E-LSTM: Efficient Inference of Sparse LSTM on Embedded Heterogeneous System

Runbin Shi[1] Junjie Liu[1] Shuo Wang[2] Yun Liang[2] Hayden So [1]

[1]Department of Electrical and Electronic Engineering
The University of Hong Kong

[2]Center for Energy-efficient Computing and Applications
School of EECS, Peking University

Design Automation Conference, June 2019

Table of Contents

Iterative Cell Evaluation in LSTM Inference

An illustration of the LSTM-cell iteration.

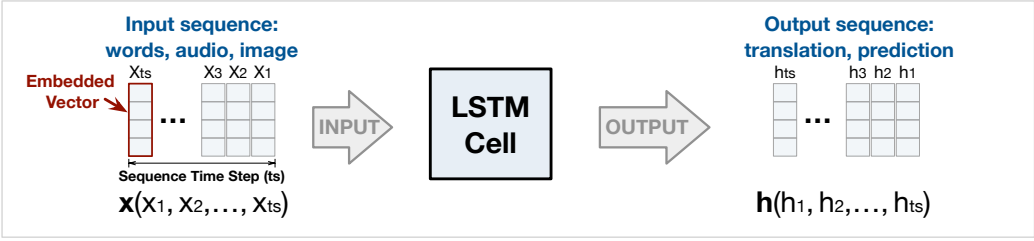


Figure: The LSTM cell and its iterative evaluation over temporal sequence.

Iterative Cell Evaluation in LSTM Inference

An illustration of the LSTM-cell iteration.

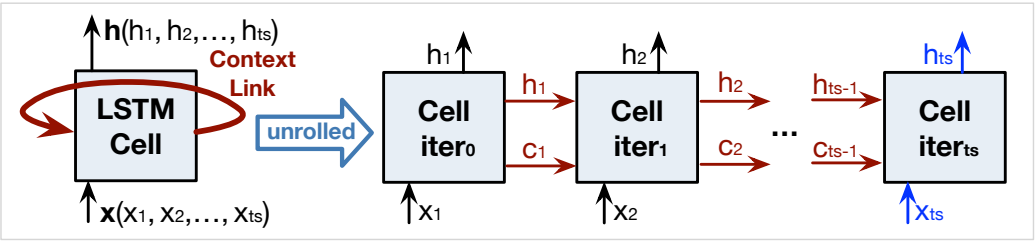


Figure: The LSTM cell and its iterative evaluation over temporal sequence.

Arithmetic of LSTM-cell Computation

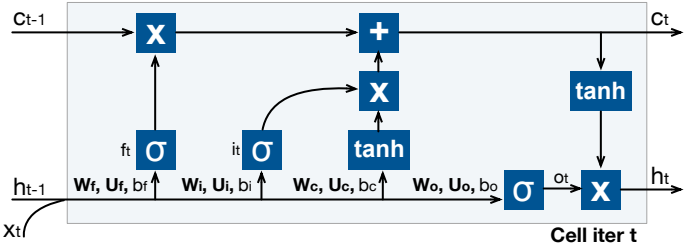


Figure: Detail dataflow in the LSTM cell.

$$f_t = \sigma(\boldsymbol{W_f} x_t + \boldsymbol{U_f} h_{t-1} + b_f) \tag{1}$$

$$i_t = \sigma(\boldsymbol{W_i} x_t + \boldsymbol{U_i} h_{t-1} + b_i) \tag{2}$$

$$c_t = f_t \cdot c_{t-1} + i_t \cdot \tanh(\boldsymbol{W_c} x_t + \boldsymbol{U_c} h_{t-1} + b_c) \tag{3}$$

$$o_t = \sigma(\boldsymbol{W_o} x_t + \boldsymbol{U_o} h_{t-1} + b_o) \tag{4}$$

$$h_t = o_t \cdot tanh(c_t) \tag{5}$$

Heavy Workload *v.s.* Low Performance of Embedded CPU

Main Computational Workload of LSTM

Matrix-vector multiplication:

