Fast Burrows Wheeler Compression Using All-Cores

Aditya Deshpande* and P J Narayanan

Centre for Visual Information Technology
International Institute of Information Technology
Hyderabad
ASHES, 2015

*Now at the University of Illinois at Urbana Champaign
Outline

- **What?**
  - Use all-cores (CPU + GPU) for a common end-to-end application
  - Our focus: *Burrows Wheeler Compression (Bzip2)*

- **How?**
  - Use fast GPU String Sort *[Deshpande and Narayanan, HiPC’13]*
  - Domain specific techniques for GPU BW Compression
  - All-core framework to use both CPU and GPU together

- **Why?**
  - Commodity computers have multi-core CPU + many-core GPU
  - All-core end-to-end applications help end user leverage them both
Previous Work

- **Multi-core CPUs** (Coarse/Task Parallelism) –
  - LU, QR, Cholesky Decomposition, Random PDF Generators etc.
  - FFT, PBzip2, String Processing, Bioinformatics, Data Struct. etc.
  - Intel MKL and other libraries

- **Many-core GPUs** (Fine/Data Parallelism) –
  - Scan, Sort, Hashing, SpMV, Lists, Linear Algebra etc.
  - Graph Algorithms: BFS, SSSP, APSP, SCC, MST etc.
  - cuBLAS, cuFFT, NvPP, Magma, cuSparse, CUDPP, Thrust etc.

- The focus is typically not **end-to-end** and/or **all-core** applications
Burrows Wheeler Compression

End-users compress/de-compress files on daily basis. Best compressor BW Compression (or Bzip2), a three step procedure:

1. **Burrows Wheeler Transform**
   Suffix sort and use the last last column (*Most compute intensive*)

2. **Move-to-Front Transform**
   Similar to run-length encoding (~10% of runtime)

3. **Huffman Encoding**
   Standard frequency of chars based encoding (~10% of runtime)

I meant what I said
and I said what I meant
From there to here
from here to there
I said what I meant

Amenable
to RLE

INPUT

After BWT
Burrows Wheeler Transform

- Input String: \( I, S \)

- Sort all cyclic shifts of \( S \)

- Last column of sorted strings, with index of original string is BWT

- \( O(N) \) strings are sorted, each with length \( O(N) \)

- Suffix sort in BWT has long ties \( 10^3 \) to \( 10^5 \) characters

- Need a good GPU String Sort that works on longer ties
Textbooks teach us many popular sorting methods:

- Quicksort
- Mergesort
- Radixsort

Data is always numbers!

Real data is beyond just numbers:
- Dictionary words or sentences
- DNA sequences, multi-dimensional db records
- File Paths

Can we sort strings efficiently?
Irregularity in String Sorting

- **Number Sorting (or Fixed Length Sorting)**
  - Fixed Length Keys (8 to 128 bits)
  - Standard containers: `float, int, double` etc.
  - Keys Fit into registers
  - Comparisons take O(1) time

- **String Sorting (or Variable/Long Length Sorting)**
  - Keys have no restriction on length
  - Iteratively load keys from main memory
  - Comparisons **not** O(1) time
  - Suffix Sort (`1M strings of 1M length!`)
Previous String Sort

CPU

- Multi-key Quicksort
  [Bentley and Sedgewick, SODA’97]
- Burstsort
  [Sinha et al., JEA’07]
- MSD Radix Sort
  [Kärkkäinen and Rantala, SPIRE’08]

GPU

- Thrust Merge Sort
  [Satish et al., IPDPS’09]
- Fixed/Var. Merge Sort
  [Davidson et al., InPar’12]
- Hybrid Merge Sort
  [Banerjee et al., AsHES’13]
- Our String Sort [HiPC’13]
  (Radix Sort)
Merge Sort: Iterative Comparisons

- Repetitive loading for resolving ties in every merge step
- Davidson et al. show that “After every merge step comparisons are between more similar strings”
- Previous GPU String Sorts are based on Merge Sort

Iterative comparisons = High Latency Global Memory Access = Divergence

We develop a Radix Sort based String Sort
Radix Sort for String Sorting

Future Sorts
Seg ID +
\( k \)-char prefix
as Keys

First Sort

MSB Segment ID
(proxy for prefix)

