Scalable Clustering for Vision using GPUs

Thesis submitted in partial fulfillment
of the requirements for the degree of

MS By Research
in
Computer Science

by

K Wasif Mohiuddin
200807007
wasif.m@research.iiit.ac.in

CVIT
International Institute of Information Technology
Hyderabad - 500 032, INDIA
Dec 2011

International Institute of Information Technology
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled ”Scalable Clustering for Vision using GPUs” by K Wasif Mohiuddin (200807007), has been carried out under my supervision and is not submitted elsewhere for a degree.

Date	Adviser: Prof. P J Narayanan

To my loving Parents and Sisters

Acknowledgments

Firstly i would like to thank my guide Prof P J Narayanan for always being there to support and encourage throughout my masters program. Our meetings helped me develop my thinking ability and come up with various possible solutions for a particular problem. I am also grateful to him for helping me out at various stages of my research and improve my paper presentation skills during deadlines. I would also like to thank faculty members of our lab and institute for providing knowledge and support in various courses.

My interactions with Phd students Gopal Joshi, Pavan, Pramod and Natraj have been very fruitful. These discussions helped me come up with innovative research problems, ideas for complex problems. I was glad to have experienced people around whenever my research met a dead end. Special thanks to Sashidhar who always used to be around and help me tackle any technical problems or debugging i used to encounter. He always used to stay till the end until things got fixed. I thank Sheetal for helping me out during my initial paper submissions with figures, latex. I would also like to mention names of my fellow lab and batch mates Mihir, Maneesh, Prachi, Pratyush, Chhaya, Mrityunjay, Chandrika, Raman, Omkar, Rahul, Rasagna, Rohit, Laxit, Sumit, Suhail and Jyothish for helping me out at various stages of my research.

Finally i am really grateful to my parents and sisters who have been caring, motivating throughout my research.

Abstract

Clustering algorithms have wide applications in Computer Vision, Data mining, Data Visualization, etc. Clustering is an important step for indexing and searching of documents, images, video, etc. Clustering large numbers of high-dimensional vectors is very computation intensive. CPUs are unable to handle such load and consume sometimes days and even weeks to cluster large data. GPUs are being used for general purpose computing of wide range of problems which require high computational power. Today's GPUs deliver as high as 1.5 TFLOPs. The GPU has evolved over the time not only by increasing number of cores but also major architectural changes like faster access of data, more shared memory, threads per block, etc. Such changes have enabled the GPU programmers to exploit its architectural features to the fullest and achieve high performance. In this thesis, we focus on K-Means algorithm which is a widely used unsupervised clustering algorithm. We develop a GPU based K-Means implementation for large datasets; also we have used this implementation to develop a video organizing application.

We present the design and implementation of the K-Means clustering algorithm on the modern GPU. All steps are performed entirely on the GPU efficiently in our approach. We also present a load balanced Multi-node, Multi-GPU implementation which can handle up to 6 million, 128-dimensional vectors. We use efficient memory layout for all steps to get high performance. The GPU accelerators are now present on high-end workstations and low-end laptops. Scalability in the number and dimensionality of the vectors, the number of clusters, as well as in the number of cores available for processing are important for usability to different users. Our implementation scales linearly or near-linearly with different problem parameters. We achieve up to 2 times increase in speed compared to the best GPU implementation for K-Means on a single GPU. We obtain a speed up of upto 170 on a single Nvidia Fermi GPU compared to a standard sequential implementation. We are able to execute single iteration of K-Means in 136 seconds on off-the-shelf GPUs to cluster 6 million vectors of 128 dimensions into 4K clusters and in 2.5 seconds to cluster 125K vectors of 128 dimensions into 2K clusters.

Video data is increasing rapidly along with the capacity of storage devices owned by a lay user. Users have moderate to large personal collections of videos and would like to keep them in an organized manner based on its content. Video organizing tools for personal users are way behind even the primitive image organizing tools. We present a mechanism in this thesis to help ordinary users organize their personal collection of videos based on categories they choose. We cluster the PHOG features extracted from selected key frames using K-Means to form a representation for each user-selected category during

the learning phase. During the organization phase, labels from a K-NN classifier on these cluster centers for each key frame are aggregated to give a label to the video while categorizing. Video processing is computationally intensive. To perform the computationally intensive steps involved, we exploit the CPU as well as the GPU that is common even on personal systems. Effective use of the parallel hardware on the system is the only way to make the tool scale reasonably to large collections that will be available soon. Our tool is able to organize a set of 100 sport videos of total duration of 1375 minutes in about 9.5 minutes. The process of learning the categories from 12 annotated videos of duration 165 minutes took 75 seconds on a GTX 580 card. These were on a standard desktop with an off-the-shelf GPU. The labeling accuracy is about 96% on all videos.

The ideas, approaches proposed in this thesis have been implemented and validated with experimental results. For large data-sets we developed a scalable, efficient K-Means clustering on GPU along with a Multi GPU framework and used it to develop a video organizer application providing high accuracy.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Computer vision is a vast area which includes methods for acquiring, processing, and understanding images. It deals with modeling and replicating human vision using computer software and hardware. This field is a study on how to reconstruct, interpret and understand a 3D scene from its 2D image in terms of properties of structures present in the scene. The goal of Computer vision is to process images in order to produce a representation of objects in the world. The motive behind this field is to duplicate the abilities of human vision by electronically perceiving and understanding an image in the best possible manner. As a scientific discipline, computer vision is concerned with the theory behind artificial systems that extract information from images. An image undergoes different steps of computer vision which typically include classification, recognition, detection, tracking, etc. Humans can easily recognize content of an image based on objects present, scene etc. For the system to understand an image we need to consider global as well as local information present. Visual recognition problems like classification requires a thorough analysis of the visual content in an image. Image classification is among the most explored area in the field of computer vision. Much work has been done and techniques proposed for classification purposes. In the modern approach the image data is being represented in the form of high dimensional descriptive vectors by means of various feature extracting algorithms. Large collection of images or videos often require extensive machine learning as a part of training the system which can be used later on. Such tasks are extremely computationally expensive due to large data, high dimensionality, etc. Unsupervised structure finding is the first step in many approaches related to vision. In order to solve the general vision problem we will have to come up with answers deep and fundamental questions about representation and computation at the core of human intelligence.

1.1 Classification

An approach to classifying images treats them as a collection of regions, describing only their appearance and ignoring their spatial structure. Figure 1.1 shows series of steps involved in the generation of visual words. Features are extracted from the given set of input images. Images can be represented by a group of key points but typically the groups vary in size and lack a meaningful order. In order to

address this issue vector quantization technique is used which clusters the descriptors in their feature space into large number of clusters using K-Means clustering algorithm. Unsupervised structure finding is the first step in many approaches. The main reason for quantisation is the large range of values and their sensitivity towards small image perturbations. So clustering makes the process robust by partitioning data into groups of more related samples. These features are clustered into meaningful groups as shown in step 3 of the figure. Each cluster is treated as *visual word* which represents a local pattern shared by key points present in that cluster. These clusters are used to develop histogram of visual words extracted from the shoe images. The clustering process generates *visual-word vocabulary* [45]. The size of the vocabulary depicts the number of clusters which is typically of the order thousands for large scale classification. Each key point is associated with a visual word and there by the entire image represents bag of visual words.

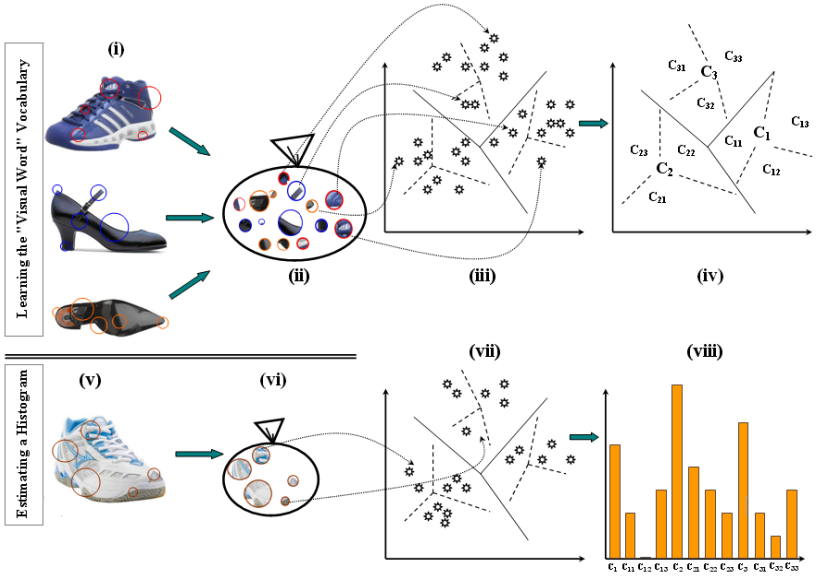

Figure 1.1 Bag of Words for Vocabulary Generation. Tomasik *et al.* [46]

This model has been successfully used in the text retrieval for analyzing documents and are known as ”bag-of-words” model, since each document is represented by a distribution over fixed vocabulary. Using such a representation, methods such as probabilistic latent semantic analysis (pLSA) [19] and latent dirichlet allocation (LDA) [4] have provided promising results for document collections in an unsupervised manner. Many current approaches for image classification are based on visual words (clusters of local descriptors), the per image frequency of these visual words (bag of local features), and a Support Vector Machine (SVM) classifier for bags of local features.

1.2 Objective

A number of techniques have been proposed lately targeting the accuracy and optimisation of classification algorithms. Figure 1.2 showcases the classification of a test image using the prior information gathered from the training images. We have focused on the aspect of optimisation involved in achieving desired results. Thus we analyze the computationally expensive steps involved in classification like feature extraction, clustering, etc. Using hybrid accelerators we intend to develop an optimised solution for these time consuming steps. Another aspect is to get familiar with these latest architectures and understand them in order to come up with a high performance solutions for the same. The idea is to learn and develop a solution for large scale clustering. Clustering being the elementary step involved in various classification techniques. The emphasis is not only to optimize the steps but also to keep the accuracy consistent.

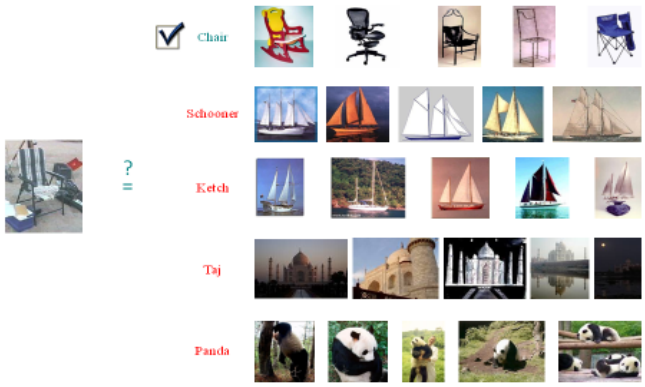

Figure 1.2 Image Classification

Unsupervised clustering is a means to discover the structure inherent in a large volume of data. Applications of this are abound in Computer Vision, Data Mining, Search, etc. The derived clusters help in understanding and visualizing the original data more efficiently and effectively. Many problems use large numbers of data items of high dimensionality and large number of natural groupings or clusters. Our objective is to provide a solution that scales well with all aspects of the problem size and the number of available cores.

Video recording devices are now commonly available to the average consumer. The amount of personal video content generated by users is increasing exponentially as a result. With the popularity of video sharing services and increased access to web, a great number of videos are available today. We intend to develop a clustering based tool for videos so that a user can organize his or her personal collection of videos. Using modern accelerators we plan to develop a fast clustering based application which may be used by a user for his personal collection.

1.3 Motivation

Clustering large high dimensional data is computationally very expensive. The users of such processing are no longer limited to large institutions; smaller institutions, research groups, and even individuals may need similar processing. For instance, clustering could be a first step towards organizing one's personal collection of photographs using computer vision based techniques. The state-of-the-art scene descriptors like SIFT [31] used in computer vision are typically 128 dimensional, in case of GIST [33] upto 580 dimensional. Many times researchers are forced to use low dimensional data due to computational limitations. The computation needed will be heavy even for modest photo collections. A fast, scalable, and available clustering approach is necessary to solve this problem.

Taking into account the volume of the data, tasks such as organization or searching through the available content can be very time consuming and tiresome. Therefore, such tasks need to be done automatically, preferably using the content of the videos to accomplish this. Collecting videos of interest has become a hobby lately given the fact that people have large storage devices. Everyone has different category of interest in terms of video. Figure 1.3 shows cluster of random videos and set of organized videos as per their category. Clearly the user will find the organized collection not only visually appealing but also the ease of access with which he can browse a video category of his choice.

Figure 1.3 A set of mixed Videos on left-hand side and on right-hand side a set of Videos organized as per their categories

Sports is a genre which has large following around the world. People tend to store full recorded matches or selected highlights of sport events of interest on their personal storage devices. An individual may have interest in multiple categories of sports and would like to maintain a personal collection in an organized fashion. There is no tool to do this today for videos; even those available for images are not sufficiently sophisticated. Other genres of interest to everyday users would be family events, outings, graduation ceremonies, etc., which also need to be organized appropriately for easy access in the future.

We concentrate on the highly available multi- and many-core accelerator architectures to increase the reach of the approach. Using CPU and GPU provide us high computational power.