$$\boldsymbol{W}x_t, \quad \boldsymbol{W} = (\boldsymbol{W_f}, \boldsymbol{W_i}, \boldsymbol{W_c}, \boldsymbol{W_o})^T \in \mathbb{R}^{4n \times m}, x_t \in \mathbb{R}^m$$
$$\boldsymbol{U}h_t, \quad \boldsymbol{U} = (\boldsymbol{U_f}, \boldsymbol{U_i}, \boldsymbol{U_c}, \boldsymbol{U_o})^T \in \mathbb{R}^{4n \times n}, h_t \in \mathbb{R}^n$$

In a benchmark layer for machine comprehension, $m = n = 1500$, one sequence has 35 time steps (cell iteration). $630,000,000$ MACC operations for each sequence. One LSTM layer costs 0.63 second on a CPU with 1 GOp/s.

Heavy Workload *v.s.* Low Performance of Embedded CPU

Main Computational Workload of LSTM

Matrix-vector multiplication:

$$\boldsymbol{W}x_t, \quad \boldsymbol{W} = (\boldsymbol{W_f}, \boldsymbol{W_i}, \boldsymbol{W_c}, \boldsymbol{W_o})^T \in \mathbb{R}^{4n \times m}, x_t \in \mathbb{R}^m$$
$$\boldsymbol{U}h_t, \quad \boldsymbol{U} = (\boldsymbol{U_f}, \boldsymbol{U_i}, \boldsymbol{U_c}, \boldsymbol{U_o})^T \in \mathbb{R}^{4n \times n}, h_t \in \mathbb{R}^n$$

Sparsity in Weight Matrix

$$Sparsity(\boldsymbol{W}, \boldsymbol{U}) \in [0.2, 0.8]$$

CPU performance decreases while computing Sparse matrix-vector multiplication (SpMV).

Embedded Solution for LSTM Inference

A heterogeneous system coupling CPU and a generic LSTM accelerator.

Target Platform: Tightly-coupled Heterogeneous System

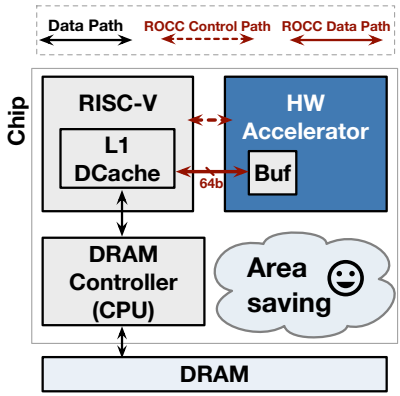


Figure: Tightly-coupled Arch.

Advantages:

- **lower latency:** 30 cycles (DRAM access) *v.s.* 1 cycle (DCache access via ROCC)
- **smaller area:** ~~DRAM Controller (Accel)~~

Limitations:

- **chip-area limitation:** off-chip weight storage
- **ROCC bandwidth:** 64bits/cycle

Table of Contents

eSELL: Area-saving Sparse Weight Format

Access Coalescing

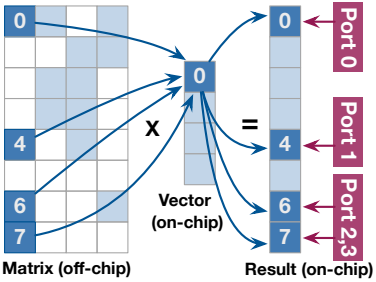


Figure: Column-major SpMV with Compressed Sparse Column (CSC) format, 4 MACC per cycle.

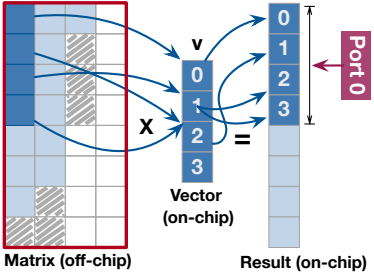


Figure: Coalesced access to result buffer, 4 MACC per cycle, 63% area reduction to CSC.

