Design of Multimedia Matching Service using Manycores and GPGPU

Dong Jin Kim, Chulmin Kim, Jong Hun Choi, and Kyu Ho Park
Computer Engineering Research Lab., Electrical Engineering
Korea Advanced Institute of Science and Technology
Daejeon, Republic of Korea
{djkim, cmkim, jhchoi}@core.kaist.ac.kr and kpark@ee.kaist.ac.kr

***Abstract*—Nowadays, many users are spending their time with enjoying multimedia, and this trend spreads rapidly according to the propagation of smartphones. Multimedia services have tried to attach tag for information of merchandizes related to multimedia, so the multimedia consumers can get detail of those merchandizes, such as price or purchasing information, using attached tag. However, multimedia provider can provide only limited tags while they even cannot support real-time video. In this situation, Multimedia Matching as a Service (MaaS) system is conceptually suggested to let the consumers find items as their preference on real-time video. This system extracts images from video stream, and finds various objects from those images.**

Since the MaaS system needs a lot of computation power, it should utilize the cloud service which is the recent trend in the computing industry. Due to the circumstance, an efficient internal design for MaaS is required to reduce the cost of using cloud service itself.

For the purpose, in this paper, two designs for MaaS system is introduced which are based on manycore system and GPGPU system respectively. One thread per one image method is used for manycore system, and hundreds of cores per one image method is adapted for GPGPU system. Using the intensive experiments, we prove that our design is appropriate for each system. More on that, we compare two designs to obtain the intuition for real system adaption.

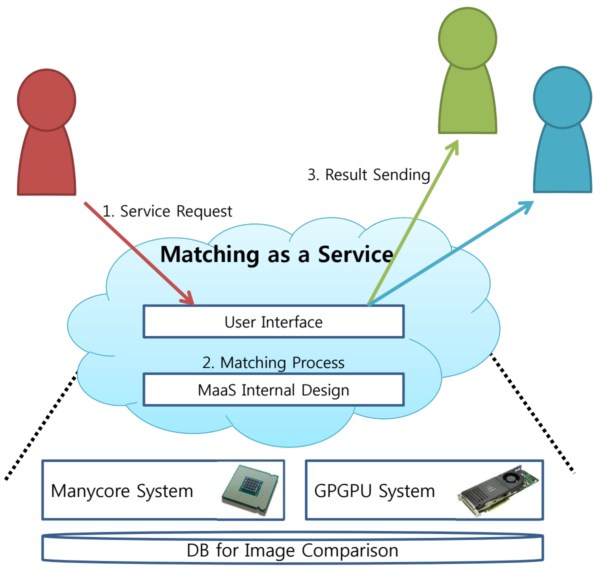

Fig. 1. Overall Architecture of Matching as a Service

I. Introduction

In these days, many users are spending much of their time to enjoy multimedia such as TV and videos. As the handheld device, represented by the smartphone, propagates dramatically, this trend spreads rapidly over most of users and becomes their life style.

Multiple multimedia service providers have tried so far to commercialize the emerging multimedia trend. In the very beginning, the service providers [1] let the multimedia producers attach tags for information of merchandizes related with the multimedia content. This method has worked for the web based multimedia service such as YouTube [2]. The multimedia consumers can reach to the website or e-commerce site of merchandizes through the tags.

However, the tag attached by the multimedia producer has limitations. The tag only covers the interests of the multimedia producers, not the multimedia consumers. Moreover, the method cannot work for the multimedia with real-time characteristic, because producers cannot know which item will be on the real-time video stream.

To overcome the limitations, multimedia matching services have been launched. At the first time, several services were started targeting image matching queries. Goggles [3] provides the image matching service on the web. More on that, Picasa [4] and Pudding apps [5] were developed, which make users use the image matching service on the handheld devices. These image matching services for images could satisfy the multimedia consumers' needs.

Recently, the service named MaaS (Matching as a Service) [6] has been conceptually suggested to provide the similar matching service for not only images, also real-time videos. It periodically extracts images from real-time videos, and matches various objects in the extracted images with pre-captured objects in DB. Information of the matched ones directly sent to the multimedia consumers. The service is described in Fig. 1. The multimedia providers supply the real-time contents to MaaS system, and the multimedia consumers receive the contents with the information obtained by the pre-processing.