1.4 Challenges

Computer vision often deals with large numbers of images and videos. It also has vast number of algorithms used for purposes like Classification, Recognition, Detection, etc. Classification algorithms require intense computation and typically take too much time even during processing of images. At times it even consumes days for training or testing the given data. Due to limited CPU resources researchers have often felt this computational barrier hindering the progress of their research. Typically these algorithms are iterative in nature and need to be carried out till a certain condition is met. In today's fast moving world no one likes to wait for days in analyzing their experiments, people need instant results at their finger tips.

K-Means is the most commonly used clustering technique [32], with a sequential time complexity of O($n^{kd+1} \log n$), where n is the input size, d the dimensionality, and k the number of clusters. K-means is a time consuming algorithm, clustering 125k vectors of 128 dimension into 2k clusters takes nearly 8 minutes on a CPU. The challenge lies in bringing its running time to a practical range by exploiting computing resources. Reasonable amount of work has been done to optimize the K-Means algorithm, so we need to come up with an optimal solution which is capable of fixing the loop holes in existing methodologies and deliver reasonable performance.

Videos are bulky and so content based organization can be really computationally heavy. On the other hand, we can use evidence from multiple parts of the video for its categorization. The challenging part in developing a clustering video organizer is to extend the image classification techniques to videos. These videos may span across wide class of categories. A number of image representations have been proposed lately, so an appropriate representation technique must be chosen which serves our purpose for feature extraction. Extracting features from a simple video itself is an expensive step. Doing so for a collection of videos is very challenging. This task demands a fast and scalable algorithm design which is capable of clustering videos quickly and accurately with the help of hybrid accelerators.

1.5 General Purpose computing using GPU

Graphics Processing Units (GPUs) are now part of most PCs and laptops. GPUs have been increasingly used for a wide range of problems involving heavy computations in graphics, computer vision, scientific processing, etc. Figure 1.4 shows a typical GPU card which fits in a laptop or desktop and may be used for general purpose computing. The main attraction is the high computation power per unit cost; today's off-the-shelf GPUs deliver 1.35 TFLOPs of single precision power. While the specific hardware available may change with time but, their availability is likely to remain. We need algorithms which are scalable over a wide range of current and forthcoming parallel architectures. GPU

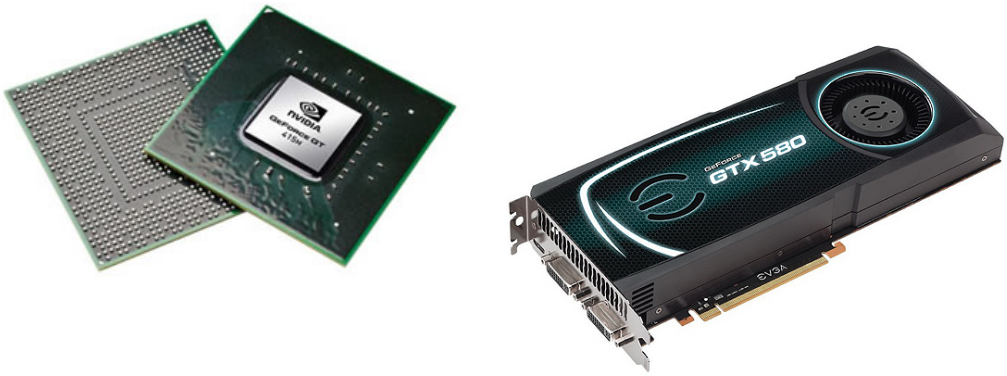

Figure 1.4 NVIDIA GPU cards

are being used in a modified form of a stream processor to allow a general purpose unit. This turns the massive floating-point computational power of a modern graphics accelerator's shader pipeline into general-purpose computing power, as opposed to being hard wired solely to do graphics operations. In certain applications requiring massive vector operations, this can yield several orders of magnitude higher performance than a conventional CPU.

Recently, Nvidia began releasing cards supporting an API extension to the C programming language called CUDA (Compute Unified Device Architecture), which allows specified functions from a normal C program to run on the GPU's stream processors. This makes C programs capable of taking advantage of a GPU's ability to operate on large matrices in parallel, while still making use of the CPU where appropriate. CUDA is also the first API to allow CPU-based applications to access directly the resources of a GPU for more general purpose computing without the limitations of using a graphics API.

1.6 Contributions of the thesis

In this thesis we present a complete GPU based K-Means clustering implementation. For large data we propose a Multi-GPU approach for the same based on the computational capabilities of individual GPU nodes. We made use of the primitives for efficient memory access and data management. A thorough analysis varying several parameters like n, d, k, cores, etc was done on various GPUs. Graphic Processor Units (GPU) have become popular on even personal systems and have tremendous compute power in them. We have also developed a video organizer application using clustering for daily users who like to keep their personal video collection in a systematic manner. Video processing is computationally expensive. We exploit all compute power available in a typical desktop or laptop of the user

to achieve this. Our system exploits the parallelism of multicore CPUs as well as the GPUs found on personal desktops and laptops today. This application makes use of our GPU based clustering we developed along with other computational steps being performed on GPU. The computation time came down significantly making it possible for laptop/desktop with low end graphic card to process within no time.

The main contributions of this thesis are the following:

1. We develop an efficient K-Means implementation for GPU, exploiting intra-vector parallelism within each data item as well as among the different data items. This is essential to utilize the hardware resources under the massively multi-threaded model. We use the K-Means++ [2] algorithm on GPU for generating initial cluster centers for better clustering.

2. We perform label assignment and mean evaluation entirely on the GPU. The row major layout for input data provided coherent memory access and data management. Use of efficient primitives like Split, Gather, Transpose, Scan, Compact, etc ensured coalesced memory access and were highly efficient for clustering high dimensional large datasets. The data transfer between CPU and GPU is minimized as a result.

3. Implementation on different generations of GPUs was done to study their respective architectural features. We achieve performance that is up to 170 times faster than a standard single-core CPU implementation on the GPU. Our approach provides almost twice speed-up compared to prior GPU implementations for high dimensional vectors. We achieve linear scaling in the running time in the number of vectors and the dimensionality, and super linear scaling in the number of clusters. The running time scales linearly with number of data objects, feature space and clusters chosen.

4. For large data we implemented a Multi-GPU approach, distributing the number of input vectors as per the computational capability of individual nodes using MPI. These nodes compute partial sums for the assigned vectors and send it back to the master node. We scale our algorithm for large n, using multi-GPU approach.

5. Video segmentation is assisted by the GPU and the PHOG computation is performed entirely on the GPU. The GPU also does bulk of the K-Means clustering to select representative vectors in the learning phase. In the categorization phase, the distance evaluation for the K-NN classifier for each frame is performed on the GPU in addition to feature extraction. The final aggregation and decision making takes place on the CPU.

6. Our tool is able to organize a set of 100 sport videos of total duration of 1375 minutes in about 9.5 minutes. The process of learning the categories from 12 annotated videos of duration 165 minutes took 75 seconds. These were on a standard desktop with an off-the-shelf GPU.

Chapter 2

Background and Related Work

In this chapter we present a literature survey about various existing implementations for optimizing the K-Means algorithm. These optimizations were not only limited to CPUs but various hybrid accelerators. We discuss about the various hybrid accelerators, which are being used for computationally intensive tasks and are providing high performance. These accelerators are now being used to perform computation in various applications. We also explore video clustering techniques employed to handle various categories using wide range of learning techniques. Aspects like dependency on prior learning, features, accuracy, feasibility, etc have been discussed.

2.1 Accelerator Architectures

We briefly explain about NVIDIA's GPU, IBM's CellBE architectures which would provide a basic understanding. GPU has a manycore architecture where in multiple cores perform a task. The architecture of the CellBE chip lies somewhere between other modern chip multiprocessors and a high end GPU, since in some views the eight Synergistic Processing Unit (SPU) mimic pixel shader units. Unlike GPUs, however, the Cell can chain its processors in any order, and have them operate independently.

2.1.1 GPU Architecture

GPUs are powerful computational devices used for graphics rendering. These days GPU are also being used for applications requiring high computational power, termed as General Purpose Computing on GPU (GPGPU). NVIDIA's Fermi architecture [35] has 16 Streaming Multiprocessors (SM) as shown in Figure 5.1 with each SM having 32 cores, so on the whole it has 512 CUDA cores. Every core in an SM executes the same instruction at all times. A set of threads forms a block. Many blocks put together form a grid. Blocks are assigned to SM for execution. SM processes one warp at a time where each warp is of 32 threads from a block. The function calls are made in the form of kernel which unleash multiple threads to perform a task in a Single Instruction Multiple Data (SIMD) fashion. The Compute

Unified Device Architecture (CUDA) [34] programming model allows programmers to design a kernel.

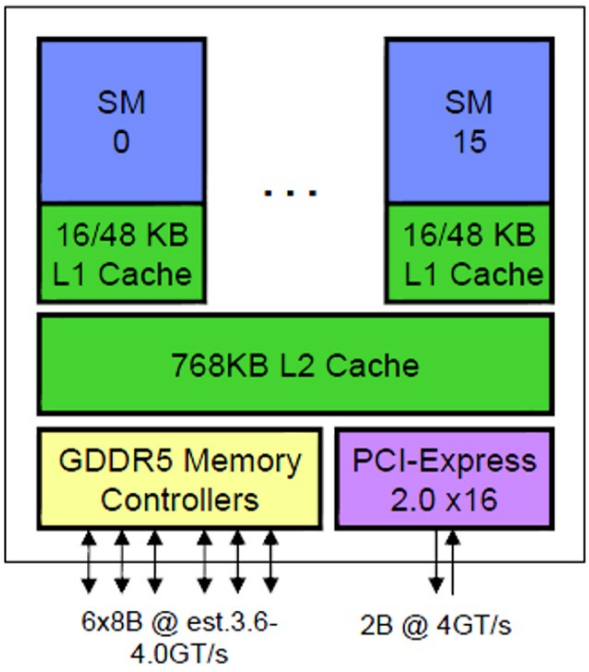

Figure 2.1 The Fermi Architecture

Every SM has registers divided equally amongst its threads. Each thread has some private local memory. The off-chip global device memory per card can be accessed by every thread in the grid but consumes hundred of clock cycles for a single fetch. NVIDIA GT200 series has 16 KB of shared memory per SM. The Fermi architecture has a single unified memory request path for loads and stores using the L1 cache per SM multiprocessor and unified L2 cache that services all operations. L1 cache is configurable to support both shared memory and caching of local and global memory operations. The 64 KB memory can be configured as either 48 KB of Shared memory with 16 KB of L1 cache or vice-versa. By configuring 48 KB of shared memory, programs that make extensive use of shared memory perform up to three times faster. The lifetime of this memory is same as that of a block. Fermi features a 768 KB unified L2 cache that services all load, store, and texture requests. The L2 provides efficient, high speed data sharing across the GPU. Apart from the global device memory the GPU also has Constant and Texture memory. Unlike previous generation GPU's, some salient features of Fermi architecture are faster double precision, faster context switching, faster atomic operations and multiple kernel execution. The GPU has been able to provide significant speed ups for irregular algorithms like List ranking [41], Graph algorithms [13, 16].

2.1.2 CellBE Architecture

CellBE [21] processor can be divided into two units as shown in Figure 5.2 namely the Power Processing Unit (PPU) and the Synergistic Processing Units (SPU). The PPU is a 64bit Reduced Instruction Set Computer (RISC) Power Architecture 2 threaded processing unit. SPUs are special processing elements that are generally assigned tasks from the PPU. The SPUs implement altivec SIMD instructions that operate on 128 bit registers. One of the key architecture feature of the SPUs is it is user managed. Each SPU has a fast 256 KB local store of memory it can access. This can conceptually be thought of like a cache. An SPU can access data in the main memory by issuing a Direct Memory Access (DMA) instruction to pull the data into a specified address in its local store.

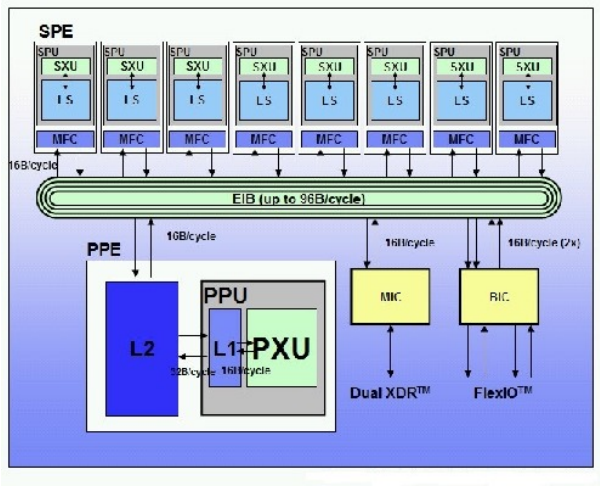

Figure 2.2 CELL Architecture. *Source M Gschwind et al., Hot Chips 17, August 2005.*

The local store has a size of 256 KB so data must be sent in chunks of size less than 256 KB. Instructions which are repeated for multiple data elements should be properly utilized to exploit the SIMD. All processors are interconnected with a high-bandwidth ring network called the EIB (Element Interconnect Bus). The CellBE's design is one that favors bandwidth over latency, as the memory model does not include a hierarchical cache. In addition, it favors performance over programming simplicity. All memory accesses must be performed by the programmer through DMA transfers calls, and the local cache at each SPE is managed explicitly by the programmer. Each SPE contains an SPU and an SPF. The SPF consists of a DMA controller, and an MMU (memory management unit) to interact with the common interconnect bus (EIB). Bandwidth from an SPE to the EIB is about 25 GB/sec, both upstream and downstream (see Figure 2). SPEs are SIMD, capable of operating on eight 16 bit operands in one instruction. Applications like Minimum Spanning Tree for graphs [23], Sorting [22], etc have been implemented on CellBE and provided significant speed-up compared to a sequential implementation.

