Programming Convolution

Name : Dekas Dimitrios

AEM : 3063

Topics

- Convolution's calculation between two vectors.
- Convolution's calculation between two WAV files.
- CUDA programming in order to parallelize convolution's caclulations using GPU ar.

Convolution beteen two vectors

```
141  /**
142   *  The code used to complete the first task.
143   */
144  void firstTask() {
145      // Get the size of vector A
146      std::cout << "Enter the wished size of vector A: " << std::endl;
147      int size = getIntInput();
148      std::vector<double> a;
149
150      // Fill vector A with random numbers
151      fillVector( &a, size);
152
153      // Define vector B
154      std::vector<double> b = {0.2, 0.2, 0.2, 0.2, 0.2};
155
156      // Calculate the convolution between A and B
157      std::vector<double> convolution = myConvolve(a, b);
```

- Use the getIntInput() function to ensure that the input given by the user is indeed an integer
- Use the fillVector() function to fill the vector A with random numbers
- Use the myConvolve() function to calculate the convolution's result between the two vectors

MyConvolve() function's implementation

```
99      /**
100      *  Calculates the convolution of two functions.
101      *
102      *  @param x : a vector representing the x function on certain values.
103      *  @param h : a vector representing the h function on certain values.
104      *  @return a vector containing the result of the convolution of the two inputs, x and h.
105      */
106     std::vector<double> myConvolve(const std::vector<double> &x, const std::vector<double> &h) {
107         const int xs = x.size();
108         const int hs = h.size();
109         const int cs = xs + hs - 1; // Size of the convolution's vector
110         std::vector<double> c(cs,  value: 0.0);
111         for(auto i(0); i < cs; i++) {
112             c[i] = 0.0;
113             const unsigned long jmin = (i >= xs - 1) ? i - xs + 1 : 0; // The lower bound for the j-loop
114             const unsigned long jmax = (i < hs - 1) ? i : hs - 1; // The upper bound for the j-loop
115             for(auto j(jmin); j <= jmax; j++) {
116                 c[i] += x[i - j] * h[j]; // Use convolution's formula
117             }
118         }
119         return c;
120     }
```

Excecute and Timing First Task

We choose to initialize the vector A with 1500000 random elements.

```
dekasdimitrios@pop-os:~/Desktop/AUTH/projects/signals-convolution$ g++ -Ofast main.cpp
dekasdimitrios@pop-os:~/Desktop/AUTH/projects/signals-convolution$ time ./a.out
Enter the wished size of vector A:
1500000
Simple convolution task completed.

real    0m6.702s
user    0m0.414s
sys     0m0.012s
```

After excecuting and timing the program that calculates the convolution of the two vectors, we got the above results.

WAV files library

```
/** @file AudioFile.h
 *  @author Adam Stark
 *  @copyright Copyright (C) 2017  Adam Stark
 *
 * This file is part of the 'AudioFile' library
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program.  If not, see <http://www.gnu.org/licenses/>.
 */
```

Convolution beteen two WAV files (a)

- Initialize an AudioFile and use load() function to get the WAV content.
- Get WAV's first channel samples using the samples.at.(0) command.
- After calculating the convolution of the two files, save the result with the save() function.

```
/**
 *  The code used to complete the second task.
 */
void secondTask() {
    // Load Files
    AudioFile<double> sampleAudio_file;
    AudioFile<double> pinkNoise_file;
    sampleAudio_file.load(SAMPLE_AUDIO_FILE_PATH);
    pinkNoise_file.load(PINK_NOISE_FILE_PATH);

    // Get Files' Samples
    std::vector<double> sampleAudio_samples = sampleAudio_file.samples.at( n: 0);
    std::vector<double> pinkNoise_samples = pinkNoise_file.samples.at( n: 0);

    // Calculate the samples' convolution
    std::vector<double> pinkNoise_sampleAudio_samples = myConvolve(sampleAudio_samples, pinkNoise_samples);

    // Save the result to a wav file
    AudioFile<double> pinkNoise_sampleAudio_file;
    pinkNoise_sampleAudio_file.samples[0] = pinkNoise_sampleAudio_samples;
    pinkNoise_sampleAudio_file.save(PINK_NOISE_SAMPLE_AUDIO_FILE_PATH,  format: AudioFileFormat::Wave);
```

Convolution beteen two WAV files (b)

```
194     // Define the white noise vector
195     std::vector<double> whiteNoise_samples;
196
197     // Generate the white noise signal, a vector containing values between -1 and 1
198     fillVector( &: whiteNoise_samples, WHITE_NOISE_SIZE);
199     for (auto i(0); i < WHITE_NOISE_SIZE ; i++) {
200         whiteNoise_samples[i] =  (2 * whiteNoise_samples[i]) - 1;
201     }
202
203     // Calculate the samples' convolution
204     std::vector<double> whiteNoise_sampleAudio_samples = myConvolve(sampleAudio_samples, whiteNoise_samples);
205
206     // Save the result to a wav file
207     AudioFile<double> whiteNoise_sampleAudio_file;
208     whiteNoise_sampleAudio_file.samples[0] = whiteNoise_sampleAudio_samples;
209     whiteNoise_sampleAudio_file.save(WHITE_NOISE_SAMPLE_AUDIO_FILE_PATH,  format: AudioFileFormat::Wave);
210 }
```

- Create the white noise vector by initializing it with random values between -1 and 1.
- After calculating the convolution of the two files, save the result with the save() function.

