Efficient Parallel Image Clustering and Search on a Heterogeneous Platform

Dong Ping Zhang[1], Lifan Xu[2], and Lee Howes[1]

[1]Research Group, Advanced Micro Devices, [2]Department of Computer and Information Sciences, University of Delaware

Keywords: Parallel GIST descriptor generation, Hierarchical K-means, hybrid vector-based breadth first search (vBFS), OpenCL image search, energy efficiency

Abstract

We present a parallel image clustering and search framework for large scale datasets that does not require image annotation, segmentation or registration. This work addresses the image search problem while avoiding the need for user-specified or auto-generated metadata. Instead we rely on image data alone to avoid the ambiguity inherent in user-provided information.

We propose a parallel algorithm exploiting heterogeneous hardware resources to generate global descriptors for the set of input images. Given a group of query images we derive the global descriptors in parallel. Secondly, we propose to build a customisable search tree of the image database by performing a hierarchical K-means (H-Kmeans) clustering of the corresponding descriptors. Lastly, we design a novel parallel vBFS algorithm to search through the H-Kmeans tree and locate the set of closest matches for query image descriptors.

To validate our design we analyse the search performance and energy efficiency under a range of hardware clock frequencies and in comparison with alternative approaches. The result of our analysis shows that the framework greatly increases the search efficiency and thereby reduces the energy consumption per query.

1. INTRODUCTION

Web-scale multi-media search is the key challenge that search engines and social networking sites are trying to tackle after the text search domain has become relatively mature and well established. With rapidly growing volume of multi-media data, image clustering and search both in terms of service quality and energy consumption have become increasingly important for the purpose of scene reconstruction, object recognition, copyright attack detection, robotic vision, adverts placement and so on. Moreover, supporting web-scale image search is not cheap, it often requires a large amount of hardware with significant energy consumption, which in turns leads to a high cost per query for the service provider.

In this paper, we present a parallel image clustering and search scheme that exploits the heterogeneity available in most hardware resources consisting of both throughput-oriented (e.g., GPUs) and latency-oriented (e.g., CPUs) components but yet not fully utilised in the large scale image search context. We evaluate both the performance and energy efficiency of our approach. We finish by discussing the potential benefits from the co-design of applications and general-purpose computer architecture for large scale image analysis.

2. BACKGROUND AND MOTIVATION

The large scale general image search project addresses the following problem: given a query image, find in a large set of images the ones that represent the same object or location that is possibly viewed from different viewpoints and in the presence of occlusions and clutter as compared with the query image. Typical applications include finding images containing particular objects as well as detecting full or partial image duplicates and deformed copies.

In this work, we do not consider the specific visual recognition tasks or semantic queries. Face detection and recognition is an example for the former category, for that purpose, specific descriptors and detectors have been developed in literature, e.g., Viola et al. [1]. For the latter category, queries like "retrieve all images containing a dog" or other visual object recognition tasks are not in our scope. For general image search, our goal is to find existing images in a large dataset that are identical or most similar to a list of query images and to achieve this with high computational and energy efficiency.

Related Work

There is a wealth of literature on the topic of image search. Currently large scale image search [2, 3, 4] is mainly performed with the "bag of visual words" scheme, relying on local desriptors together with indexing and retrieval schemes. Search based on global image data is constrained mainly to a relatively smaller scale, due to its much higher requirement on the memory system and computational resources. In a recent work [5], global descriptor has been evaluated for web-scale image search, however, in this work, the global descriptor has to be computed off-line before the image classification, search and verification steps.

The image descriptors consist of various categories: global descriptor (e.g., GIST [6]), local descriptor (e.g., SIFT [7], HoG [8], ORB [9], SURF [10]), or other regional descriptor. Regional descriptors may include the basic properties of a region: average greyscale, variance, color, object area etc. We focus on the global and local descriptors in this work.

While the global descriptor can represent the image relatively efficiently both in terms of storage space and its computational cost, global-descriptor-based image comparison can be less robust and less discriminant compared with local approaches. For example, global descriptor can be sensitive to transformations, changes of viewpoint, illumination changes, cluttering, cropping and occlusion. Often the local approaches can be very expensive to compute and its feature sets are often much larger than a global descriptor for storage. The insensitivity of local descriptors to rotation and scaling, and robustness towards other formats of image variations gained it popularity in image search domain, in particular as the main component of bag of features (BoF) approaches (e.g., [11]).

To this end, our work presents a hybrid method that can combine the merits of both global and local descriptors yet without their drawbacks, in a heterogeneous system.

GPU Architecture

GPU architectures generally follow a throughput computing design with a large number of ALUs operating in wide SIMD arrays, or using significant amount of hardware multiplethreading or both. The graphics processor in this work is based on AMD's Southern Islands (SI) GPU architecture [12], shown in Figure 1(a) as a high-level overview.