2.2 K-Means Clustering

K-Means is an unsupervised clustering algorithm for vectors proposed by MacQueen [32]. Given a set $X \subset R^d$ of n points in a d-dimensional space and an integer $k \geq 2$, the problem is to partition X into k disjoint subsets $(S_1, S_2.., S_k)$ along with a set $C = c_1, ..., c_k$ of corresponding centers such that the vectors in S_i is closest to the center c_i than any other c_j for a pre-defined distance measure. The sequential K-Means in Algorithm 1 iteratively calculates the distance from every input vector to each of the current centers and assigns it a label i between 1 and k based on the center c_i it is nearest to. In a second step, each center c_i is updated to the mean of all vectors with the label i. This 2-step process is repeated until no vector changes the label or a suitable convergence condition is met. The computations incurred per iteration may be given by $O(nkd)$ as the distance from each input point to each center needs to be calculated.

Algorithm 1 K-Means Algorithm

1: Input $\{x_i | i = 1...n\} \subset R^d$ and a set $C = c_1, ..., c_k$ of initial centers
2: Membership evaluation : Assign each vector x_i to cluster c_j with a label j for which the distance(x_i,c_j) is minimum among the current cluster centers.
3: Mean evaluation : Evaluate new centers c'_j as the mean of all vectors x_i that was assigned the label j.
4: Check condition for convergence, if true then convergence is achieved else go back to step 1 with the set of new centers C'

The first task is to select the initial centers in order to cluster the input data. In Figure 2.3 we can see centers are selected randomly from the set of points. Then membership is assigned based on Euclidean distance. New means are computed using which the process of relabelling is done again until convergence. This iterative process could result in a large number of iterations before convergence. We use the K-Means++ scheme to generate the initial centers which are spread out [2]. In K-Means++, the first center is selected at random from the input vectors. The subsequent ones are selected based on the probability of distance $D(\mathrm{x}_i)^2$ from the previous selected center. K-Means++ is computationally more expensive but reduces the overall computation time in practice by lowering the number of iterations.

K-Means has been worked on by many researchers. Pelleg and Moore [38] employed kd-tree to improve the K-Means algorithm. This technique defines regions in n-dimensional space. While assigning labels, it is checked if points lie in bounding box or not. However, Weber and Zezula [50] found that bounding trees do not scale well with increasing dimensions. Elkan [11] used triangle inequality to reduce unnecessary distance calculations based on distance from the previous centers and maintaining a look up table between old and new centers. Although there was a reduction in distance evaluations by a factor of nearly 10, for high values of k the book keeping turned out to be a dominant expense. Multithreaded parallelism on cluster of Symmetric MultiProcessors (SMP) architectures using OpenMP or MPI [17] may be used on very large data able to achieve high performance but were limited by number of SMP nodes. There have been approaches for K-Means on GPU. Hall and Hart [14] in their

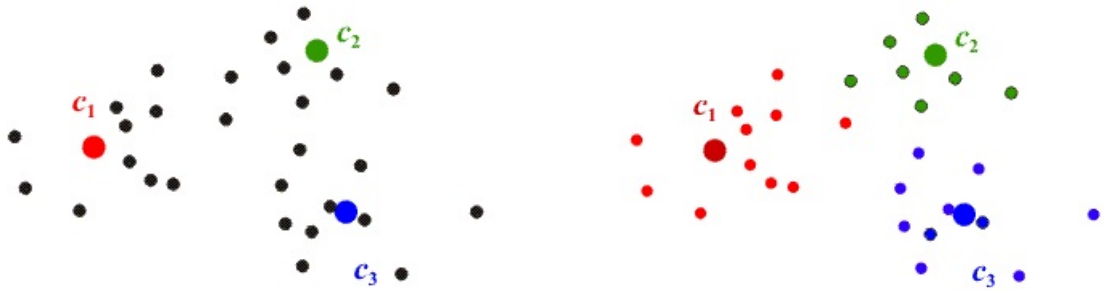

Figure 2.3 K-Means Clustering: Selecting Centers and assigning labels accordingly

implementation on NVIDIA's GeForceFX 5900 Ultra used the fragment shader to fetch input data from texture memory and cluster center from the constant memory for metric evaluation. Their approach was constrained in dimensionality due to texture memory limitations.

Che *et al.* [7] and Zechner *et al.* [54] put partial steps of K-means onto the GPU, where every thread is associated with a data object sequentially evaluating its label but the evaluation of new means was done entirely on the CPU. Hong *et al.* [20] in their approach further moved the new center evaluation partially on the GPU, achieving a speed-up of 70 on NVIDIA GeForce 8800 GTX for small clusters but the rearrangement of input vectors as per labels was done on the CPU. In GPUMiner Ren *et al.* [51] discussed two approaches for small and large input data due to the limited device memory on GPU and streamed data whenever size exceeded and achieved a speed up of 50-88 on a GTX 280. It used bitmap approach based on the closest center changes the suitable bit into true. A bitmap stores occurrences of item in a transaction and encode information on GPU. Each thread is responsible for one centroid and finds corresponding data point from bitmap. The drawback with GPUMiner is poor memory utilization and bitmap approach is not elegant for high performance also consuming more space for large k. HPKmeans by Wu *et al.* [52] considered GPU memory hierarchy utilizing bandwidth efficiently. They used Constant and texture memory for their cache mechanism and shared memory for data requiring frequent access. Li *et al.* [29] moved the mean evaluation on GPU using a divide and rule approach. The input data is divided into n/M chunks where M is a multiple of number of SM's. Temporary centroids are evaluated on GPU iteratively and eventually sent to CPU for final processing. The approach would not be beneficial for large n. These approaches exploited the parallelism of the multiple data items only. Data objects were independent and were assigned to a single thread. The evaluation on each dimension of that data item is independent and can be parallelized.

There have been implementations of K-means on CellBE platform. Buehrer *et al.* [6] made use of single instruction multiple data (SIMD) intrinsics for faster distance evaluations [44] and took care of load balancing and synchronization amongst the SPU's. They worked on datasets of size upto 400K with low number of clusters.

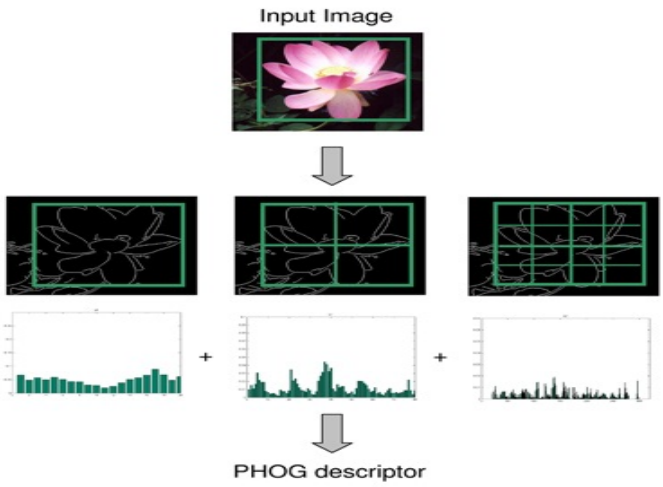

Figure 2.4 PHOG Feature Descriptor. Source A Bosch et al., CIVR 2007.

2.3 Video Applications

Much has been done in the field of computer vision towards analyzing the image content for scene classification, object detection, image search, etc. Less work has gone on doing the same on videos. A video genre can be discriminated from the other based on the analogous features and attributes that is disparate from other genres. Based on the content of a video, it could be categorized into different genres such as Cartoon, Sports, Commercials, News, Serials, etc.

A number of algorithms have been designed for key frame extraction for videos based on flow. Lui *et al.* proposed a triangular model of perceived motion energy to model motion patterns in videos [30]. The key frames were the turning points of motion acceleration and deceleration. Chen *et al.* extracted optical flows and used them to cluster human crowds into groups in unsupervised manner using adjacency-matrix based clustering (AMC) [8]. Gross image features such as motion and color were used to classify video genre, along with a decision tree classifier [47] concentrated on background or camera motion and the foreground object motion using Gaussian Mixture Model (GMM) as the classifier [42]. Ekenel *et al.* addressed the problem of video genre classification for five classes with a set of visual features, with SVM used for classification [10]. They used temporal and spatial information to build an HMM classifier. The technique described in Vakkalanka *et al.* [48] uses different types of spatial and temporal features. The features are modeled using two different classifier methodologies, namely Hidden Markov Model (HMM) and Support Vector Machines (SVMs). Rea *et al.* near-automatically classified tennis videos by modeling the spatio temporal behaviour of the serving player [40]. They then summarized a match using a number of key-frames on a synthesized court.

A survey of techniques for automatic indexing and retrieval of video data can be found in Lebart *et al.* [27] and Wang *et al.* [25]. A number of descriptors have been proposed for image representation like GIST [33], PHOG [5], etc. GIST has been found useful for scene classification while PHOG has been used for object identification [9]. PHOG descriptor was chosen as it depicts the extent to

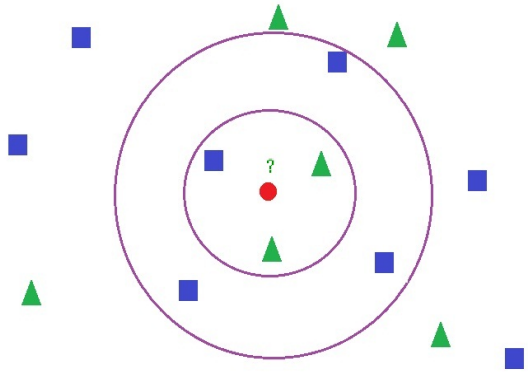

Figure 2.5 K-NN Classification

which the images have similarity in any of the shapes and correspond in their spatial layout. In PHOG descriptor we have histogram of orientation gradients over each of the image subregion as shown in Figure 5.5 at different levels. Clubbed together these form pyramid of histogram of oriented gradients. Image is divided into increasing spatial grids by doubling number of divisions. This forms a pyramid representation as number of points in a cell is summation of all the cells int was divided at next level. At each level the cell counts are bin counts for histogram representing that level. For each of the grid cell HOG vector is computed, which is eventually concatenate with others to get the PHOG descriptor. We follow the current approach using PHOG to classify videos into appropriate categories. The learning phase uses a few videos tagged by the user of each sports category. We extract a few keyframes from each video and build a representation using K-Means clustering. Use of other clustering algorithms apart from K-Means has also been seen. K-NN has also been very successful in cases where a data object may belong to multiple categories. Especially in cases of videos which may belong to different categories on the whole but may have set of similar features or content common. For example a Cricket and Football video have similar frames consisting of crowd, grass, etc. Figure 2.5 shows an example where the test sample (Red) is to be classified between (Green) and Squares (Blue) if we consider K=3 nearest neighbour the test sample gets assigned to Triangle class and if we take K=5 it would be assigned to class of Squares. This algorithm becomes very handy when dealing with such closely related categories in case of videos. The naive version of the algorithm is easy to implement by computing the distances from the test sample to all stored vectors, but it is computationally intensive, especially when the size of the training set grows. Using appropriate neighbors for the application makes the K-NN algorithm computationally tractable for large datasets.

Image/Video processing algorithms often deal with processing of large number of images/frames, iterations, high dimensional feature descriptors, etc. They require tremendous computational power. Processing such algorithms on a CPU often takes weeks and even months in some cases to produce

results. A number of algorithms are being ported on hybrid accelerators which are capable of dealing with such parameters.

2.4 Computer Vision on the GPU

Optimized computer vision libraries for CPU often consume too many of the CPU cycles to achieve real-time performance, leaving little time for other tasks. GPUs are being widely used for computer vision applications. The operations involved in vision map efficiently on the GPU architecture. Computationally intensive tasks which have extensive applications in vision algorithms are being ported onto GPUs like SIFT features extraction by Heymann *et al.* [18], Graph Cuts by Vineet *et al.* [49], Singular Value Decomposition by Sheetal *et al.* [24], Even applications like 3D reconstruction which require learning from large dataset of images has been implemented [12] on GPU. GPUs are found on most personal computers and often exceed the capabilities of the CPU. Thus, we can use the GPU to accelerate computer vision computation and free up the CPU for other tasks. Furthermore, multiple GPUs can be used on the same machine, creating an architecture capable of running multiple computer vision algorithms in parallel.

Chapter 3

K-Means on Multicore and Many-core Accelerators

K-Means algorithm has been implemented on various architectures like GPU, CellBE, OpenMP, MPI etc. We have focused on two accelerators namely the GPU and CellBE. We were able to optimize the per iteration runtime of K-Means algorithm compared to conventional approaches. Also we varied our approach as per the architecture so that our implementation is adaptable to evolving architectures. The cores have been increasing with development in GPU architectures. The interaction between and CPU and GPU was reduced making the entire algorithm run on GPU. One of the most important thing was to ensure that the architectural features like shared memory, registers, threads per block, global read/write, etc are exploited along with other optimizations. We ensured that data access is coalesced and does not involve concurrent writes. In case of CellBE focus was on load balancing, minimize SPU to PPU data transfer, SIMD intrinsics for vector operations, etc.

3.1 Single GPU Implementation

The implementation is divided into two parts: membership evaluation and mean evaluation. We have extended parallelism to the computation done on each component of each input and center vector compared to a typical one thread per input vector approach. In our approach shared memory, global memory, L2 cache, etc were used appropriately.

