ACCELERATING QUERY BY SINGING/HUMMING ON GPU: OPTIMIZATION FOR WEB DEPLOYMENT

Chung-Che Wang, Chieh-Hsing Chen, Chin-Yang Kuo, Li-Ting Chiu and Jyh-Shing Roger Jang

Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan
{geniusturtle, chchen, jimmy, jesmallchiu, jang}@mirlab.org

ABSTRACT

This paper presents the use of GPU for implementing a parallelized comparison method of linear scaling in a query by singing/humming system, which can compare a user's acoustic input to the database containing about 13,000 songs. We focus on the comparison from anywhere in a song, and the optimum setting is found through 3 different schemes of parallelization. With a speedup factor of 66, the proposed scheme with the optimum setting has been successfully implemented in a public QBSH system that is available from the internet.

Index Terms— Music retrieval, Query-by-singing/humming, linear scaling, GPU

1. INTRODUCTION

Query by singing/humming (QBSH) is an intuitive method for music retrieval. With a QBSH system, users are able to retrieve intended songs by singing or humming a portion of them. Ghias et al. published one of the early papers of query by humming, which used three different characters ('U', 'D', and 'S') to represent pitch contours [1]. McNab et al. enhanced the representation by considering rhythm information of segmented notes [2]. Jang et al. proposed the first QBSH system using linear scaling (LS) over frame-based pitch contours, which accommodates natural singing/humming for better performance [3]. An online demonstration system, which is called MIRACLE (Music Information Retrieval Acoustically with Clustered and paralleL Engine), was proposed by Jang et al. by using clustered computing [4].

With the availability of GPUs (Graphic Processing Units), a number of time-consuming recognition-related tasks have been speeded up such that commercial or real-time applications become possible. In particular, Poli et al. applied dynamic time warping (DTW) on voice password identification system on GPUs, where four-digit passwords were used in their experiments [5]. Li et al. evaluated the probability of hidden Markov models on GPUs, where forward probability were calculated and summed up in parallel [6]. Sart et al. parallelized DTW on GPUs and FPGAs (Field Programmable Gate Arrays) to perform subsequence search of ECG (Electrocardiography) traces and star light curves [7]. They parallelize the search for a query in a time sequence. The procedure is repeated when there are many time sequences needed to be searched.

A similar study of accelerating QBSH on GPUs is proposed by Ferrao et al. [8]. However, the comparison only started from the beginning of a song in the database, which may violate people's singing habits. In this paper, comparison starts from anywhere in a song where notes begin, and the schemes of parallelization are directly designed for searching many sequences (i.e. many songs in the database). The proposed method is used to improve a web-deployed QBSH system called MIRACLE [9] which won the championship of 2011 CUDA programming contest in Taiwan, hosted by NVIDIA [10].

The remainder of this paper is organized as follows. Fundamentals of LS and GPUs are described in section 2 and 3 respectively. Three different schemes of parallelization are introduced in section 4. Experimental results and analysis are shown in section 5. We conclude this paper and address directions for future work in section 6.

2. LINEAR SCALING

Linear scaling is a simple yet effective method for query by singing/humming [3]. Since the keys and tempos may be different between the input pitch vector and the songs in the database, we need to transpose the key and scale the tempo of the input vector. Key transposition can be simply handled by shifting the mean values of pitch vectors of query input and the intended songs in the database to the same value. Usually we shift one of them to match another when comparing these two vectors. For tempo scaling, since tempo variation is usually linear, we can apply linear scaling to the input pitch vector for comparison. Assuming that the input pitch vector has a duration of d seconds, then we need to compress or stretch the vector to obtain r versions of the original vector, with durations equally spaced between $s_{min} \times d$ and $s_{max} \times d$, where s_{min} (<1) and s_{max} (>1) are the minimum and maximum of the scaling factor, respectively. The distance between the input pitch vector and a particular song is then the minimum of the r distances between the r vectors and the song, as shown in figure 1 where we compress/stretch the d-second vector to obtain 5

vectors with lengths equally spaced between $0.5 \times d$ and $1.5 \times d$. The best result is obtained when the scaling factor is 1.25.

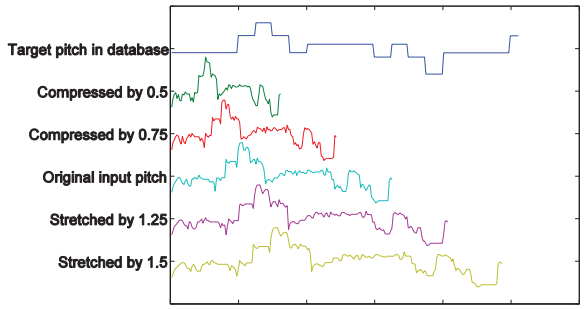

Figure 1. An example of LS.