SRAM Area Estimation [?]

$$area \propto (\#bits)^{0.9} \times (\#port)^{0.7}$$

eSELL: Area-saving Sparse Weight Format

Weight Format Construction

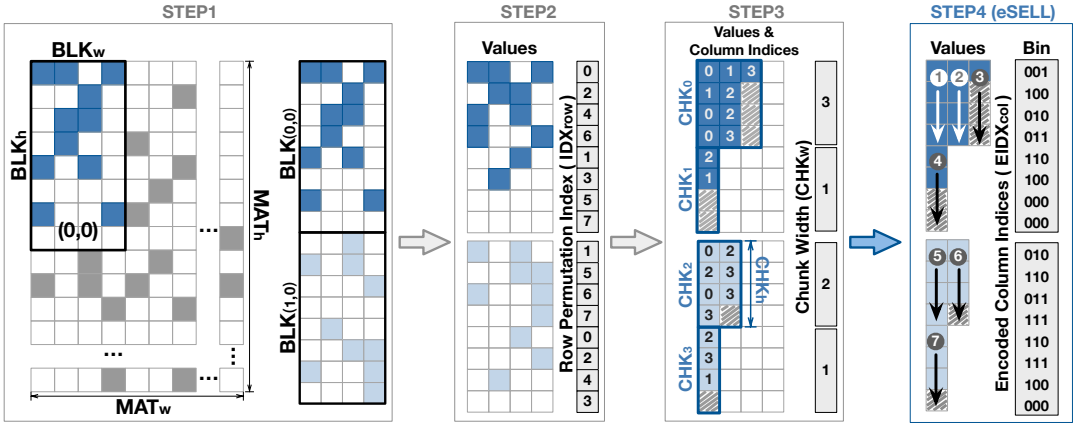


Figure: Steps for eSELL weight format construction.

eSELL: Area-saving Sparse Weight Format

Alignment to ROCC Interface

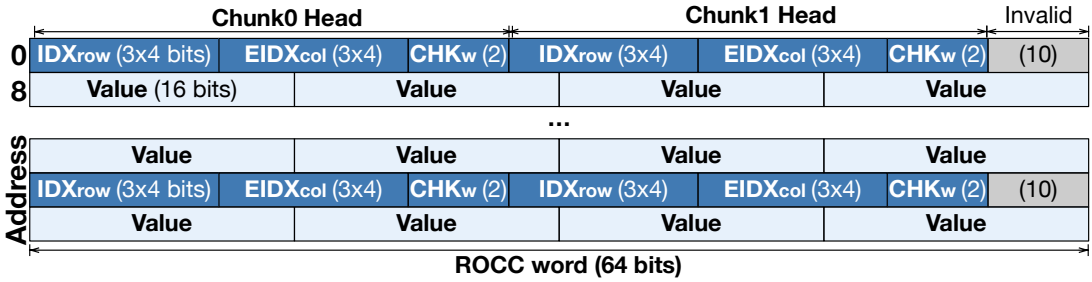


Figure: eSELL storage / transmission pattern aligned with ROCC 64-bits interface.

