Optimization for Data-Aware Prefix Caching

Motivation

The adoption of Large Language Model technologies faces the challenge of slow inference which is related to the inherent nature of generative transformer-based architectures. To mitigate this issue, LLMs are typically deployed with a key-value cache which is used for storing frequently used computations.

For example, when users repeatedly query the same long document (e.g., a manual or report), the document can be processed only once storing computation results in the KV cache. Then all future requests can avoid recomputing the long document by reusing the cache. Similarly, when many user prompts share the same prefix, or input prompt, the KV cache can store one copy of the prefix in advance to reduce redundant computation. If two requests share a prefix, the first request is processed loading results into the KV cache and the subsequent request can directly reuse these values for further inference without having to recompute them [1].

The caching technique allows LLM to serve queries with a higher throughput and is particularly useful when some information about queries to be massively issued to LLM is known in advance. For example, this is the case for scenarios when LLM inference is triggered from some app implementing a fixed busines logic (implying a fixed shape of queries and prompts) [2] or over a collection of structured textual data with repeated parts [3]. In this challenge, we focus on the latter scenario.

Challenge Content (informal)

Given a finite collection of (abstract) sentences, each containing n words, we aim at finding a partitioning of this collection into batches and for each batch, finding a permutation of word positions, which gives the longest prefix (concatenation of first several words) shared by each sentence (under the permutation) in the batch. The goal is to mutually optimize the size of batches and the length of prefixes shared by sentences inside batches under a permutation.

Example

To provide some intuition, consider an example of two sentences, each consisting of 3 words:

$$\begin{matrix} d & bc & a \\ e & c & ab \end{matrix}$$

Then there is a permutation transforming both sentences into

$$\begin{matrix} a & bc & d \\ ab & c & e \end{matrix}$$

having shared prefix abc.

Now consider the collection of sentences:

$$\begin{matrix} d & bc & a \\ e & c & ab \\ xy & a & az \\ y & ax & aw \end{matrix}$$

They all have shared prefix a under a permutation, but on the other hand, this collection can be partitioned into two batches, with two sentences in the first batch sharing the longer prefix abc under one permutation, and with sentences in the second batch sharing axy under another permutation, respectively. From the intended practical point of view, batches of sentences with

permutated words can then be processed one-by-one supporting caching of the corresponding prefix.

Problem Statement

In mathematical terms, the problem can be formulated as follows. Given a $n \times m$ matrix M of words over an alphabet A, where $n, m \geq 1$, the goal is to partition the set of rows of M into k disjoint subsets and for each subset, find a permutation of columns of M under which the weighted sum of sizes of prefixes shared by concatenations of words from each subset is maximal:

$$\underset{B}{\operatorname{argmax}}\left(\sum_{i=1}^{k} |B_i| \times \max_{P(i,x)} (prefixsize^i(\mathrm{P}(i,1), \ldots, \mathrm{P}(i,n))) \right)$$

where B is a partitioning of $1, .., m$ into (mutually disjoint) subsets $B_1, .., B_k$
and for all $i \in \{1, .., k\}$,

- $\mathrm{P}(i,x)$: $\{1, \ldots, n\} \to \{1, .., n\}$ is a permutation;
- $prefixsize^i$ is the number of symbols in the longest prefix shared by words concat($M_{P(i,1),j}$, …, $M_{P(i,n),j}$), for all $j \in B_i$

where concat denotes concatenation.

Call for Proposals

Within the scope of this challenge, we seek proposals including, but not limited to:

- references to literature on related optimization problems and review of methods to solve them
- relaxations of the given optimization problem to ensure computational efficiency while conforming with the original practical motivation of prefix caching
- standalone algorithms to solve the problem or encodings into the language of solvers

We would greatly appreciate any insights, recommendations, or innovative approaches that could help in solving this challenge. Your expertise and contributions could empower our collaborative efforts, paving the way for deeper exploration and future partnerships.

References

[1] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In Proceedings of the 29th Symposium on Operating Systems Principles (SOSP '23).

[2] Zheng, Lianmin, et al. "Sglang: Efficient Execution of Structured Language Model Programs." *Advances in Neural Information Processing Systems* 37 (2024): 62557-62583

[3] Shu Liu, Asim Biswal, Audrey Cheng, Xiangxi Mo, Shiyi Cao, Joseph E Gonzalez, Ion Stoica, and Matei Zaharia. Optimizing LLM Queries in Relational Workloads. arXiv preprint arXiv:2403.05821, 2024.