Since MaaS needs a lot of computation power and unified DB to store pre-captured objects, it should utilize large

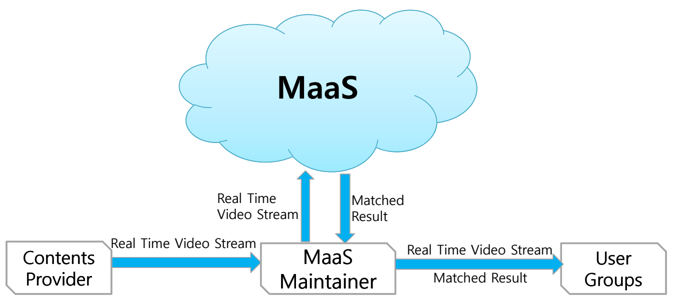

Fig. 2. Outlook of MaaS System

computing environments such as the cloud service which is the recent trend in the computing industry. In perspective of MaaS maintainers, it is very crucial to design MaaS internal efficiently. Since the maintainers should spend the cost in proportional to the time used the resources, the utilization of computing resources is directly related with the cost. However, there is no precise design so far for the service yet though MaaS is expected to satisfy the needs of the multimedia consumers.

In this paper, we tackle the absent efficient design of MaaS system. Based on the suggested concept of MaaS, we define internal operations and user interface of MaaS. Since the current cloud services equips with manycores and GPGPU which are appropriate to execute massive computing jobs like MaaS operation, we design two kinds of MaaS system using manycores and GPGPU respectively focusing which operations can be parallelized efficiently. Also, we thoroughly experiment our designs to measure the performance in realistic environments. Comparison between two designs lets us know which design is better depending on the environments.

The remainder of the paper is consist of the followings. Section II describes background knowledge and related works of MaaS system. Section III explains our designs in detail. Section IV shows the performance of our MaaS system designs. Section V concludes this paper by listing our further works.

II. Background and Related works

A. *MaaS: Matching as a Service*

The concept of MaaS(Matching as a Service) [6] is suggested to provide multimedia matching service, especially for real-time video. The use case of MaaS is shown in Fig. 2. According to this case, the only input to MaaS is real-time video stream and MaaS returns the matched results continuously. For the output, MaaS should capture a image frame of the input and find images from its DB which are similar with the objects in the image frame. Since the input is the real-time video stream, MaaS should do the job above continuously and as many times as possible to provides the qualitative output.

MaaS assumes that it operates on the computing resources of the cloud service. Since Pay-as-you-go model of the cloud service charges the cost in proportional to the execution time, it is important to design the MaaS internal efficiently. So far, the paper which suggested MaaS [6] only described the concept and the forecasted challenges.

B. *Feature Matching Algorithm*

During the MaaS operation, it should compare which object in the image frame is similar with which image in its DB. For the purpose, MaaS should have the algorithm which judges the similarity between the object and the image. It is called the feature matching algorithm, and a lot of algorithm researches [7]–[9] have done so far to enhance the functionality of the feature matching.

Among them, SIFT [7] is the most promising one in the research area and it is widely being used even for commercialized services. This algorithm has several stages. On scale-space extrema detection stage, it finds interest points using difference-of-Gaussian(DoG) function. On keypoint localization stage, keypoints are selected. On orientation assignment stage, based on local image gradient directions, some orientations are assigned to keypoint, and it will provide scale invariance. Last stage is keypoint description, which indicates local shape distortion and illumination change. Through these stages, several features can be found on one image frame.

To compare features come from two images, the method with KD-tree is used widely [10]. This method is well-known for matching features area. With this method, each feature will be compared with features of the another image one-by-one, and the matching ratio can be calculated with the number of matched features.

C. *Parallelism of Manycores and GPGPU*

MaaS has several needs for the parallelism. First, it is expected to have lots of users simultaneously. Second, the algorithm and the DB queries used for MaaS operation can be parallelized highly. Thus, it should be operated on the environment with large computing resources such as the cloud service.

The recent cloud services equip with manycores and GPGPU as the hardware technology evolves. Literally, manycores refers the machine with lots of cores per chip. Each core of the manycore system has strong computation power while the parallelism is limited to a few tenths per 1 node machine. On the other hand, GPGPU is the abbreviation of general purpose graphic processor unit which is evolved from GPU. GPGPU provides a few thousands of parallelism per unit though each core has less computation power than a core in the manycore system.

Though GPGPU provides more parallelism, it does not always overwhelm manycores. GPGPU has a limitation that an application should tolerate memory transfer overhead. GPGPU cannot share the main memory space of the host system, so that the memory transfer is unavoidable. More on that, it is connected with the host system using PCI-e interface, which has bottleneck of memory transfer.

