A Scalable Hardware Architecture for Efficient Learning of Recurrent Neural Networks at the Edge

TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

Yicheng Zhang, Manil Dev Gomony, Henk Corporaal, Federico Corradi
Eindhoven University of Technology

Neuromorphic Edge Computing Systems Lab

Abstract

- Motivation for RNN edge learning
  - RNN is powerful and compact at the edge for sequential tasks, e.g. speech recognition
  - Fine-tuning for personalized data enables better performance
  - Edge learning has advantages in energy efficiency, latency, and privacy over cloud
- Problem
  - traditional Back Propagation Through Time (BPTT) algorithm, demanding in memory and computing resources, hardly fits the edge devices
- Our contributions
  - devise the routine for the partition based algorithm, Forward Propagation Through Time(FPTT) at the edge to save memory
  - customized Chipyard-based hardware system for the routine to achieve further efficiency and trade-off options

Evolution of Sequential Learning

- traditionally in one go: update the weight **once** by all network states generated after processing the entire sequence

$$W_{new} = W_{old} - \eta \frac{\partial \sum_{t=1}^{T} l_{(t)}}{\partial W_{old}} \quad (1)$$

- Forward Propagation Through Time: update by the current network state **after every time step**; application of regularization term $R_{(t)}$ for stabilization

$$W_{(t+1)} = W_{(t)} - \eta \frac{\partial (l_{(t+1)} + R_{(t+1)})}{\partial W_{(t)}} = W_{(t)} - \eta \frac{\partial L_{(t+1)}}{\partial W_{(t)}} \quad (2)$$

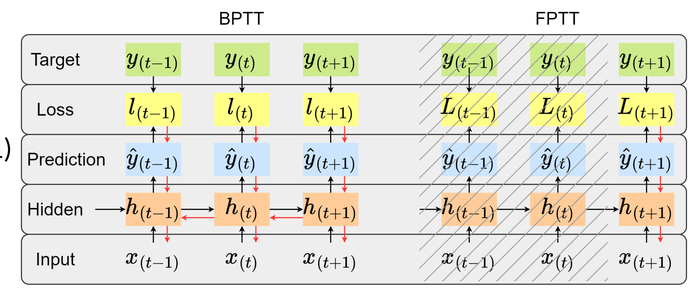

Figure 1. Unrolling of computation graph: BPTT vs FPTT

Memory Saving: memory for the network state can be **released** at every time step (usually use fewer partitions)

Experimental Set-up

- Sequential MNIST task, T=784; 4 samples as fine-tuning
- tiny model at the edge: 128x10 (LSTMxFully Connected)
- partition the sequence into K parts: 1,2,7,14,28,56
- FireSim with AMD UltraScale+ VCU118 for architecture simulation
- NVIDIA GPU L4 and V100 for baseline

Customized embedded platform

Two customizations applied:

1. **Compression of Gemmini**
   - global application of Brain Float 16 (BF16)
   - halved memory, usage of Weight Stationary only
2. **System Scaling**
   - 2 mesh sizes of the systolic array in Gemmini: 4x4, 8x8
   - 4 types of CPU cores: 1,2,4,8
   - leads to 8 design points