3. GPUS' ARCHITECTURE AND PROGRAMMING

A GPU consists of several streaming multiprocessors (SMs), each of which is composed of dozens of cores (streaming processors, SPs), on-chip shared memories, and registers. There are also constant memory, texture memory, and global memory shared by all of the SMs. Constant and texture memories can be accessed rapidly, but they are read-only by the GPU. Global memory is much larger, and it can be written by the GPU. But the access time is usually several hundreds times longer than those of constant and texture memories. Figure 2 shows the basic architecture of the GPU. Arrows indicate the directions of data transfer.

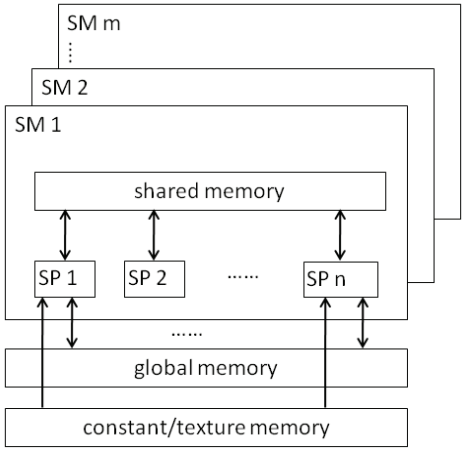

Figure 2. GPUs' architecture.

CUDA (Compute Unified Device Architecture) is a parallel computing framework developed by NVIDIA for their recent GPUs. It can be viewed as an extension of C, which allows programmers to define C functions (called kernels) to be executed in parallel by different CUDA threads. Several threads are grouped in a block. Data in shared memory is shared by all threads within a block. In this study, NVIDIA GeForce GTX 560 Ti was used in our experiments, which contains 384 cores (48 cores per SM), that share a global memory of 1 GB with 256-bit interface width providing a throughput of 128 GB/sec. The number of threads within one block is limited up to 1024.

4. PARALLEL IMPLEMENTATION

At the time of system startup, we first expand the music notes in the database into pitch vectors (for frame-based comparison), and then move the whole database to global memory. For compressing/stretching the input pitch vector, we simply launch r threads for each different scaling factor. Then we move the scaled vectors back to the main memory, and then move them to GPUs' constant memory to speed up the access. Since our system allows frame-based comparison from any note, our memory access pattern is quite different from those in [7, 8] since each note may have different duration. Thus we turn to investigate different schemes for optimum parallelization, as shown next.

- In the first scheme, we simply launch N threads for comparing N different songs in the database. The current song segment for comparison is copied to local storages for speeding up the computing.

- In the second scheme, we launch r threads for one song. These r threads are grouped into a block, with N blocks in total. However, though the degree of parallelization is higher, the computation time is even longer. The reason is that there are many blocks but only a few of threads within one block – the SPs are not fully utilized.

- In the third scheme, we still have N blocks for N songs, but now each block has k threads in a block. The computation tasks starting at different notes in a song are equally distributed to the k threads. Since k is usually larger than r, the utilization of SPs is better than that of scheme 2. Moreover, since there are multiple threads for one song, we obtain the minimum distance between the input pitch vector and this song in parallel by using these threads.

After obtaining the distance between the query input and each of the songs in the database, we then sort all the distances on CPU to obtain the top-n list. We did try the sorting using GPU, but the performance is not satisfactory due to excessive access time over the global memory.

5. EXPERIMENTAL RESULTS AND ANALYSIS

We used the public corpus MIR-QBSH [11] for our experiments of QBSH with GPU. Note that in this corpus, the anchor positions for all queries are from the beginning of a song. In order to test the accuracy of "anchor anywhere", we duplicate the last one fourth of each song and prepend it to the beginning of the song. The corpus contains 6197 clips which correspond to 48 children's songs. To increase the

complexity of the comparison, we added 12887 noise songs (which correspond to pop songs in the past decades) to the database, such that the number of songs in the database is 12935. Figure 3 shows the distribution of song lengths, in terms of number of music notes and number of pitch points, respectively. This plot indicates the complexity of our task. The number of music notes indicates how many positions we need to start the comparison algorithm, while the number of pitch points represents how long the sequence we need to run through the comparison algorithm.

