Accelerating Generalized Linear Models with MLWeaving: A One-Size-Fits-All System for Any-Precision Learning

Zeke Wang, Kaan Kara, Hantian Zhang,
Gustavo Alonso, Onur Mutlu, Ce Zhang

Systems Group, ETH Zurich






Outline

Quick Background (5mins)

Stochastic Gradient Descent (SGD)

Synchronous vs. Asynchronous

Low Precision

MLWeaving (10mins)

Arbitrary-precision Training

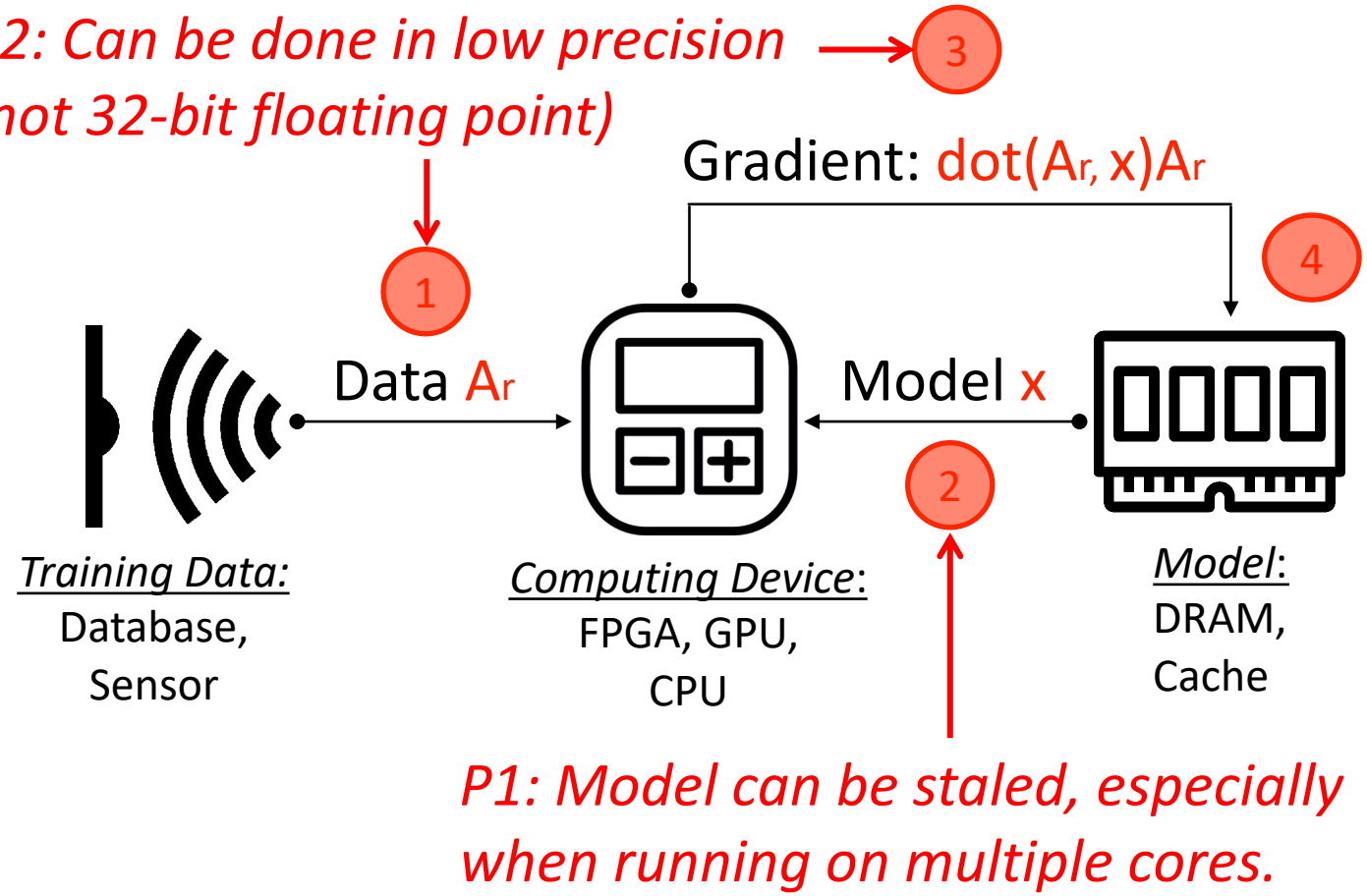
MLWeaving Memory Layout

MLWeaving Hardware Design

Efficient Synchronous Design

OK, how does SGD work?