3.1.1 Data Layout

The n input vectors are arranged in a row major format as shown in Figure 3.1, This provides perfectly coherent memory accesses as consecutive components of each vector is worked on by consecutive CUDA threads. For coalesced access consecutive threads must access contiguous memory locations. This makes the job easier during membership evaluation since consecutive threads are mapped to the row major input vector. The input vectors and cluster centers are stored in row major format to enable fast coalesced reading of the whole vector using d threads.

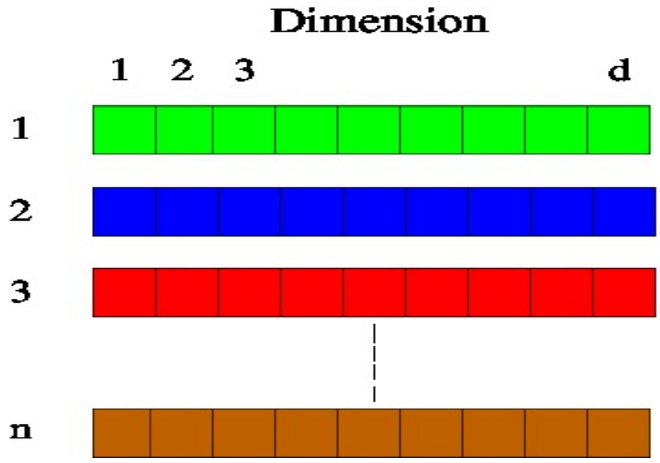

Figure 3.1 d Dimensional Input data stored in a row major format.

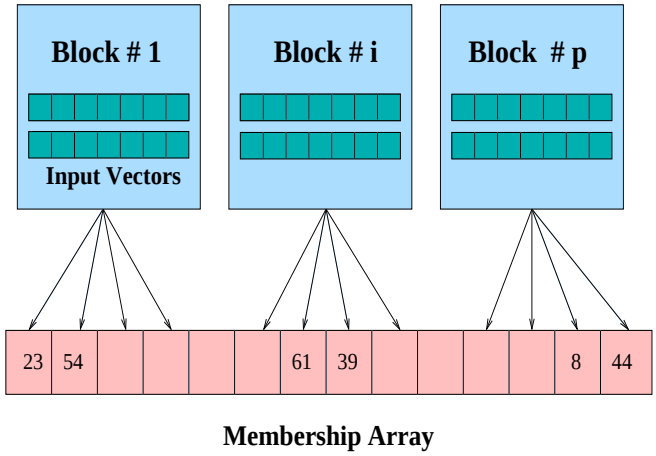

Figure 3.2 Layout of the CUDA blocks. Each block evaluates the new labels for q input vectors and writes to the membership array

3.1.2 Membership Evaluation

To generate new membership labels, we need to evaluate the distance of each input vector with all cluster centers. We process q input vectors in each CUDA block as shown in Figure 3.2. The membership kernel processes these using a $q \times d$ geometry for the block. The total number of blocks is $p = n/q$. The minimum square distance and the corresponding cluster number is stored for each of the q vectors in the shared memory of the block in a sequential manner processing say r centers at a time based on number of threads and amount of shared memory. Component-wise distances are evaluated in parallel and stored in the shared memory. The distance for a pair of vector and r cluster centers are evaluated using a log reduction of these values. This is compared with the current minimum distance for updation if necessary.

Algorithm 2 Membership Evaluation

$Input$: *I_Data*, *Centers*
$Output$: *Membership*
$t_i \leftarrow$ Thread Id in a block
$dim \leftarrow$ Vector dimension
$dist_{yz} \leftarrow$ Distance between vector y and center z
$min_y \leftarrow$ Minimum distance of vector y from a center
$s\_dist_{yz} \leftarrow$ Distance components

1: **for** $y = 1$ to n in parallel **do**
2: *membership_y*, *s_data_y*[*dim*]
3: $s\_data_y[] \leftarrow I\_Data_y[]$
4: **for** $z = 1$ to k clusters **do**
5: $s\_dist_{yz}[t_i] = (s\_data_y[t_i] - center_z[t_i])^2$
6: SyncThreads
7: **for** $i = dim/2$ to 0 **do**
8: **if** $Tidx \leq i$ **then**
9: Add $s\_dist_{yz}[2 * t_i + i]$ to $s\_dist_{yz}[t_i]$
10: **end if**
11: SyncThreads
12: $i \leftarrow i/2$
13: **end for**
14: $dist_{yz}$ = $s\_dist_yz[0]$
15: **if** $min_y \leq s\_dist_{yz}[0]$ **then**
16: $min_y \leftarrow s\_dist_{yz}[0]$
17: $Membership_y \leftarrow y$
18: **end if**
19: **end for**
20: **end for**

The distance and membership evaluation are performed using Algorithm 2. We have ignored the square root function for distance evaluation as it increases the computational cost. We store the following onto the shared memory: s_data holds the input vectors, s_dist stores square of the differences for every dimension, min_y and $membership_y$ store global minimum distance and label for vector id y. On Tesla GPUs, shared memory per SM was restricted to 16 KB. The centers were accessed via texture memory and l of them were loaded first onto the shared memory. The ql distance evaluations were done by the kernel then. The current generation GPUs (Fermi) have an L2 cache and thrice as much as shared memory. This helps in dedicating the shared memory for input vectors and distance evaluations. We load the cluster centers directly from the global memory. The L2 cache help us achieve good performance as all blocks access the same set of centers. We observed that the performance was good for 1500 centers of 128 dimensional vectors. The performance deteriorated sharply when the number of centers were increased. This is because L2 cache exceeded its limit and relied on global memory for accessing the additional centers. So on a Fermi(GTX 480), we send centers in batches of approximately 1500 making use L2 cache efficiently. It is, however, important to select the optimum block dimension,

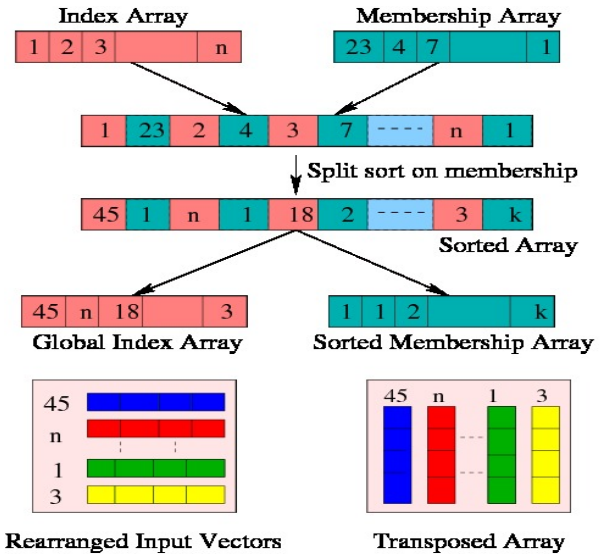

Figure 3.3 The membership array is sorted on the labels. The input vectors are then rearranged to bring those with the same labels together

number of vectors processed per block, shared memory utilization on a per block basis, etc., so that the GPU is efficiently utilized. The aim is to get better occupancy with more number of active SM's. The $syncThreads$ call ensures that threads in a block which have completed the task wait for other threads to finish the task before executing the next instruction.

3.1.3 New Center Evaluation

Center evaluation involves finding the sum of all vectors with the same label. For a parallel approach this task involves concurrent writes since data objects having the same membership may add to the same histogram bin at a time even to count the frequencies. Our contribution to earlier GPU implementations is performing the entire process of mean evaluation on the GPU itself. Algorithm 3 shows the sequence of kernel calls made for center evaluation. In our approach, the input data is partitioned into k clusters of different sizes to find the sum. We sort the input vectors based on their labels, using the *split* primitive [37]. The $SplitSort$ kernel brings the records with each label together. This is done by forming a list of 64-bit records combining the new label value and global index of the input vector. We split this using the label value as the key as shown in Figure 3.3, shuffling the original global index order. The *gather* primitive [37] is used subsequently to rearrange the input vectors in the order of labels using the index after the split. The gather primitive moves data efficiently, using coalesced read/write operations. We mark the cluster boundaries using the function $get\_Boundaries()$ based on the change in sorted labels. These are used to sum the vectors belonging to the same label using a segmented scan [43]. Using histogram() function we finalize the count of each cluster. The vectors with similar label are now

grouped together using $Rearrange()$ so that the list can be summed component-wise to compute the new centers.

The row major storage of the vectors make component-wise addition uncoalesced in memory accesses and hence inefficient on the GPUs. We solve this problem by converting the input vectors to a column major format, i.e., to a $n \times d$ arrangement from the $d \times n$ one. This is done using an efficient transpose operation [3] that uses the shared memory efficiently. The components are now arranged consecutively. Using transpose operation $Transpose()$ makes a significant contribution towards the mean evaluation. We use a segmented sum scan of the list of nd elements divided into kd segments. The kernel $Compact()$ then extracts the new means from the result obtained by segmented scan. Transpose operation is performed once again to revert back the values from row major storage. The final kernel call $Divide()$ gives us the new means for this iteration. All steps are performed using full occupancy and exploit the hardware well. Once we have the new means we check for the convergence condition which if failed we go back with these new centers and evaluate new labels again else we terminate the algorithm. Our mean evaluation was able to fix the concurrent write problem which the previous approaches failed to solve. Also the pure coalesced memory access for data rearrangement proved to be vital step in enhancing the mean evaluation on GPU.

Algorithm 3 Kernel Sequence for evaluation of New centers

Input: I_data, $Membership$
Output: $new\_centers$

1: $sorted\_Membership \leftarrow$ SplitSort(*Membership*, *Key*)
2: $flag\_hist \leftarrow$ get_Boundaries(*sorted_Membership*)
3: $hist\_scan \leftarrow$ Segscan(*Sorted_Membership*, *Flag_hist*)
4: $Histogram \leftarrow$ Histogram(*Hist_scan*, *Flag_hist*)
5: $data\_sorted \leftarrow$ Rearrange(*I_data*, *Sorted_index*)
6: $transp\_data \leftarrow$ Transpose(*Data_sorted*)
7: $flag\_scan \leftarrow$ get_Boundaries(*sorted_membership*)
8: $seg\_scan \leftarrow$ Segscan(*transp_data*, *flag_scan*)
9: $sum \leftarrow$ Compact(*seg_scan*)
10: $total\_sum \leftarrow$ Transpose(*sum*)
11: $new\_centers \leftarrow$ Divide(*total_sum*, *histogram*)

3.2 CellBE Implementation

Parallelization of K-Means on CellBE architecture is done by partitioning the input data set equally based on the number of Synergistic Processor Units (SPU) for load balancing. The boundary of each processor segment is aligned on a 16 byte boundary. Records must be padded accordingly. Large numbers of distance evaluations are to be performed in K-Means algorithm so all data objects and centers have to be sent to the SPU. The local storage is not sufficient to store all the centers. Since input is of large size it is wise to send input vectors to SPE only once and pass the center multiple times. For DMA

transfers of sizes greater than or equal to 16K bytes, the memory address in both the local store and main memory must be 16 byte aligned. Using SIMD we can calculate distance to multiple centers at once, distance between a center and multiple data points at once and multiple dimensions at once. We use two functions:

$$v_3 = vector\_subtract(v_1, v_2) \tag{3.1}$$

$$v_4 = vector\_multiply\_add(v_1, v_2, v_3) \tag{3.2}$$

Operation 3.1 subtracts vectors v_1, v_2 and stores the result in v_3 and Operation 3.2 multiplies elements in v_1, v_2 and adds the result to v_3 to store in v_4. The load is to be divided amongst the SPU equally. We divide the centers accordingly and store them till the end. The chunk of vectors assigned to a particular SPU are passed to it later. As the centers are accessed frequently for every input vector, we store the center in the SPUs local storage until all the input vectors assigned to that particular SPU are done with their processing. This way the DMA load can be reduced to great extent. We would have to pass the center vectors just once to the SPU and stream the input vectors accordingly. Most of the time is spent on computations when compared to that of pulling in memory. The CellBE PPU performs quite badly compared to CPU so one must off load the PPU as much as possible.

3.2.1 SPU for membership evaluation

For every vector assigned to a SPE we perform its distance evaluation from every center and decide its membership based on it. A chunk of vectors are assigned to the SPU along with its address, dimension, number of centers. Algorithm 4 gives distance evaluation on a SPU which is assigned R vectors. Using Algorithm 5 we compute the final labels for each input vector. When distances are evaluated we need to store a local distance minima and corresponding center id and update the global membership once all the centers have been processed by the SPU. If centers do not fit into the local store then we use stream.

Algorithm 4 Distance Evaluation on SPU

Input: *I_data*, *Center*
Output: *distance*

1: **for** $i = 0$ to *Dim*/4 **do**
2: *vector sub =vector_Subtract(I_data[i],Center[])*
3: *vector sum =vector_Multiply_Add*(*sub*,*sub*,*sum*)
4: $i + 1 \leftarrow i$
5: **end for**
6: *distance = sum[0]+sum[1]+sum[2]+sum[3]*

Algorithm 5 Membership on SPU

1: Get the chunk of data; centers to be loaded through DMA
2: For every vector of that SPU compute Distance(*V*,*C*)
3: Get the remaining centers from PPU
4: Compare distances for a vector against all the centers and assign membership to the center closest to it
5: Return the global membership to PPU

3.2.2 SPU for center evaluation

Unlike Buehrer *et al.* [6] we evaluate our mean evaluation on SPU instead of PPU. Mean evaluation on PPU is efficient for small n. For mean evaluation we send the vectors with similar memberships to a single SPU, and use SIMD intrinsic to evaluate their mean as shown in Algorithm 6. This requires gathering all the data objects belonging to a cluster. The center evaluation involves two steps grouping vectors with similar label and send them to SPU as per the labels. Since all the vectors sent to SPU bear same membership we just have to perform vector addition on the entire set of vectors. Once evaluation is done we send new centers to PPU. Since cluster had variable sizes the load could not be balanced amongst the SPUs. The new centers need to be sent to the PPU for verifying convergence condition. The memory usage is high for increasing clusters and input vectors.