Excecute and Timing Second Task

```
dekasdimitrios@pop-os:~/Desktop/AUTH/projects/signals-convolution$ g++ -Ofast main.cpp
dekasdimitrios@pop-os:~/Desktop/AUTH/projects/signals-convolution$ time ./a.out
WAV files task completed.

real    8m7.063s
user    8m6.914s
sys     0m0.052s
```

After excecuting and timing the program that calculates the convolution of the two WAV files, we got the above results.

Convolution in the GPU using CUDA : Setting Up the Arrays

```
212  /**
213   *  The code used to complete the third task.
214   */
215  void thirdTask() {
216      // Get the size of vector A
217      std::cout << "Enter the wished size of vector A in CUDA: " << std::endl;
218      int size = getIntInput();
219      double* a;
220      cudaMallocManaged(&a , size: size * sizeof(double));
221
222      // Fill A array with random numbers
223      fillArray(a, size);
224
225      // Define B array
226      int bsize = 5;
227      double* b;
228      cudaMallocManaged(&b,  size: bsize * sizeof(double));
229      for (auto i(0); i < bsize ; ++i) {
230          b[i] = 0.2;
231      }
232
233      // Define Convolution's array
234      int csize = size + bsize - 1;
235      double* convolution;
236      cudaMallocManaged(&convolution,  size: (size + bsize - 1) * sizeof(double));
237
```

- Create the necessary arrays and allocate memory for them using cudaMallocManaged()
- Fill the A array with random numbers
- Fill the B array with 0.2
- Just define the convolution array

Convolution in the GPU using CUDA : Invoke CUDA kernel

```
238     // Calculate the convolution between A and B
239     int numThreads = 512;
240     int numBlocks = (csize + numThreads - 1) / 512;
241     cudaConvolve<<<numBlocks, numThreads>>>(a, b, size, bsize, convolution);
242
243     // Wait for GPU to finish
244     cudaDeviceSynchronize();
```

- Use a multiple of 32 as the ammount of threads we with to use (optimal purpose)
- Use the needed number of blocks to fully parallelize the process
- Call the cudaConvolce() function using the <<<>>> syntax
- Use the cudaDeviceSynchronize() function in order to make the main program excecuting in th CPU wait for the device program to complete its processes

Convolution in the GPU using CUDA : Always free allocated memory

```
258         // Free memory
259         cudaFree(a);
260         cudaFree(b);
261         cudaFree(convolution);
```

Use cudaFree() function in orded to free up the allocated space

Convolution in the GPU using CUDA : Kernel implemantation

```
122  /**
123   *  CUDA Kernel function that calculates the convolution of two signal functions.
124   *
125   *  @param x : a double array representing the x function on certain values.
126   *  @param h : a double array representing the h function on certain values.
127   *  @param xs : x array size
128   *  @param hs : h array size
129   *  @param c : a double array containing the result of the convolution of the two inputs, x and h.
130   */
131  __global__ void cudaConvolve(const double* x, const double* h, const int xs, const int hs, double* c) {
132      int tid = threadIdx.x + blockIdx.x * blockDim.x;
133      c[tid] = 0.0;
134      const unsigned long jmin = (tid >= xs - 1) ? tid - xs + 1 : 0; // The lower bound for the j-loop
135      const unsigned long jmax = (tid < hs - 1) ? tid : hs - 1; // The upper bound for the j-loop
136      for(auto j(jmin); j <= jmax; j++) {
137          c[tid] += x[tid - j] * h[j];
138      }
139  }
```

The tid variable is used to store the thread index for every cudaConvolve() invocation. Using this function in the GPU instead of the normal CPU excecution we benefit by the big ammount of parallelization we are able to obtain.

Excecute and Timing Third Task

```
dekasdimitrios@pop-os:~/Desktop/AUTH/projects/signals-convolution$ nvcc ./main.cu -o conv_cuda
dekasdimitrios@pop-os:~/Desktop/AUTH/projects/signals-convolution$ time ./conv_cuda
Enter the wished size of vector A in CUDA:
1500000
CUDA task completed.

real    0m3.689s
user    0m0.483s
sys     0m0.105s
```

After excecuting and timing the program that calculates the convolution of the two vectors using CUDA, we got the above results.

CPU parallelization benefits

The arrays used in the above examples (A and B) do not make the benefits obvious. In order to understand and appreciate the value of the parallelization we should use two sized arrays and calculate the convolution between them.

If we were to calculate the convolution of two arrays with size equal to 500.000 elements each, the CPU code would take up to 30 minutes long.

On the other hand, for the two arrays mentioned above, the CUDA program excecuting in the GPU would only take 4 seconds!!!