In the past, many other applications needed the parallelism carefully decided whether to choose manycores or GPGPU

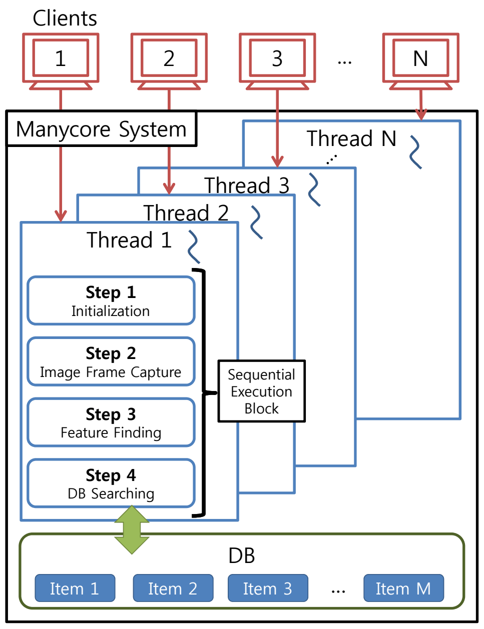

Fig. 3. MaaS design for manycore

for its parallelism. MaaS also can have the decision problem. Through this paper, we try to find the best solutions for both directions. Designing and comparing those will give us valuable knowledge for MaaS system design.

III. Design

In this section, MaaS is designed for both manycore system and GPGPU acceleration system. As explained, for MaaS operation, it should compare which object in the image frame is similar with which image in its DB. Therefore, designs should have following steps: initialization, image frame capturing, feature finding, and DB searching. However, their detail design is somewhat different, and they can be used well on different situation.

Among four steps, feature finding and DB searching can be parallelized. For MaaS, there are two available parallelization methods - parallelizing operation itself or parallelizing whole operation with many threads. The former method parallelizes each thread to work on multiple cores concurrently. The latter method parallelizes multiple threads, with one thread per one core.

A. MaaS design for manycore

On manycore system, the latter method, which is parallelization for one thread per one core, is more appropriate. Since manycore system has NUMA architecture [11], when one core tries to access another core's memory, it requires larger overhead than local memory access. On MaaS, image data should be read from memory to multiple cores, this overhead can be increased. Therefore, if the number of clients is guaranteed, this parallelization method is better.

Fig. 3 shows MaaS design for manycore. For N clients, N threads will be made, and each thread will serve each client. All threads will execute the same operation steps, and it will loop until corresponding client leaves the system.

1) Initialization: At first for initialization step, a thread will construct DB with items which client wants to find on video stream. As client's demand, the items can be clothes, snacks, appliances, or other sets, and they are stored on the disk as image files. Using SIFT algorithm, it will find features from item images, and the features will be stored on the disk for later use. If the item image itself is not changed, features can be used permanently.

After DB construction, the thread will get ready to receive images from video stream. To receive real-time video streams, the thread needs network device or other I/O devices. Therefore, it will get permission to access the memory of those devices and get ready to read image data from them.

2) Image frame capturing: To find items from video stream, the thread requires image frame from I/O devices. During those devices receive video data, the images will be stored on the memory. Therefore, all for the thread is just copying the image data from device memory space to its local memory space. If the copy operation is successful, it can find matched items on later steps.

3) Feature finding from image frame: In this step, the thread will find features from image frame, using SIFT algorithm [7]. As explained before, this algorithm will find features consist of keypoints and descriptors. These features will be used as bases of matching decision on the next step. Not as DB items, since image frame will be changed on the next operation, the features are temporal and they will not be stored on the disk.

4) DB searching: Final step is actual matching operation between image frame and DB item images. For all DB items, image frame has to be compared using their features. On the initialization step, each DB item images are already transformed as features. The thread reads one DB file from disk, and then that file will have feature information comes from corresponding image.

For finding matched features between two images, algorithm with KD-tree is used [10]. Thread makes a tree using features of one image, then features of the another image will be searched using that tree, one by one. With this operation, matching percentage can be calculated between two images. If that percentage is greater than some threshold value, the thread will send the information about matched item. If there are M DB items on interest, the thread needs M iterations to check matching result of all DB items.

After DB searching is finished, thread will go back to the image frame capturing step, and it will operate with next image frame. This loop will be finished if the client is disconnected.