Table of Contents

Generic Accelerator Hardware for Embedded LSTM

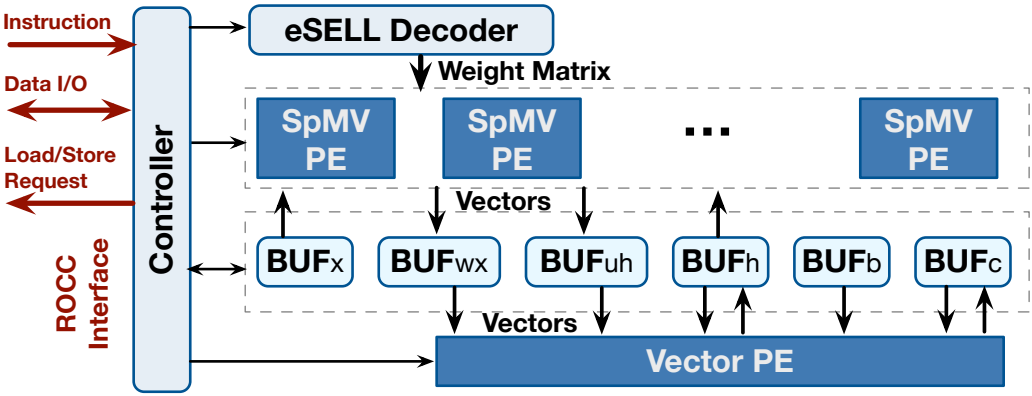


Figure: Accelerator architecture in E-LSTM.

Throughput Bottleneck

Pipeline diagram for single SpMV-PE case.

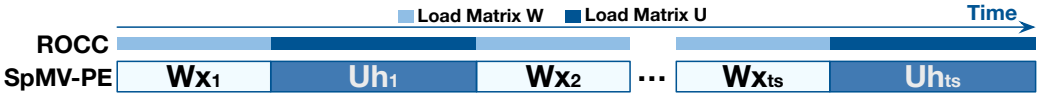


Figure: Process cell iterations in sequence; both ROCC and PE are fully utilized.

Throughput Bottleneck

Pipeline diagram for multiple SpMV-PE case.

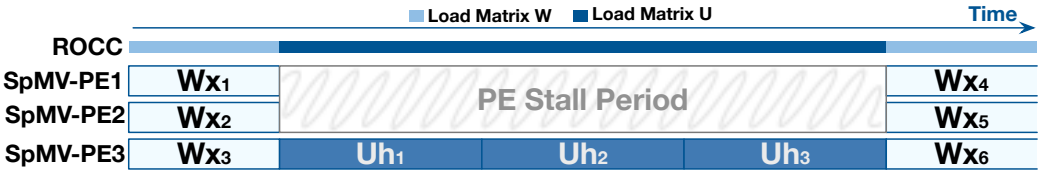


Figure: Process $\boldsymbol{W}x_t$ in parallel and $\boldsymbol{U}h_t$ in sequence.

Throughput Bottleneck

Pipeline diagram for multiple SpMV-PE case.

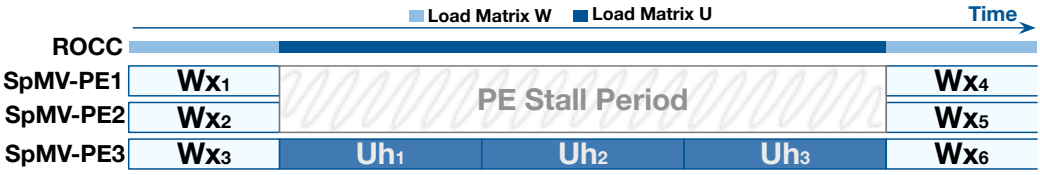


Figure: Process $\boldsymbol{W}x_t$ in parallel and $\boldsymbol{U}h_t$ in sequence.

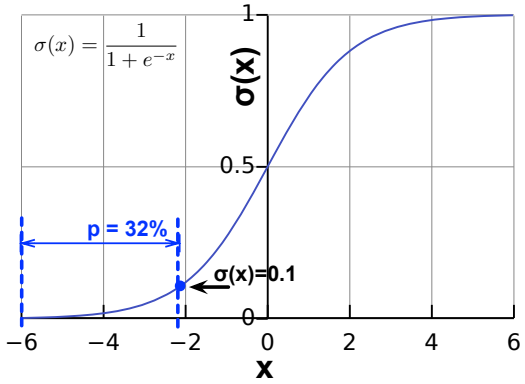
Pipeline Stall

$\boldsymbol{W}x_t$ and $\boldsymbol{U}h_t$ cannot be computed concurrently, as the ROCC can only load one word of $\boldsymbol{W}$ or $\boldsymbol{U}$ in each cycle. Thus the stall of PE is unavoidable, and $\boldsymbol{U}h_t$ becomes the throughput bottleneck.