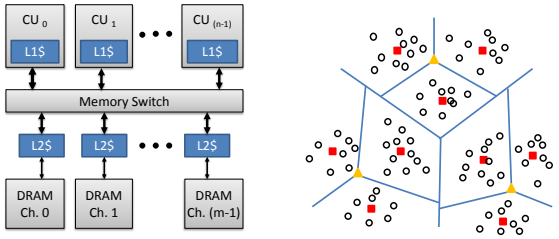

Figure 1: (a): An overview of SI GPGPU architecture; (b): An illustration of H-Kmeans with $L = 2$ and $K = 3$

The computing resources are grouped into Compute Units (CU), which are conceptually similar to CPU cores. Each CU includes four 16-wide SIMD units with registers and vector ALUs private for each SIMD. It also consists of a scalar unit, an instruction scheduler, a read/write L1 cache and a shared local memory (LDS). The off-chip DRAM is organised as a set of memory channels, each with an associated slice of the L2 cache. For OpenCL terminology, execution, memory and programming model, we refer to [13].

3. METHODOLOGY

We present a hybrid image search mechanism for heterogeneous systems. This work addresses the image search problem while requiring no additional metadata or tags to be specified by users. By using image data alone it eliminates any dependency or ambiguity on user-provided information. As the number of images increases in the dataset, brute-force search requiring the exhaustive pair-wise comparison becomes impractical and also the two views of same images may be evaluated with significant difference by a distance based similarity metric without registration that typically incurs high computational cost. To address this issue for the image search, we propose a new parallel processing paradigm for large-scale image clustering and search, including four main components, each of which is a parallel process and is detailed in this section:

- Fast and efficient representation of images using a global image descriptor: parallel GIST descriptor;
- Hierarchical K-means classifications to identify the hierarchical clusters and to form the search tree;
- A hybrid vector-based breadth first search (vBFS);
- Final verification based on local descriptor: SURF.

A coarse classification of the image database into multiple hierarchical clusters facilitates the search efficiency. In a nutshell, we propose to first compute the global descriptor and perform coarse-level classification and image search to reduce the search space and cost for a local descriptor approach significantly; secondly, a local descriptor approach is deployed to perform more localised match and verification.

3.1 Parallel GIST Descriptor Generation

The color GIST descriptor [6] introduced the concept of spatial envelope as a holistic descriptor of the main structure of an image, to define what a "scene" is, as opposed to an "object". This work is motivated by human capability of instant recognition of the "gist" of an image as quickly and as accurately as a single object. During the viewing process, the human brain extracts enough visual information to accurately recognize the functional and categorical properties of an image, and overlooks the details of the objects and their locations for fast scene recognition. The spatial envelope model provides a holistic description of the scene where local object information is not taken into account. In this work, we use the spatial envelope attributes to compute the similarities between two scenes, since these attributes provide a meaningful representation of the space captured in the images.

The GIST descriptor represents the global scene with a much more condensed descriptor compared with the local image descriptors (e.g., SIFT [7], SURF [10], HoG [8]). The local image description is founded on the premise that images can be characterized by attributes computed on local regions of the image. It represent an image with a number of feature vectors in a high dimensional space, which often requires orders of magnitude more storage space and computational cost than the GIST global descriptor.

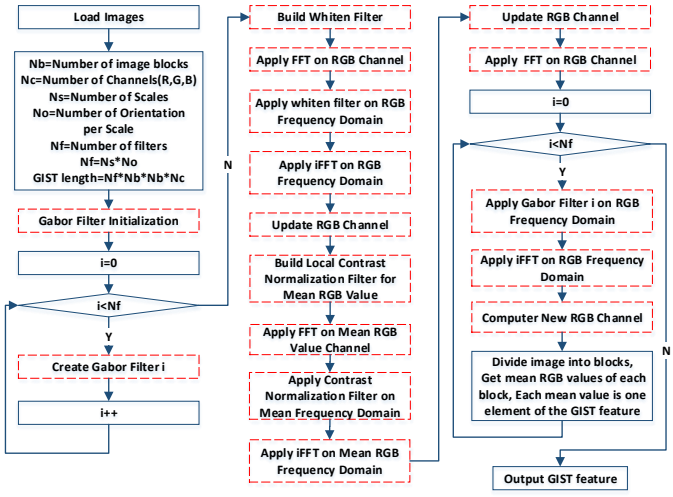

Figure 2: Flow chart of parallel GIST descriptor generation. The red dotted blocks mark the components parallelised as GPGPU compute.

Our parallel GIST descriptor computation is implemented in OpenCL to utilise the heterogeneous compute resources. Figure 2 illustrates the algorithmic flow of the parallel GIST

generator. Static and dynamic application profiling are used to analyse the program flow and identify the application hotspots from baseline serial implementation [5].

3.2 Parallel Hierarchical Kmeans

The K-means algorithm is extensively used as a clustering method for data analysis in many scientific domains. It randomly selects k data points as the initial centroids for k clusters. The algorithm then repeats a two-step procedure until reaching a pre-defined stopping criterion to group the data points into clusters:

- Stage one (assignment): k clusters are formed by assigning each data point to its closest centroid according to a predefined distance metric.
- Stage two (update): For each cluster, an updated centroid is calculated as the mean of all data points in this cluster.

The hierarchical K-means (H-Kmeans) divides the task of finding a large number of centroids for a large-scale data set into L levels, by keeping the number of centroids low for each K-means procedure and yet achieving much finer degree of clustering at each subsequent level. This property reduces the search space while querying images in a large-scale dataset.