May 29, 2015
AsHES 2015
Results

- **After-Sort Tie Length**: Indicates difficulty of sorting a dataset

  - Suffix sort of BWT has still higher ties and requires many sort steps
  - We develop domain-specific sort techniques for BWT

Code from [cvit.iiit.ac.in](http://cvit.iiit.ac.in)
Modified String Sort for BWT

- Doubling MCU length of String Sort
  - MCU length determines sort steps
  - Large sort steps for long ties and thus, longer runtime 😞
  - Use fixed length MCU initially, then double to reduce sort steps
  - 1.5 to 2.5x speedup
Partial GPU Sort and CPU Merge
- Cyclically shifted strings have special property
- We can sort only 2/3\textsuperscript{rd} strings, synthesize rest w/o iterative sort
- Sort all \( \text{(mod 3)} \neq 0 \) strings iteratively
- 1\textsuperscript{st} char of \( \text{(mod 3)} = 0 \) string, rank of next in 2/3\textsuperscript{rd} sort enough to sort remaining 1/3\textsuperscript{rd} strings
- Non-iterative overlapped merge also possible \textbf{(CPU)}
Datasets GPU BWT

### Datasets
- **Enwik8**: First $10^8$ bytes of English Wikipedia Dump (96MB)
- **Wiki-xml**: Wikipedia xml dump (151MB)
- **Linux-2.6.11.tar**: Publicly available linux kernel (199 MB)
- **Silesia Corpus**: Data-compression benchmark (208MB)

### Tie-Length vs. Block Size

#### Max. Tie-Length Statistics

<table>
<thead>
<tr>
<th>Dataset</th>
<th>1000000</th>
<th>10000</th>
<th>100</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>enwik8</td>
<td>960</td>
<td>960</td>
<td>960</td>
<td></td>
</tr>
<tr>
<td>wiki-xml</td>
<td>996</td>
<td>960</td>
<td>960</td>
<td></td>
</tr>
<tr>
<td>silesia</td>
<td>16320</td>
<td>16320</td>
<td>16320</td>
<td></td>
</tr>
<tr>
<td>linux</td>
<td>65472</td>
<td>65472</td>
<td>262080</td>
<td></td>
</tr>
</tbody>
</table>

#### Avg. Tie-Length Statistics

<table>
<thead>
<tr>
<th>Dataset</th>
<th>900KB</th>
<th>4.5MB</th>
<th>9MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>enwik8</td>
<td>298</td>
<td>576</td>
<td>813</td>
</tr>
<tr>
<td>wiki-xml</td>
<td>614</td>
<td>874</td>
<td>929</td>
</tr>
<tr>
<td>silesia</td>
<td>1406</td>
<td>4075</td>
<td>8430</td>
</tr>
<tr>
<td>linux</td>
<td>2836</td>
<td>10078</td>
<td>27340</td>
</tr>
</tbody>
</table>
GPU BWT vs. Bzip2 BWT

No speedup for small blocks
GPU not utilized sufficiently
Speedup on large blocks
GPU still slow for worst-case linux dataset

Average runtime (secs/per block) for CPU and GPU BWT Algorithms, Block Size : 900KB

Average runtime (secs/per block) for CPU and GPU BWT Algorithms, Block Size : 4.5MB

Average runtime (secs/per block) for CPU and GPU BWT Algorithms, Block Size : 9MB
String Perturbation

- Large sort steps result from repeated substrings/long ties
- Runtime reduces greatly if we break ties
- Perturbation ‘add random chars at fixed interval’ to break ties
- Useful for applications where BWT transformed string is irrelevant, and BWT+IBWT are used in pairs (viz. BW Compression)
- Fixed Perturbation can be removed after IBWT

![Graphs showing time for GPU BWT and CPU BWT vs. % perturbation for linux-2.6.11.tar, 4.5MB and 9MB]

Linux-9MB Blocks, 8.2x speedup with 0.1% perturbation

May 29, 2015 AsHES 2015
All-Core Framework

- System made of CoSt’s:
  - GPU with controlling CPU thread a CoSt
  - Other CPU cores are CoSt’s
- Split blocks across CoSt’s, dequeued from work-queue

Only CPU+GPU CoST
= Hybrid BWC
Hybrid BWC on CPU+GPU CoSt

- Patel et al. did all 3 steps on GPU, 2.78X slowdown
- Map appropriate operation to appropriate compute platform
- GPU for sorts of BWT, CPU does sequential merge, MTF, Huff
- Pipeline blocks such that CPU computation overlaps with GPU
- Throughput BWC = BWT, barring first and last block offset

Map appropriate operation to appropriate compute platform.
GPU for sorts of BWT, CPU does sequential merge, MTF, Huff.
Pipeline blocks such that CPU computation overlaps with GPU.