B. MaaS design for GPGPU

On GPGPU system, the former method, which is parallelization of whole operation with many threads, is more appropriate. Since GPGPU is SIMD architecture [12], GPGPU cannot be shared by multiple threads, so it is hard to operate multiple images with multiple threads simultaneously. Therefore, GPGPU system uses this parallelization method.

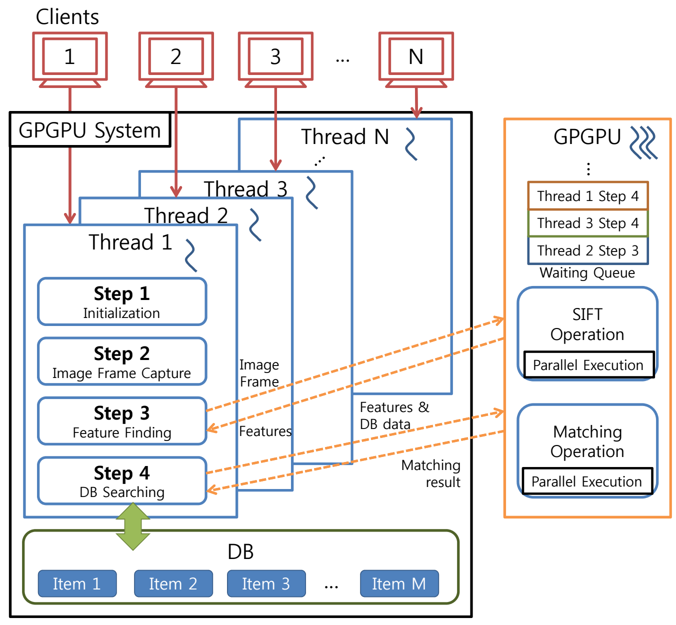

Fig. 4. MaaS design for GPGPU

Step	Parallelizability	Design	
		Manycore	GPGPU
Initialization	x	Single Thread	Single Thread
Image frame capturing	x		
Feature finding	o	1 thread / 1 image	336 cores* / image
DB searching	o		

* The number of CUDA cores for GTX460

TABLE I
MAAS DESIGN PARALLELIZABILITY AND IMPLEMENTATION METHOD

Fig. 4 shows the scheme for GPGPU system. As manycore sysetm case, if there are *N* clients, *N* threads will be made. For GPGPU operation, however, they need to wait until previous thread finishes its operation on GPGPU.

1) Initialization: What to do on initialization is same as manycore case, but SIFT operation for DB items can be done by GPGPU.

2) Image frame capturing: Since this step has just memory copy operation on CPU main memory region, there is no difference with manycore case.

3) Feature finding from image frame: SIFT algorithm itself is same as the one of manycore system, and it is just done by GPGPU. On GPGPU, not like manycore system, SIFT operation itself is parallelized [13], [14], and only one thread can use GPGPU at a time.

Additionally, when each thread uses GPGPU for SIFT, it has to first load image frame from its memory space to GPGPU memory space, and vice versa for the result after operation. Since their memory space cannot be shared, this loading is unavoidable. Therefore, SIFT operation itself will be accelerated by GPGPU, with waiting overhead and memory transfer overhead.

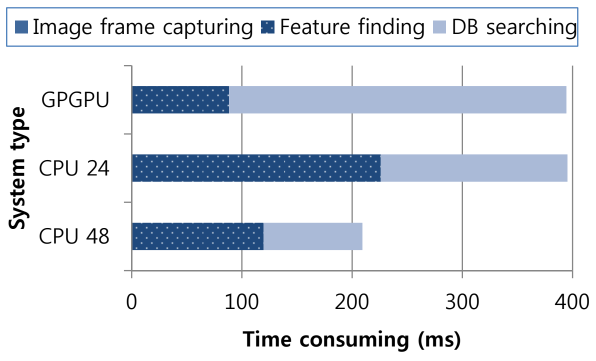

Fig. 5. Time Consuming for Image Frames

4) DB searching: Searching method is same as the one of manycore system. Here, the matching operation itself is accelerated with parallelization. During matching step, each DB item will be loaded on GPGPU, matched and returned result, one by one. However, waiting overhead and memory transfer overhead can be occurred for this design.

C. Design summary

Table I shows the summary of two designs. Among four steps, feature finding step and DB searching step can be parallelized. On manycore design, parallelization method with multiple threads is used because of memory hierarchy property. On GPGPU design, parallelization method within one thread is used because it is non-preemtable.

