Artifical Neural Networks for Computer Vision
Assignment-1 (CUDA programming)

Arulkumar S (CS15S023)

August 14, 2015

1 Dot product of two vectors

Source code

1.1 Approach

* The program asks for an input (Number of components in the vectors) from the user
* The program randomly generates two vectors whose components count is equal to the Number of components given by the user
* The equivalent component (or) co-efficient of all the basis (indexes) are passed to each thread which is running in device.
* The number of threads in each block is 512 & the number of blocks is computed dynamically according to the Number of components in the vector.

  Number of Blocks = (Vector_Dimension % 512) + 1;
  Number of threads = 512
* To compute dot-product, Each thread will multiply the appropriate index-ed component (identified by BlockID & ThreadID) of given vectors. There is a shared variable available for all threads to store the product of appropriate components of the two vectors.
* At the end, the main program will calculate the addition of all the products which are stored in shared variable and write the output to the data file

1.2 Why this approach?

To avoid the sequential processing of each componenet of the vector (going through all the components one-by-one and to calculate the product, add them), I have used multiple blocks and 512 threads per each block, so that the product of vector components will be done in parallel.

1.3 Observed Results

Average time taken for Vectors with 10000 components

* PC execution: 90 to 100 microseconds
* GPU emulation execution: 15 to 25 microseconds

1.4 Output file

Task1 output file

2 Multiplication of two matrices

Source code

2.1 Approach

* The program will ask for the total number of rows and columns for each matrix.
* The program will randomly generate two matrices according to the total number of rows & columns given by the user
* The program calls the device function with
  Number of blocks = Total rows of First matrix
  Number of Threads per block = Total columns of second matrix
* Each thread of every block will take the elements of particular row from Matrix-1 (identified by BlockID) & multiply them with appropriate column in Matrix-2 (identified by ThreadID), store the result in appropriate index of output parameter.
* After the device execution completed, the result is copied to Host & printed into report text file.

2.2 Why this approach?

To avoid the sequential processing of each row of Matrix-1 + column of Matrix-2 multiplication (going through all the rows of Matrix-1 one-by-one and to multiply them with columns of Matrix-2, add them), I have used multiple blocks and threads to parallelize the multiplication into GPU cores.

2.3 Observed Results

Average time taken for Matrix-1 (2000 x 400), Matrix-2 (400 x 500) is

* PC execution: 3.2 to 3.5 seconds
* GPU emulation execution: 20 to 30 microseconds

2.4 Output file

Task2 output file

3 Convolution of a big matrix A over a small matrix S

Source code

3.1 Approach

* The row count & column count of (Big) Matrix to be convoluted & the (small) filter matrix is to be inputted by the user.
* The program will randomly generate contents of the Big matrix & small matrix according to the dimension given by the user.
* After having the two matrices ready, the program launches the device function to compute convolution for each element of the big matrix.
* the number of blocks = the row count of the big matrix
  the number of threads = the column count of the big matrix
* each thread in every block will calculate the convolution for a matrix element (identified by BlockID & ThreadID) & store it in appropriate index of "Output" parameter

3.2 Why this approach?

To avoid the sequential processing to compute convolution for each matrix element (going through all the big matrix elements one-by-one and to calculate the convolution), I have tried to use the parallel processing power of GPU and parallelize the convolution calculation of each matrix element.

3.3 Observed Results

Average time taken for Big matrix (500 x 500), Filter matrix (5 x 5) is

* PC execution: 100 to 110 milli-seconds
* GPU emulation execution: 15 to 25 microseconds

3.4 Output file

Task3 output file