Throughput BWC = BWT, barring first and last block offset.
Hybrid (CPU+GPU) BWC

INPUT

Work Item Work Item Work Item Work Item Work Item

FIFO WORK QUEUE

Atomic dequeue of Work Items and Parallel Execution

CoSt CoSt CoSt CoSt

Enqueue to Output Queue

Output Output Output Output Output

OUTPUT QUEUE

Seward’s Bzip2 used for blocks on CPU-only CoSt’s
Results: Hybrid BWC

<table>
<thead>
<tr>
<th>Dataset (Size)</th>
<th>Block Size</th>
<th>0% Perturbation</th>
<th>0.01% Perturbation</th>
<th>0.1% Perturbation</th>
<th>1% Perturbation</th>
</tr>
</thead>
<tbody>
<tr>
<td>enwik8 (96MB)</td>
<td>900KB</td>
<td>10.07, 10.81, 27.66</td>
<td>10.03, 10.85, 27.70</td>
<td>9.91, 10.88, 28.09</td>
<td>8.87, 10.97, 31.32</td>
</tr>
<tr>
<td>wiki-xml (151MB)</td>
<td>900KB</td>
<td>36.88, 38.29, 15.29</td>
<td>36.56, 38.16, 15.39</td>
<td>33.85, 37.63, 16.19</td>
<td>23.49, 32.07, 21.89</td>
</tr>
<tr>
<td></td>
<td>4.5MB</td>
<td>30.42, 60.78, 13.66</td>
<td>30.14, 60.76, 13.77</td>
<td>26.97, 60.55, 14.55</td>
<td>15.96, 48.52, 19.82</td>
</tr>
<tr>
<td>linux-2.6.11.tar (199MB)</td>
<td>900KB</td>
<td>84.86, 24.93, 35.35</td>
<td>48.01, 24.69, 35.46</td>
<td>32.84, 23.21, 36.44</td>
<td>22.78, 22.17, 44.19</td>
</tr>
<tr>
<td></td>
<td>4.5MB</td>
<td>133.54, 45.66, 33.10</td>
<td>41.37, 44.02, 33.23</td>
<td>24.17, 39.88, 34.26</td>
<td>14.24, 26.65, 42.31</td>
</tr>
<tr>
<td></td>
<td>9MB</td>
<td>196.64, 53.50, 32.51</td>
<td>45.55, 51.77, 32.65</td>
<td>23.81, 32.11, 33.69</td>
<td>14.37, 29.64, 41.81</td>
</tr>
<tr>
<td>silesia.tar (203MB)</td>
<td>900KB</td>
<td>39.56, 29.65, 52.08</td>
<td>36.14, 29.69, 52.17</td>
<td>28.98, 29.32, 52.97</td>
<td>23.06, 27.46, 59.49</td>
</tr>
<tr>
<td></td>
<td>4.5MB</td>
<td>34.60, 39.57, 50.06</td>
<td>29.52, 39.63, 50.19</td>
<td>22.97, 32.67, 51.08</td>
<td>16.81, 38.07, 57.54</td>
</tr>
<tr>
<td></td>
<td>9MB</td>
<td>36.10, 46.73, 49.57</td>
<td>28.85, 46.92, 49.70</td>
<td>24.55, 46.31, 50.55</td>
<td>17.74, 41.94, 57.11</td>
</tr>
</tbody>
</table>

Compression Ratio improves with increase in Block Size
GPU runtime is better with larger blocks compared to CPU

GPU runtime improves with perturbation, CPU runtime stays the same
Compressed file size increases, but reasonable till 0.1% (< state-of-the-art)