- Step 1 (initialisation): Assuming all data points (here gist descriptors) are in one big cluster initially, at first level $l = 1$, H-Kmeans identifies the k centroids c_1^l, c_2^l, ..., c_k^l and their corresponding k clusters C_1^l, C_2^l, ..., C_k^l, using K-means;
- Step 2: Before progressing to the next level $l+1$ of H-Kmeans, to improve data locality we rearrange the data points by moving the ones with same cluster labeling to the same memory region.
- Step 3: Within each cluster $C_{k,0<=k<K}^l$, if the number of data points in this cluster exceeds the exit threshold, K-means is performed to find the centroids $c_{k,0<=k<K}^{l+1}$ and identify the clusters $C_{k,0<=k<K}^{l+1}$.
- Step 4: Iterate Step 2 and 3 for all sub-clusters before progressing to next level until reaching the limit L.

In summary, at l_{th} level, the algorithm identifies k centroids using K-means algorithm for each cluster C_i^{l-1} identified at previous $l-1_{th}$ level. Figure 1 (b) illustrates the H-Kmeans algorithm with an example drawing of two levels and three clusters for each Kmeans block. The black circles mark the data points to be clustered. The yellow triangles show the centroids of first level H-Kmeans, and the red squares highlight the centroids found at the second level of H-Kmeans.

Our proposed H-Kmeans approach allows the user to specify the maximum number of hierarchies, the maximum number of clusters for each K-means procedure and the minimum number of images for a leaf cluster. There must be enough images for each non-leaf node to perform K-means. We set the limitation that the number of images in each non-leaf node must be equal to or greater than the multiplication of the minimum number of images per leaf node and the maximum number of clusters of the K-means routine. Otherwise, this inner node will be marked as a leaf node even it is not in the bottom level of the hierarchy.

Clustering the data points using the H-Kmeans algorithm produces an H-Kmeans tree that contains the node and edge information, the centroids at each level, and a list of data points for each leaf node. In this work, each data point is a feature vector representing the GIST descriptor.

3.3 Parallel Vector-based Breath First Search

Breadth first search (BFS) is one of the most extensively used graph search algorithms. Breadth first traversal is also frequently evaluated as a benchmark to provide a high-level study on the performance of data intensive analytical workloads on GPGPU or supercomputing systems (e.g., Rodinia [14], SHOC [15] and Graph500 [16]).

Different from any prior BFS work focusing on scalar values, we first propose a vector-based approach: Each node in the search tree is a high-dimensional vector, and each query is a vector of same dimensionality. Furthermore, a second layer of parallelism is added on top of this to allow searching from multiple queries concurrently.

To search for one query image from a collection of images, we perform the preliminary search using the query image's GIST global descriptor. To achieve high efficiency of a large-scale parallel search of a group of query images, we propose a parallel vector-based hybrid breadth first search algorithm. It takes a group of GIST descriptors as input, one per query image. Then for this group of GIST descriptors, a parallel vector-based breadth first search is conducted to find the closest node in the H-Kmeans tree for each descriptor concurrently. At the coarse level, the search set is equivalent to the number of input descriptors, for which, a number of vector-based breadth first searches are launched concurrently. At the fine level, within each search, the hybrid BFS is performed at the dimension of GIST descriptor.

Algorithm 1 Vector based Breadth First Search

```
1:  function VECTORBFS(query)
2:      root ← 0
3:      while true do
4:          if leaf[root] == 1 then
5:              return root
6:          end if
7:          dist_min ← FLT_MAX
8:          index ← −1
9:          for i = 0 → n_cluster do
10:             next_node ← node_children[root][i]
11:             dist ← Euclidean(query, next_node)
12:             if dis < dist_min then
13:                 dist_min ← dist
14:                 index ← next_node
15:             end if
16:         end for
17:         root ← index
18:     end while
19: end function
```

In traditional BFS, each node in the search tree is usually labeled with a number or a string. The search terminates when

it finds a node's label matching the query. In our vector-based scheme each node in the search tree is a cluster center. For an incoming query, the distance between the query vector and each node at the first level of the search tree is calculated and compared. The closest node is picked as the new root node and the search continues on this new root node's child nodes. The search ends when it reaches a leaf node that stores a list of GIST descriptors belonging to this cluster. Algorithm 1 illustrates this process for a single query. In our parallel vector based BFS, each input is a set of queries.

Another contribution of our parallel vector-based BFS approach is to deploy a hybrid strategy that allows the flexibility of two ways of mapping tasks to the GPU compute units:

- One option is to assign one work-item (one execution instance of a kernel in the OpenCL model) with the task of searching for the best matching cluster center for one query and this work item calculates distances between the query image and all nodes (here cluster centroids).
- The other option is to assign one work-group (a cluster of cooperating work-items) with the task of searching for the best matching node for one query, which leads to one work-item in this work-group only computing the distance between this query and one node at a time in each level.

The option is determined by the compute and memory architecture of the compute units, data sharing among work items, the resource requirement of each work item and the number of queries.

3.4 Verification via Local SURF Descriptor

SURF [10] is a widely used scale and rotation invariant feature detector and descriptor. It consists of using integral images for image convolutions, Hessian matrix based measure for the detector, and a distribution based descriptor. This Hessian based interest point detection computes the Hessian matrix at each pixel by convolving the second order Gaussian derivative with the image at each pixel.

