Scaling Back-Propagation by Parallel Scan Algorithm

Shang Wang[1,2] | Yifan Bai[1] | Gennady Pekhimenko[1,2]

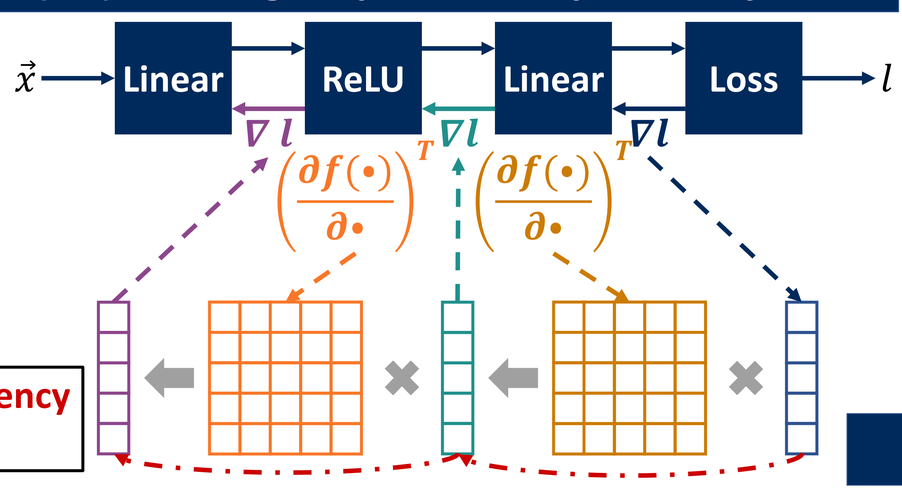
[1] Computer Science, University of Toronto

[2] Vector Institute

Back-propagation (BP)'s Strong Sequential Dependency

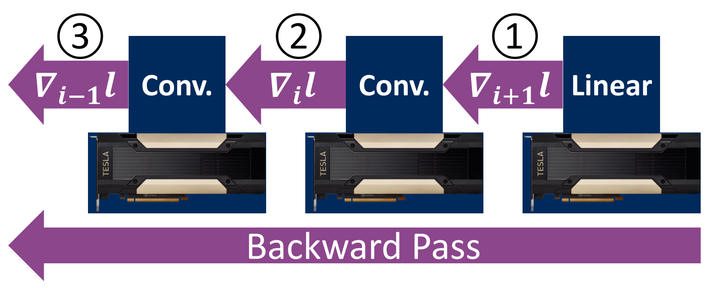
$$\nabla_{\vec{x}} l = \left(\frac{\partial f(\vec{x})}{\partial \vec{x}}\right)^T \nabla_{f(\vec{x})} l$$

Strong Sequential Dependency along layers.



Model Parallel Training

Strong sequential dependency **limits scalability** on parallel systems.



Pipeline Parallel Training:

- **Linear** per-device space complexity.
- "**Bubble of idleness**" vs. **convergence affect**.

What is a Scan Operation?

Binary, associative op.: **+**, input: 1 2 3 4 5 6 7 8

Exclusive scan: 0 1 3 6 10 15 21 28

Worker (**p**): an instance of execution; e.g., a core in a multi-core CPU.

Number of Elements (**n**)

On a single worker: scan linearly: **n** steps.

With more workers: **sublinear** steps?

Reformulate BP as a Scan Operation

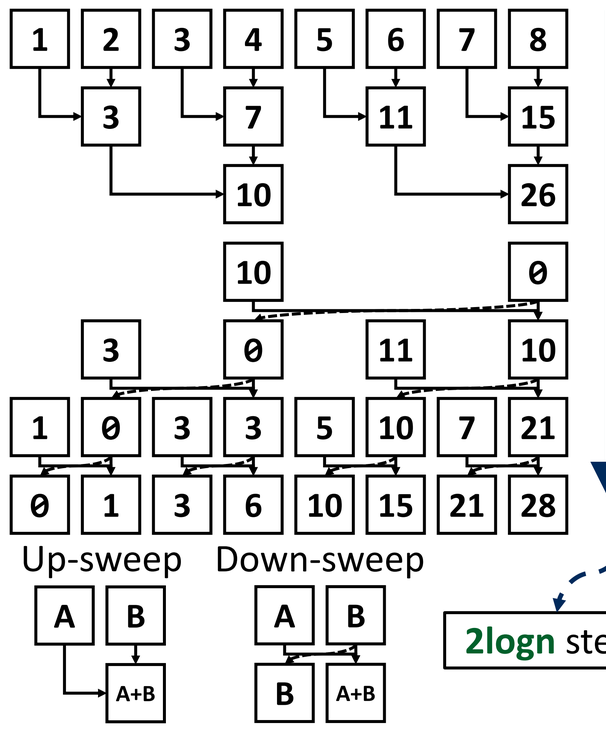
Key Insight: matrix multiplication in **BP** is also **binary** & **associative**!