Runtime & compressed file size better than state-of-the-art (Bzip2, 900KB)
Note, CPU does much less work using 900KB blocks, GPU uses 9MB.
Results: Hybrid BWC

<table>
<thead>
<tr>
<th>Dataset (Size)</th>
<th>Block Size</th>
<th>(i) Compression time for our hybrid BWC (s)</th>
<th>(ii) Compression time for CPU BWC (s) [Seward 2000]</th>
<th>(iii) Compressed file size in MB's (same for both)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0% Perturbation</td>
<td>0.01% Perturbation</td>
<td>0.1% Perturbation</td>
</tr>
<tr>
<td>enwik8 (96MB)</td>
<td>900KB</td>
<td>10.07, 10.81, 27.66</td>
<td>10.03, 10.85, 27.70</td>
<td>9.91, 10.88, 28.09</td>
</tr>
<tr>
<td></td>
<td>9MB</td>
<td>8.31, 15.23, 24.86</td>
<td>8.30, 15.22, 24.91</td>
<td>8.33, 15.82, 25.33</td>
</tr>
<tr>
<td>wiki-xml (151MB)</td>
<td>900KB</td>
<td>36.88, 38.29, 15.29</td>
<td>36.56, 38.16, 15.39</td>
<td>33.85, 37.63, 16.19</td>
</tr>
<tr>
<td></td>
<td>4.5MB</td>
<td>30.42, 60.78, 18.66</td>
<td>30.14, 60.76, 13.77</td>
<td>26.97, 60.55, 14.55</td>
</tr>
<tr>
<td>linux-2.6.11.tar (199MB)</td>
<td>900KB</td>
<td>84.86, 24.93, 35.35</td>
<td>48.01, 24.69, 35.46</td>
<td>32.54, 23.21, 36.44</td>
</tr>
<tr>
<td></td>
<td>4.5MB</td>
<td>133.54, 45.66, 33.10</td>
<td>41.37, 44.02, 33.23</td>
<td>24.17, 39.88, 34.26</td>
</tr>
<tr>
<td></td>
<td>9MB</td>
<td>196.64, 53.59, 32.51</td>
<td>45.55, 51.77, 32.65</td>
<td>23.81, 32.11, 33.69</td>
</tr>
<tr>
<td>silesia.tar (203MB)</td>
<td>900KB</td>
<td>39.56, 29.65, 52.06</td>
<td>36.14, 29.69, 52.17</td>
<td>28.98, 29.32, 52.97</td>
</tr>
<tr>
<td></td>
<td>4.5MB</td>
<td>34.60, 39.57, 50.06</td>
<td>29.52, 39.63, 50.19</td>
<td>22.97, 32.67, 51.03</td>
</tr>
<tr>
<td></td>
<td>9MB</td>
<td>36.10, 46.73, 49.57</td>
<td>28.85, 46.92, 49.70</td>
<td>24.55, 46.31, 50.55</td>
</tr>
</tbody>
</table>

Speed Up and Percent Reduction in Compressed File Size

- Hybrid BWC (9MB): Speed Up vs. CPU BWC (900 KB)
- Hybrid BWC (9MB): Speed Up vs. CPU BWC (9MB)
- % Reduction in File Size (with 9MB Blocks) (right y-axis)
Results: All-Core BWC

### Using CPU CoSt’s only: 3.06x speedup

### Using all CoSt’s (CPU and GPU): 4.87x speedup
Conclusions

- Developed techniques for efficient use of all-core (CPU + GPU) systems

- String sort outperforms state-of-the-art significantly, adapts to future GPUs

- Our CPU+GPU hybrid GPU BWC shows a speed up for the first time on BWC using GPUs

- All-Core BWC shows improvement over using only the CPU or GPU cores for BWC

- Our results should encourage other developers to focus on development of fast end-to-end applications
Thank you!

Questions?

All codes are available for download at http://cvit.iiit.ac.in/ or https://web.engr.illinois.edu/~ardeshp2, CVIT/Personal Webpage

Please contact ardeshp2@illinois.edu or pjn@iiit.ac.in for more details

We thank the ‘Indo-Israeli Project’ by Department of Science and Technology for partial financial support for this work