Stochastic Gradient Descent (SGD)



Linear Regression

$$\min_x \frac{1}{2}\sum_r (A_r x^T - b_r)^2$$

```
A_r = get_data()
x = get_model()
g = comp_grad(x,A_r)
x = x - g
set_model(x)
```

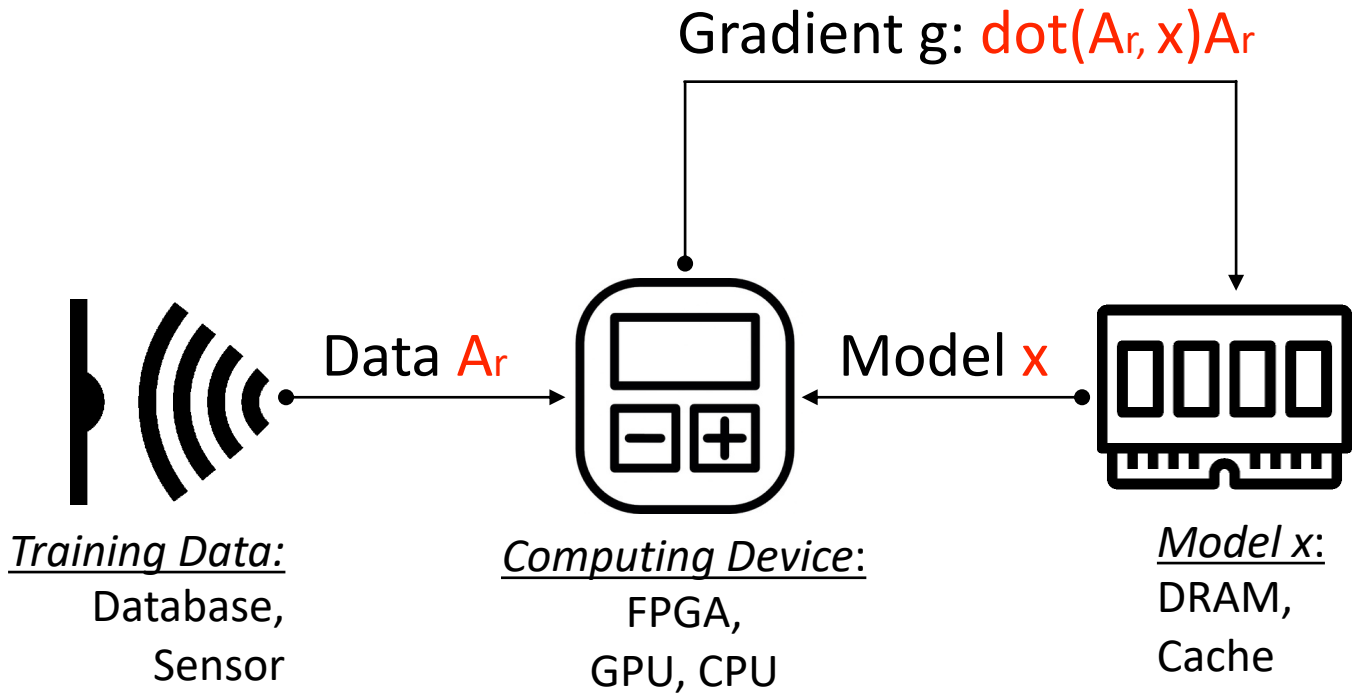
Two Interesting Properties

SGD on the CPU: synchronous or asynchronous?

Sync. Single-Core SGD: Low Throughput

CPU – Single Core

```
A_r = get_data()

x = get_model()

g = comp_grad(x, A_r)

x = x - g

set_model(x)
```



Read After Write (RAW) Dependency Regarding the Model x

Causes Problem When Using Multiple Cores.

Async. Multi-Core SGD: High Throughput

Multi-core SGD relies on asynchrony.

HogWild! [1]

ModelAverage [2]

[1] Hogwild: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent. In NIPS. 2011.

[2] Parallelized Stochastic Gradient Descent. In NIPS. 2010.

Synchrony vs. Asynchrony on CPUs

	Hardware Efficiency (Throughput)	Statistical Efficiency (Convergence Rate)
Single-core SGD (Synchrony)	Low 	High 
Multi-core SGD (Asynchrony)	High 	Low 

Why Low Precision?

Why Low Precision?

"It is a cat" (>0.5)

Full precision

1.310245

X 0.602069

0.788857897

Low precision

about 1.3

X about 0.6

about 0.78

Relax, It is only Machine Learning.

Current Hardware Supports Limited Precisions

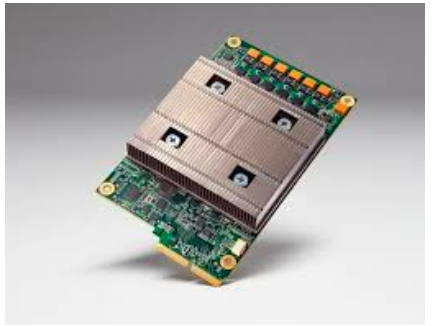
CPU

Char (8-bit),
Short (16-bit)

GPU

FP8 (8-bit),
FP16 (16-bit)

TPU



INT8 (8-bit)

Goal of This Work

For Generalized Linear Model training, can we enable things that cannot be well done on CPUs?