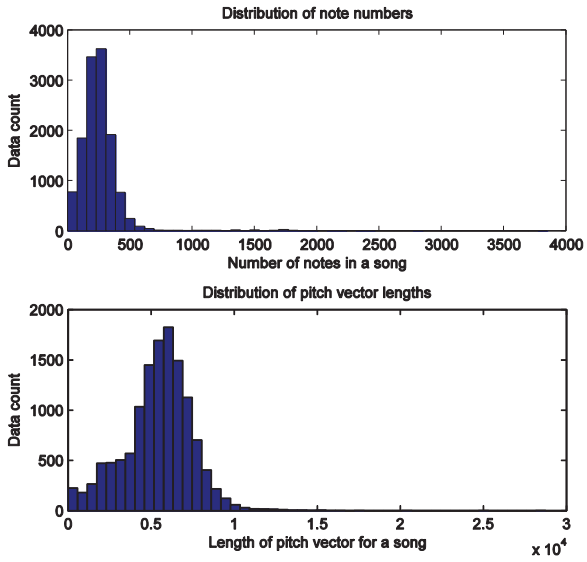

Figure 3. Distribution of song lengths. The upper plot shows the distribution of note numbers in a song, while the lower plot shows the distribution of pitch vector lengths.

In our experiment, the scaling factor was varied from 0.6 to 1.5 to obtain 31 compressed or stretched versions of the original query input vector. Moreover, the frame size is 256 points with no overlap; the sample rate is 8 KHz, leading to a pitch rate of 31.25/sec. The top-n recognition rate is shown in figure 4.

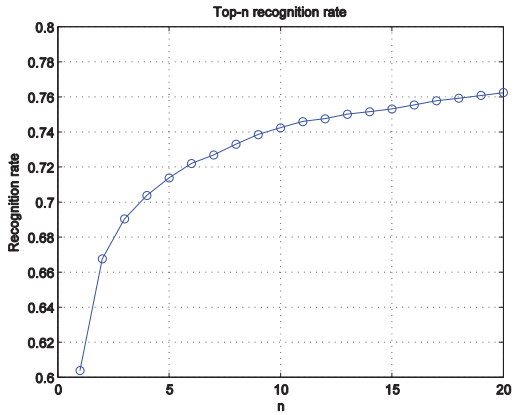

Figure 4. Top-n recognition rate.

Figure 5 and 6 show the computation time per query with respect to the number of songs in the database for the three different schemes of parallelization. In figure 5, numbers of threads in a block are 1024 for schemes 1 and 3, and 31 for scheme 2. In figure 6, numbers of threads in a block are 128 for schemes 1 and 3, and 31 for scheme 2. As shown in these figures, scheme 3 is the fastest, follow by schemes 1 and 2. More specific, scheme 3 is about 66 times faster than the original CPU version, which is running on a PC with and i7-2600 processor and 16 GB DDR-3 1600 memories, demonstrating the effectiveness of the proposed method.

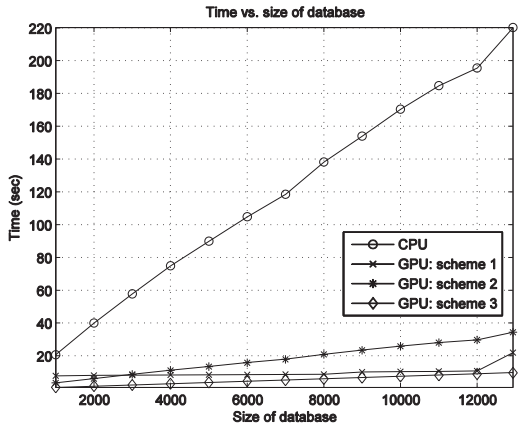

Figure 5. The computation time per query with respect to the number of songs in the database for the three different schemes of parallelization. The number of threads in a block is 1024 for schemes 1 and 3, and 31 for scheme 2.

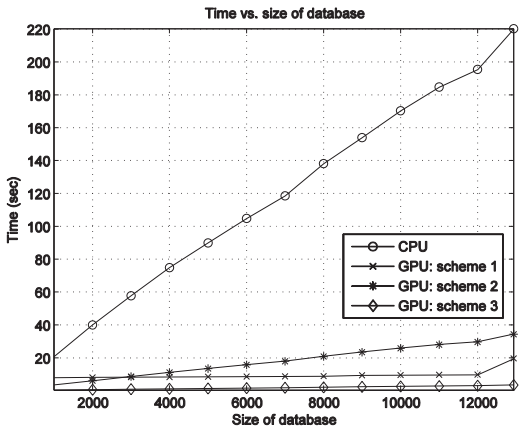

Figure 6. The computation time per query with respect to the number of songs in the database for the three different schemes of parallelization. The number of threads in a block is 128 for schemes 1 and 3, and 31 for scheme 2.

