GPU based implemenation of Burrows-Wheeler Transform

Keith Daigle
Advised by Dr. Kjell
Central Connecticut State University, Department of Computer Science
keith.dgl@gmail.com

1. Abstract

In recent years there has been a great increase in the use of graphics processing units, or GPUs, in many areas of computing. Due to the vast computational resources provided by GPUs some of the most recent additions to the top 500 supercomputers in the world have included graphics processing units[1]. While GPUs possess great computational abilities the programming model for these chips differs significantly from the normal sequential style of programming. Software written for GPUs must be designed to maximize parallel operations on the data used in the computations through data-parallel algorithms. This poster provides results of a parallel implementation of the Burrows-Wheeler transform for a GPU.

2. What is a GPU?

- A graphics processing unit, abbreviated GPU, is a computer processor designed to render graphics to a monitor or screen.
- Immense computational resources provided by GPUs makes them attractive for computationally intensive tasks.
- Software toolkits like CUDA and Thrust have helped make the power of GPUs more accessible to problems outside the domain of computer graphics.
- Toolkits make writing parallel software for GPUs easier, but still not as easy as writing sequential software for CPUs.
- Both NVIDIA's CUDA toolkit and Thrust template library were used in this project along with a NVIDIA GeForce GPU and a Tesla GPU.

3. What is the Burrows-Wheeler transform?

- Reversible transform that can be applied to a set of data.
- Operations to apply transform:
  - Rotate input data once for each byte
  - Sort rotations
  - Take last column and index of original data as output.
- Used in data compression to keep runs of like data together to make compression more efficient[2].
- Used in bioinformatics for DNA read alignment[3].
- Transformed data and index can be used to reverse transform and retrieve original input.
- Example on input of 'abracadabra' in next column.

3.1 Rotation of input with first input character in red

```
vsqueezebox: ~
File  Edit  View  Terminal  Help
vsqueezebox:~$ ./bwt.pl abracadabra rotate
Rotations of input: abracadabra
index   rotated text
-----   ------------
0       abracadabra
1       bracadabraa
2       racadabraab
3       acadabraabr
4       cadabraabra
5       adabraabrac
6       dabraabraca
7       abraabracad
8       braabracada
9       raabracadab
10      aabracadabr
vsqueezebox:~$
```

3.2 Sorted rotations with first input character in red

```
vsqueezebox: ~
File  Edit  View  Terminal  Help
vsqueezebox:~$ ./bwt.pl abracadabra sort
Sorted rotations of input: abracadabra
index   sorted rotations
-----   ----------------
10      aabracadabr
7       abraabracad
0       abracadabra
3       acadabraabr
5       adabraabrac
8       braabracada
1       bracadabraa
4       cadabraabra
6       dabraabraca
9       raabracadab
2       racadabraab
vsqueezebox:~$
```

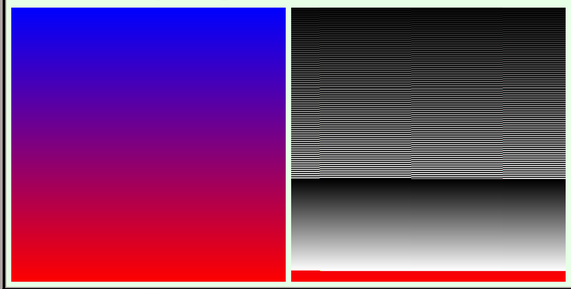
3.3 Output from last column in red

```
vsqueezebox: ~
File  Edit  View  Terminal  Help
vsqueezebox:~$ ./bwt.pl abracadabra final
Last column of sorted rotations:
index   sorted rotations
-----   -----------
10      aabracadabr
7       abraabracad
0       abracadabra
3       acadabraabr
5       adabraabrac
8       braabracada
1       bracadabraa
4       cadabraabra
6       dabraabraca
9       raabracadab
2       racadabraab
Output of BWT is: r d a r c a a a a b b with index: 2!
vsqueezebox:~$
```

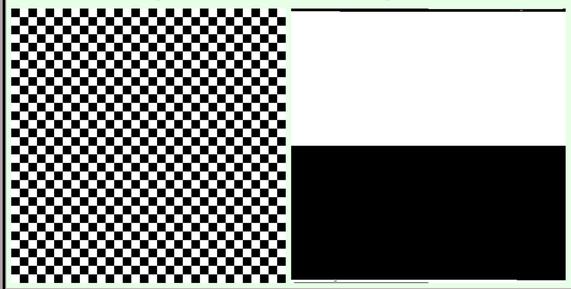
4. Transformed images

In order to test the performance of the GPU implementation some images were chosen to provide average and worst case performance along with an image of the author's dog. In each of the examples below the original image is on the left while the image on the right contains the image after being transformed with the BWT. The images had the inverse transform applied to verify that original data can be obtained and in each case the GPU and reference versions produced identical output.

