Accelerating a BWT-based exact search on multi-GPU heterogeneous computing platforms

David Alberto Baião da Constantina Jácome Nogueira

Thesis to obtain the Master of Science Degree in Electrical and Computer Engineering

Supervisor: Doctor Nuno Filipe Valentim Roma
Co-supervisor: Doctor Pedro Filipe Zeferino Tomás

Examination Committee
Chairperson: Doctor Nuno Cavaco Gomes Horta
Supervisor: Doctor Nuno Filipe Valentim Roma
Member of the Committee: Doctor Luís Manuel Silveira Russo

November 14, 2014
• Introduction
  • Motivation
  • Problem definition
  • Objectives
• Algorithmic background
• Proposed solution
• Experimental evaluation
• Conclusions and Future Work
Motivation

- Exact string matching is of extremely importance in several domains:
  - Bioinformatics;
  - Pattern recognition;
  - Document matching and text mining;
  - Detecting plagiarism;
  - Intrusion detection systems (IDS);
  - Image and signal processing, etc.
Problem definition

- Exact string matching

  - Input:
    - Reference string (length $m$)
    - Short string, called pattern or query (length $n$, $n << m$)

  - Output:
    - List of all occurrences of pattern in reference string
Objectives

• Development of a pattern matching tool:
  • for parallel offline exact search
  • BWT and the FM-Index
  • Mainly targeting GPU cards
  • Scale throughput with the number of GPUs
  • Heterogeneous platforms
    • e.g., CPUs and GPUs

BowMapCL
Index

- Introduction
- **Algorithmic background**
  - Indexed string matching
  - How to create the index?
  - How to search with the index?
- Proposed solution
- Experimental evaluation
- Conclusions and Future Work
Indexed string matching

• Sequentially searching the string **directly** in the reference text, without any data structure to support it (online search);

• Through an indexed approach: takes as input a previously computed data structure called **index** (offline search):
  - leads to a reduction in the search execution time;

• Index-based approaches:
  - Hash-tables;
  - Suffix trees;
  - Suffix arrays;
  - **Burrows-Wheeler Transform and FM-index** (Chosen approach).
How to create the index?

Algorithmic background: Burrows-Wheeler Transform

Burrows-Wheeler Transform

<table>
<thead>
<tr>
<th>1</th>
<th>mississippi$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>ississippi$m</td>
</tr>
<tr>
<td>3</td>
<td>ssissippi$mi</td>
</tr>
<tr>
<td>4</td>
<td>sissippi$mis</td>
</tr>
<tr>
<td>5</td>
<td>issippi$miss</td>
</tr>
<tr>
<td>6</td>
<td>ssippi$missi</td>
</tr>
<tr>
<td>7</td>
<td>sippi$missis</td>
</tr>
<tr>
<td>8</td>
<td>ippi$mississ</td>
</tr>
<tr>
<td>9</td>
<td>ppi$mississi</td>
</tr>
<tr>
<td>10</td>
<td>pi$mississip</td>
</tr>
<tr>
<td>11</td>
<td>i$mississipp</td>
</tr>
<tr>
<td>12</td>
<td>$mississippi</td>
</tr>
</tbody>
</table>

a) Original rotation matrix.

| 12 | $mississippi |
|11 | i$mississipp |
| 8 | ippi$mississ |
| 5 | issippi$miss |
| 2 | ississippi$ |
| 1 | mississippi$ |
|10 | pi$mississip |
| 9 | ppi$mississ |
| 7 | sippi$missi |
| 4 | sissippi$mi |
| 6 | ssippi$miss |
| 3 | ssissippi$m |

b) Sorted cyclic shifts.

mississippi$ → BWT(T) → ipssm$pissii
How to create the index?