The SURF local descriptor of an image represents the image with a set of key feature points. Each feature point is a 64-element vector. The size of the SURF descriptor, effectively a matrix, is proportional to the number of feature points. Although more compact compared with SIFT, the computational and storage cost of the SURF descriptor precludes computing it for all images of very large-scale datasets. To increase the efficiency, we perform on-demand computation of SURF only from the short-listed potential matches of the query images.

Furthermore, the computation of the SURF descriptor consists of multiple components that can be parallelised. We based our work on the clSURF project [17] that provides the baseline OpenCL implementation of the SURF algorithm. We analysed the performance of this work and addressed limitations by improving the packing of work-items into work-groups. This change in the density of execution on hardware resources increases the efficiency of execution and resource utlisation. As an example, we found on average there are 240 SURF features, requiring 60KB storage space per image for the downsampled Dataset 1 as detailed in §4.1.

3.5 Parallel Search Framework

For a set of query images Q, parallel GIST descriptor generation is performed to compute their feature vectors. For each GIST feature vector, the vBFS presented in §3.3 is applied to search this feature vector in the H-Kmeans tree formed in §3.2. This procedure returns the ranking of the image global descriptors from the best matched leaf node. To verify this global descriptor based search results, local SURF descriptors based evaluation is used to compute the distance of the query image and each image returned from the vBFS, according to the descending order of the rankings. The final matches are then returned based on the descending order of the number of matched SURF features.

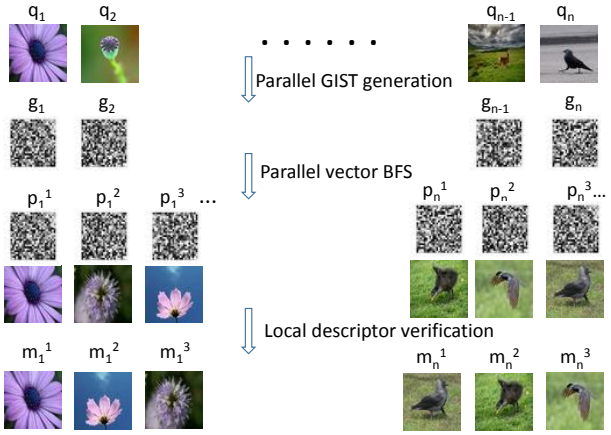

Figure 3: An illustration of parallel image search pipeline.

Figure 3 illustrates the parallel image search pipeline with examples. This figure presents the query results from searching the H-Kmeans tree, using the parallel vBFS. The preliminary search results are then re-ranked and pruned by the SURF descriptor verification. The example query q_1 demonstrates the outcome for an in database search; the example query q_n shows the results of querying an image that does not present in the image set used to build the H-Kmeans tree. Note as shown in this figure, queries $q_1, ..., q_n$ are processed in parallel.

4. IMPLEMENTATION

In this section, we present the image dataset used for this work, the hardware specification and the runtime system. We also detail the implementations of the proposed algorithms on the chosen platform. Lastly, we discuss the optimisation techniques deployed for achieving high performance. Both the implementation details and the optimisation methods presented are not limited to the platform that this work is based on.

4.1 Experiment Setup

For evaluation purpose, we studied two datasets:

- Dataset 1: A randomly selected subset with 10 images from each category of the SUN Database[18], which contains 397 categories and 130,519 images.
- Dataset 2: We use the MIRFLICKR collection [19] consisting of one million images from the social photo sharing website Flickr. This dataset was formed by downloading up

to a thousand photos per day that were evaluated as the most interesting by Flickr.

Before computing the GIST descriptor and building a search tree, we preprocess the input image data set with Lanczos filter and normalise the images to a fixed size of 256 by 256 pixels. GIST global descriptor and SURF local descriptor when needed are computed from the resampled images.

We perform experiments on a heterogeneous system consisting of an APU and a discrete GPU (dGPU) with specification as follows:

- Radeon^{TM} HD 7970 (device), A10-5800K APU (host)
- CentOs 6.4 with GCC version 4.4.7, OpenCL 1.2 AMD-APP 1124.2, CodeXL 1.2.2484, clFFT 1.10.321.

4.2 Implementation Details

We implemented the proposed search framework in OpenCL 1.2 and C++, and performed the experiments on the system detailed above. In this work, we use a gist descriptor configuration with 4, 4, and 4 orientation bins. The total dimension of the gist descriptor is computed as: (the number of blocks) * (the number of blocks) * (the sum of orientation bins) * 3. Here we divide the three-channel color images to a grid of 4 by 4 blocks, resulting in a total of 576 dimensions for the final configuration.

The proposed parallel GIST descriptor generation is completed with the fast fourier transforms routines (clFFT) from the open source clMath libraries. Through calling the clFFT library functions, the FFT OpenCL kernels are automatically generated for the specific underlying hardware platform and enqueued on the device. However our experiments show the set of OpenCL FFT routines generated are not well optimised for the SI dGPU used, in particular, a large number of registers are allocated for each unit of execution. On the SI architecture, as dGPU register files are dynamically allocated to thread contexts (wavefronts in AMD terminology), this limits the amount of concurrency available to cover memory stalls and hiding memory latency. Dynamic profiling shows these clFFT routines are latency bound, instead of being constrained by memory bandwidth or computational resources. Hence, increasing memory bandwidth or CU frequency do not provide significant performance gain. It is part of our future work to eliminate these bottlenecks imposed by the clFFT library.