IV. EXPERIMENT

For experiment, we used AMD opteron 48-core 2.1GHz, 64GB DDR3 main memory and GPGPU GTX460 machine, with Ubuntu 11.10 and Linux Kernel 3.1.0. For video, we used 1280x720 video stream, and 50 DB item images of snacks, image size from 150x150 to 200x200.

A. Performance comparison between two system

At first, we tried to check the performance of each steps, image frame capturing, feature finding and DB searching, on both manycore system and GPGPU system. For same workload, which means same image frames and same DB items, time consuming on each step is measured.

Fig. 5 shows the measurement result. In both case, image frame capturing step shows only a little overhead, compared with other steps. For manycore system, time values are measured as average time consumption for all clients, in case of 48 clients which shows maximum throughput on the later experiment and 24 clients which shows about half throughput of maximum. In this case, it has more time consumption on feature finding. Because parallelization scheme is adapted for multiple clients, that step has maximum consuming, which has more amount of calculations than the others.

For GPGPU system, time values are measured in case of only one client, which provides maximum throughput on the later experiment. In this case, DB searching step has more time consuming than the others. It means, GPGPU operation

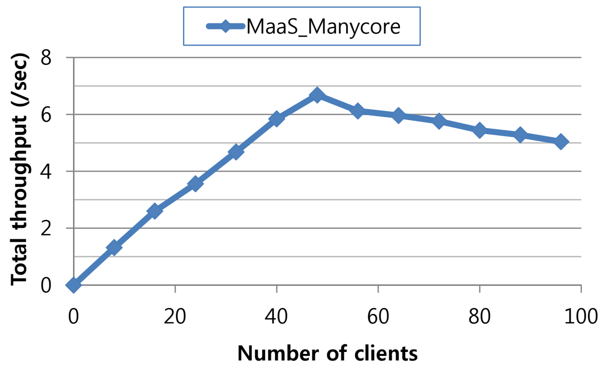

Fig. 6. Manycore System Throughput with Various Client Numbers

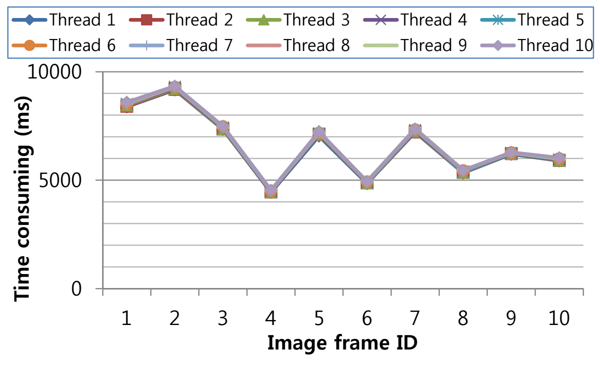

Fig. 7. Time Comsuming for each clients of Manycore System

itself does not have much time consumption, and memory transfer overhead is a lot larger than operation. Therefore, DB searching, which has fewer amounts of calculation and more amount of memory transfer, needs more time to work.

B. Manycore system parallelization measurement

1) Throughput with client number variation: We checked total throughput change with increasing the number of clients. The unit of throughput is one loop per second, and each loop consists of image frame capturing, feature finding and DB searching steps. For various number of clients, total throughput is checked.

The result is shown in Fig. 6. Until the number of clients reaches to the number of cores, 48, throughput is increased proportionally to client number. It is caused by well-parallelized clients. If there is some interference among clients, the throughput value will be bounded before the number of cores.

If there are number of clients more than the number of cores, they have to compete to have a core, so it shows total throughput decrease on this region.

2) Operation time consuming for each client: Next, we try to check operation time consumed by each client, with same workload. For 10 concurrent clients, same image frame is inserted, and operation time for each client is measured.

Fig. 7 shows the result. There are 10 image frames, from 1 to 10, on the vertical axis, which means 10 different work-

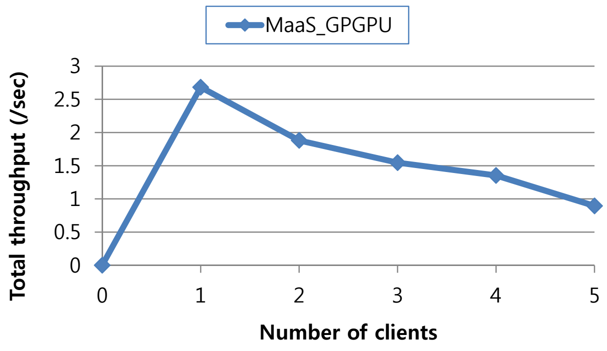

Fig. 8. GPGPU System Throughput with Various Client Numbers

loads. Because each image frame has different workload, there are some time consuming variation among image frames. On clients, however, if they have same workload, their operation time is almost same with each other. This result implies that each client are well-parallelized, independent to various workload.