Algorithm 6 Mean Evaluation on SPU

1: Get data objects belonging to same cluster and the histogram count.
2: Summation using intrinsics.
3: Evaluate the new mean taking average for each.
4: Return the new centers to PPU.

3.3 Results

We evaluated the performance of our approach on NVIDIA's 8600, GTX 480, IBM CellBE with 8 SPUs, and for sequential implementation used 32 bit Intel (2.4 GHz, 1Gb RAM). For the sequential results, we used standard Bardia Sardi's [15] implementation which has produced results for sequential K-Means. The input for our K-Means implementation were SIFT vectors. SIFT vectors are high dimensional vectors (128 dimensions) which represent a key point in an image, which are clustered to build vocabulary for classification purposes. These are generated using Lowe's [31] implementation. We made comparison with the latest approach on GPU. We comparing with a standard CPU implementation as a reference to show how the accelerators can enhance the performance in practice for someone using the CPU. The timings given below are average timing for 20 iterations. The iteration time includes labeling and new mean evaluation. In our experiments, we used K-Means++ for selecting initial clus-

Input Size: n	Number of Centers: k	CPU 2.4 GHz	GPU Tesla	GPU GTX 480	IBM CellBE
10 K	80	1.3	0.119	0.18	0.389
50 K	800	71.3	2.73	1.73	11.2
125 K	2,000	463.6	14.18	7.71	64
250 K	4,000	1320	38.5	27.7	149
500 K	4,000	5985	58.2	30.9	339
1 M	8,000	28936.2	268.6	177.61	1356

Table 3.1 Running time of our K-Means implementations per Iteration on GPUs, the CellBE, and sequential implementation on the CPU. The times given are in seconds.

ters. It is an expensive evaluation consuming a time equivalent to 2-4 iterations of K-Means for large n but it has found to drastically reduce the number of iterations. This extra time can be shared by all the iterations which is high for large input data. For an input size of 10K we observed it took almost 74 iterations to converge using random centers. Initialization using K-Means++ provided a 2-fold speed compared to initialization using random initial centers. We have performance timings for input sizes upto 1 million vectors of 128 dimension. By varying the cluster centers k, dimension d and input size n, we study the scalability of our approach.

Each CUDA block uses 256 threads in a row major format contributing two membership values after looping over all centers. For the Tesla GPU device, we process 2 vectors per block (i.e., $q = 2$). We loop over the cluster centers, taking two centers (i.e., $l = 2$) a time. With more shared memory on Fermi GPU, we process 2 input vectors per block (i.e., $q = 2$) looping over the cluster centers, taking four centers (i.e., $l = 4$) a time. The centers were accessed globally via texture memory for the Tesla and the global memory for the Fermi via L2 cache. The amount of shared memory used by 2 input vectors and performing 8 distance evaluations was 5136 KB. We achieved an occupancy of 83% for the above parameters on GTX 480. This combination of parameters gave the best performance in practice. The looping over all centers is the time consuming part of membership kernel.

On the IBM CellBE, We perform the membership on the SPU's by streaming the cluster centres in each iteration. Centre evaluation is also performed on the SPU by bringing vectors of selected labels to it. The SIMD intrinsics work very well for the large vectors we use. Buehrer *et al.* [6] made use of single instruction multiple data (SIMD) intrinsics. The architecture is slightly dated, but our results help to compare it with the GPUs. The running times shown in Table 3.1 provide a comparison between CPU, CellBE and different generations GPUs for different combinations of n and k. The speed-up obtained is from 170 on a GTX480 and almost 25 on CellBE. Figure 3.4 shows a plot of the time per iteration for different n with a fixed dimension of 128 and $k = 4000$. A single GPU provided a speed up ranging from 50-170 with increasing n. The variation of performance was nearly linear and scalable with increasing n.

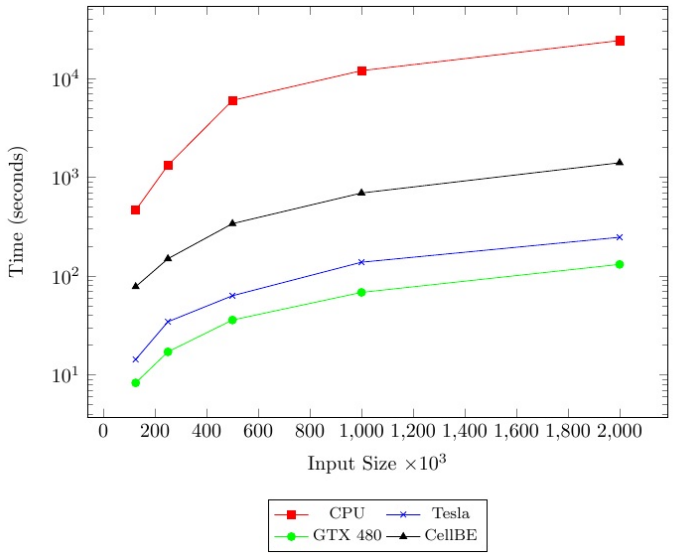

Figure 3.4 Running time per iteration in seconds for different input sizes for $d = 128$ and $k = 4$K.

Figure 3.5 shows the variation of running time with the dimension d. We can deduce that the performance varies nearly linearly with the input size and the number of dimensions. The distance evaluation time vary linearly with d and the number of distances linearly with n.

In Figure 3.6 we vary the number of cluster centers and found that as number of centers decrease the gap between GPU and CellBE was reducing mainly due to the fact that CellBE had to send the centers multiple number of times to the SPU but with less number of centers it was evident that its performance would improve drastically. So it could be said that performance of CellBE is highly dependant on number of cluster centers chosen and also the size of the input. It also shows the dependence of the running time on k, the number of cluster centers. The variation in performance was slightly more than linear. Increasing centers not only add up more work in membership evaluation step but also divergence in the mean evaluation on GPU.

On CellBE a speed up of 7.26 folds was achieved by increasing the number of SPUs from 1 to 8. In Figure 3.7 speed-up factor was dependent on number of SPUs used and variation with increasing number of SPU's gave almost a linear speed-up with increasing SPUs The performance time was almost linear also with respect to dimension of the vector.

K-Means involves large amount of distance evaluations. Most of them do not affect the current membership of the data object. We may reduce the computational load by terminating the distance computation for an input vector and a particular center as soon as the partial component-wise distance exceeds the local minima for the input vector. We then switch to the next center there by avoiding the unnecessary computation. This approach is referred as Lazy Evaluation. We added a lazy evaluations step in which the threads whose running sum of the distance exceeds the present minimum exiting early. The can eliminate unnecessary evaluations, but can introduce thread divergence on the GPUs. This is not very serious as the terminated threads only abstain from the computations. Table 3.2 compares the

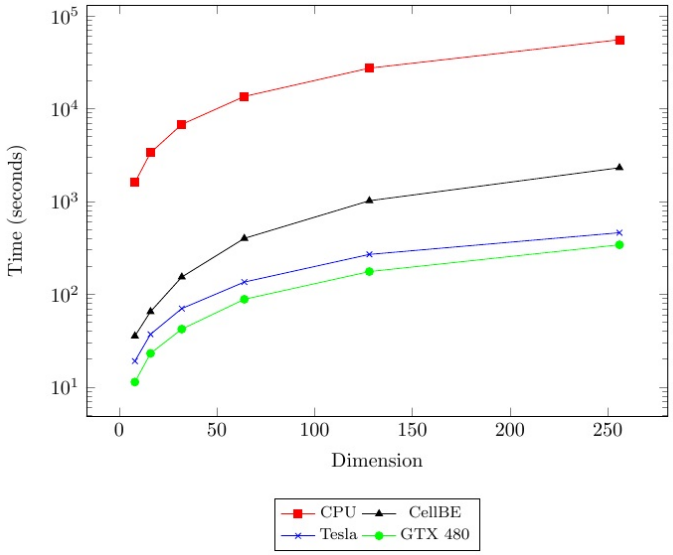

Figure 3.5 Running time per iteration in seconds for vectors of different dimensionality for n = 1M and k = 8K

Input Size: n	Number of Centers: k	Approach [20]	[20] with Lazy eval	Our Version of [20]
30 K	1,000	2.955	1.63	0.945
50 K	2,000	10.11	5.2	3.14
125 K	2,000	18.8	8.1	7.41
250 K	4,000	78.3	37.4	26.7

Table 3.2 Running time in seconds on a GTX480 of Hong-Tao *et al.* [20] approach with and without lazy evaluation and our approach for d=128

result of the implementations of the prior method with and without lazy evaluations to that of our own method. Our fully parallel implementation is faster by a factor of roughly 1.5 over the lazy evaluation method for large n and$k$. In few cases the lazy evaluations did catchup with our results. In case of small dimensional data lazy evaluations are not effective.

We compared our GPU approach with that of Li *et al.* [29] which claims to provide the best performance and also HP K-Means. Table 3.3 shows the comparison of various approaches on GTX 280 hardware. Unfortunately HP K-Means did not provide results for high dimensional data. The computation iterates over the dimensions and cluster centers. Our performance was nearly 4 times better than HP K-Means. For low dimensional data and centers Li *et al.* performance was marginally better but as the dimension, clusters increased our performance was almost better by a factor of 2.

Membership evaluation consumes a major percentage of total time of each K-Means iteration. Table 3.4 shows time division for label evaluation and new mean evaluation. The efficient mean evaluation

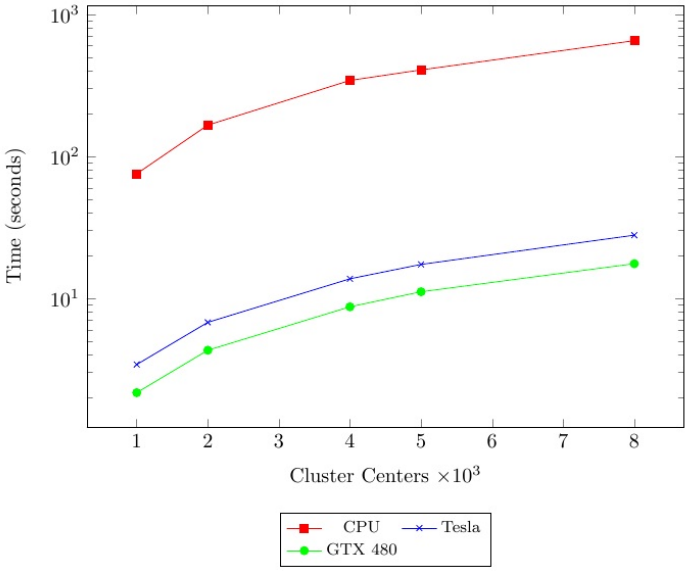

Figure 3.6 Running time per iteration in seconds for different centers for $n = 50$K and $d = 128$

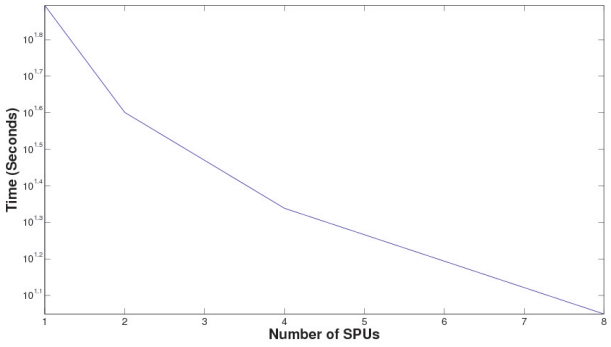

Figure 3.7 Running time by varying number of SPUs for 50k vectors of 128 dimension in 2k centers

performed on the GPU results in its time share to about 6% of total time for large input of high dimension. For low d, k the mean evaluation consumes major time but with increasing parameters its share reduces.

3.3.1 Performance: Discussions

The membership evaluation step is both computation and memory bandwidth intensive. Each input vector needs to compute its distance with each cluster centre. A single distance evaluation needs $3d$ floating point operations (one subtraction, one multiplication, and one addition to find the sum). The max finding uses K comparisons per vector. Thus the total computations are bound by $4NKd$. This comes to $4 \times 2^{20} \times 2^{12} \times 2^7$ or 2 terra floating point operations per iteration for 1M vectors of 128

Input Size: n	Dimension d	Number of Centers: k	Our K-Means	Li *et al.* [29] K-Means	Wu *et al.* [52] K-Means
2 Million	8	400	1.27	1.23	4.53
4 Million	8	100	0.734	0.689	4.95
4 Million	8	400	2.4	2.26	9.03
51,200	64	32	0.191	0.403	-
51,200	128	32	0.282	0.475	-

Table 3.3 Running time in seconds of our approach and those of Li *et al.* and Wu *et al.* on a GTX280.

n, d, k	CPU		1 Tesla		GTX 480	
	New Label	New Mean	New Label	New Mean	New Label	New Mean
50K, 32, 34	33.5	2.28	0.104	0.24	0.073	0.2
0.5M, 32,34	207.78	16.67	0.316	0.4	0.248	0.29
0.5M, 128,2k	2499	548	42.3	1.9	24.5	2.9
1M, 128, 4k	11864	2604	113	7.6	69.2	4.1

Table 3.4 Times for the labelling and mean evaluation steps per iteration in seconds

dimensions with 4K clusters. A single Tesla, with a peak compute power of 1 TFLOPS, should be able to perform these in 2 seconds if fully utilized. Membership evaluation also loads all K cluster centers from the global memory for each input vector. This results in a memory traffic of $4NKd$ bytes per iteration for 4-byte float data. This comes to 2 terrabytes of memory reads, which should take about 20 seconds if the rated peak bandwidth of 100GB per second can be sustained. Table 3.4 shows the total time to be about 113 seconds on 1 Tesla GPU. This shows that the average effective performance we obtain is off by a factor of 7 from the peak. This leaves the possibility for significant future speed-up by utilizing the resources more efficiently.