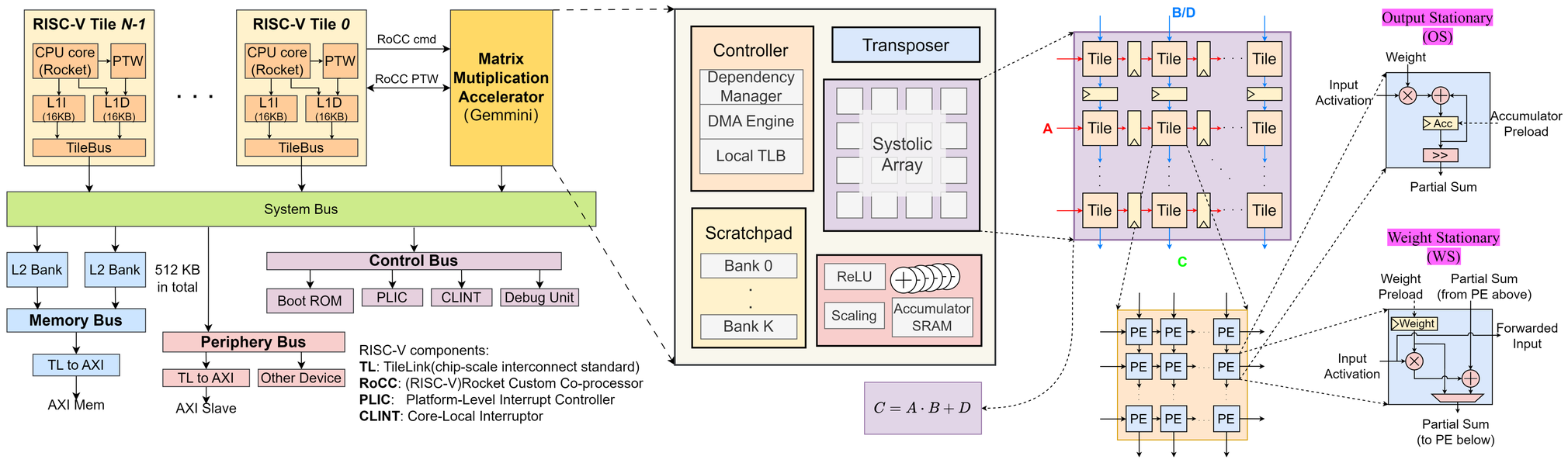

Figure 2. Chipyard-based System Architecture

Results

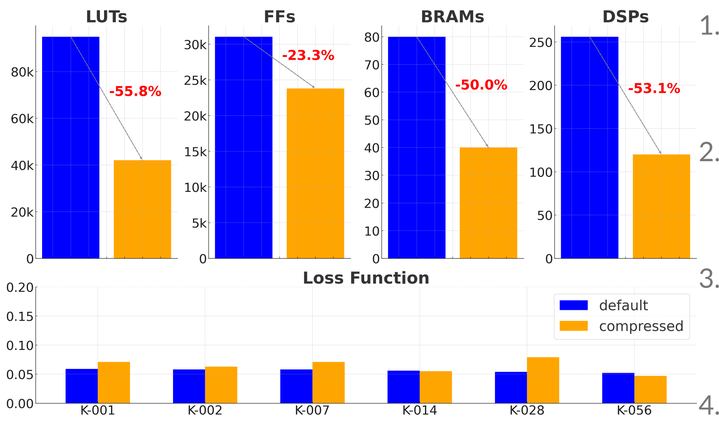

Figure 3. Effect of Gemmini Compression

1. Effect of Gemmini compression
   - Figure 3 shows significant decreases in resource utilization but similar performance in loss function before and after Gemmini compression.
2. Edge over cloud
   - For most clusters of K in Figure 4, customized architectures outperform two GPU platforms in latency.
3. Trade-off in design points
   - We can observe a trade-off between latency and resource utilization, given any cluster of K in Figure 4, and figure 5.
4. Degree of partition
   - For the cluster in Figure 4, a benchmark of larger K, i.e., finer partition over a sequence leads to a slight latency increase but significant memory saving.

Figure 5. Resource Utilization of design points

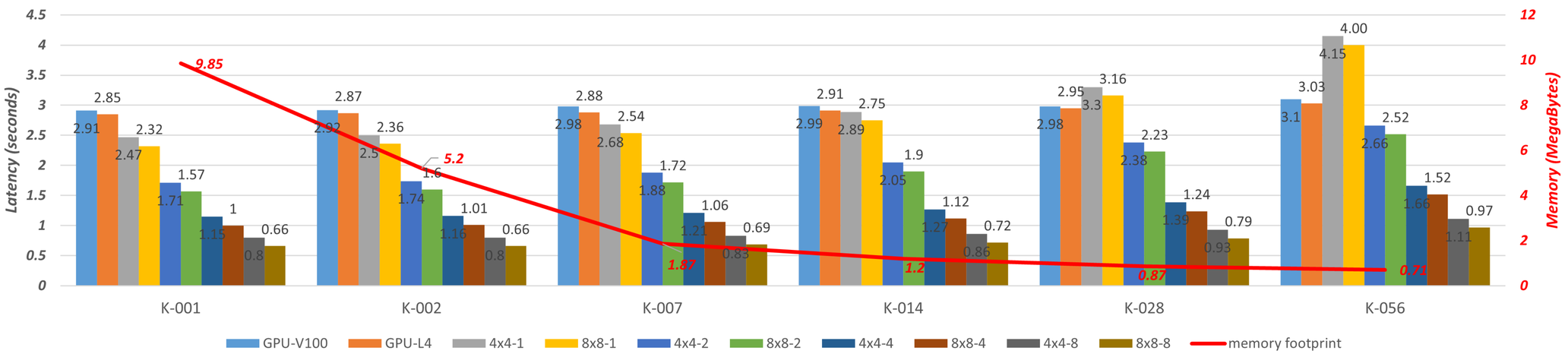

Figure 4. Latency and Memory of FPTT benchmarks of six K values on ten architectures (All MS-C are normalized to 500MHz)