Cost-Efficient Cluster Design for High Dimensional Similarity Search

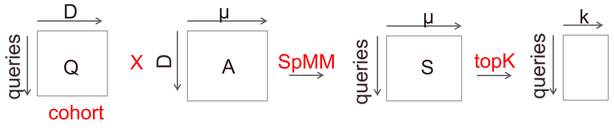
Sandeep R Agrawal, Christopher M Dee, Alvin R Lebeck
Department of Computer Science, Duke University

1. Similarity Search

- Given
  - D dimensional space
  - N sparse points we wish to search across
  - Query stream with an arrival rate of α
  - A distance metric in this space
- Problem
  - Find the k (k << N) most similar matches (nearest neighbors) for each of the query points
- Datacenter constraint
  - Solve the problem within a deadline T seconds achieving maximum throughput and efficiency

2. Algorithms for Search

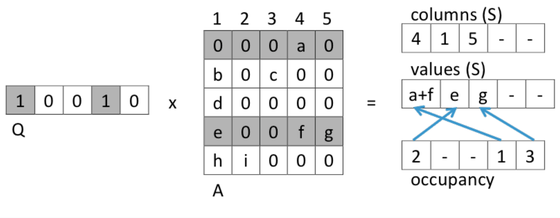
- Assume cosine similarity as a similarity metric
- Exhaustive search => Computing similarities between each point and a query cohort becomes a Sparse Matrix Matrix Multiplication (SpMM) operation
- Partition *shard* into tiles of size μ each
- Assume tiles are stored in CSR format.
- Compute the Top-k similarities for each query point



3. SpMM – Our Algorithm

- Columns(S) does not need to be sorted as we are only interested in Top-k results
- Store Occupancy array per query (essentially a perfect hash function generated at runtime)
- For nonzero A[m][n] at index p in sparse array,

```
if occupancy[n] = 0
   values(S)[p] = A[m][n]
   columns(S)[p] = n
   occupancy[n] = p
else
   values(S)[occupancy[n]] += A[m][n]
```



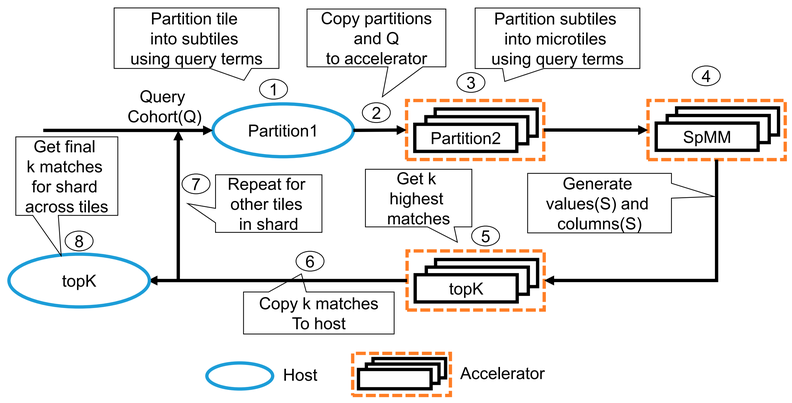
4. SpMM - Host

- Tiles – Better cache locality
- Partition Query cohort uniformly across threads/cores
- Cores work on different queries (no atomics) but share tile
- More threads => More ILP, but More inflight query state => cache contention
- For single queries (cohort size = 1), use one tile per thread

5. SpMM – Accelerator

- Partition tile into microtiles at runtime to increase parallelism per query
- Two step partition function due to PCI-E constraints
  - Partition2 creates parallelism for SpMM
  - Partition1 creates parallelism for Partition2
- Tiles reside in accelerator memory, avoiding PCI-E transfers
- Use scratchpad memory to store occupancy array