$G_i = \nabla_{\vec{x}_i} l$ Define op.: **A ◊ B = BA**, input: G_7 J_7 J_6 J_5 J_4 J_3 J_2 J_1

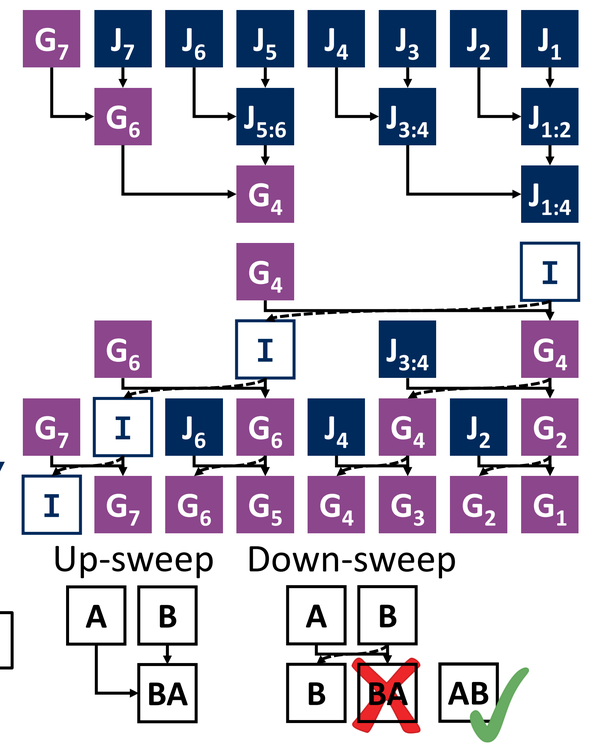
$J_{i+1} = \left(\frac{\partial \vec{x}_{i+1}}{\partial \vec{x}_i}\right)^T$