**Algorithmic background: FM-index**

Two additional data structures are created using the BWT output:

$$\text{BWT}(T') = \text{BWT}(\text{mississippi}$) = ipssm$pissii$$

**C vector**

$C(x)$ presents number of lexicographically smaller characters than character $x$ in text

<table>
<thead>
<tr>
<th></th>
<th>i</th>
<th>p</th>
<th>s</th>
<th>s</th>
<th>m</th>
<th>$</th>
<th>p</th>
<th>i</th>
<th>s</th>
<th>s</th>
<th>i</th>
<th>i</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**OCC matrix**

$\text{Occ}(x, y)$ presents the number of occurrences of character $x$ in the prefix $\text{BWT}(T')[1…y]$

<table>
<thead>
<tr>
<th></th>
<th>i</th>
<th>p</th>
<th>s</th>
<th>s</th>
<th>m</th>
<th>$</th>
<th>p</th>
<th>i</th>
<th>s</th>
<th>s</th>
<th>i</th>
<th>i</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>i</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>m</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>p</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>s</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>
How to search with the index?

Algorithmic background: Backward search algorithm

```
procedure BACKWARD SEARCH(queries[ ][ ], C[ ], OCC[ ][ ])
    for every query j do
        i := size of query j, i.e., i := queries[j]
        c := last character of query j, i.e., c := queries[j][i]
        FIRST := C[c]+1
        LAST := C[c+1]
        while (FIRST ≤ LAST) AND (i ≥ 2) do
            c := get previous last character of query j, i.e., c := queries[j][i-1]
            FIRST := C[c] + OCC[c][FIRST-1] + 1
            LAST := C[c] + OCC[c][LAST]
            i := i - 1
        end while
        if LAST < FIRST then
            return not found
        else
            return values of indexes between FIRST and LAST
        end if
    end for
end procedure
```
How to search with the index?

Algorithmic background: Backward search algorithm

Query string: “ssi”

FIRST=1
LAST=12
How to search with the index?

Algorithmic background: Backward search algorithm

Query string: “ssi”

FIRST=1
LAST=12

FIRST=2
LAST=5
How to search with the index?

**Algorithmic background:** Backward search algorithm

**Query string:** “ssi”

<table>
<thead>
<tr>
<th>FIRST</th>
<th>LAST</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
</tr>
</tbody>
</table>
How to search with the index?

Algorithmic background: Backward search algorithm

Query string: “ssi”

FIRST=1
LAST=12

FIRST=2
LAST=5

FIRST=9
LAST=10

FIRST=11
LAST=12

Accelerating a BWT-based exact search on multi-GPU heterogeneous computing platforms
How to search with the index?

**Algorithmic background:** Output conversion with SA

| 1  | mississippi  | 1  | mississippi  |
| 2  | i$mississipp | 2  | i$mississipp |
| 3  | ippi$mississ | 3  | ippi$mississ |
| 4  | issippi$miss | 4  | issippi$miss |
| 5  | issississippi | 5  | issississippi |
| 6  | mississippi$ | 6  | mississippi$ |
| 7  | pi$mississip | 7  | pi$mississip |
| 8  | ppi$mississi | 8  | ppi$mississi |
| 9  | sippi$missis | 9  | sippi$missis |
| 10 | sississippi$ | 10 | sississippi$ |
| 11 | issippi$missi | 11 | issippi$missi |
| 12 | ssississippi$ | 12 | ssississippi$ |

SA

1 12
2 11
3 8
4 5
5 2
6 1
7 10
8 9
9 7
10 4
11 6
12 3

Backward search output conversion with suffix array

Obtain the actual positions of the query ssi in the original text
• Introduction
• Algorithmic background
• **Proposed solution**
  • Main challenges
  • General purpose hardware architectures comparison
  • Proposed parallelization approach
  • Multiple buffering and events dependencies graph
  • Index partitioning
  • Other enhancements
• Experimental evaluation
• Conclusions and Future Work
Main challenges

Main challenges to overcome:

• Unpredictable/irregular memory access pattern
• How to explore existent parallelism – architectures comparison
  • Cope with reduced GPU memory space
  • Scale the throughput with number of GPUs
  • Overlap computation and data transfers in GPUs
• Reduce I/O operations overhead
• Accept any data type alphabet
• Accept any input file size
General purpose hardware architectures comparison