Chapter 4

Clustering on Multiple GPUs

K-Means is a highly computational and memory consuming algorithm. Overloading a single device with large amount of data to cluster will consume time and large data transfers even if performed in batches. A single GPU device has limited global memory, hence makes it hard to process very large number of high dimensional input vectors. Registers and shared memory also restrict accommodation of more number of vectors to process at a time. The scalability of the algorithm also becomes an issue with limited GPU memory.

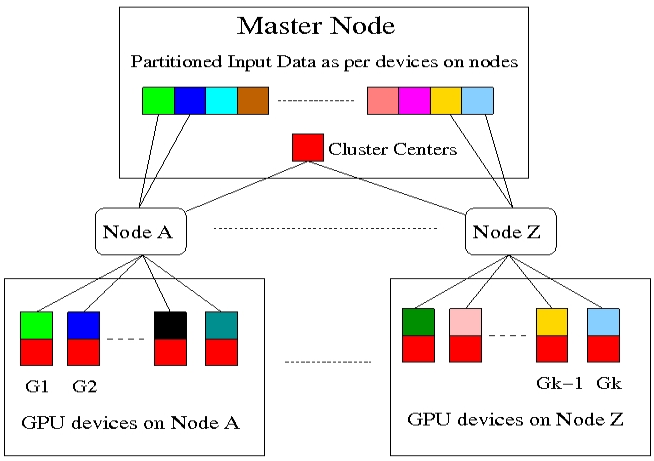

Figure 4.1 For Multinode, Multi-GPU configuration, the input vectors are partitioned among the GPUs of all nodes without duplication. The cluster centres are copied to each GPU in each node.

4.1 Multi-GPU Implementation

We propose an implementation on the multiple GPU's where in multiple threads efficiently compute label for a single data object. Algorithm 7 describes our approach using G_k GPU devices and Z nodes. Figure 4.1 shows the data partition amongst the G_k GPUs and Z nodes. The input vectors are partitioned uniformly among all available GPUs. When devices have different capabilities, the partitioning should

be done on the basis of the capability of the device like number of cores and available device memory. The cluster centres are copied to all GPUs as typically it occupies much less space. One of the CPUs serves as the master node (Figure 4.1) and distributes the data to other nodes. The cluster centers are broadcast to all nodes and copied to all GPUs at the start of each iteration. We can truly exploit the computational power of each device and balance the load distribution. This frame-work may be adopted in cases where we have large number of GPU devices available on multiple nodes.

Algorithm 7 Multi-GPU K-Means algorithm

Input: I_data, $Centers$, No of Nodes Z, No of GPUs - G
Output: $Membership$, $new\_centers$

1: Partition the input data into chunks propotional to number of cores, Global memory on each GPU for every node and transfer them to each of the GPUs.
2: Broadcast the k cluster centers to all the nodes, GPUs.
3: **for** G GPUs in parallel **do**
4: Perform membership evaluation using Algorithm 2 for own partition.
5: Perform mean evaluation using algorithm 3 to calculate partial centers on each device and send them back to respective Nodes.
6: **end for**
7: Every node performs summation of partial means collected from their respective GPUs.
8: Nodes send these partial sum to the Master node.
9: Perform final summation to evaluate new centers for the next iteration on the Master Node.
10: Check condition for convergence, if true then convergence is achieved else go back to step 2 with new centers.

Each GPU performs operations similar to the single GPU implementation. It first computes the new membership for each input vector in its partition. Then, the partial sums of its share of vectors for each of the K clusters is computed on the GPU. This data is sent to the CPU of the node along with the size of each new cluster. This data is accumulated at the CPU and send to the master node. The master node computes actual cluster centres from the data, which is broadcast to each GPU via its node CPU for the next iteration. This ensures that all $O(n)$ work is performed on the parallel GPUs while some of the $O(k)$ work is performed on the CPU. In practice, the work on the CPU is small compared to membership evaluation.

The number of clusters k is much smaller than the number of vectors n. Typically $k = O(\sqrt{n}))$ and hence it is fine to have all centres stored in each GPU. In Computer Vision, visual vocabulary is built using high values of n – in tens of millions – depending upon the application. The number of clusters is typically 5000 for such data. In each node, the GPU to CPU bandwidth is reasonably high, to the order of 4GB/s. The cluster centres consist of kd numbers, which comes to 2MB of data for 5K vectors of 128 dimensions. The partial sums can thus be sent in less than 1 millisecond to the CPU. Since the node CPU aggregates all data from its GPUs, an equal amount of data needs to be sent from each node to the master node. We used a commodity LAN to connect the network which takes 10-20 milliseconds for such a transfer. A proper HPC cluster will have better bandwidth using which message passing between nodes can be made efficient and faster. At the start of each iteration, the new cluster

Input Size: n	dim d	1 Tesla Device	GTX 480 Device	4×Tesla Devices	4×Teslas + GTX 480
1 M	128	120.4	73.3	34.6	22.8
1.5 M	128	181.7	95.6	47.2	34.8
3 M	128	364.2	-	94.671	67.4
6 M	128	-	-	183.8	136.7
16 M	16	220.4	-	57.8	40.9
32 M	16	-	-	116	84.3

Table 4.1 Running time per iteration in seconds for different input sizes on 1 T10, GTX480, 4× Tesla devices, and 4× T10 + GTX480 for k=4K.

centres need to be broadcast to all nodes, which involve exactly the same amount of data transfer as partial sums. Transferring input data across the nodes is a one time cost. The communication times are not significant for such large data sets as the membership evaluation takes several seconds even on the GPUs.

4.2 Results

We have performed our Multi-GPU implementation using $4 \times Tesla$ T10 devices and a GTX 480 card. Using MPI we performed node to node communication for data transfers. The load was divided as per the number of cores present in each of the devices to ensure load balancing and efficient utilization of resources. Table 4.1 shows a comparison of timings for increasing input size on single GPU and Multi-GPU devices. The blank values in the table are due to limited global memory on devices we were not able to run experiments on single GPU device for large datasets and had to rely on Multi-GPU approach.

The scalability in the number of cores of the GPU can be seen on low end card like 8600 with 32 cores and limited device memory (256 MB). We limit the input size to 200K vectors and compared the individual performance of 8600 (peak performance 100 GFLOPS) with that of Tesla (peak performance of 1TB) and GTX 480 (peak performance of 1.35TB). For low input size the performance of 8600 compared to other high end cards is shown in Figure 4.2. GTX 480 boosted the speed on an average by a factor of 1.5-1.8 compared to Tesla. The $4 \times Tesla$ T10 devices gave speed-up of nearly 3.8 compared to a single Tesla T10 device. The significant points are that the 8600 also was able to give good performance and the algorithm scales well with increasing number of cores.

We have used Open Message Passing Interface (MPI) for node to node communication in case of multi node multi GPU using 4×Tesla devices on node A and GTX 480 on node B. The node to node communication took place over local area network (LAN), using a SMP based cluster of nodes we may able to reduce the communication overhead. The scalability of n is dependant on total number of devices across the nodes. We maximized n up till the point where devices can handle data since they

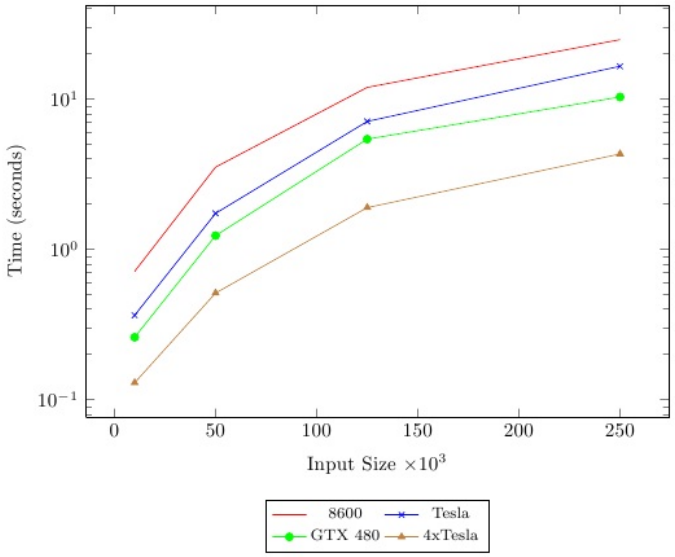

Figure 4.2 Running time on 8600, T10, GTX480 and $4 \times T10$ GPUs for various input values, d=128 and k=1k.

have global memory restrictions. So if we have large number of devices available we may scale our performance to a larger n and d. Dividing the input data based on cores present on a node ensured that GPU resources have been properly utilized achieving load balancing. This approach will be useful when we have cluster of GPU nodes with different computational capability.

Figure 4.3 shows the performance of Multi-GPU using $4 \times Tesla$ T10 devices for n upto 6 million. The variation is pretty much linear with increasing n. We were able to overcome the memory limitations faced on a single GPU device and able to scale our approach to n. Figure 4.4 shows the graph for variation of number of centers and how it affects the performance. The variation is mainly due to the architectural differences each of these devices have. The architecture has evolved from low end card 8600 to Fermi architecture. Features like shared memory, L2 cache, etc have added up to the performance. Also the implementation has been altered as per the GPU architecture, in case of Tesla the centers were accessed via Texture memory where as in Fermi via L2 cache.

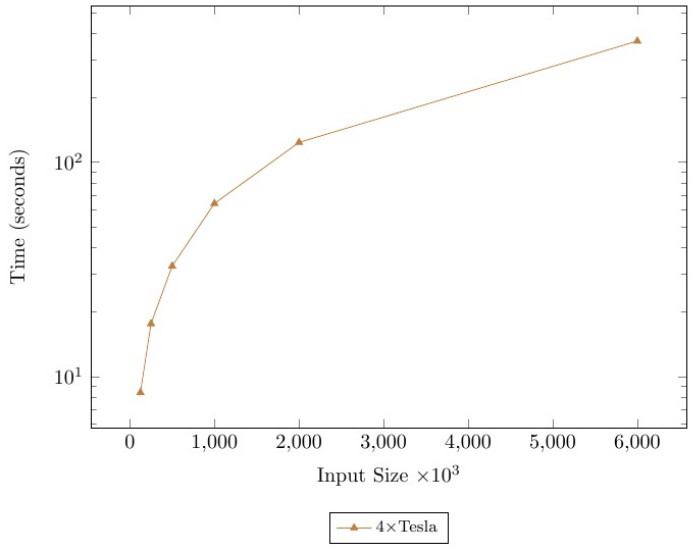

Figure 4.3 Running time per iteration in seconds for different number of input vectors for $k = 8$k and $d = 128$

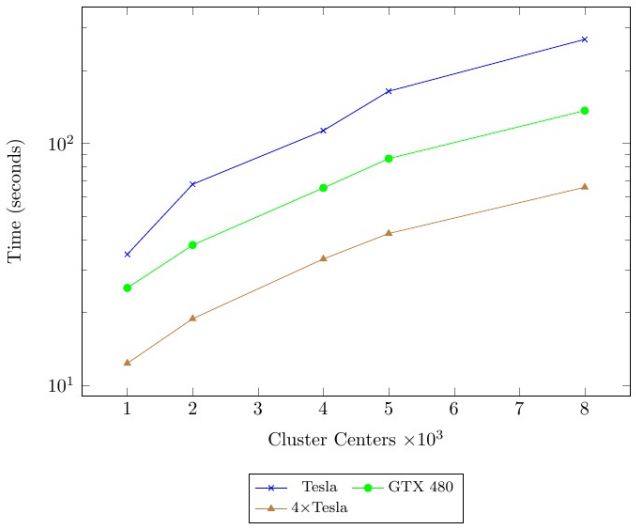

Figure 4.4 Running time per iteration in seconds for different centers for $n = 50$K and $d = 128$

Chapter 5

GPU Assisted Video Organizer

We present in this chapter the design and implementation of a scheme that helps categorize large collections of videos on a desktop or a laptop. We focus on the sports genre which has wide range of categories and will be a good test subject for our approach. Our categorization approach is an extension to videos of current scene classification [1, 26, 53] and object detection [28, 36] research carried out by computer vision community. The user annotates a few videos or parts of it by one of the category names which he wishes to use. Our scheme uses the Pyramidal Histogram of Oriented Gradients (PHOG) features [5] and their clustered collection as the representation for each category the user provides, which is computed by the system in the learning phase. A K-NN classifier and aggregation of votes by individual key frames are used to assign categories to the unlabeled collection in the categorization phase. The focus is on a scalable implementation in this work and not on pushing the state of the art on image/video classification.

