Low latency DSSM hardware accelerater

Sixiao Zhu, Ningyi Xu, MSRA

5/20/2016

DSSM: introduction

- DSSM stands for Deep Structured Semantic Model, or more general, Deep Semantic Similarity Model. DSSM, developed by the MSR Deep Learning Technology Center, is a deep neural network (DNN) modeling technique for representing text strings (sentences, queries, predicates, entity mentions, etc.) in a continuous semantic space and modeling semantic similarity between two text strings (e.g., Sent2Vec).
- DSSM has wide applications including information retrieval and web search ranking, ad selection/relevance, contextual entity search and interestingness tasks, question answering, knowledge inference, image captioning, and machine translation, etc.

Necessarity for FPGA acceleration

- Bing performs DSSM model inference on **real time**.
- When a query enters, Bing uses DSSM to encode the query into a 300 dimension vector, and then calculate cosine similarity with offline calculated encoding of saved web docs. This similarity is used as a ranking fector for search results.
- Latency is crucial in this process.

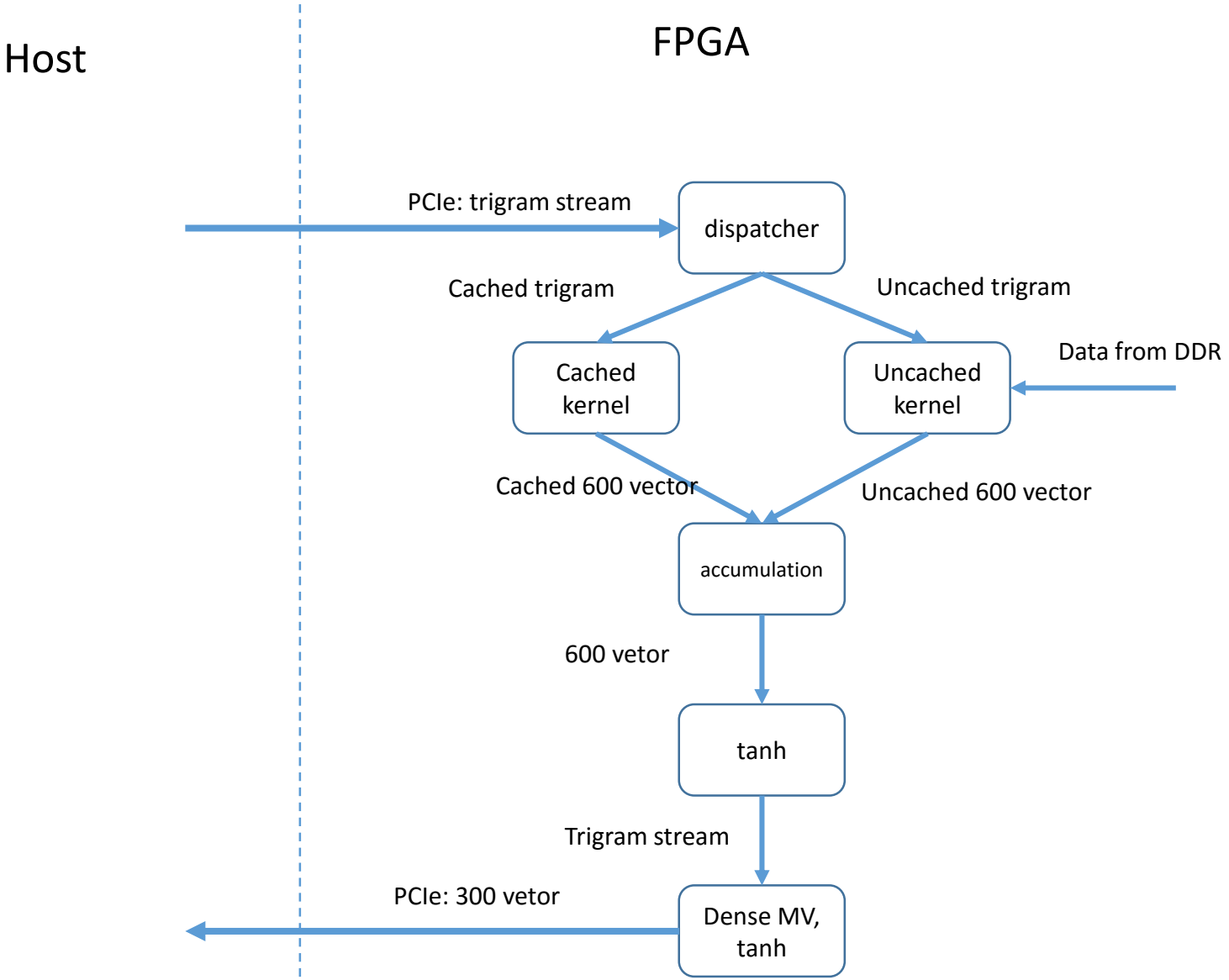
DSSM: inference mechanism

- Consider “apple”, in this model, it is spilted into a series of trigrams, “#ap”, “app”, “ppl”, “ple”, “le#”.
- The list of trigrams of a certain sentence could be regarded as a 49292 dimension sparse vector.
- The sentence “aaaa fa” is represented as [(“#aa”, 1), (“aaa”, 2), (“aa#”, 1), (“#fa”, 1), (“fa#”, 1)]

DSSM: inference mechanism

- $W_d \in M_{600\times49292}(\mathbb{R})$
- $W_d \in M_{300\times600}(\mathbb{R})$
- $v_{input} \in \mathbb{R}^{49292}$ (represented as sparse vector)
- $v_{result} \in \mathbb{R}^{300}$

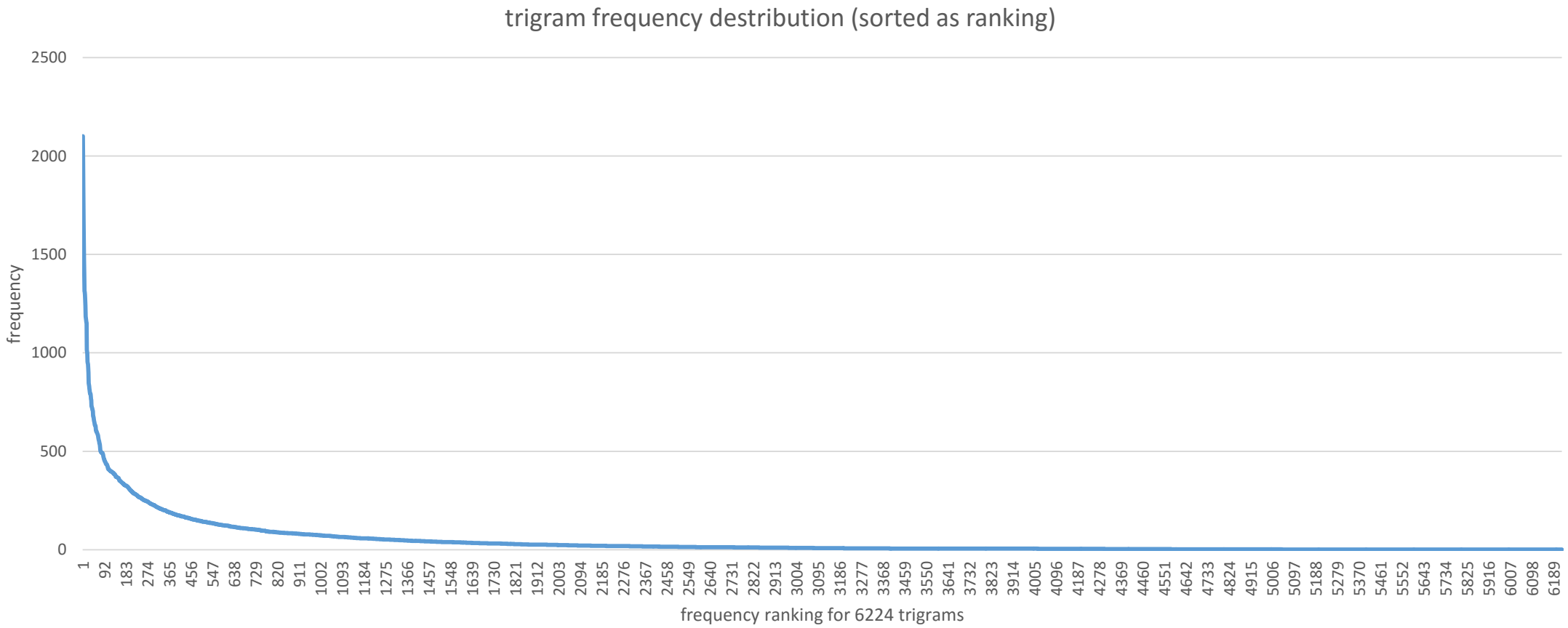
- DSSM inference process could be simplely summarized as:
- $v_{result} = \tanh(W_d \times \tanh(W_s \times v_{input}))$