Optimization1: Shorten $\boldsymbol{U}h_t$ period with inherent sparsity of h_t

Backtrace of h_t computation:

$$o_t = \sigma(\boldsymbol{W_o}x_t + \boldsymbol{U_o}h_{t-1} + b_o)$$
$$h_t = o_t \cdot tanh(c_t)$$



Inherent sparsity of h_t

As $P(o_t < 0.1) \approx 0.32$, and $tanh(c_t) \in (-1, 1)$, a considerable portion of h_t is closed to zero that can be regarded as zero in $\boldsymbol{U}h_t$ computation.

Optimization1: Shorten $\boldsymbol{U}h_t$ period with inherent sparsity of h_t

Sparse-Matrix Sparse-Vector Multiplication (SpMSpV) in $\boldsymbol{U}h_t$

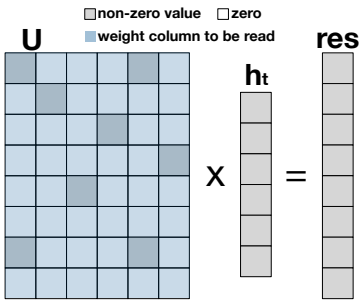


Figure: SpMV: original computation of $\boldsymbol{U}h_t$.

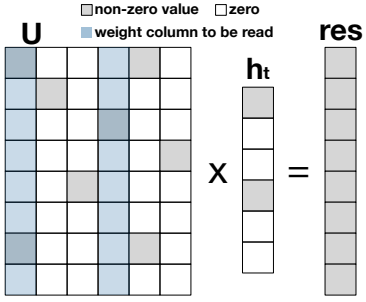


Figure: SpMSpV: $\boldsymbol{U}h_t$ computation considering inherent sparsity of h_t.

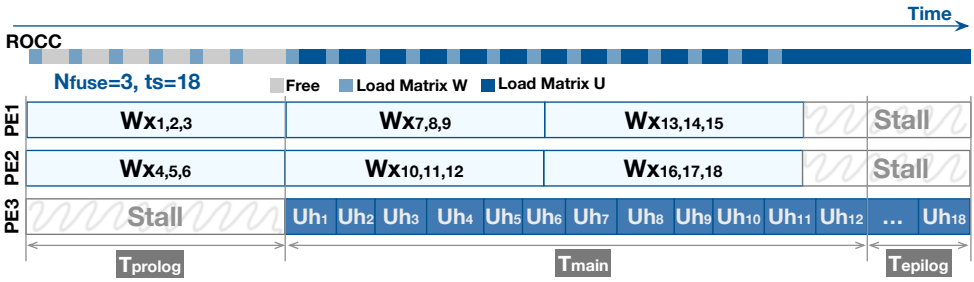
In this example, SpMSpV achieves $3\times$ speedup on $\boldsymbol{U}h_t$ computation.

Optimization2: Scheduling with Cell-fusion

Inter-cell Parallel Scheme: Cell-fusion

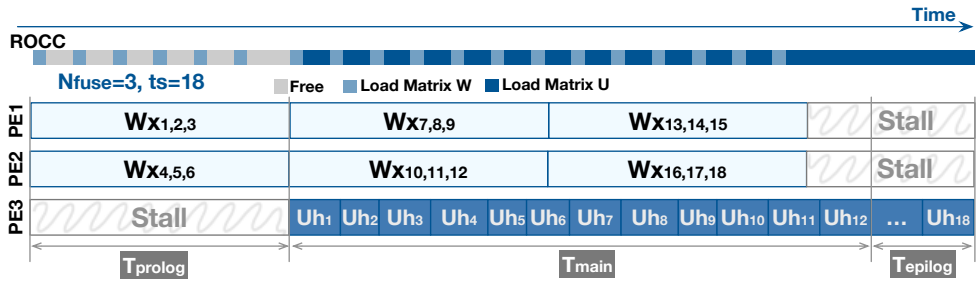
Cell-fusion Scheme

Assuming there are N_{pe} PEs, we set $(N_{pe} - 1)$ of them process $\boldsymbol{W}x_t$ (SpMV) and the rest one process $\boldsymbol{U}h_t$ (SpMSpV). Besides, each SpMV-PE process $\boldsymbol{W}x_t$ of N_{fuse} cell iterations in interleave.



