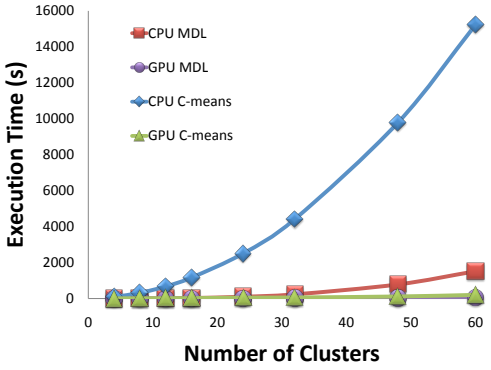

Figure 8. C-Means Execution Time

clustering. To investigate this, several test data sets were generated with known clusters using the *mvtnorm* library for *R*. Functionality was shown by selecting the known cluster centers even when originally looking for more clusters than logically exist. After C-means converges on a set of cluster centers and the MDL Q matrix is generated, the Tabu Search takes over and identifies which cluster to include and which to ignore. The correct number of clusters and cluster centers could always be identified, however the MDL parameters turned out to be quite sensitive and needed to be tuned depending on the number of starting clusters selected.

The second objective is to achieve performance improvements that realize the potential of the CUDA framework and Tesla Architecture. As detailed in the preceding section, multiple levels of parallelism exist in the application and were exploited in the implementation. To gather performance data, a single data set size of 100,000 elements was used. With this held constant, the number of clusters and the number of dimensions were varied. The essential trends have been condensed into the following Figures, and some tabular results are included at the close of this section.

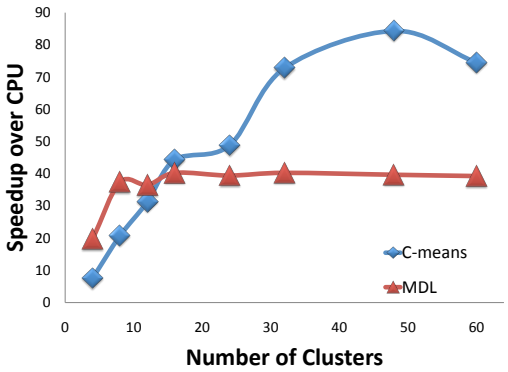

Figure 7. C-Means Speedup vs. Clusters

As is readily apparent from Figures 7, 8, and 9, the CUDA enabled GPU version far outperformed the sequential CPU version. As the amount of work increases with the number of clusters, the CPU experiences a swift increase in execution time while the GPU retains a low rate of increase. This results because any increase in work can be executed in parallel with other work on the GPU, but the CPU requires directly increased execution time to complete the work. The C-means problem is O(NC^2) where N is the number of events and C is the number of clusters. The GPU escapes this quadratic increase by exploiting the increased parallelism that results from increased clusters and only increasing the amount of work done in a thread block linearly with increasing numbers of clusters. MDL is even more dramatic, as it is O(NC^3). The faster sloping increase in the MDL execution time demonstrates this and since the GPU again only increases the work in a thread block linearly, much of the remaining quadratic increase can be absorbed through parallel computation.

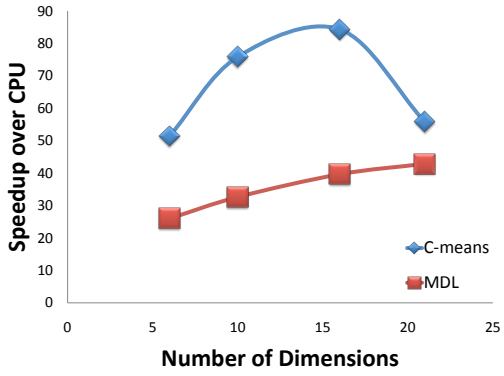

Figure 9. C-Means Speedup vs. Dimensions

Figure 9 shows the speedup change as the number of dimensions change while holding the number of clusters constant at forty-eight. This shows an interesting result for both C-means and MDL. C-means peaks at 16 dimensions and then falls slightly (although maintaining a significant speedup). This occurs primarily because the number of concurrent threads had to be decreased to accommodate

the larger memory requirements that come with additional dimensions. MDL continues to slightly increase as the memory demands are less and the same type of execution time reduction through parallelism that drove speedups in Figure 7 continue to be beneficial. The following Tables show the raw execution time data and speedup results. The highest observed speedup are 84.34 times for each C-means iteration and 43.4 time for MDL Q Matrix Generation.