Exclusive scan: I G_7 G_6 G_5 G_4 G_3 G_2 G_1

Blelloch Scan



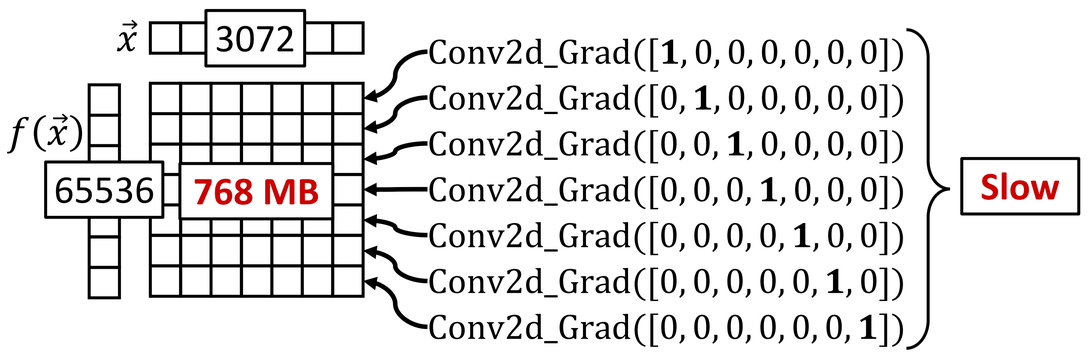
Scale BP by Blelloch Scan



Jacobians are Memory & Compute Hungry

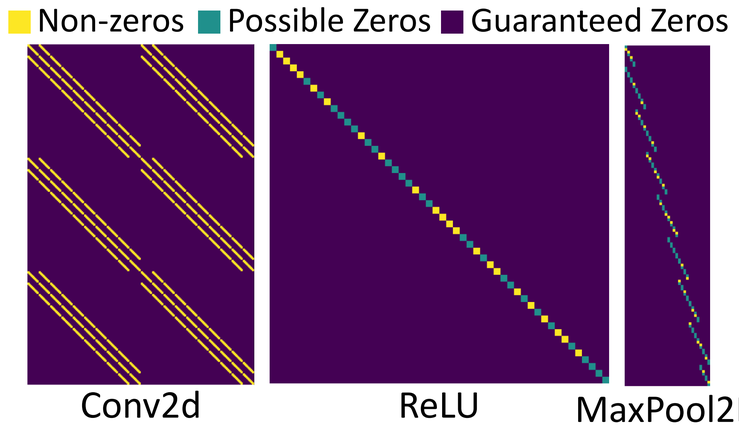
A full Jacobian: **prohibitively expensive**.

e.g., 1st convolution in VGG-11 on CIFAR-10 images:



Generated by `Op_Grad(basis vectors)` one by one.

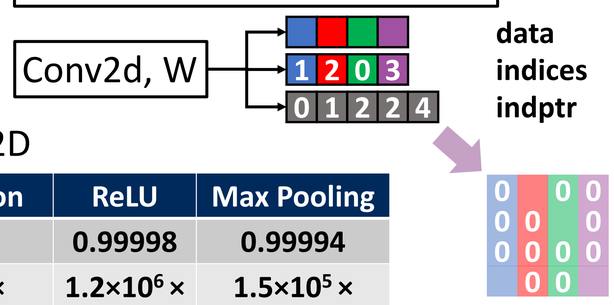
Leverage the Sparsity in the Jacobians



Deterministic pattern. ✓ Known **ahead of training time.** ✓

Potentially **better** SpGEMM performance.

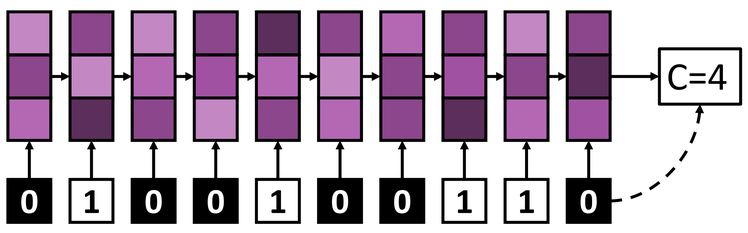
Generated **directly** into **CSR**:



First three ops of VGG-11 on CIFAR-10	Convolution	ReLU	Max Pooling
Sparsity	0.99157	0.99998	0.99994
Jacobian Calculation Speedup	$8.3\times10^3\times$	$1.2\times10^6\times$	$1.5\times10^5\times$

