(../../index.php)

CS 475: Parallel Programming

Computer Science Department (http://www.cs.colostate.edu)

Fall 2016

Assignment 6 - Back Propagation Learning in CUDA

Colorado State University (http://welcome.colostate.edu/)

Introduction

The purpose of this exercise is to (i) Implement bunch-mode backpropagation learning(BPL) algorithm on GPUs using CUDA. (ii) Study the performance(execution time) with varying bunch size. Here, bunch size means the number of samples trained together in a single pass. (iii) (Optional and limited analysis) Study the trade-off of model accuracy versus the gains of training time that we obtain from bunch BPL.
For this assignment, you will modify the provided host code and parallelize it using CUDA. Download and untar the PA6.tar (PA6.tar) file. Follow the instructions in Lab-6 for compiling and running CUDA program.

Resources that may be useful for this assignment include:

- Backpropogation Learning (https://en.wikipedia.org/wiki/Backpropagation)
- Activation Function (https://en.wikipedia.org/wiki/Activation_function)
- Softmax function (https://en.wikipedia.org/wiki/Softmax_function)
- NVIDIA CUDA Programming Guide (https://developer.nvidia.com/nvidia-gpu-programming-guide).

Provided Code

Here is a set of files/folders provided in PA6.tar:

- README
- Reference/
- Bunch-ANN/
- Bunch-ANN-flatten/

The README file has a detailed description of the contents of the folders, usage of the program, sample input/output and code description.

Bunch-mode BPL in CUDA

The given code (Bunch-ANN/bpl.cu) implements the Bunch-mode BPL on CPU. It is a single hidden layer ANN with N input neurons, M hidden neurons and P output neurons. S is the total number of training samples and I is the bunch size. The following is the sequence of computations found in the code.

Forward Phase	Backpropagation Phase
$Z_h[i][j] = \sum_{k=0}^{N} X[i][k] * W_{xh}[k][j]$ (1)	$dW_{hy}[i][j] = \sum_{k=0}^{I-1} H^\intercal[i][k] * E[k][j]$ (6)
$H[i][j+1] = tanh(Z_h[i][j])$ (2)	$W_{hy}[i][j] -= dW_{hy}[i][j] * learningrate$ (7)
$Z_y[i][j] = \sum_{k=0}^{M} H[i][k] * W_{hy}[k][j]$ (3)	$H[i][j] = \sum_{k=0}^{P-1} E[i][k] * W_{hy}^\intercal[k][j]$ (8)
$P[i][j] = \frac{e^{Z_y[i][j]}}{\sum_{k=0}^{P-1} e^{Z_y[i][k]}}$ (4)	$Z_h[i][j] = H[i][j+1] * (1 - tanh^2(Z_h[i][j]))$ (9)
$E[i][j] = P[i][j] - Y[i][j]$ (5)	$dW_{xh}[i][j] = \sum_{k=0}^{I-1} X^\intercal[i][k] * Z_h[k][j]$ (10)
	$W_{xh}[i][j] -= dW_{xh}[i][j] * learningrate$ (11)

Please look into the provided code/README file for what each term means.

In the code provided, each equation in the above table is executed with a host function call except for equation (4) which is broken down into 3 separate function calls. In order to parallelize this in CUDA, you will have to write and make multiple Kernel calls. Use the code provided in PA5 as your starting point. Modify them appropriately to implement the computations. As a guidance,

1. `Matmult Kernel:` Can be modified to implement equations 1,3,6,8,10.
   Notice that some matrices are transposed and multiplied. You can either write seperate Kernels to reflect this or use a single kernel that can handle all cases.
2. `Vecadd Kernel:` Can be modified to implement equations 2,5,7,9,11.
   Notice that vecadd Kernel involved point-wise addition of a single vector. Here it involves point-wise operations on matrices. You can change the operation from addition to the ones required. Again, you can write separate Kernels or a general one to handle all the cases.
3. `Equation 4 (Softmax function):` It is a bit tricky to implement as it involves reduction of a matrix into a column vector. Again, you can break down the computations or use a single Kernel call.

Please document your implementation in a new README file so as to understand your code.

Report

Use any GPU found in CS department machines. (GPUs found in 120(state-capitals), 225 and 325 lab machines have compatible CUDA driver and runtime versions)

1. Explain briefly how you were able to parallelize bunch-mode BPL algorithm in GPU using CUDA. In particular, how work is distributed among threads/thread-blocks.
2. Collect and compare the time bpl_CPU and bpl_GPU takes for the following 3 input specifications:
   - N=M=P=S=I=1024, L=10.
   - N=M=P=S=I=512, L=10.
   - N=M=P=S=I=256, L=10.
3. Collect and compare the time bpl_GPU takes for various bunch sizes for the input specification(N=M=P=S=1024, L=10, vary I from 1 to 1024).
4. Can you formulate any trade-off between the accuracy of prediction and the execution time?

Submission

Submit your report(PA6.R.pdf) through Checkin.

Submit a tarball(PA6.tar) of your code that contains atleast

- Makefile
- bpl.cu
- util.h
- bunch_ann.h
- timer.cu
- timer.h
- README(Not the given one. A simple document to understand your files)

Include other Kernel files and header files if you have used.
Your code should be able to run with a single "make" command.
There is only one Phase submission for this assignment. No late submissions allowed.
There is no Preliminary testing for this assignment.
In order to verify your code, compare the weight matrices (W_{xh} and W_{hy}) that your code produces to the one that given code produces. It should be the same barring minor precision difference.
DO NOT INCLUDE EXTRA PRINT STATEMENTS WHEN SUBMITTING THE FINAL CODE. The following is a sample output your code should produce during final testing.

```
With verbose:
    $./bpl_CPU -N1 -M4 -S30 -I5 -L10 -P5 -v
    Time: 0.000351

    input/hidden weights:
       0.07025    -0.02358     0.05504     0.06093
       0.08233    -0.06049    -0.03296     0.05365

    hidden/output weights:
      -0.03527     0.01972     0.00444     0.03458    -0.01791
       0.00331     0.09106     0.08386     0.02775     0.04409
      -0.07188     0.02119    -0.09694    -0.05162    -0.07276
       0.06135    -0.06817    -0.01931    -0.07355    -0.07773
       0.10034    -0.05581     0.00313     0.06835     0.02308

Without verbose:
    $ ./bpl_CPU -N1 -M4 -S30 -I5 -L10 -P5
    Time: 0.000364
```