Table II
EXECUTION TIME AND PERFORMANCE DATA FOR 16 DIMENSIONAL DATA

	Single CPU (ms)		GPU (ms)		Speedup	
n	cmeans	MDL	Cmeans	MDL	Cmeans	MDL
4	90	0.51	11.88	0.02	7.57	19.69
8	318	3.80	15.35	0.10	20.71	37.38
12	676	12.79	21.61	0.35	31.28	36.50
16	1168	30.56	26.36	0.75	44.30	40.26
24	2512	99.37	51.40	2.51	48.87	39.44
32	4418	235.14	60.69	5.83	72.79	40.28
48	9792	785.57	116.09	19.80	84.34	39.67
60	15222	1519.10	204.36	38.67	74.48	39.27

Table III
EXECUTION TIME AND PERFORMANCE DATA FOR 21 DIMENSIONAL DATA

	Single CPU (ms)		GPU (ms)		Speedup	
n	cmeans	MDL	Cmeans	MDL	Cmeans	MDL
4	95	0.64	16.94	0.03	5.60	16.44
8	330	4.76	21.57	0.15	15.29	29.98
12	672	16.25	29.44	0.45	22.82	35.99
16	1138	38.13	36.86	0.92	30.86	41.19
24	2464	123.71	72.70	2.94	33.88	42.08
32	4302	293.19	86.66	6.75	49.64	43.40
48	9428	987.33	168.95	23.02	55.80	42.88
60	14774	1930.56	267.26	44.89	55.27	43.00

VIII. CONCLUSIONS

The performance results demonstrated from the two approaches explained in this paper show excellent speedup and make effective use of the massively parallel Tesla architecture using the CUDA framework. Further work is required to investigate data quality and intelligently move forward with improvements. The availability of efficient implementations of data clustering algorithms will revolutionize how flow cytometry data is analyzed. We will further optimize our algorithms to achieve even better performance and investigate other clustering techniques.

ACKNOWLEDGMENT

Work conducted by Gregor von Laszewski is supported (in part) by NSF CMMI 0540076 and NSF SDCI NMI 0721656.

REFERENCES

[1] Cuda zone. NVIDIA Corp. [Online]. Available: www.nvidia.com/cuda

[2] A. Selb, H. Bischof, and A. Leonardis, "Fuzzy c-means in an mdl-framework," in *15th International Conference on Pattern Recognition (ICPR'00)*, vol. 2, 2000, p. 2740.

[3] D. R. Parks, M. Roederer, and W. A. Moore, "A new logicle display method avoids deceptive effects of logarithmic scaling for low signals and compensated data," *International Society for Analytical Cytology*, vol. Cytometry Part A 69A:, pp. 541–551, 2006.

[4] K. M. Abbas, Y. Lee, H. Wu, and G. von Laszewski, "e-science environment for objective analysis of flow cytometry data," gregors Flow Cytometry Paper.

[5] K. Gulati and S. P. Khatri, "Accelerating statistical static timing analysis using graphics processing units," in *3rd Annual Austin Conference on Integrated Systems & Circuits 2008*, 2008.

[6] H.-Y. Schivea, C.-H. Chiena, S.-K. Wonga, Y.-C. Tsaia, and T. Chiueha, "Graphic-card cluster for astrophysics (gracca)," in *AstroGPU*, 2007.

[7] M. K. J. Tolke, "Towards three-dimensional teraflop cfd computing on a desktop pc using graphics hardware," Feb 2008, institute for Computational Modeling in Civil Engineering, TU Braunschweig.

[8] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym, "Nvidia tesla: A unified graphics and computing architecture," *Micro, IEEE*, vol. 28, no. 2, pp. 39–55, March-April 2008.

[9] Nvidia cuda programming guide 2.0. Nvidia Corp.

[10] A. P. Reynolds, G. Richards, and V. J. Rayward-Smith, "The application of k-medoids and pam to the clustering of rules," in *Intelligent Data Engineering and Automated Learning*, ser. Lecture Notes in Computer Science. Springer Berlin, 2004, pp. 173–178.

[11] G. Schwarz, "Estimating the dimension of a model," *The Annals of Statistics*, vol. 6, pp. 461–464, 1978, bayesian Information Criterion (BIC).

[12] J. B. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability*, vol. 1. Berkeley, University of California Press, 1967, pp. 281–297.

[13] J. C. Dunn, "A fuzzy relative of the isodata process and its use in detecting compact well-separated clusters," *Journal of Cybernetics*, vol. 3, pp. 32–57, 1973.

[14] J. C. Bezdek, *Pattern Recognition with Fuzzy Objective Function Algoritms*. Plenum Press, New York, 1981.

[15] G. Gan, C. Ma, and J. Wu, *Data Clustering Theory, Algorithms, and Applications*, M. T. Wells, Ed. Society for Industrial and Applied Mathematics, 2007.

[16] P. D. Grunwald, *The Minimum Description Length Principle*. The MIT Press, 2007.

[17] A. Leonardis and H. Bischof, "An efficient mdl-based construction of rbf networks," *Neural Networks*, vol. 11, issue 5, pp. 963–973, 1998.