The above figures suggest that number of threads in a block is an important factor for scheme 3. Thus we investigated the effect of number of threads in a block for scheme 3, as shown in figure 7. The best performance is achieved when there are 128 threads in a block. Utilization

of cores in GPU will be lower if we launch fewer threads in a block. On the other hand, if we have more threads than 128, then it becomes time consuming to compute the minimum distance for a song which is obtained from these threads. The snapshot of our system is shown in figure 8.

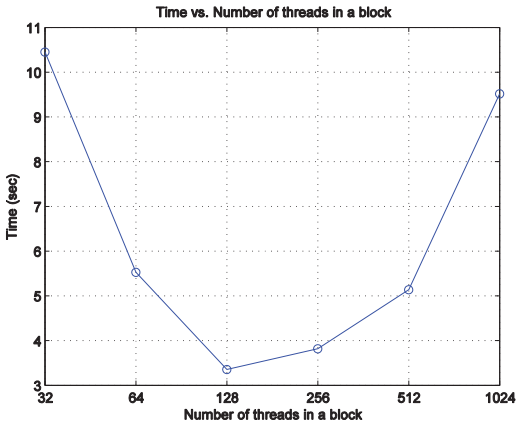

Figure 7. Computation time versus number of threads in a block.

Figure 8. A snapshot of our system.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we have proposed three parallelized schemes of LS on GPU for QBSH, where the comparison starts from anywhere in the songs. The speedup factor is about 66, and the response time has been reduced from 3 minutes to 3 seconds, which is a critical factor when we are considering the usability of a web-deployed QBSH system available at [9].

Several directions for immediate future work are under way. Currently, linear scaling is employed in our system, which can deal with uniform scaling. For non-uniform scaling, we can apply DTW to achieve a higher degree of flexibility, which is the goal of our current focus. Moreover, due to efficiency of GPU-based computation, now it becomes possible to incorporate multiple recognizers for achieving a better accuracy, which is the second task of our future work.

7. REFERENCES

[1] A. J. Ghias, D. C. Logan, and B. C. Smith, "Query by humming-musical information retrieval in an audio database," *in Proc. ACM Multimedia'95*, San Francisco, pp. 216–221, 1995.

[2] R. J. McNab, L. A. Smith, I. H. Witten, C. L. Henderson, and S. J. Cunningham, "Toward the digital music library: Tune retrieval from acoustic input," *in Proc. ACM Digital Libraries*, pp. 11–18, 1996.

[3] J.-S. R. Jang, H.-R. Lee, and M.-Y. Kao, "Content-based Music Retrieval Using Linear Scaling and Branch-and-Bound Tree search," *in Proc. of IEEE International Conference on Multimedia and Expo*, August 2001.

[4] J.-S. R. Jang, J.-C. Chen, and M.-Y. Kao. "MIRACLE: A Music Information Retrieval System with Clustered Computing Engines," *in Proceedings of the 2nd International Conference on Music Information Retrieval, ISMIR 2001*, 2001.

[5] G. Poli, A. L. M. Levada, J. F. Mari, J. H. Satio, "Voice Command Recognition with Dynamic Time Warping (DTW) using Graphics Processing Units (GPU) with Compute Unified Device Architecture (CUDA)," *in Proceedings of the 19th International Symposium on Computer Architecture and High Performance Computing , SBAC-PAD 2007*, Brazil, pp. 19–25, 2007.

[6] Jun Li, Shuangping Chen, Yanhui Li, "The Fast Evaluation of Hidden Markov Models on GPU," *in IEEE International Conference on Intelligent Computing and Intelligent Systems, Shanghai*, vol. 4:426-430, Nov., 2009.

[7] D. Sart, A. Mueen, W. Najjar, E. Keogh, and V. Niennattrakul, "Accelerating Dynamic Time Warping Subsequence Search with GPUs and FPGAs," *in ICDM '10 Proceedings of the 2010 IEEE International Conference on Data Mining*, pp. 1001-1006, 2010.

[8] P. Ferraro, P. Hanna, L. Imbert, and T. Izart, "Accelerating Query-by-Humming on GPU," *in Proceedings of the 10th International Conference on Music Information Retrieval, ISMIR 2009*, pp. 279–284, 2009.

[9] Chung-Che Wang, Chieh-Hsing Chen, Chin-Yang Kuo, Li-Ting Chiu and Jyh-Shing Roger Jang, "Welcome to Miracle!" http://cuda.mirlab.org, 2011

[10] NVIDIA, "2011 CUDA programming contest in Taiwan," http://nvidia.ithome.com.tw/cuda/, 2011

[11] J.-S. R. Jang, "MIR-QBSH Corpus", MIR Lab, CS Dept, Tsing Hua Univ, Taiwan. Available at the "MIR-QBSH Corpus" link at http://mirlab.org/jang.