Furthermore, a parallel search scheme for a group of queries with configurable size does not only increase the efficiency of resource utilisation, but also amortizes the cost of creating an OpenCL context, building OpenCL kernels, and setting up the kernel execution pipeline. Further implementation and optimisation details are presented in §4., followed by the comparison of our approach and other approaches in §5.

4.3 Optimisation Techniques

The amount of memory available to OpenCL applications on the device is limited by the device driver on AMD hardware. By default, the size of a single memory allocation is limited to a quarter of the total device memory, and the total amount of device memory that can be allocated to OpenCL applications is limited to half of the total device memory. The former can be changed by setting GPU_MAX_ALLOC_PERCENT to increase the maximum buffer size; the latter can be changed by increasing the value of the environment variable GPU_MAX_HEAP_SIZE from the default 50% to the percentage of total dGPU memory that could be exposed. This limitation constrains all OpenCL applications, with no exception in our case. However, we designed a group-wise scheme to bypass this issue by domain decomposition. The domain decomposition partitions the data set for each compute stage to a number of sub-sets that can be processed on the dGPU without frequently swapping the data set between the dGPU memory and the host memory, which would lead to further API and data transfer overhead.

This section presents the optimisation techniques we explored using an example: K-means, the building block of H-Kmeans algorithm. Three methods of mapping the K-means computation to the hardware resource are presented here, and their evaluations and comparison are in §5.

- Method 1: assignment stage on the device dGPU, centroid update stage on the host APU.
- Method 2: Both assignment and centroid update stages are performed on dGPU. The update stage first uses atomic operations to compute the number of images for each cluster, secondly creates a GPU work-item for each dimension of the feature vector. Each GPU work-item loops through all feature vectors to identify which cluster this image feature vector belongs to, and subsequently update the corresponding cluster centroid.
- Method 3: To improve the data locality in method 2, after using atomic operation to compute the number of images for each cluster, we create a starting offset array by summing the size of each cluster to get the cluster starting index. We then create an index array that groups the images in the same cluster together. The size of the array is equal to the total number of images. There is an atomic counter for each cluster. We assign one work item to one image, find its belonging cluster and the cluster starting index created earlier, add the cluster counter atomically and store the image number in the index array with position [cluster starting index + cluster counter value before atomic add]. The last step is to update the cluster center. The global work size is the multiplication of the number of clusters and the number of dimensions of the feature vector. Each work item updates one dimension of one cluster. So each work item finds its cluster ID, dimension ID, cluster size, and cluster starting offset. Then it goes to the index array, finds all points belonging to this cluster and performs the update.

We also evaluated the cost and benefit of utilising local data share (LDS) for the update stage. The low latency and high bandwidth of LDS potentially offers benefit with low power consumption, provided there is enough data reuse. However, every access to LDS requires the LDS address to be computed ahead. Furthermore, extra overhead is incurred for the first time usage of one data item in LDS, since the data needs to be read from cache, written to LDS and finally read from LDS. If L1 is fully utilised, moving frequently reused data to LDS increases the effective capacity of the data close to compute. However, if not, using LDS is essentially migrating data

from L1 cache to LDS, which gains nothing but costs extra operations and synchronisation. For H-Kmeans, we do not observe benefits from using LDS.

5. RESULTS AND DISCUSSIONS

To uncover the runtime characteristics of the OpenCL applications on heterogeneous platforms, for each OpenCL application both the OpenCL API trace file and hardware performance counters are collected by dynamic profiling. Through these collected statistics we analyse the task execution time on the APU and dGPU as well as the communication traffic between the two. With the API traces and hardware performance counters as input, our analytical tool provides the following functionality to allow us an schematic runtime overview of the applications on the underlying hardware systems:

- Summarise the statistical performance data from the OpenCL application to identify the bottlenecks in the application and interpret the execution characteristics of the program on the specific hardware.
- Project the potential improvement from accelerating data transfer or kernel execution time given architecture design modifications and/or code optimisations.

An example summary of this analysis on parallel GIST is presented in Table 1. The API latency only includes the unhidden latency of OpenCL memory object creation and release, kernel launch, read and write launch, and event release. The "others" category here covers the execution time on the APU host.

Table 1: Application trace breakdowns of parallel GIST.

Kernel execution	Data transfer	API latency	Others
29.99 ms	46.12 ms	25.26 ms	39.61 ms
21%	33%	18%	28%

To keep it concise, we only show the kernel level analysis for the GIST generation. For H-Kmeans, we show the impact of varying memory controller and CU frequency on the kernel performance. However, all analyses are applicable to other GPGPU applications. Finally we present our analysis and evaluations of the parallel search and verification.

5.1 Parallel GIST Generation and Analysis

Through runtime profiling, we gathered the statistics about the OpenCL kernel execution of each run of each individual kernel during the GIST computation for Dataset 1. The curve in Figure 4 shows the percentage distribution of total kernel execution time among the 12 OpenCL kernels in descending order. The bars show the execution time spent on each kernel per invocation. For each image, a total of 139 kernel launches are performed to compute the GIST on the dGPU device.