5.1 Video Classification

Describing the entire video using a single descriptor, as is often done in images is not easy. Videos for many purposes can be considered as a collection of images which have a certain connection or pattern with each other. In our discussions, we use the term frame to refer to an image from the video stream. A video can be broken into small shots which are a collection of similar frames. Our classification approach has two parts: Category Determination and Category Assignment. We assume no off-line training data is available; so we need to form a basis for classification in order to categorize the remaining test videos. Also our training process adapts to the user's collections.

5.1.1 Category Determination

In this phase, we use a few videos tagged with their categories by the user. These are used to train our video organization system. Figure 5.1 shows the flow of the algorithm for this step. The representation formed at the end is the basis for categorizing the videos. We extract the key frames

using color histogram differencing. We remove small shots from the obtained result using a threshold on the number of frames to keep only major content. Each remaining shot is represented using a small number of key frames for further processing.

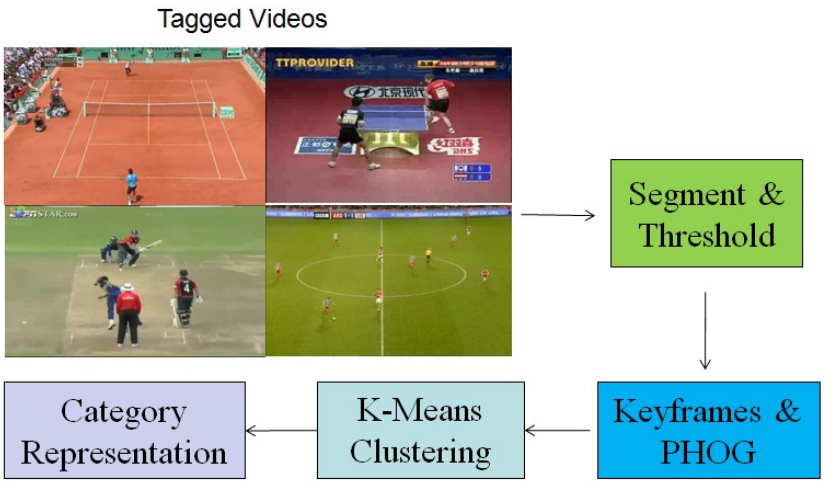

Figure 5.1 Category Determination: Algorithm Flow

We use PHOG feature descriptor. In our experience, video frames are best distinguished based on objects present in the frames like pitch in the case of cricket, net in the case of tennis, etc. In the image, each region is represented as a Histogram Of Gradients (HOG) as explained in [9]. The HOG vector is computed for each grid cell at each pyramid resolution level. The final PHOG descriptor for the image is a concatenation of all the HOG vectors. In forming the pyramid, the grid at level l has 2^l cells along each dimension. Level 0 is represented by a V-dimensional vector corresponding to V bins of the histogram, level 1 by a $4V$ vector, etc. Three spatial pyramid levels are used ($1 \times 1, 2 \times 2, 4 \times 4$). The PHOG vector is normalized to sum to unity. This normalization ensures that images with more edges or texture rich or larger are not weighted more strongly than others. The dimension of our PHOG descriptors is 640. We compute descriptors for all key frames of the tagged videos. The distance between two PHOG image descriptors then reflects the extent to which images contain similar shapes and corresponding in their spatial layout. We use K-Means clustering to group similar frames so that minor variations can be accounted for, similar to how visual words are formed in the object process of Video Google [45]. This results in a set of meaningful centers to represent each category. Our aim is to reduce the large number of descriptors to those that describe a particular category. We typically use 200 clusters to represent each video category. The cluster centers of form the representation for each category.

5.1.2 Category Assignment

The above representation is used to assign categories to other videos in the user's collection. Shot segmentation and PHOG extraction on user videos are carried out as described earlier. We use a K-

Algorithm 8 Category Assignment

Input: Test Keyframe Descriptors, Centers
Output: Category Label
l: number of key frames, *kF*: Key frame

for $y = 1$ to l in parallel **do**
  K-NN (kF$_y$) /* Returns K neighbors, distances */
end for
for $y = 1$ to l **do**
  $d1 \leftarrow$ distance(1^{st}-NN(kF$_y$))
  $d2 \leftarrow$ distance(2^{nd}-NN(kF$_y$))
  if $d1/d2 > r$ **then**
    Allocate kF$_y$ a single category, of Centre[1]
  else
    Allocate kF$_y$ to all K categories for the frame
  end if
end for
topper $\leftarrow$ Highest count among categories for video
runner-up $\leftarrow$ Next highest count
if topper is 20% more than runner-up **then**
  Assign video to category of the highest count
else
  Ask user for category assignment
end if

nearest neighbor (K-NN) approach using Euclidean distance to assign a label to each key frame of the video. Algorithm 4 shows flow for category assignment phase. Some of the frames may be classified wrongly due to the similarity in individual frames between different categories, such as the ground seen in different sports videos. We give all labels to each frame based on the distances to K nearest neighbors. K may be decided based on the closeness of these sport categories. We use a number that is close to half the number of categories for K in practice.

Lowe's criterion [31] considers the distances $d1$ and $d2$ ($d2 > d1$) to the first and the second nearest neighbors. A query and its closest neighbor are matched when the ratio $r = d1/d2$ between these two distances is below a threshold. This criterion is more robust than a global threshold on distances and behaves well when the structure to be matched is present exactly once in the candidate database. In our case, if the top match passes the ratio test, only one category is assigned to that frame. Otherwise, multiple labels are assigned to it as per the K nearest neighbors. In this manner, we are able to ensure that multiple close matches against the training frames are considered for final scoring. Each frame has one or more labels at the end of this process. We aggregate them for a video to give it a final label. We compute a category histogram for each video by combining the labels of its key frames. Frames with multiple labels contribute to multiple bins. We assign the most probable label to the video based on the final histogram, provided its score is clearly above all others. To assign a category based on score

we require that the top category of a video to have 20% more score than the next best category. We let the user decide the category manually if a clear label cannot be assigned using this criterion. Involving user at the end ensures even ambiguous videos are classified properly also in our view, preferable over mislabeling them. Fewer than 5% of the videos required user intervention in our experiments.

5.2 Implementation Details

The computationally intensive tasks like segmentation, PHOG feature extraction, K-Means clustering and K nearest neighbor are performed on GPU. Figure 5.2 indicates the division of work between CPU and GPU for different steps. We adapted the histogram sample code provided by NVIDIA for segmentation and key frame extraction. After the histograms are evaluated, we perform difference between consecutive frames to mark boundaries for shots. PHOG feature descriptors are evaluated on GPU using the approach by Prisacariu and Reid [39]. Their approach uses one thread per pixel and the thread block size is 16×16. Trilinear interpolation is used in cell/block configuration and pixel contributes to up to 4 histograms (one for each cell), and up to 2 bins per histogram. To compute the color gradients, we use two separable convolution kernels similar to the ones from the NVIDIA CUDA SDK. There are two kernels, the first kernel convolutes the row with the centered 1-D mask, while the second kernel computes the column convolution, gradient orientations and magnitudes. HOG is computed for different scales and then merged to get PHOG descriptors. Using the results we get from K-NN we perform the final scoring on CPU.

5.2.1 K-Means on GPU

We use our own GPU implementation of K-Means as referred in previous chapter. Fast GPU based clustering was used in order to organize similar frames extracted from all the videos. Our implementation is divided into two parts: membership evaluation and mean evaluation. We extended parallelism to the computation done on the d components of each input and center vector. Also the mean evaluation is performed in a coalesced manner. The detailed implementation has already been discussed in chapter

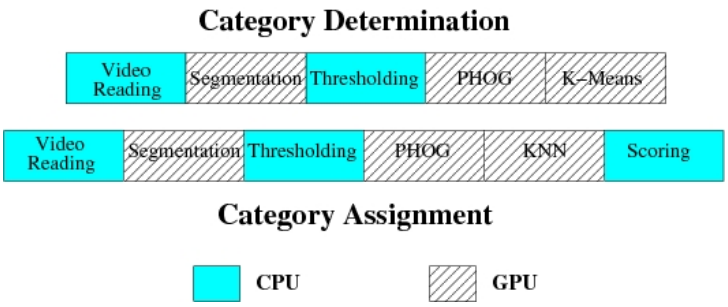

Figure 5.2 Work division between CPU and GPU.

3. In this case we deal with high dimensional PHOG vectors (520). Using our implementation which scales well with increasing dimension provided quick results clustering the set of frames.

5.2.2 K-NN on GPU

Algorithm 9 KNN in parallel for all keyframes

1: Each block handles l new keyframes at a time loops over all keyframes
2: Find distances for each keyframe against all centers sequentially using Algorithm 5.1.2
3: k distances are in the global memory
4: Sort the distances using Splitsort for each keyframe

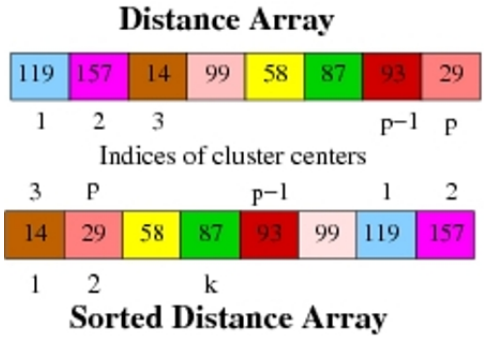

Figure 5.3 The distance array is sorted as per the distance values. The top k values represent the K Nearest Neighbors

The basic process of K-NN algorithm involves finding K nearest neighbors for each keyframe of our test video. To achieve this, we find the distances to the given centers for each test vector and sort them on a distance criteria. The class label of the point is given by the labels of the closest K vectors. For K nearest neighbor evaluations, we use Algorithm 9. Distance of a vector from a centre is evaluated using Algorithm 2. Data elements which are frequently accessed are stored onto the shared memory. We store the following onto the shared memory: s_data holds the input keyframe vector, s_dist stores square of the differences for every dimension, min_y and $membership_y$ store the global minimum distance and label for vector id y. The $syncThreads$ ensures that threads in a block which have completed the task to wait for other threads to finish the task before executing the next instruction. We have all the distances for the testing frames from the centers. The number of testing frames is typically large. We use the SplitSort operation to get K nearest centers. We sort the index of center based on their distance from a single data object, using the *split* primitive [37]. This is done by forming a list of 64-bit records combining the distance and center index. We split this using the distance value as the key as shown in Figure 5.3, shuffling the original center index order. After sorting the centers index based on distance we send the top K centers from the sorted list to the CPU for the final categorization.

GPU Device	No of Videos	No of Key frames	Segmenting video	PHOG features	K-Means Clustering
8600	4	756	182.7	139.6	3.94
8600	12	2432	584.3	468.4	14.3
280	4	756	24.8	19.2	0.59
280	12	2432	76.9	61.8	1.97
580	4	756	11.8	9.1	0.26
580	12	2432	37.91	30.2	0.89

Table 5.1 Time taken in seconds to process the Category Labeling phase on NVIDIA 8600, GTX 280 and GTX 580 cards

5.3 Results

We worked on a collection with four sport category videos, including cricket, tennis, football, table tennis, for our initial experiments. We downloaded highlights of these sports from popular websites like You Tube, Vimeo, Metacafe, etc., ranging over a period of 5-6 years. The videos typically have 30 frames per second and range from 8-15 minutes of duration. For our experiments, we took 100 videos belonging to 4 categories. Each video roughly had 20K frames. The user tagged 12 videos belonging to four categories. We performed our experiments using computers which have a dual-core Intel CPU and a GPU. A high-end Nvidia GTX 580 and a low end Nvidia 8600 GPUs under the CUDA programming model were tried to gauge the effect of performing the classification on a desktop and on a laptop. In case of cricket videos, we observed that the following shots emerged as key frames: Full screen Score card, Pitch, Wicket celebration, Boundary, Focus on a player, Hawkeye prediction and Crowd (Figure 5.4). Even in case of football we found such similar frames (Figure 5.5) of player positioning, goal post, goal celebration, football trajectory, crowd, etc. Pretty similar was the case for other categories. During the category determining phase, we performed video segmentation, PHOG evaluation, and K-Means clustering on the GPU device. Instead of representing a shot by single frame we take 4 frames to represent a single shot. In this manner, we get an entire summary of the shot consisting of adequate details. Figure 5.6 shows the frames extracted during the category determining phase using the tagged information. Similar frames extracted from every video are clubbed together using clustering. We perform 10 iterations of K-Means on each category to get representative centers for each. The final representation consists of about 200 key frames for each category, distributed evenly.

During the labeling phase we perform the video segmentation, PHOG feature descriptor, K-nearest neighbor on the GPU. Table 5.1 shows the time taken for each stage of training process for 4 videos of one category and 12 videos of all categories by the user. The times in seconds are shown on a low-end (8600) and a hig-end (GTX 280, GTX 580) GPU showing that the algorithm can be scaled to number of cores. The comparison shows that even the low-end users can benefit from our application.

Table 5.2 shows the processing time of K-NN algorithm for the category assignment phase on 8600, GTX 280 and GTX 580. The time consuming steps in our application are the video segmentation and PHOG feature extraction. The ratio test was useful for frames which had close ups of an individual or the crowd. Such frames were labeled in multiple categories. Frames consisting of the cricket pitch, the table in tennis table, etc., were classified clearly.