Optimization2: Scheduling with Cell-fusion

Inter-cell Parallel Scheme: Cell-fusion

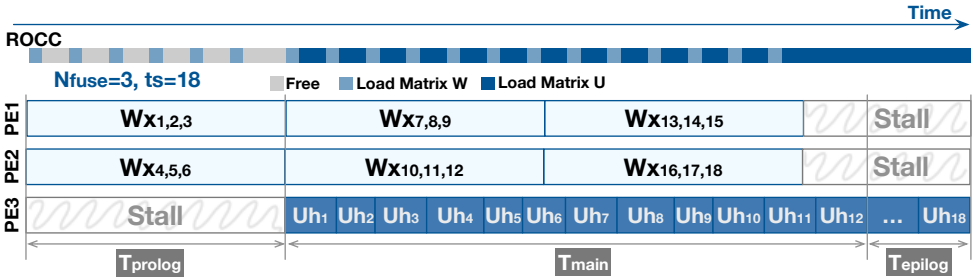


Advantage: $\boldsymbol{W}x_t$ and $\boldsymbol{U}h_t$ are processed in concurrent. In every N_{fuse} cycles, ROCC interface is occupied by loading $\boldsymbol{W}$ for 1 cycle and loading $\boldsymbol{U}$ for $(N_{fuse} - 1)$ cycles.

Optimization2: Scheduling with Cell-fusion

Fine-tuning fusion factor (N_{fuse}) in Cell-fusion

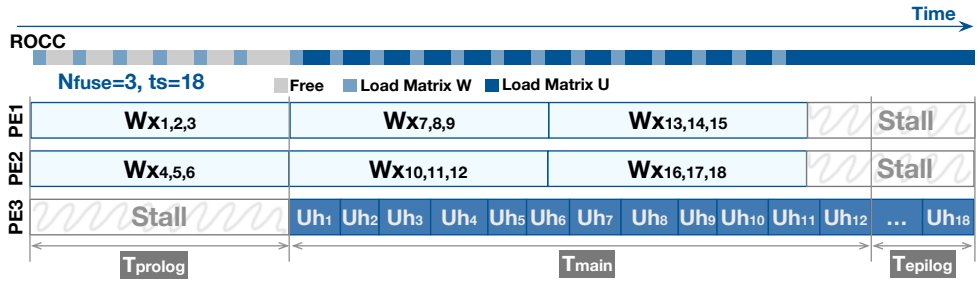
Fine-tuning N_{fuse} for the minimum computation period.



Optimization2: Scheduling with Cell-fusion

Fine-tuning fusion factor (N_{fuse}) in Cell-fusion

Fine-tuning N_{fuse} for the minimum computation period.



Optimization target:

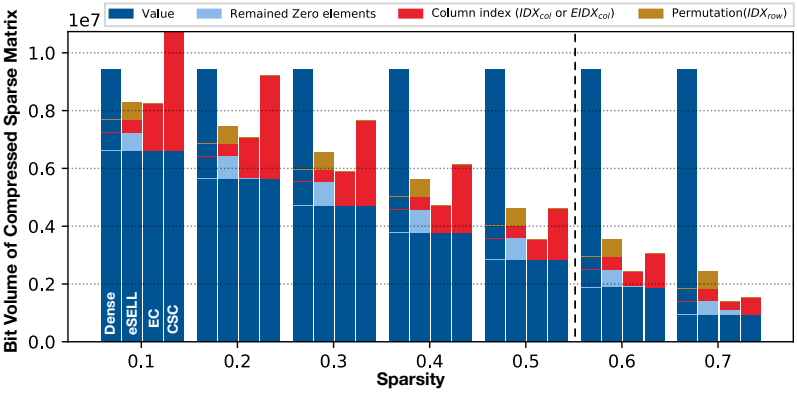
$$\underset{N_{fuse}}{\text{minimize}} \quad T(N_{fuse}) = T_{prolog} + T_{main} + T_{epilog}$$

with time consumption model: $T = f(N_{fuse}, N_{pe}, ts, len(x), len(h), Sp_w, Sp_u, Sp_h)$