We improved upon the serial algorithm of the GIST descriptor [5] by eliminating the initialisation overhead and allowing the batch processing of multiple images in parallel. This improved GIST descriptor computation is used as baseline on the APU-CPU to compare with the throughput-oriented version in OpenCL on the dGPU. Our experiments with Dataset 1 show the baseline takes on average 885.79 ms to compute the gist of one image, and the proposed parallel approach takes on average 140.98ms, offering a 6.28 times acceleration. This evaluation includes I/O, kernel compilation, kernel dispatch, data transfer overhead and others that are observed on existing hardwares with current OpenCL runtime support. We subsequently compute the GIST of Dataset 2 using our approach as an offline pre-processing stage prior to the image clustering.

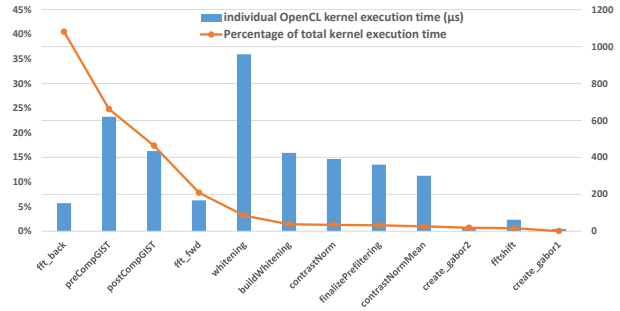

Figure 4: OpenCL kernels of parallel GIST generator.

5.2 H-Kmeans

Figure 5 shows the impact of memory controller frequency and dGPU CU frequency on the total runtime, kernel execution, read and write time; and also the impact on the execution time of six individual kernels of Kmeans method 3. This shows us the "create index", "update 1" and "create start" kernels are computation bound, hence the increase of the performance is dominated by the increase of the CU frequency. On the other hand, the performance of the other three kernels are significantly impacted by the bandwidth made available via the increase of memory controller frequency instead of the computational throughput. This analysis shows adaptive frequency scaling at kernel granularity is necessary to obtain the optimal performance with minimum energy consumption.

Figure 6 compares the execution time of three Kmeans implementations on the heterogeneous platform, with the assignment step consistently performed on the dGPU. The results shows as the number of clusters increases, the approach of updating the cluster centroids on the dGPU with optimisations achieves similar performance as the one with update stage on the host CPU. Furthermore, as the number of dimensions increases from 32 to 576 gradually, the performance of updating the centroids on the dGPU improves. The performance comparison includes the data transfer and OpenCL runtime overhead counting as 28% and 12% of the total application runtime respectively, which are to be further eliminated in future heterogeneous system architectures.

However, even in the best case the dGPU is only competing with, not defeating, the CPU when including the centroid update state. This highlights the necessity to carefully balance different parts of a computation across different types of nodes in a heterogeneous system. As the balance of compute resources changes, for example using a smaller GPU in a fused SoC such as an AMD APU, the balance changes. In particular, the balance of devices on future SoCs may lean more towards GPU resources than CPU resources if graphics continues to drive innovation, and this may lead to a GPU-favouring resource balance on such future devices.

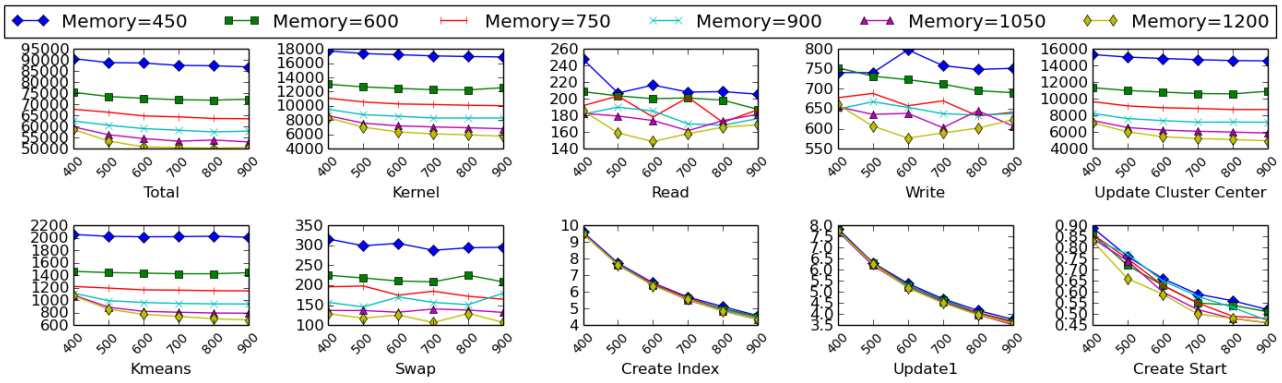

Figure 5: H-Kmeans: varying memory and CU frequency of the dGPU. For each sub-figure, horizontal axis denotes CU frequency (MHz), vertical axis represents the time (ms), each line as memory controller frequency (MHz).