Host
FPGA
PCIe: trigram stream
dispatcher
Cached trigram
Uncached trigram
Cached kernel
Uncached kernel
Data from DDR
Cached 600 vector
Uncached 600 vector
accumulation
600 vetor
tanh
Trigram stream
PCIe: 300 vetor
Dense MV, tanh

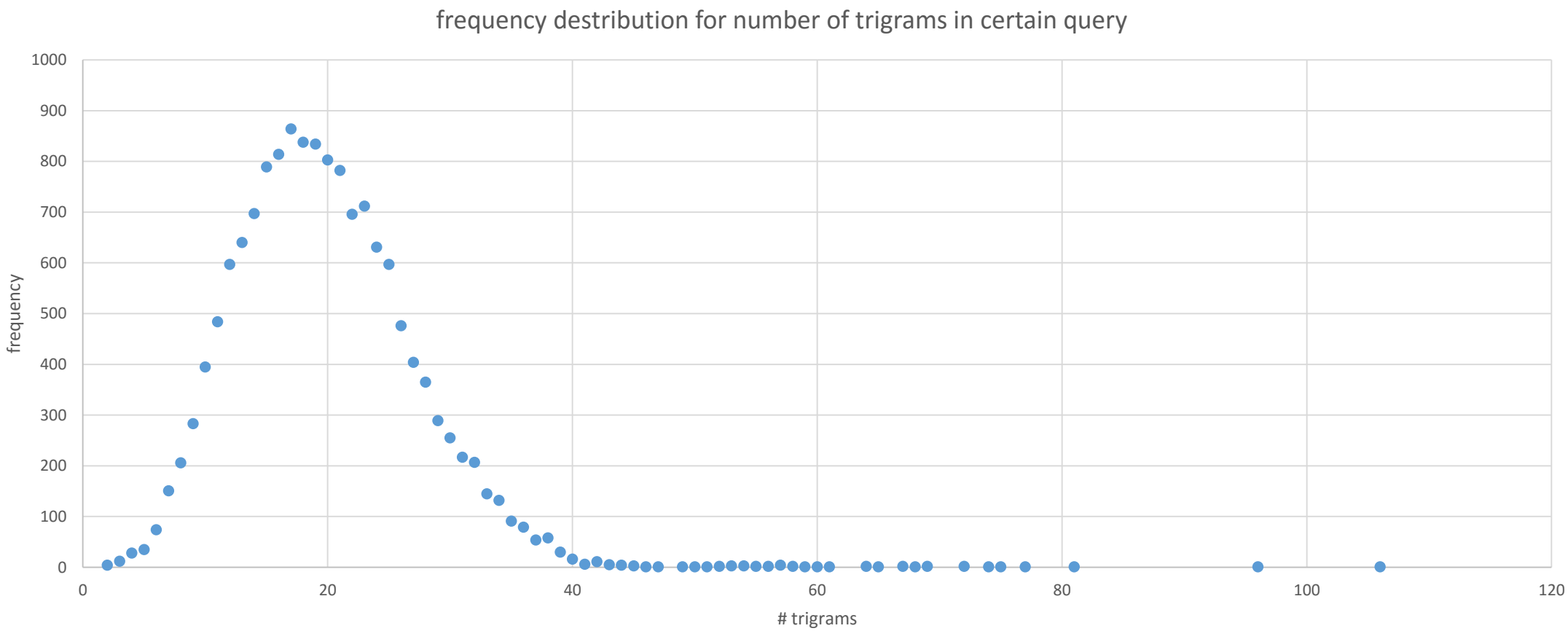
Cache: analysis

- On the sparse MV stage, FPGA could easily perform a high parallel vector element-wise addition.
- 10GB/s DDR delivers about 16 float point per cycle under 200MHz , ideally.
- Noticed that by the nature of language, some trigram appears frequently, like “con”, “er#”, and “#th”. (“#”means beginning or ending)
- We could cache frequently hit v600 on BRAM, and use FIFO to decompose the cached fetch and uncached fetch process.