C. GPGPU system parallelization measurement

1) Throughput with client number variation: As manycore case, total throughput is measured with various numbers of clients. Fig. 8 shows the result.

In this case, there are only one GPGPU on the system for MaaS operation. Therefore, if there is more than one client, they should compete with each other to access GPGPU. It means, for one GPGPU, access from only one client can make highest throughput. This problem comes from SIMD architecture, as explained. Therefore, for GPGPU system, parallelization on amount of cores per one image is better approach.

V. Conclusion

In this paper, we designed MaaS, real-time multimedia matching system, with focusing on parallelization efficiency. On manycore system, one thread per one image parallelization method is used, according to NUMA overhead on this system. This system is better when the number of clients increase more and more. On GPGPU system, hundreds of cores per one image parallelization method is used, because of SIMD architecture of GPGPU. This system has advantage when some clients want much faster response for matching operation.

For further works, multiple GPGPU system can be considered. If there are more than one GPGPU, one GPGPU per one image parallelization can be applied, and it can reduce competition overhead among several clients. Additionally, manycore and GPGPU hybrid system can be used. This hybrid system can get the profits of both manycore system and GPGPU system.

References

[1] G. Geisler and S. Burns, "Tagging video: conventions and strategies of the youtube community," in *Proceedings of the 7th ACM/IEEE-CS joint conference on Digital libraries*, ser. JCDL '07. New York, NY, USA: ACM, 2007, pp. 480–480. [Online]. Available: http://doi.acm.org/10.1145/1255175.1255279

[2] Youtube. Youtube, multimedia sharing. [Online]. Available: http://www.youtube.com/

[3] Google. Google goggles. [Online]. Available: http://www.google.com/mobile/goggles/

[4] ——. Picasa. photo tagging, categorizing service. [Online]. Available: http://picasa.google.com/

[5] KT. Pudding camera. [Online]. Available: http://itunes.apple.com/us/app/id379411152

[6] J. H. Choi, K. W. Park, S. K. Park, and K. H. Park, "Multimedia matching as a service: Technical challenges and blueprints," in *Proceedings of the 26th international technical conference on circuits/systems, computers and communications*, ser. ITC-CSCC'11, 2011, pp. 632–635.

[7] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vision*, vol. 60, no. 2, pp. 91–110, Nov. 2004. [Online]. Available: http://dx.doi.org/10.1023/B:VISI.0000029664.99615.94

[8] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," in *Computer Vision ECCV 2006*, ser. Lecture Notes in Computer Science, A. Leonardis, H. Bischof, and A. Pinz, Eds., vol. 3951. Springer Berlin / Heidelberg, 2006, pp. 404–417, 10.1007/11744023_32. [Online]. Available: http://dx.doi.org/10.1007/11744023_32

[9] M. Trajkovi and M. Hedley, "Fast corner detection," *Image and Vision Computing*, vol. 16, no. 2, pp. 75 – 87, 1998. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0262885697000565

[10] J. Beis and D. Lowe, "Shape indexing using approximate nearest-neighbour search in high-dimensional spaces," in *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*, jun 1997, pp. 1000 –1006.

[11] K. Klues, B. Rhoden, A. Waterman, D. Zhu, and E. Brewer, "Processes and resource management in a scalable many-core os," in *Proceedings of the 2nd USENIX workshop on Hot Topics in Parallelism*, jun 2010.

[12] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable parallel programming with cuda," *Queue*, vol. 6, no. 2, pp. 40–53, Mar. 2008. [Online]. Available: http://doi.acm.org/10.1145/1365490.1365500

[13] S. N. Sinha, J. michael Frahm, M. Pollefeys, and Y. Genc, "Gpu-based video feature tracking and matching," In Workshop on Edge Computing Using New Commodity Architectures, Tech. Rep., 2006.

[14] R. Hess, "An open-source siftlibrary," in *Proceedings of the international conference on Multimedia*, ser. MM '10. New York, NY, USA: ACM, 2010, pp. 1493–1496. [Online]. Available: http://doi.acm.org/10.1145/1873951.1874256