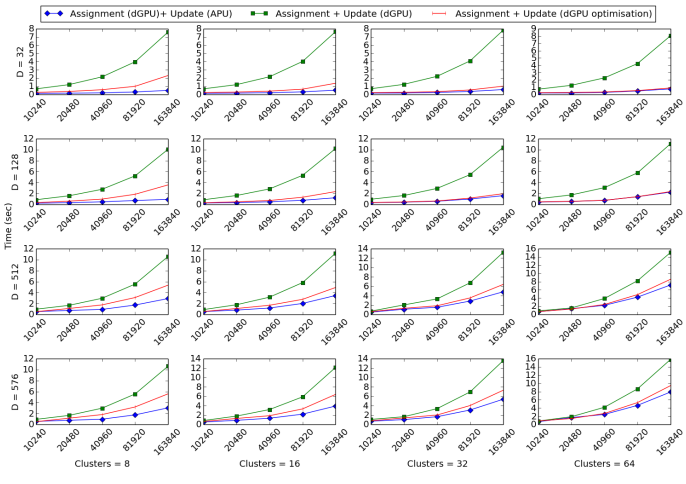

Figure 6: Kmeans: comparison of the total run time of three approaches as presented in §4.3. The horizontal axis of each sub-graph represents the number of data points to be clustered.

5.3 Parallel Search and Verification

Matsubara et al. [20] presented a similarity-based image retrieval algorithm and optimised the search by data alignment, reporting an average of 300ms of search time. In our work, the average search time in the Dataset 2 of 1 million images is significantly reduced. For example, we constructed a H-Kmeans tree with $L = 9; K = 8$ from Dataset 2. To measure the search accuracy and performance, we randomly seletected four sets consisting of 1K, 2K, 4K, and 8K images respectively from Dataset 2 to query concurrently. The average time on finding the closest cluster matching each query is shown below on the 2nd row. The average time per query spent on verification based on the local SURF descriptor compared with all the candidates in the closest cluster found is listed on the 3rd row. This local descriptor verification takes 12.6ms per pair of query and candidate image on average. All four sets of queries correctly indentified the images in Dataset2.

Number of concurrent queries	1024	2018	4096	8192
Locate the cluster	86.8μs	47.8μs	24.4μs	16.6μs
Local descriptor verification	72.8ms	59.1 ms	58.1ms	68.6ms

Figure 7 illustrates a verification via a SURF feature points matching scheme, where the image (a) is the query, the image (2) is one potential match. Figure 7 (e) shows the 106 correspondences established among the SURF key points from both images. We also searched for images that are not in the dataset and performed visual inspection. The results are very promising at finding the closest cluster for the query image according to the global descriptor and ranking according to the local descriptor. One key part of our future work is to design quantitative metrics for evaluating these results.

(a) (b) (c)

Figure 7: An example of local-descriptor SURF based verification (c), correctly matching the same Oxford building in the two images (a) and (b), where the SURF features are marked.

We summarise the 30 OpenCL kernels of all applications presented in this work to study the diversity and its challenge for the architecture. To provide an overall characterisation of the OpenCL kernels, we propose two simple metrics: average bandwidth usage, defined as the sum of fetch size and write size per second, and the average number of vector ALU instructions executed per work-item per second, which are representative but not meant to cover all aspects of kernel runtime features, e.g., cache effect, scalar operations, memory unit status. Both metrics are derived from the statistics collected from the dynamic profiling on the system presented in §4.1. From a bird's eye view, Figure 9 shows the variations of the two characteristics of the application compute kernels that are used in our study. Each blue diamond represents one kernel in the two-dimensional space of aforementioned two metrics.

We also analyse the impact of frequency scaling on OpenCL kernel execution. The findings show that for general image search presented here, a wide range of behaviors are observed when scaling the clock frequencies for the OpenCL kernels. To accurately select the best configuration for each kernel to attain best performance and minimum energy cost requires a

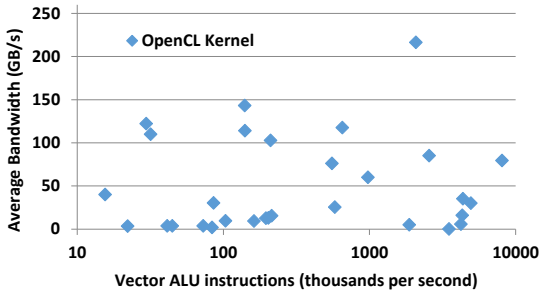

Figure 8: Bandwidth utilisation and vector ALU instruction characteristics of the 30 OpenCL application kernels.

fine-grained dynamic scaling model. We consider this the most important next step in our work. The scaling graphs of kernels in Figure 9 show three examples: bounded by computation throughput, memory bandwidth or a combination of both.

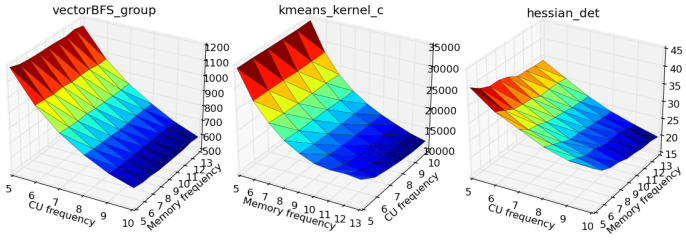

Figure 9: Variation of the impacts of frequency scaling on kernel execution. The vertical axis is execution time (ms) with dynamic profiling. The frequencies are at hundreds MHz scale.

