



MOCCASIN: Efficient Tensor Rematerialization for Neural Networks

Speaker: Burak Bartan

Senior Engineer

Qualcomm Technologies, Inc.

*Qualcomm AI Research**

Email: bbartan@qti.qualcomm.com

Introduction

- The deployment and training of neural networks on edge computing devices pose many challenges.
- The low memory nature of edge devices is often one of the biggest limiting factors encountered in the deployment of large neural network models.
- Tensor rematerialization or recompute is a way to address high memory requirements for neural network training and inference.

Introduction

- Consider execution of the nodes in the order

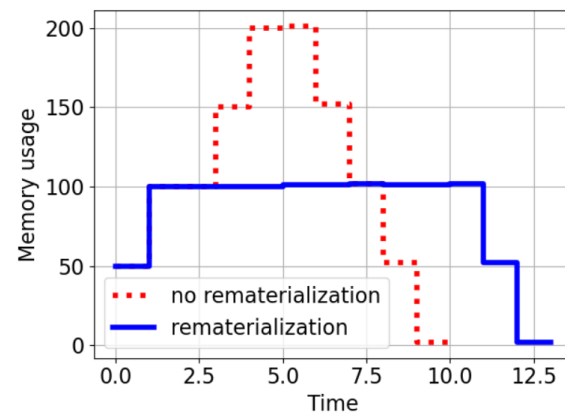
0, 1, 2, 3, 4, 5, 6, 7, 8, 9

- We need to keep the output of node 1 in memory until we execute node 8 (similarly for 2&7)
- Rematerialization:** Discard output of node 1 after node 2 is computed and recompute it later:

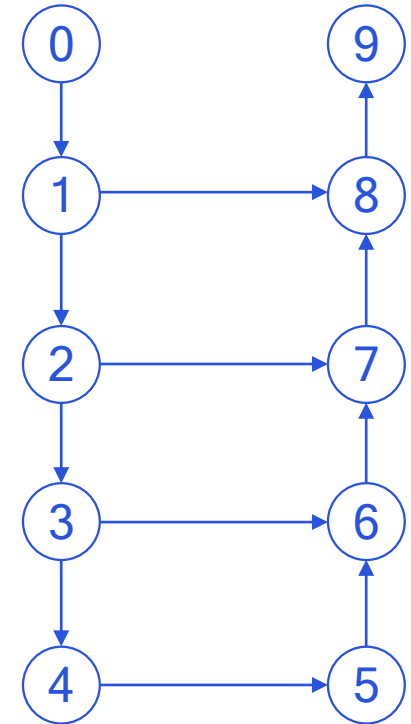
0, 1, 2, 3, 4, 5, 6, 7, 8, 9

discard 1 recompute 1

- To recompute node 1:** We need to recompute 0 or have it available in memory - another decision that needs to be made by the algorithm



Memory usage vs time



10-node computation graph where

- Output sizes are 50 for nodes on the left, and 1 for the ones on the right.
- Duration is 1 for all nodes.

Introduction

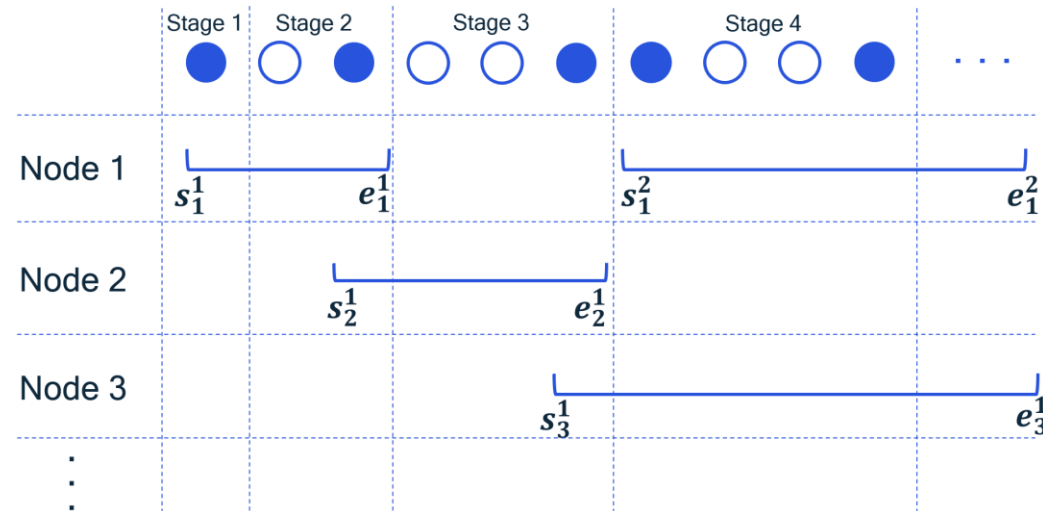
- This problem studies the trade-off between memory and compute.
 - By allowing nodes to be rematerialized (i.e. extra compute), it is possible to lower the memory footprint of a sequence.
 - The problem of determining which node outputs to keep in memory and which to discard (and rematerialize later) to reduce peak memory is a combinatorial optimization problem.

MEMORY-CONSTRAINED COMPUTATION GRAPH SEQUENCING WITH REMATERIALIZATION
minimize total execution duration $seq(G)$
subject to
– $seq(G)$ meets the data dependencies of G
– peak memory footprint of $seq(G)$ \leq local memory capacity M

- We have developed a new optimization problem formulation keeping scalability in mind, which greatly simplifies the problem.
 - Key assumption: Don't rematerialize any given node more than C times.