CPU

Intel Haswell architecture

- Faster clock frequency
- Low-latency access to memory (due to multiple levels of cache)
- Complex control for out-of-order and speculative execution

GPU

NVIDIA Kepler 110 architecture

- Data parallel instructions
- High-throughput computation
Proposed parallelization approach
Asynchronous multi-threaded host execution
Proposed parallelization approach
Asynchronous multi-threaded host execution

Usage of:
• a producer-consumer scheme with multiple threads dedicated to each GPU device and to the involved I/O operations
• a multiple buffering technique

Allowed to:
• overlap the I/O operations from disk to main memory with the string matching procedure
• overlap the OpenCL data transfers between the host device and the target devices with the kernel execution on those same devices.
Multiple buffering and events dependencies graph

a) Multiple buffering scheme is not used.

b) Multiple (double) buffering scheme is used.

- **RQ**: Read subset of queries from input file
- **WQ**: Copy subset of queries to device memory
- **MK**: Execute the exact string matching kernel on the device
- **RO**: Copy kernel output from device memory
- **WS**: Convert with SA and write solutions to output file
What if the index size happens to be larger than the device memory? *Index partitioning*

- No longer presents any relevant restrictions:
  - on the host memory size (RAM),
  - nor on the global memory of the GPUs;
- It is independent of the size of the considered reference input.
Other enhancements

• To the kernel:
  • Local memory usage
  • Coalesced memory accesses

• To the index data structures:
  • OCC matrix bitmap encoding and sampling
  • SA sampling and on-the-fly computation
• Introduction
• Algorithmic background
• Proposed solution
• Experimental evaluation
  • OCC matrix bit encoding and index sampling
  • CPU-based tools comparison
  • CUDA GPU-based tools comparison
  • Index and query set size scalability
  • GPU scalability and multiple buffering study
  • Load balancing
• Conclusions and Future Work
OCC matrix bit encoding and index sampling

Memory footprint reduction and runtime performance variation

- FM-index space footprint
- Execution time increase

Sampling rate

Increase in program execution time (against version with no sampling)

FM-index space footprint (bits per input bit)
CPU-based tools comparison

![Bar chart comparing execution time of various CPU-based tools across different numbers of queries. The tools compared are bowtie-1.1.1 (8 threads), bwa-0.7.10 (8 threads), soap2.21release (8 threads), and BowMapCL (1 GPU).]

![Bar chart comparing speedup of BowMapCL against CPU-based tools across different numbers of queries. The tools compared are bowtie-1.1.1 (8 threads), bwa-0.7.10 (8 threads), and soap2.21release (8 threads).]
CUDA GPU-based tools comparison (1): CUSHAW

Accelerating a BWT-based exact search on multi-GPU heterogeneous computing platforms

David Nogueira

22/11/2014
CUDA GPU-based tools comparison (2): HPG-BWT

Execution time (in seconds)

Speedup of BowMapCL against HPG-BWT

Number of queries in alignment procedure

HPG-BWT (2GPUs)  BowMapCL (1 GPU)

HPG-BWT (2GPUs)

Accelerating a BWT-based exact search on multi-GPU heterogeneous computing platforms
CUDA GPU-based tools comparison (3): SOAP3

- Execution time (in seconds)
- Number of queries in alignment procedure
- Speedup of BowMapCL against SOAP3

- soap3-r146-x64-cuda3.2 (1 GPU)
- BowMapCL (1 GPU)

Accelerating a BWT-based exact search on multi-GPU heterogeneous computing platforms

22/11/2014 David Nogueira
## Index and query set size scalability

### Variable index size, fixed query set size

<table>
<thead>
<tr>
<th>(100M queries)</th>
<th>E. Coli genome</th>
<th>Human Chromosome 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution time (s)</td>
<td>26.03 s</td>
<td>28.95 s</td>
</tr>
</tbody>
</table>

### Fixed index size, variable query set size

<table>
<thead>
<tr>
<th>(Human Chromosome 1)</th>
<th>10 M queries</th>
<th>100 M queries</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution time (s)</td>
<td>2.95 s</td>
<td>28.95 s</td>
</tr>
</tbody>
</table>
Index and query set size scalability

Variable index size, fixed query set size

<table>
<thead>
<tr>
<th>(100M queries)</th>
<th>E. Coli genome</th>
<th>Human Chromosome 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution time (s)</td>
<td>26,03 s</td>
<td>28,95 s</td>
</tr>
</tbody>
</table>

Almost same performance, with different index size and same query set size.

