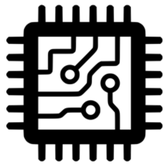
CoSA: Scheduling by Constrained Optimization for Spatial Accelerators

Qijing Huang, Minwoo Kang, Grace Dinh, Thomas Norell, Aravind Kalaiah,
James Demmel, John Wawrzynek, Yakun Sophia Shao
Email: jennyhuang@nvidia.com Git repo: https://github.com/ucb-bar/cosa

Scheduling is a big challenge

Intractable scheduling space



Exponentially growing algorithm complexity

Rapidly increasing hardware capacity

- **~10^{13} possible mappings for a typical ResNet layer on a 3-level architecture**

Accelerator-oriented scheduling

Key DNN accelerator properties to leverage:

Workload Regularity | Known HW Constraints | Explicit Data Management

CoSA: a one-shot approach

Brute-force
Timeloop
dMazeRunner
Triton Interstellar
Marvel

- Costly
- Sample invalid space
- Hard to generalize

Feedback-based
AutoTVM Halide
FlexFlow Gamma
MindMapping

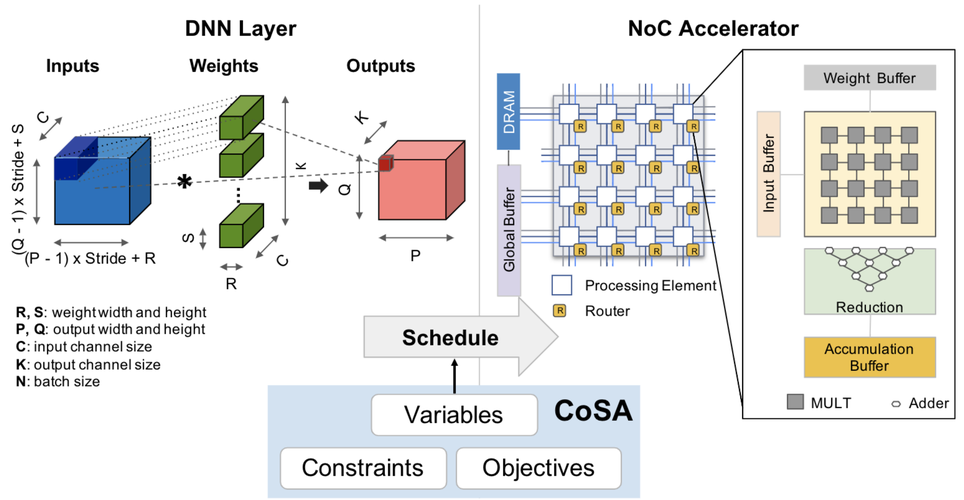
Constrained Optimization
Polly+Pluto TC
Tiramisu

- Unable to determine tiling factor sizes

CoSA - addresses key scheduling decisions

1 2

DNN scheduling formulation with CoSA



Three scheduling decisions

DRAM level
for q2 = [0 : 2) :
Global Buffer level
for q1 = [0 : 7) :
for n0 = [0 : 3) :
spatial_for r0 = [0 : 3) :
spatial_for k1 = [0 : 2) :
Input Buffer level
for c1 = [0 : 2) :
for p1 = [0 : 2) :
Weight Buffer level
for p0 = [0 : 2) :
spatial_for k0 = [0 : 2) :

1. Tiling Factors
How do we break up our tensors?

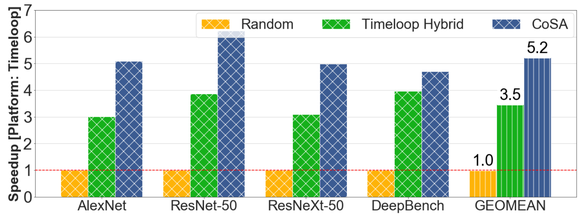
2. Spatial / Temporal
Which loops do we parallelize?

3. Loop Permutation
How do we order the loops?
What dataflow do we use"

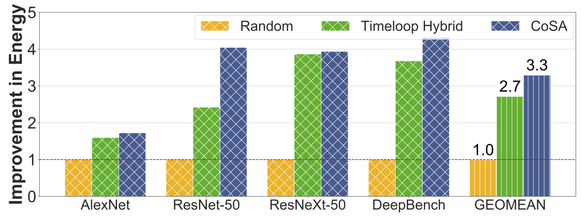
CoSA Evaluation

- Baselines:
  - Random (best out of 5 valid schedules)
  - Timeloop Hybrid (best out of 16K valid schedules)
- Platforms:
  - Timeloop Simulator

1.5x latency speedup



1.2x better energy efficiency



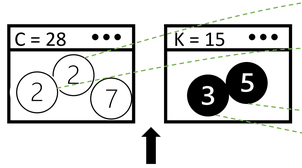
90x faster time-to-solution

	CoSA	Random	Timeloop Hybrid
Runtime / Layer	4.2s	4.6s (1.1x)	379.9s (90.5x)
Samples / Layer	1	20K	67M
Evaluations/ Layer	1	5	16K

4

3 Key idea: prime factor allocation

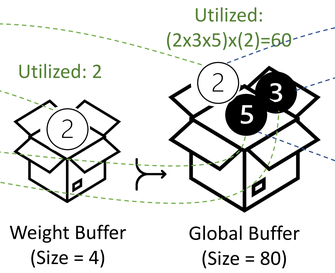
Prime factor items :



Example workload:

Matrix-vector mult:
for c in [0:C) // C = 28
for k in [0:K) // K = 15
OA[k] += IA[c] × W[c,k]

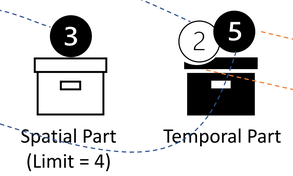
Local buffers:



1.Tiling Factors

	C=28			K=15	
Prime Factors	2	2	7	3	5
WeightBuf	✓				
GlobalBuf		✓		✓	✓
DRAM			✓		

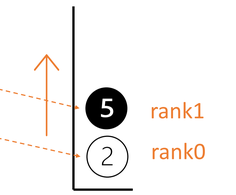
2.Spatial/Temporal Factors



GlobalBuf	C=28			K=15	
Prime Factors	2	2	7	3	5
Spatial				✓	
Temporal	✓				✓

3.Loop Permutation

Rank in Global Buffer:



GlobalBuf	C=28			K=15	
Prime Factors	2	2	7	3	5
rank0	✓				
rank1					✓
rank4					

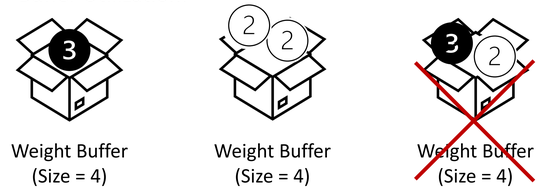
binary allocation variable **X**
✓ - 1
- 0

CoSA Variable X

CoSA optimizes the binary allocation variable **X** using the **constraints** and the **objectives**

CoSA Constraints

Buffer Utilization:



CoSA Objectives