GPU Device	No of Videos	No of Key frames	K-NN
8600	88	16946	40.33
280	88	16946	5.39
580	88	16946	2.46

Table 5.2 Time taken in seconds for K-NN during the category assignment phase on NVIDIA 8600, GTX 280 GTX 580 cards

In Table 5.3 we see the percentages of individual frames which are correctly classified for different number of neighbors for a single test video in each category. In case of 3 neighbors, we label the category of a frame based on the majority neighbor criteria. In cases where there was no majority the frame may belong to any of the 3 neighbor categories. Improvement could be seen in categories which have frames consisting of field, players, etc. Not much improvement could be seen in table tennis as majority of the frames consisted of the table views which were easy to match. Using K-NN was beneficial in boosting the classification of frames. We did have some misclassification in certain categories like football due to lack of information from majority of the key frames which have field, players in it. Our tool is able to organize a set of 100 sport videos of total duration of 1375 minutes in about 9.5 minutes. The process of learning the categories from 12 annotated videos of duration 165 minutes took 75 seconds. We achieved an accuracy of nearly 96% on our testing dataset.

Figure 5.4 Key Frames for Cricket video

Figure 5.5 Key Frames for Football video

Figure 5.6 Training Frames accumulated for various categories

No of Neighbors	Cricket Videos	Football Videos	Tennis Videos	TT Videos
1	64%	58%	69%	82%
3	73%	66%	77%	84%

Table 5.3 The percentage of frames correctly classified using K-NN

Chapter 6

Conclusions

We presented the design and implementation of the K-Means clustering algorithm entirely one or more GPUs in this thesis. We achieved high performance on different GPU generations using the massive parallelism supported by the CUDA model. Our implementation is scalable in the problem size, the number of dimensions, the number of centers, and the number of available cores on the available GPU. The multi-GPU approach produced nearly linear speed-ups with large data. Our multi-GPU framework can efficiently solve the clustering problem using multiple nodes each with multiple GPUs. Even low end GPUs found on laptops are shown to provide speed-ups of 10-20 compared to the CPU version. The performance we achieve are indicators of the performance that can be obtained on future accelerators, in our view. Such accelerators are becoming more common and are likely to play key roles in different computation steps performed by individuals on their PCs. Scalability to large vectors and problems is important as all aspects of data handled by even individual users is growing very fast. With architectures evolving there will always be scope to improve our approach taken for K-Means algorithm. Work could be carried out in developing a hybrid many-core implementation utilizing the resources of both GPU and CPU.

We also presented the design and initial results from a GPU-assisted system to organize a personal collection of videos. We used a combination of the CPU and the GPU to get good computational performance. Using GPU K-Means approach we developed implementation for K-NN algorithm. We used simple methods adapted from object recognition literature to classify video frames. In future we would like to use more sophisticated methods from the visual object detection and scene classification literature in the future. We have to be selective about this as computational requirements have to be kept reasonable for any personal system. We intend to test the system on larger databases with more categories popular in personal videos. In future, with people having thousands of videos, GPU devices on their machines such application will always be handy for maintaining an organized collection of personal videos.

GPU is a good platform and its architecture has evolved making it suitable for wider range of applications. These graphics cards are highly available in the current market and will certainly play a major role in time to come. Usage of both CPU and GPU at the same time, will surely be an approach for ef-

ficient resource utilization. We have certainly made a beginning in developing GPU based applications. In future wide range of GPU based applications on personal laptops/desktops can be developed.

Related Publications

K Wasif Mohiuddin, P. J. Narayanan. Scalable Clustering using Multiple GPUs, International Conference on High Performance Computing (HiPC), December 2011.

K Wasif Mohiuddin, P. J. Narayanan. A GPU-Assisted Personal Video Organizing System, ICCV Workshop on GPU in Computer Vision Applications, November 2011.

Bibliography

[1] F. akir, U. Gdkbay, and z. Ulusoy. Nearest-neighbor based metric functions for indoor scene recognition. *Computer Vision and Image Understanding*, 115(11):1483–1492, 2011.

[2] D. Arthur and S. Vassilvitski. K-Means++: The advantages of careful seeding. In *Symposium on Discrete Algorithms*, 2007.

[3] M. Bader, H. Bungartz, D. Mudigere, S. Narsimhaan, and B. Narayanan. Fast gpgpu data rearrangement kernels using cuda. In *HiPC - High Performance Computing Student Symposium*, 2009.

[4] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, 2003.

[5] A. Bosch and A. Zisserman. Representing shape with spatial pyramid kernel. In *ACM International Conference on Image and Video Retrieval*, CIVR, 2007.

[6] G. Buehrer and S. Parathasarathy. The potential of cell broadband engine in data mining. In 33^{rd} *Int. Confernce on Very Large Databases(VLDB)*, 2007.

[7] S. Che, M. Boyer, J. Meng, and D. Tarjan. A performance study of general-purpose applications on graphics processors using cuda. In *Journal of Parallel and Distributed Computing*, pages 1370–1380, 2008.

[8] D.-Y. Chen and P.-C. Huang. Motion-based unusual event detection in human crowds. *Journal of Visual Communication and Image Representation*, 22:178–186, 2011.

[9] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 1*, CVPR, pages 886–893, 2005.

[10] H. K. Ekenel, T. Semela, and R. Stiefelhagen. Content-based video genre classification using multiple cues. In *Proceedings of the 3rd international workshop on Automated information extraction in media production*, AIEMPro, pages 21–26, 2010.

[11] C. Elkan. Using the triangle inequality to accelerate K-Means. In *International Conference on Machine Learning*, pages 147–153, 2003.

[12] J.-M. Frahm, P. Fite-Georgel, D. Gallup, T. Johnson, R. Raguram, C. Wu, Y.-H. Jen, E. Dunn, B. Clipp, S. Lazebnik, and M. Pollefeys. Building rome on a cloudless day. In *Proceedings of the 11th European conference on Computer Vision: Part IV*, ECCV'10, pages 368–381, 2010.

[13] Y. Frishman and A. Tal. Multi-level graph layout on the gpu. *IEEE Transactions on Visualization and Computer Graphics*, 13:1310–1319, 2007.

[14] J. D. Hall and J. C. Hart. Gpu acceleration on iterative clustering. In *SIGGRAPH poster.*, 2004.

[15] Har-Peled and B. Sadri. How fast is the k-Means method? In *Proceedings of the* 16^{th} *ACM-SIAM Symposium on Discrete algorithms,*, pages 877–885, 2005.

[16] P. Harish and P. J. Narayanan. Accelerating large graph algorithms on the gpu using cuda. In *Proceedings of the 14th international conference on High performance computing*, HiPC'07, pages 197–208, 2007.

[17] Y. He and C. H. Ding. Mpi and openmp paradigm on cluster of smp architectures: The vacancy tracking algorithm for multi-dimensional array transposition. In *International Workshop on OpenMP Application and Tools WOMPAT*, volume 27, page 185, 2003.

[18] S. Heymann, B. Frhlich, F. Medien, K. Mller, and T. Wiegand. Sift implementation and optimization for general-purpose gpu. In *WSCG*, 2007.

[19] T. Hofmann. Probabilistic latent semantic indexing. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '99, pages 50–57, New York, NY, USA, 1999. ACM.

[20] B. Hong-tao, H. Li-li, O. Dant-ong, L. Zhan-shan, and L. he. K-Means on commodity gpu's with cuda. In *Proceedings of WRI World Congress on Computer Science and Information Engineering.*, volume 3, pages 651–655, 2009.

[21] IBM. Cellbe programming tutorial v 3.0, 2007.

[22] H. Inoue, T. Moriyama, H. Komatsu, and T. Nakatani. Aa-sort: A new parallel sorting algorithm for multi-core simd processors. In *Proceedings of the 16th International Conference on Parallel Architecture and Compilation Techniques*, PACT, pages 189–198, 2007.

[23] P. Kanase-Patil, A. Mittal, and K. Singh. Implementation of minimum spanning tree algorithm on cellbe. In *Proceedings of the 2010 International Conference on Advances in Computer Engineering*, ACE, pages 281–283, 2010.

[24] S. Lahabar and P. J. Narayanan. Singular value decomposition on gpu using cuda. In *Proceedings of the IEEE International Symposium on Parallel&Distributed Processing*, IPDPS '09, pages 1–10, 2009.

[25] W. Lao, J. Han, and P. H. N. de With. Automatic sports video analysis using audio clues and context knowledge. In *Proceedings of the 24th IASTED international conference on Internet and multimedia systems and applications*, pages 198–202, 2006.

[26] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2 of *CVPR*, pages 2169–2178, 2006.

[27] K. Lebart, C. Smith, E. Trucco, and D. Lane. Automatic indexing of underwater survey video: algorithm and benchmarking method. *IEEE Journal of Oceanic Engineering*, 28(4):673 – 686, 2003.

[28] B. Leibe, A. Leonardis, and B. Schiele. Robust object detection with interleaved categorization and segmentation. *International Journal of Computer Vision*, 77:259–289, May 2008.

[29] Y. Li, K. Zhao, X. Chu, and J. Liu. Speeding up K-Means algorithm by gpus. In *Proceedings of the 2010 10th IEEE International Conference on Computer and Information Technology*, CIT '10, pages 115–122, 2010.

[30] T. Liu, H. Zhang, and F. Qi. A novel video key-frame-extraction algorithm based on perceived motion energy model. *IEEE Transactions on Circuits Systems for Video Technology*, 13(10):1006–1013, 2003.

[31] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60:91–110, November 2004.

[32] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium of Mathematical Statistics and Probability*, pages 281–297, 1967.

[33] K. P. Murphy, A. Torralba, D. Eaton, and W. T. Freeman. Object detection and localization using local and global features. In *Toward Category-Level Object Recognition*, pages 382–400, 2006.

[34] NVIDIA. Nvidia cuda compute unified device architecture programming guide 2.1, 2008.

[35] V. Osipov, N. Leischner, and P. Sanders. Nvidia Fermi Architecture White Paper, 2009.

[36] C. Papageorgiou and T. Poggio. A trainable system for object detection. *International Journal of Computer Vision*, 38:15–33, 2000.

[37] S. Patidar and P. J. Narayanan. Scalable split and sort primitives using ordered atomic operations on the gpu. In *IIIT/TR/2009/99. Tech. Rep*, 2009.

[38] D. Pelleg and A. Moore. Accelerating exact K-Means algorithm with geometric reasoning. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (SIGKKD)*, 1999.

[39] V. Prisacariu and I. Reid. FastHOG - a real-time GPU implementation of HOG. Technical Report 2310/09, Department of Engineering Science, Oxford University.

[40] N. Rea, R. Dahyot, and A. C. Kokaram. Classification and representation of semantic content in broadcast tennis videos. In *IEEE International Conference on Image Processing*, ICIP, pages 1204–1207, 2005.

[41] M. S. Rehman, K. Kothapalli, and P. J. Narayanan. Fast and scalable list ranking on the gpu. In *Proceedings of the 23rd international conference on Supercomputing*, ICS '09, pages 235–243, 2009.

[42] M. J. Roach, J. D. Mason, and M. Pawlewski. Video genre classification using dynamics. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, 2001. Volume 03*, ICASSP, pages 1557–1560, 2001.

[43] M. H. S. Sengupta, Y. Zhang, and J. Owens. Scan primitives for gpu computing. In *Proceedings of Graphics Hardware*, 2007.

[44] S. A. Shalom, D. Manoranjan, and T. Minh. Efficient K-Means clustering using accelerated graphics processors. In *DaWaK- Data Warehousing and Knowledge Discovery*, pages 166–175, 2008.

[45] J. Sivic and A. Zisserman. Video google: A text retrieval approach to object matching in videos. In *Proceedings of the Ninth IEEE International Conference on Computer Vision - Volume 2*, ICCV '03, pages 1470–1478, 2003.

[46] B. Tomasik, P. Thiha, and D. Turnbull. Tagging products using image classification. In *Proceedings of the 32nd international ACM SIGIR conference on Research and Development in Information Retrieval*, pages 792–793, 2009.

[47] B. T. Truong, S. Venkatesh, and C. Dorai. Automatic genre identification for content-based video categorization. In *International Conference on Pattern Recognition*, ICPR, pages 4230–4233, 2000.

[48] S. Vakkalanka, C. Krishna Mohan, R. Kumaraswamy, and B. Yegnanarayana. Combining multiple evidence for video classification. In *Proceedings of 2005 International Conference on Intelligent Sensing and Information Processing*, pages 187 – 192, 2005.

[49] V. Vineet and P. J. Narayanan. Cuda cuts: Fast graph cuts on the gpu. *Computer Vision and Pattern Recognition Workshop*, 0:1–8, 2008.

[50] R. Weber and P. Zezula. The theory and practice of searches in high dimensional dataspaces. In *Proceedings of the Fourth DELOS Workshop on ImageIndexing and Retrieval*, 1997.

[51] F. Wenbin, L. Ka, Keung, L. Mian, X. Xiangye, L. Chi, Kit, Y. Philip, H. Bingsheng, L. Qiong, S. Pedro, V, and Y. Ke. Parallel data mining on graphic processors. In *Technical Report HKUSTCS08.*, 2008.

[52] R. Wu, B. Zhang, and M. Hsu. Clustering billions of data points using gpus. In *Proceedings of the combined workshops on UnConventional high performance computing workshop plus memory access workshop*, UCHPC-MAW '09, pages 1–6, 2009.

[53] J. Yang, Y.-G. Jiang, A. G. Hauptmann, and C.-W. Ngo. Evaluating bag-of-visual-words representations in scene classification. In *Proceedings of the International Workshop on Multimedia Information Retrieval*, MIR '07, pages 197–206, 2007.

[54] M. Zechner and M. Granitzer. Accelerating K-Means on the graphics processor via cuda. In *International Conference on Intensive Applications and Services.pp. 7-15.*, 2009.