7. Similarity Search Pipeline (Accelerator)

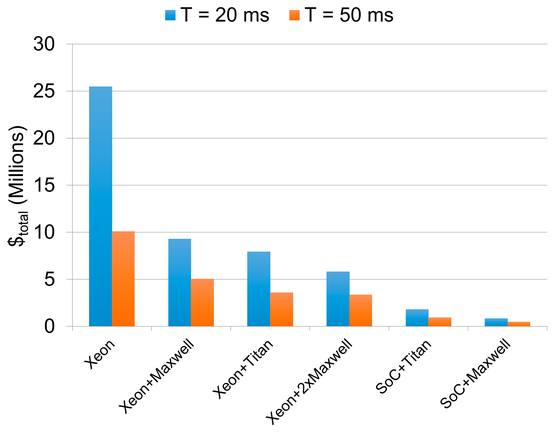


8. Experimental Setup

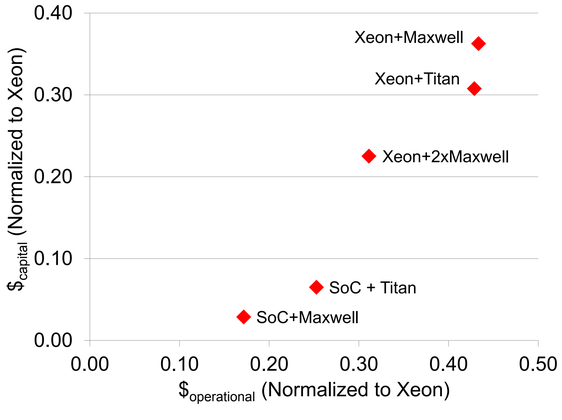
Platform	$\$_{fixed}$	$\$_{cpu}$	$\$_{acc}$	Description
Xeon	4500	2300	-	PowerEdge R720, 2 x Xeon E5-2650v2, 22 nm, 16C/32T, 8x8GB 1866MHz RDIMMs
Xeon + Titan	4500	2300	1000	PowerEdge R720, 2 x Xeon E5-2650v2, GTX Titan, 28 nm, 2688 CUDA cores, 14 SMs, 6GB GDDR5 Memory
Xeon + Maxwell	4500	2300	150	PowerEdge R720, 2 x Xeon E5-2650v2, GTX 750Ti, 28 nm, 640 CUDA cores, 5 SMs, 2GB GDDR5 Memory
SoC + Titan	200	-	1000	J1800 SoC, 22 nm, 4GB DDR3L RAM, GTX Titan
SoC + Maxwell	200	-	-	J1800 SoC, 22 nm, 4GB DDR3L RAM, GTX 750Ti

- Search across the English Wikipedia (Bag of words model)
- Queries are randomly picked from page titles
- 95% latency evaluated using Monte Carlo simulations
- Create models for $\$_{capital}$ and $\$_{operational}$ based on number of machines and energy consumed for searching across 1 Billion documents
- Pick μ and C giving lowest $\$_{total} = \$_{capital} + \$_{operational}$

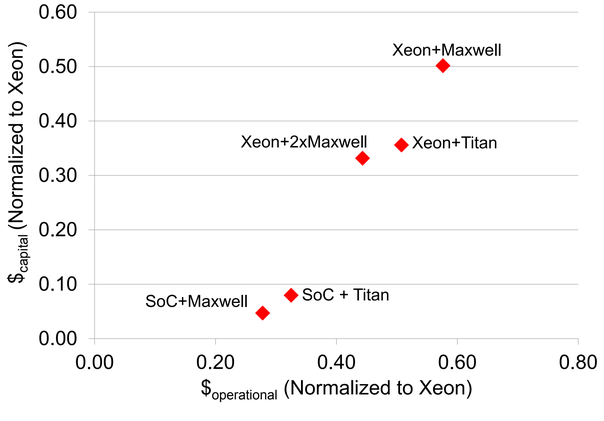
9. Evaluation



$\$_{total}$ for T = 20 ms and T = 50 ms and α = 5,000 queries/sec



Breakdown of $\$_{total}$ for T = 20 ms and α = 5,000 queries/sec



Breakdown of $\$_{total}$ for T = 50 ms and α = 5,000 queries/sec

- Augmenting existing Xeon based servers with accelerators results in a 2.7x to 4.4x cost reduction for T = 20 ms.
- Replacing a Xeon based cluster with an accelerator based cluster reduces TCO by more than 30x while consuming 4x less total power.

Funding by National Science Foundation (CCF-1335443), NVIDIA and Duke University