Trigram frequency anslysis: caching the most frequent trigrams' vector while significantly reduce fetch latency



Query analysis: Query contains mostly 10-30 trigrams

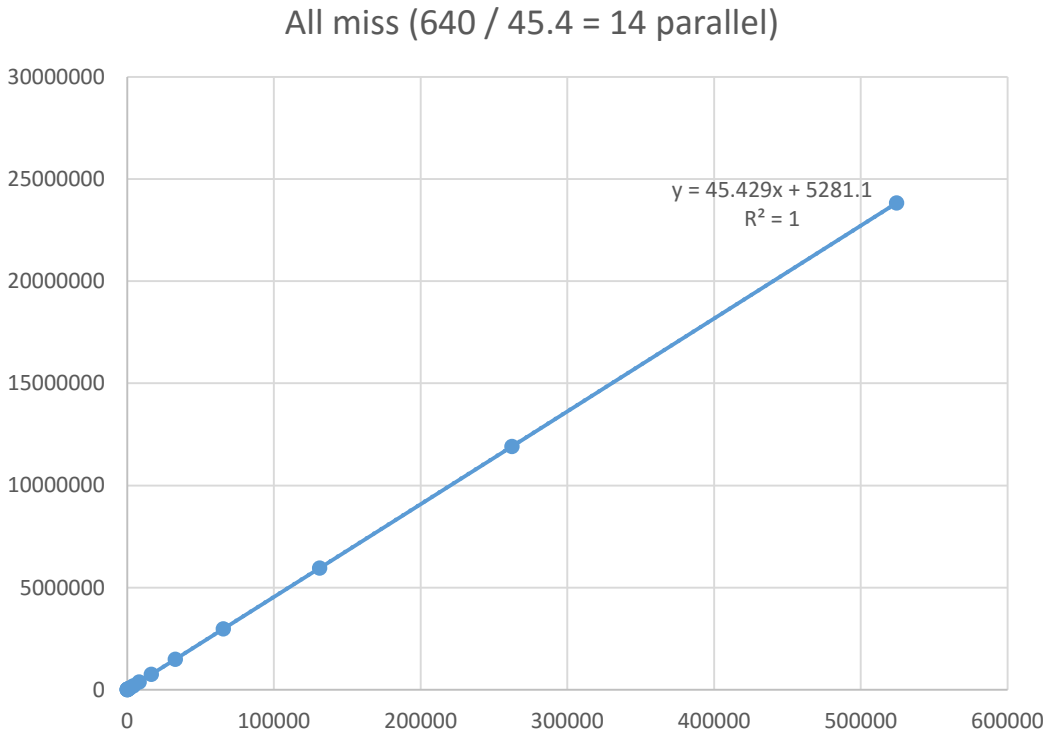


Cache performance analysis

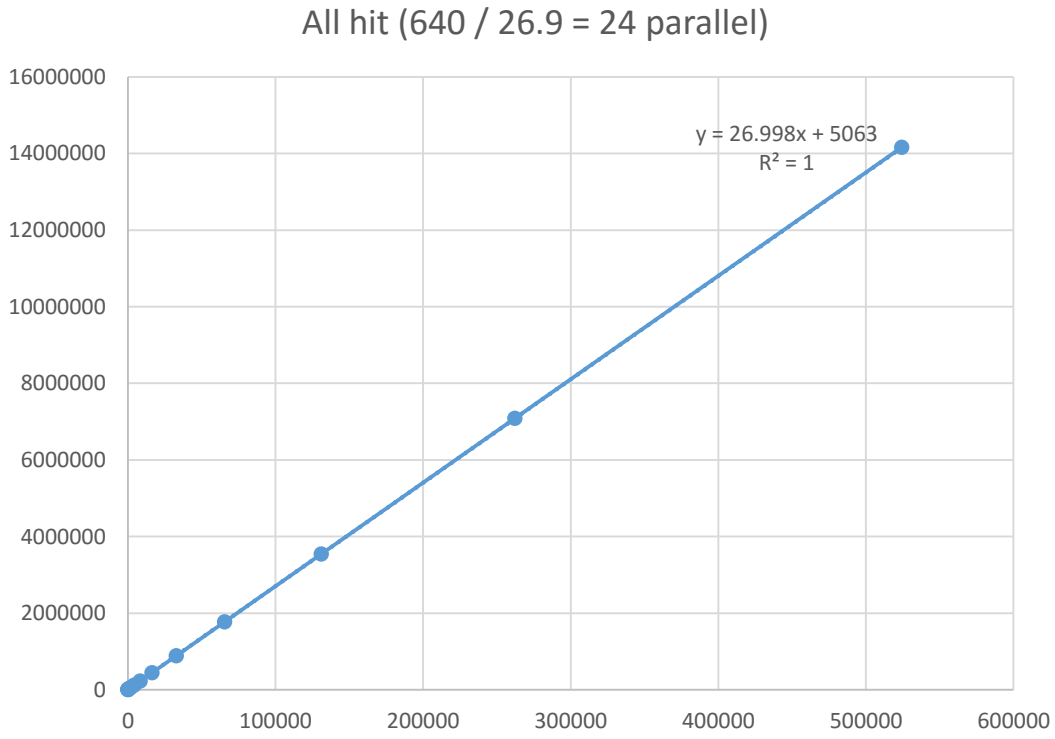
- Recall the previous FPGA acceletator structure.
- Acculumation kernel (for sparse MV) consumes fetched vector with constant rate v_a
- DDR fetching rate v_b
- Supose the cache hit rate is λ
- By average comsumption rate:
- $v_a \times \min(\frac{v_b}{\lambda v_a}, 1)$