Evaluation

Model: **RNN** Task: **Bitstream Classification**

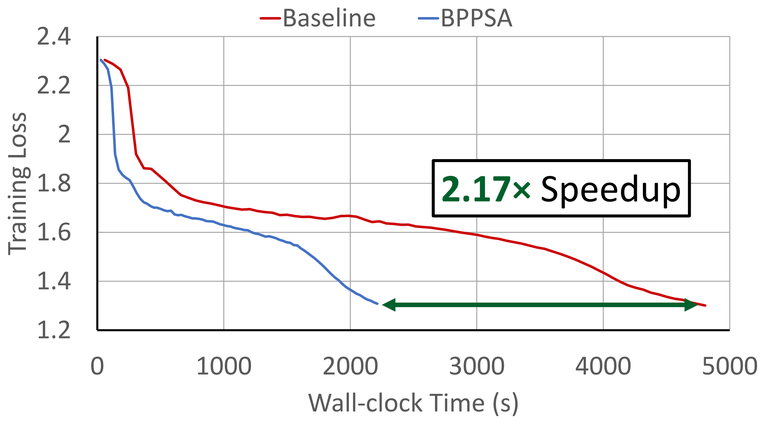


Baseline: PyTorch **Autograd** & **cuDNN**

Hardware: **RTX 2070** & **RTX 2080Ti**

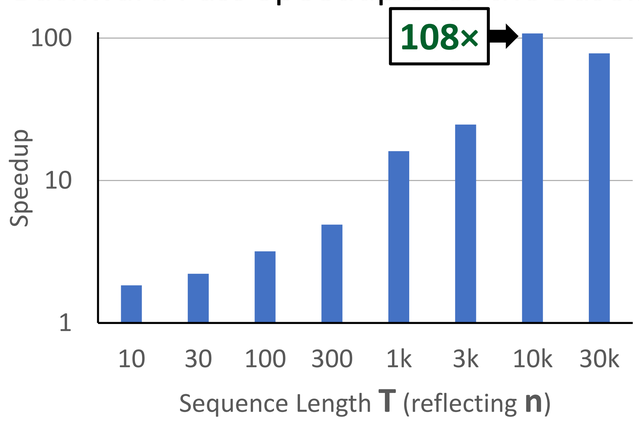
Implementation: **Custom CUDA Kernels**

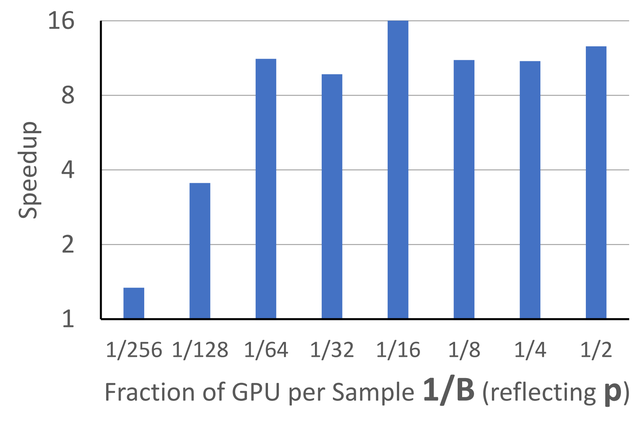
End-to-end training when Batch Size (**B**) = 16, Sequence Length (**T**) = 1000



BPPSA **reconstructs** the original BP **exactly**.

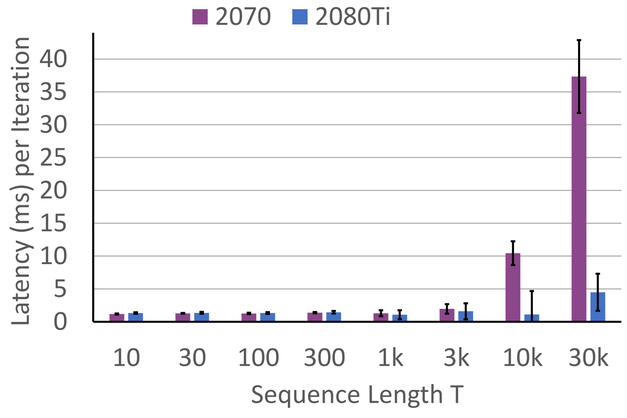
Backward Pass Speedup over the Baseline

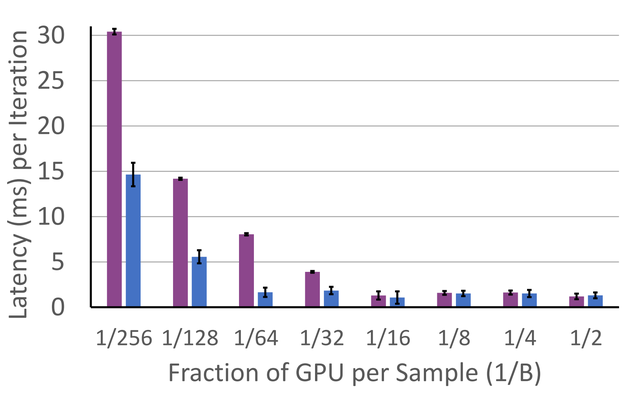




BPPSA **scales** with **n** until being bounded by **p**; and **scales** with **p**.

Hardware Sensitivity





# of SMs(2080Ti) > # of SMs(2070)
→ Latency(2080Ti) < Latency(2070).