6. CONCLUSION AND FUTURE WORK

In this work we have demonstrated that image search is an area well suited to increasing performance on heterogeneous architectures. We have shown the potential of further improvement of the performance and energy efficiency by reducing data movement and fine-grained clock frequency scaling. The overall objective is to investigate how we can improve the architecture to reach an optimal point in the energy/performance design-space and to this end we are investigating processing-in-memory architecture (PIM) [21]. In-memory processing offers the opportunity to significantly reduce aggregate data traffic across the entire system through localisation. The next step in our work is to map these image processing and search algorithms onto the PIM simulation infrastructure to explore the co-design space of PIM architecture and large-scale image search. Processing-in-memory can potentially increase search efficiency and reduce the energy consumption per query.

The limitations from clFFT library are also to be investigated further and overcome in future work. Furthermore, other local descriptors that are reported to be multi-fold more efficient than SIFT and SURF are also to be studied, such as the ORB descriptor [9]. The parallalisation challenges on heterogeneous platforms for BoF approach are also of interest as comparison. Finally, to achieve energy efficiency and performance gains, it is important to perform the co-design of applications and compute architecture.

Note: Lifan Xu's contribution to this project was completed during his internship with the Research Group at AMD.

REFERENCES

[1] P. Viola and M. Jones. Rapid Object Detection Using a Boosted Cascade of Simple Features. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, pages 511–518, 2001.

[2] F.-F. Li and P. Perona. A Bayesian Hierarchical Model for Learning Natural Scene Categories. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, pages 524–531, USA, 2005.

[3] Y. C. Gong, Q. F. Ke, M. Isard, and S. Lazebnik. A Multi-View Embedding Space for Modeling Internet Images, Tags, and their Semantics. *Computing Research Repository (CoRR)*, abs/1212.4522, 2012.

[4] Z. Wu, Q.F. Ke, M. Isard, and J. Sun. Bundling Features for Large Scale Partial-duplicate Web Image Search. *IEEE Conference on Computer Vision and Pattern Recognition*, pages 25–32, 2009.

[5] M. Douze, H. Jégou, H. Sandhawalia, L. Amsaleg, and C. Schmid. Evaluation of GIST Descriptors for Web-Scale Image Search. In *International Conference on Image and Video Retrieval*, pages 19:1–19:8, 2009.

[6] A. Oliva and A. Torralba. Modeling the Shape of the Scene: A Holistic Representation of the Spatial Envelope. *Int. J. Comput. Vision*, 42(3): 145–175, 2001.

[7] David G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *Int. J. Comput. Vision*, 60(2):91–110, 2004.

[8] N. Dalal and B. Triggs. Histograms of Oriented Gradients for Human Detection. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, pages 886–893, 2005.

[9] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. ORB: An Efficient Alternative to SIFT or SURF. In *International Conference on Computer Vision*, pages 2564–2571, USA, 2011.

[10] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool. Speeded-Up Robust Features (SURF). *Computer Vision and Image Understanding*, 110(3): 346–359, 2008.

[11] J.G. Zhang, M. Marszałek, S. Lazebnik, and C. Schmid. Local Features and Kernels for Classification of Texture and Object Categories: a Comprehensive Study. *International Journal of Computer Vision*, 73 (2):213–238, Jun 2007.

[12] AMD. White paper: AMD graphics cores next (GCN) architecture. Jun 2012.

[13] Khronos OpenCL Working Group. *The OpenCL Specification*. 1.2 edition, 2012.

[14] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron. Rodinia: A Benchmark Suite for Heterogeneous Computing. In *IEEE International Symposium on Workload Characterization*, pages 44–54, 2009.

[15] A. Danalis, G. Marin, C. McCurdy, J. S. Meredith, P. C. Roth, K. Spafford, V. Tipparaju, and J. S. Vetter. The Scalable Heterogeneous Computing (SHOC) Benchmark Suite. In *3rd Workshop on General-Purpose Computation on Graphics Processing Units*, pages 63–74, USA, 2010.

[16] R. C. Murphy, K. B. Wheeler, B. W. Barrett, and J. A. Ang. Introducing the Graph 500. Technical report, Sandia National Laboratories, 2010.

[17] P. Mistry, C. Gregg, N. Rubin, D. Kaeli, and K. Hazelwood. Analyzing Program Flow within a Many-Kernel OpenCL Application. In *Fourth Workshop on General Purpose Processing on Graphics Processing Units*, pages 10:1–10:8, USA, 2011. ACM.

[18] J. Xiao, J. Hays, K.A. Ehinger, A. Oliva, and A. Torralba. SUN Database: Large-Scale Scene Recognition from Abbey to Zoo. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3485–3492, 2010.

[19] M. J. Huiskes, B. Thomee, and M. S. Lew. New Trends and Ideas in Visual Concept Detection: The MIR Flickr Retrieval Evaluation Initiative. In *International Conference on Multimedia Information Retrieval*, pages 527–536. ACM, 2010.

[20] D. Matsubara and A. Hiroike. High-Speed Similarity-Based Image Retrieval with Data-Alignment Optimization Using Self-Organization Algorithm. In *11th IEEE International Symposium on Multimedia*, pages 312–317, 2009.

[21] D. P. Zhang, N. Jayasena, J. Greathouse, M. Meswani, M. Nutter, A. Lyashevsky, and M. Ignatowski. A New Perspective on Processing-In-Memory Architecture Design. In *ACM SIGPLAN Workshop on Memory Systems Performance and Correctness*, 2013.