Fixed index size, variable query set size

<table>
<thead>
<tr>
<th>(Human Chromosome 1)</th>
<th>10 M queries</th>
<th>100 M queries</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution time (s)</td>
<td>2,95 s</td>
<td>28,95 s</td>
</tr>
</tbody>
</table>
Index and query set size scalability

### Variable index size, fixed query set size

<table>
<thead>
<tr>
<th>(100M queries)</th>
<th>E. Coli genome</th>
<th>Human Chromosome 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution time (s)</td>
<td>26.03 s</td>
<td>28.95 s</td>
</tr>
</tbody>
</table>

### Fixed index size, variable query set size

<table>
<thead>
<tr>
<th>(Human Chromosome 1)</th>
<th>10 M queries</th>
<th>100 M queries</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution time (s)</td>
<td>2.95 s</td>
<td>28.95 s</td>
</tr>
</tbody>
</table>

It shows scalability regarding the query set size (fixed index).
It shows that:

• by using more than one set of buffers it is possible to overlap multiple concurrent operations in the same device:
  • resulting in a speed-up of around 2× when using two buffers;
• by using a higher number of buffers it is possible to further exploit the GPU spatial resources.
Load balancing on heterogeneous platform

- **Consumer thread execution time (ms)**
  - **NVIDIA**: 2099.35 ms
  - **AMD**: 2113.69 ms

- **Number of queries**
  - **NVIDIA**: 4600000
  - **AMD**: 5400000

---

- **GPU 1 (NVIDIA GeForce GTX 560 Ti)**
- **GPU 2 (AMD Radeon R9 290X)**
Load balancing

Load balancing under GPU contention

Consumer thread execution time (ms)

4194.09  4173.74

GPU 1 - overloaded shared GPU
(NVIDIA GeForce GTX 680)

GPU 2 - GPU with no load
(NVIDIA GeForce GTX 680)

Number of queries

4100000  5900000

NVIDIA
Index

- Introduction
- Algorithmic background
- Proposed solution
- Experimental evaluation
- Conclusions and Future Work
  - Conclusions
  - Future work
  - Contributions
Conclusions

• Major contributions of the proposed solution (BowMapCL):
  
  • Performance speed-ups between 10x and 15x when compared with the state-of-art CPU tools and between 1.5x and 5x with state-of-art GPU tools;
  
  • **Linear scaling** of the offered throughput with the **number of GPU** devices;
  
  • **Efficient load balanced** execution among the several devices in the heterogeneous platforms
  
  • **Agnostic** in what concerns the accepted data input (i.e., DNA, proteins, text)
  
  • Supports any number of **queries** with **no limitation** in the input **reference** size
Future Work

- Extend tool to non-exact search or integrate in pipeline of sequence alignment tool;

- Expand the tool to support text in other encodings (besides ASCII);

- Use MPI to run the application in multiple nodes;

- Allow automatic optimization of execution in other OpenCL devices (besides GPUs).
Contributions


Accelerating a BWT-based exact search on multi-GPU heterogeneous computing platforms

David Alberto Baião da Constantina Jácome Nogueira

Thesis to obtain the Master of Science Degree in Electrical and Computer Engineering

Supervisor: Doctor Nuno Filipe Valentim Roma
Co-supervisor: Doctor Pedro Filipe Zeferino Tomás

Examination Committee
Chairperson: Doctor Nuno Cavaco Gomes Horta
Supervisor: Doctor Nuno Filipe Valentim Roma
Member of the Committee: Doctor Luís Manuel Silveira Russo

November 14, 2014