Proposed solution

- Our solution is an optimization problem formulation where
 - The main building block is the concept of output retention intervals which simplifies the problem formulation greatly
 - Output retention intervals are used to model how long the output of a node is retained in memory
 - We define C retention intervals for each node where each interval represents a rematerialization (i.e. recompute) of that node
 - The starting s_v^i and end e_v^i times of the intervals are modeled as the decision variables of the optimization problem
 - We then define the precedence and memory constraints using these decision variables
 - The resulting problem could be posed as a constraint programming (CP) problem, which we could use any generic CP solver to solve numerically



Proposed solution

$$\begin{aligned}
 & \underset{s,e,a}{\text{minimize}} && \sum_{v,i} w_v a_v^i && (1) && \text{-----} \rightarrow \text{Total amount of computation} \\
 & \text{subject to} && s_v^i \leq e_v^i, \forall v, i && (2) && \text{-----} \rightarrow \text{Start of an interval comes before end} \\
 & && e_v^i \leq s_v^{i+1}, \forall v, \forall i < C_v && (3) && \text{-----} \rightarrow \text{Ordering among the intervals of a node} \\
 & && \sum_{v,i: s_v^i \leq t \leq e_v^i} m_v a_v^i \leq M, \forall t \in \mathcal{D} && (4) && \text{-----} \rightarrow \text{Memory use cannot exceed } M \\
 & && \forall (u, v) \in E, \forall i \in \{i : a_v^i = 1\}, \exists j \text{ such that} && && \\
 & && \quad a_u^j = 1, s_u^j + 1 \leq s_v^i \leq e_u^j && (5) && \text{-----} \rightarrow \text{Precedence constraints} \\
 & && s_v^i \neq s_u^j, \forall v, u \in \{v, u : v \neq u\}, \forall i, j && (6) && \text{-----} \rightarrow \text{Interval start times are all different} \\
 & && a_v^1 = 1, \forall v && (7) && \text{-----} \rightarrow \text{There is at least one active interval per node} \\
 & && s_v^i, e_v^i \in \mathcal{D}, a_v^i \in \{0, 1\}, \forall v, i. && (8) && \text{-----} \rightarrow \text{Variable domains}
 \end{aligned}$$

Notes:

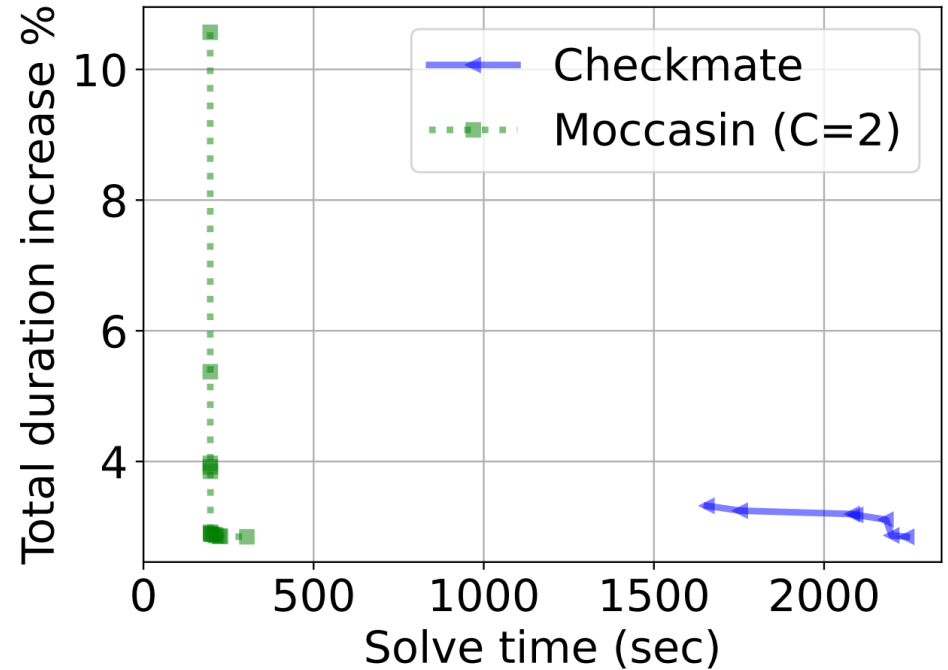
- The formulation above does not enforce any ordering on the nodes.
- We consider a fixed topological order to speed up solve.
- Number of intervals does not have to be a global parameter: C_v

Discussion and results

- The concept of retention intervals simplifies the problem formulation greatly
- More concretely, consider the complexity comparison:

- Comparison:

- Our formulation: $2Cn \log(n)$ Boolean variables
- Checkmate: $O(n^2 + nm)$ Boolean variables



Real-world graph with $n = 442$ nodes and $m = 1247$ edges

Thank you

Qualcomm

Follow us on: [in](#) [twitter](#) [instagram](#) [youtube](#) [facebook](#)

For more information, visit us at:

qualcomm.com & qualcomm.com/blog

Nothing in these materials is an offer to sell any of the components or devices referenced herein.

©2018-2023 Qualcomm Technologies, Inc. and/or its affiliated companies. All Rights Reserved.

Qualcomm is a trademark or registered trademark of Qualcomm Incorporated. Other products and brand names may be trademarks or registered trademarks of their respective owners.

References in this presentation to "Qualcomm" may mean Qualcomm Incorporated, Qualcomm Technologies, Inc., and/or other subsidiaries or business units within the Qualcomm corporate structure, as applicable. Qualcomm Incorporated includes our licensing business, QTL, and the vast majority of our patent portfolio. Qualcomm Technologies, Inc., a subsidiary of Qualcomm Incorporated, operates, along with its subsidiaries, substantially all of our engineering, research and development functions, and substantially all of our products and services businesses, including our QCT semiconductor business.

Snapdragon and Qualcomm branded products are products of Qualcomm Technologies, Inc. and/or its subsidiaries. Qualcomm patented technologies are licensed by Qualcomm Incorporated.