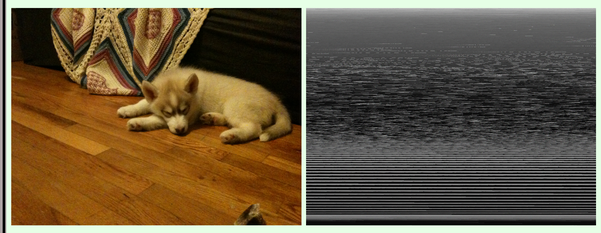
4.1 512x512 pixel gradient



4.2 512x512 pixel checkerboard pattern



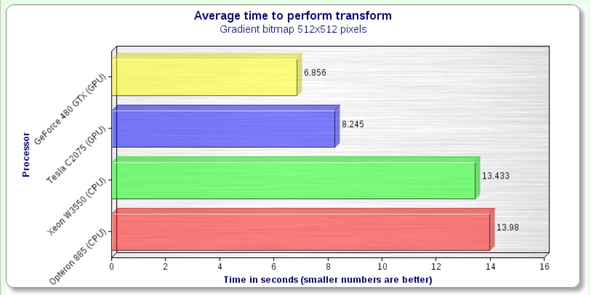
4.3 2048x1536 pixel picture of dog



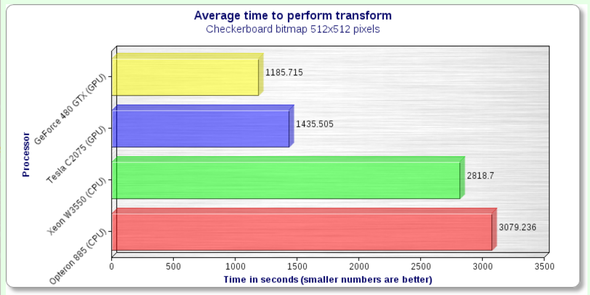
5. Performance results

The performance of the GPU based BWT implementation is summarized in the graphs below. A freely available implementation of the BWT[4] was used to provide a reference to a CPU based implementation. Results were average time to compute transform only for 5 runs for picture of dog and gradient. Checkerboard times averaged over 2 runs.

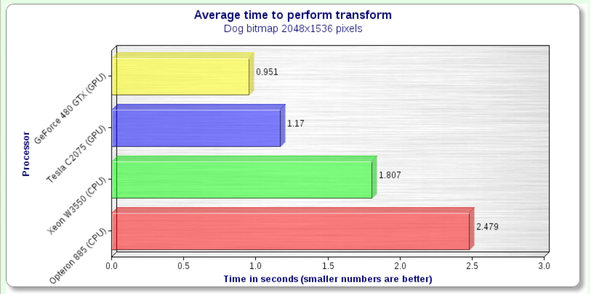
5.1 Gradient performance



5.2 Checkerboard performance



5.3 Dog performance



1: www.top500.org - rankings of publicly known supercomputers.
2: Burrows M and Wheeler D (1994), A block sorting lossless data compression algorithm, Technical Report 124, Digital Equipment Corporation - http://www.hpl.hp.com/techreports/Compaq-DEC/SRC-RR-124.pdf
3: Li H. and Durbin R. (2009) Fast and accurate short read alignment with Burrows-Wheeler Transform. Bioinformatics, 25:1754-60. [PMID: 19451168]
4: http://michael.dipperstein.com/bwt/index.html

I would like to thank NVIDIA for their generosity in donating the Tesla GPU used in this project to CCSU.
I would also like to thank the Office of Undergraduate Research for their generous grant which allowed us to buy additional hardware to help run the GPUs used in the tests.