Table of Contents

- Sparse LSTM benchmark layers: PyTorch
- Heterogeneous system with LSTM accelerator: CPP behavioral model in Spike (a gem5-like simulator in RISC-V eco-system)
- Scripts for eSELL-format model translation and N_{fuse} fine-tuning: Python
- Source code: `https://github.com/rbshi/elstm`

Bit-volume comparison between sparse formats in different sparsity.
(matrix size: 768×768)



Competitive bit volume, but 63% reduction on SRAM area-cost due to less port usage.

Experiments and Evaluation

Sparse LSTM Benchmark Layers

Table: Benchmark LSTM layers from three real-world applications.

Name	Layer	len(x)	len(h)	ts	Sp_w	Sp_u	Sp_h	Score
OCR	LSTM1	28	128	28	0.3	0.5	0.22	98.68/98.61/98.11
(MNIST)	LSTM2	128	128	28	0.2	0.4	0.29	the higher, the better
LM	LSTM1	800	800	35	0.2	0.5	0.56	81.33/81.67/88.52
(PTB)	LSTM2	800	800	35	0.2	0.6	0.41	the lower, the better
LM	LSTM1	1500	1500	35	0.4	0.5	0.37	101.63/102.15/106.5
(Wikitext)	LSTM2	1500	1500	35	0.3	0.4	0.39	the lower, the better

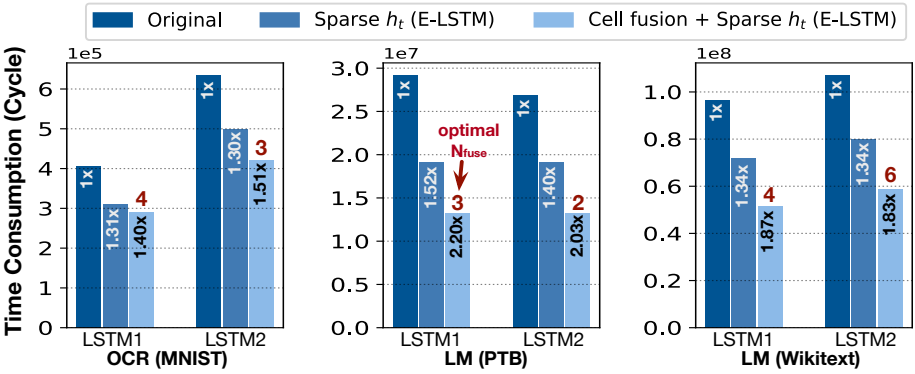
$Sp_{w/u}$: sparsity of weight matrix $\boldsymbol{W}, \boldsymbol{U}$
Sp_h : sparsity of hidden state (h_t)
len(x/h) : size of input vector (x_t) or hidden state vector (h_t)
ts : time steps (length) of a sequence

Experiments and Evaluation

Performance Comparison

Accelerator hardware configuration: 3 PEs (12 MACC Ops/cycle)

Schemes in comparison: Original v.s. Sparse h_t v.s Cell fusion + Sparse h_t



Max. Speedup: 1.52× in Sparse h_t scheme; 2.2× in Cell fusion + Sparse h_t scheme.

Table of Contents

Conclusion

- E-LSTM provides a solution for LSTM acceleration in *embedded heterogeneous system* considering the latency and chip-area cost.
- E-LSTM leverages the *inherent sparsity* in algorithm and proposes the *cell-fusion* scheme. With the fine-tuned fusion factor, a significant speed up is achieved. This scheme is also suitable to all LSTM accelerators.
- The open sourced framework contributes to the RISC-V heterogeneous system community.