Parallelization of Beat Detection Algorithm

Bryan Bastida
Gabriel Fajadro
Lissett Munoz
Oscar Padilla

Florida International University
Miami, Florida

ABSTRACT

Marco Ziccardi's Sound Energy Algorithm is an algorithm designed with the goal of detecting a song's tempo, or more specifically it's beats per minute. This beat detection algorithm divides a song into blocks of samples, finds the average energy of these samples in each block, and if the energy of 4 consecutive blocks meet the specific threshold a beat is counted. In an effort to optimize this algorithm through parallelization, we inserted a techno .wav file's samples, or frequencies, from a text file to an array and then passed it to a GPU where we put each index on its own thread and used a parallel reduction algorithm to compute the instant energies of blocks of samples more efficiently before sending back that array and continuing the next calculations back on the CPU. We concluded after our optimizations that parallelizing the Sound Energy Algorithm's process of calculating the instant energies of blocks resulted in a consistent minor slowdown of milliseconds.

I. Introduction:

Knowing the BPM (Beat Per Minute) of a song is crucial for a DJ when they're mixing samples and music together. We decided we would take one of the many beat detection algorithms out there and find a way to parallelize it. We found one written by a computer scientist named Marco Ziccardi. In his algorithm he explains the steps needed to detect a beat in a song. A song is made up of millions of frequencies. One second of a song, assuming the sampling rate is 44,100 hz is 44,100 frequencies. An instant of a song is 1/500 of a second which translates to 1024 frequencies. By dividing 44,100 / 1024 we get 43 blocks. Each one of those blocks will contain the sum of 1024 frequencies which have been squared and multiplied by 2 (because we are using mono channel rather than stereo). Then each block will be compared to C * average of all 43 blocks. If the block is greater than C * average then we've found a peak. If we have four consecutive peaks then we've found a beat. We address our problem by passing our frequencies to the GPU which then puts each frequency on its own thread. We then use reduction

in the GPU to which gives us the value of each block. We then pass that information back from the GPU to compare the blocks to C * average.

II. Related Work:

There exists many beat detection softwares and applications such as MixMeister BPM Analyzer, BPM Detector Pro, and Pistonsoft BPM Detector. Not all these softwares use the same algorithm as Marco Ziccardi which divides the frequencies of a wave file into blocks of samples and compares their energy with the energy of the preceding blocks. Using this algorithm he created his own beat detection program in Scala without parallelization. For our project we used his beat detection algorithm to create a program in Cuda that detects the beats per minute of a song on the Graphics Processing Unit using frequencies from .wav file of songs.

III. Process:

The idea was to parallelize the "Sound Energy Algorithm" provided by M. Ziccardi. However, first we coded it without any optimizations. We only concerned ourselves with the detection of the beats. We did not take into account how we would get the samples of the song. We assumed we would have a file that would be used as input to the code. A Java class called StdAudio written by Robert Sedgewick and Kevin Wayne was used for this purpose. The class took care of creating a text file with all the samples provided by a .wav file.

An array is initialized using the values from the file containing all the samples of the song. The array is passed to a function that calculates the instant energies for each block in a second. This means that 43 blocks are computed at a time. Then, the average is found by adding all the instant energies in the second and dividing by the number of blocks (43). The variance is found after since it's a summation of the difference from the block's instant energy and the average. The variance is used to calculate a constant, which is multiplied with the average to compare with each block's instant energy, one by one.

Now, once this was coded, it was time to see where parallelization could be done. Obviously, the largest computation happens when calculating the instant energies. Once again, a block contains 1024 samples. However, all these samples are combined in the summation:

$$E_j = \sum_{i=0}^{1023} channel[i]^2 * 2$$

So, first each thread squared and multiplied by two its corresponding sample and then stored it in the original array holding all the samples of the song. Once this was computed, a core

with 1024 threads added 1024 consecutive elements of the array to produce the instant energy for the block. This was done with a parallelization technique called reduction.

Reduction uses threads to break up the work of adding elements in the array. This is done by adding the element corresponding to a thread with some offsetted element and storing them in the position corresponding to the thread doing the addition. Basically, it's a division of labor between threads, and as more and more additions are made less and less threads are used until final only one thread adds two cumulated additions and finishes the addition. See the following figure to understand this more clearly.

In the end, the original array gets reduced by a factor of 1024. To understand this more clearly, with our program, the array containing about 9,200,000 samples gets reduced to about 8984 instant energies. This new array is then returned from the GPU and used to compute the final parts of the algorithm, which are implemented the same as the initial version of the program.

IV. Results:

We tested our program using various songs in the techno genre and a a clip of a metronome playing at 40 beats per minute. While testing various songs we found that our beats per minute detector could only accurately find the beat of one techno song from YouTube, "C-Bool - MBrotherTrebles(TaTaTa)". It could not find the correct beats per minute of any of the other songs we tested such as Armin van Buuren's "Ping Pong (Extended Edition)" or Darude's "Sandstorm". When we initially found that only "MBrotherTrebles(TaTaTa)" had an accurate beats per minute computed we decided to test the program using a metronome that had

4 beats per minute. Our program detected that the metronome clip had 60 beats per minute. Upon finding that we realized that our algorithm assumed that the length of a beat was four peaks, meaning a beat was detected whenever instant energy was bigger than linear regression * average energy, four times in a row. Finding this out we realized that the algorithm for computing beats per minute created by Marco Ziccardi is inefficient. We also found that computing the beats per minute using the GPU had about the same runtime as computing beats per minute without it.

V. Conclusion

Our optimizations using multithreading in the GPU for calculating instant energies resulted in a consistent minor slowdown of milliseconds resulted in a consistent minor slowdown of milliseconds, meaning parallelizing the Sound Energy Algorithm has proven ineffective using our methods. The implications of our conclusions could lead to future more efficient attempts in order to optimize beat detection algorithms, such as optimizing the reduction algorithm or using shared memory rather than global memory for the computations in the GPU.

VI. Tools

Convert MP3 to WAV
http://audio.online-convert.com/convert-to-wav

MixMeister BPM Analyzer
http://download.cnet.com/MixMeister-BPM-Analyzer/3000-2169_4-10290906.html

Works Cited

Z, Marco. "Disclaimer." *GameDev.net -- Beat Detection Algorithms*. N.p., n.d. Web. 25 Apr. 2017.

Ziccardi, Marco. "Marco Ziccardi." *Beat Detection Algorithms (Part 1) · Marco Ziccardi*. N.p., 28 May 2015. Web. 1 Apr. 2017.