Cache evaluation: calculate performance sparse performance from slope on different number of trigrams.

All miss case, actually 14 parallel / potentially 32 parallel



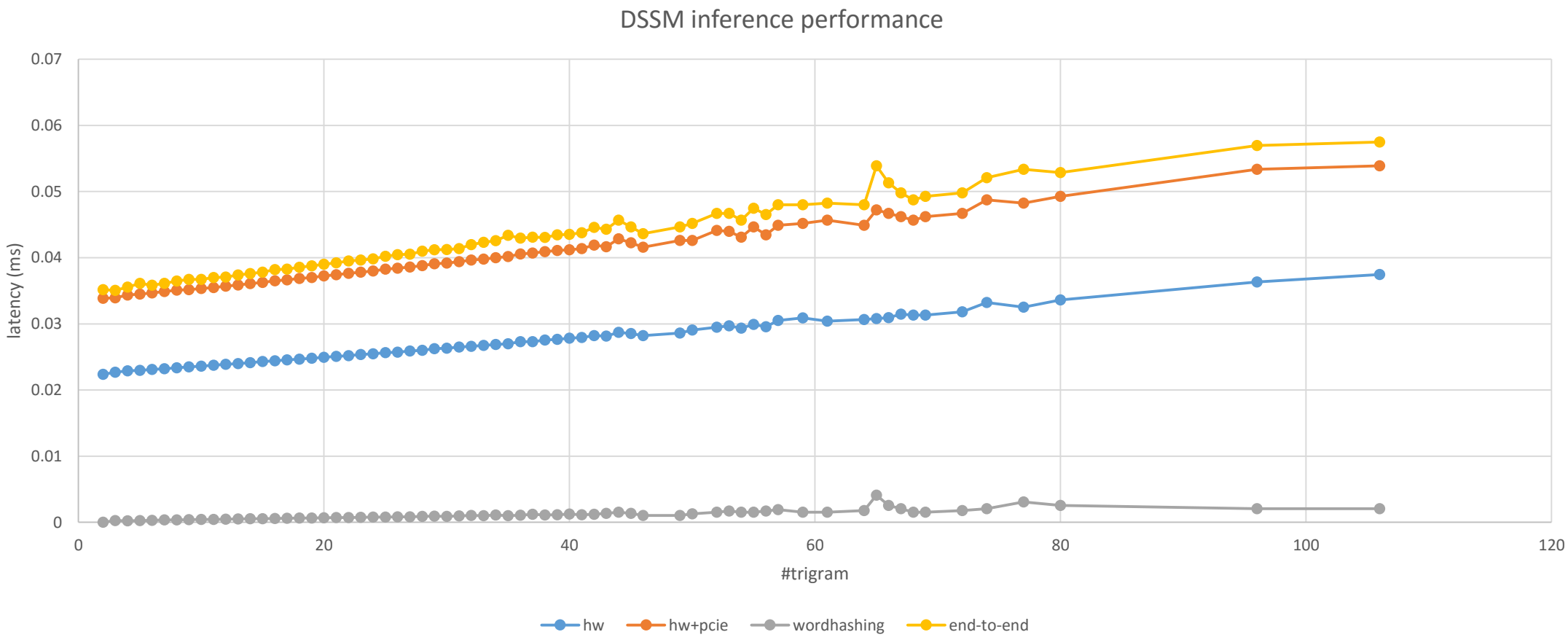
All hit case, actually 24 parallel / potentially 32 parallel



Performance

- Accroding to production team’s result, the software version latency of a single query is by average 4ms.
- Our current implementation performs on average 0.04ms, about 100 times acceleration. Correctness verified.

End-to-end profiling for prototype implementation



Thanks