Preliminary work on radix-based multi-select on GPUs

I. Komarov, R. D’Souza

University of Wisconsin-Milwaukee

dsouza@uwm.edu

March 26, 2014
Overview

1. The $k$-NN problem and applications
2. Methods for $k$-NN search
3. Radix Select
4. Radix Multi-Select
5. Results
6. Future Work
7. References
Given a set of data points $S$, and a set of query points $Q$, for each point $q \in Q$ find a subset $K \subset S$, where $\text{size}(K) = k$, such that

$$d(s_k, q) < d(s_j, q) \forall s_k \in K, s_j \in S - K$$

where $d(a, b)$ is the distance metric with the following properties

$$d(a, b) = d(b, a) \geq 0$$

$$d(a, b) = 0 \Rightarrow a = b$$
Applications

- Bioinformatics [RMW⁺07, WEZ⁺04]
- Pattern recognition [DHS01]
- Cluster analysis [MHvL09, TSL00]
- Data mining [ZH00]
Methods for $k$-NN search

- Space partitioning (exact)
  - k-D Trees, Ball Trees [JOR11, AMN$^+$94]
- Local Sensitive Hashing (approximate) [DIIM04]
- Brute force
Brute-force methods

- Computation of distance matrix
  - Formulated as a matrix multiplication operation (use CUBLAS)
- Selection of $k$-NN
  - Full sort and selection
  - Insertion sort-based selection
  - Heap sort-based selection
  - Quick select
  - Truncated bitonic sort
  - Radix select
Differences pertain only to method of selection

- Heap sort-based selection [GDNB10, BGT+11, KH10]
- Insertion sort-based selection [ARBM12]
- Truncated bitonic sort [SPS12]
Radix Select

- Partition based on MSB
- Pruning

\[ MSB_1 = 1 \quad MSB_1 = 0 \]
\[ MSB_2 = 1 \quad MSB_2 = 0 \]
\[ MSB_3 = 1 \quad MSB_3 = 0 \]
\[ l_1 > k \]
\[ l_2 < k \]
\[ l_3 = k - l_2 \]
Naïve approach (used by Cederman for quick sort [CT10])

Fig. 3: Partitioning a sequence
Issues

- Two passes $\Rightarrow$ two global memory reads (even though both are coalesced)
- Read is coalesced, write is un-coalesced
Process multiple queries at the same time
- One row of the distance matrix per thread block/thread warp

BIG IDEA?
- Incremental partitioning
- Using a combination of shared memory, __ballot(), bit shifting, and __popc()
Intrinsic functions

- `unsigned int __ballot(predicate)`
  - when predicate evaluates to TRUE the function returns a value with the Nth bit set

- `int __popc(unsigned int x)`
  - returns the number of bits in x set to 1
Partition process: Read incrementally

\[ B = \text{ballot}(x_i \land (1 \ll j)) \]

- **B**
- **Ballot function predicate**
- **Register**
- **Global memory**

- \( t_{id} \): 0 1 2 3
- \( 0 \text{ to } 30 \)
Partition process: Partition in shared memory

\begin{align*}
\text{if } (\neg (x_i \land (1 \ll j))) \text{// elements with MSB=0/} & \\
b &= \neg B \gg (31 - i) \\
sh\_mem[\_\text{popc}(b)] &= x_i \\
\text{else} & \\
b &= B \gg (31 - i) \\
sh\_mem[31 - \_\text{popc}(b)] &= x_i \\
\text{end}
\end{align*}
Partition process: Write incrementally to auxiliary array

\[ t_{id} \leftarrow \text{popc}(B) \]

\[ g < \text{piv} \]

\[ g \geq \text{piv} \]
Each thread warp in the TB works as shown previously. The start point during the write process for the left and right hand side for each warp has be to offset to account for previous warps. This is accomplished using a summing operation within the TB.
These tests were performed for elements that contained both values and indices. $-3 \leq \log_2(n/Q) \leq 20$. $n \times Q = 2^{27}$. $2 \leq k \leq 512$. Performance peaks at $\log_2(n/Q) = 7$ stays flat till $\log_2(n/Q) = 11$ and starts falling at $\log_2(n/Q) = 13$. The decrease in performance corresponds to the number of queries $Q$ falling below 128 where our method underutilizes GPU resources.
Future work

- Processing multiple MSB in one partition
  - Will require binning - juggling multiple intrinsic calls
  - Will reduce global memory reads?
- See what we can do with Maxwell architecture for decision making process
Acknowledgements

This work was partially funded by the National Science Foundation (NSF) through grant CCF-1013278. Views expressed are those of the authors alone and not necessarily those of the NSF.