Any-precision Training

High-throughput Sync. Design

Outline

Quick Background (5mins)

Stochastic Gradient Descent (SGD)

Synchronous vs. Asynchronous

Low Precision

MLWeaving (10mins)

Arbitrary-precision Training

MLWeaving Memory Layout

MLWeaving Hardware Design

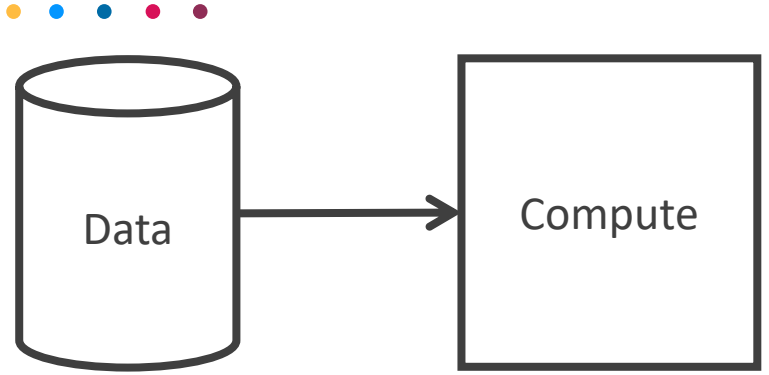
Efficient Synchronous Design

Two Goals of Arbitrary-precision Training

1, One hardware design and one copy of dataset support any-precision training.

2, Our design achieves linear speedup with lower precision.

MLWeaving Memory Layout



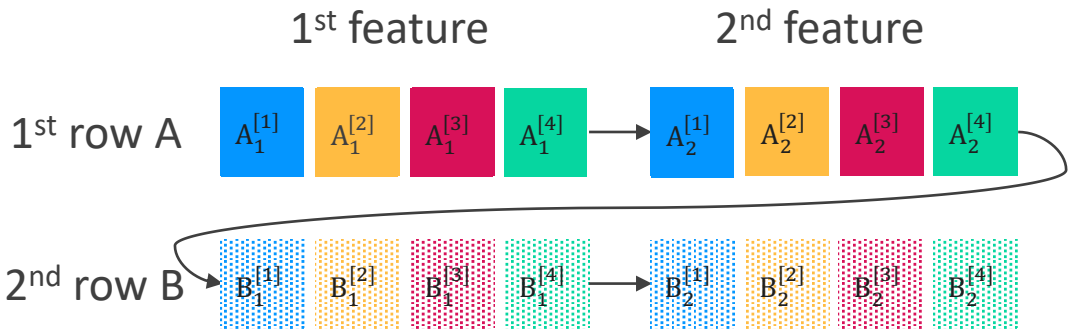
Observation 1:
Often memory bandwidth bound

Observation 2: Low precision (e.g., 8 bit fixed point) often provides reasonable quality

Observation 3: Different training task might need different precision level even on the same dataset

Can we store the data in a new data structure that efficiently supports arbitrary precision data movement?

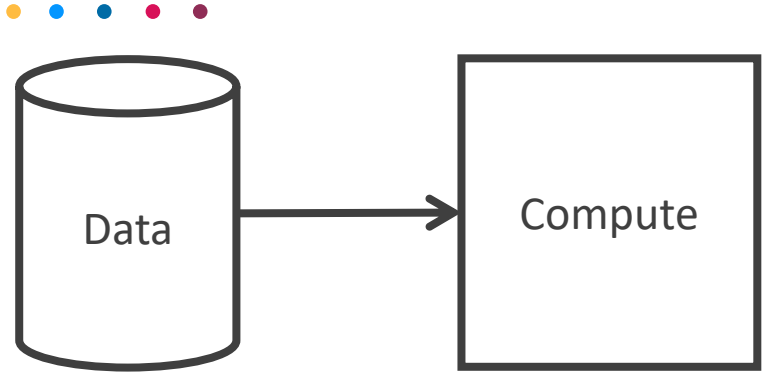
How most systems store ML data today:



MLWeaving:

1st row A

MLWeaving Memory Layout



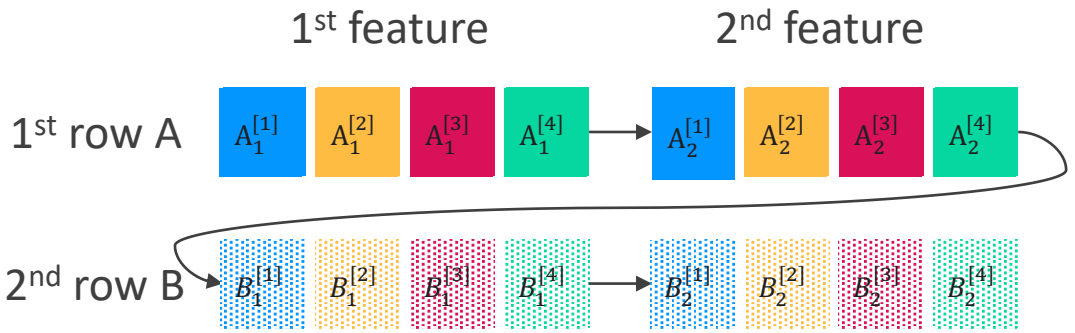
Observation 1:
Often memory bandwidth bound

Observation 2: Low precision (e.g., 8 bit fixed point) often provides reasonable quality

Observation 3: Different training task might need different precision level even on the same dataset

Can we store the data in a new data structure that efficiently supports arbitrary precision data movement?

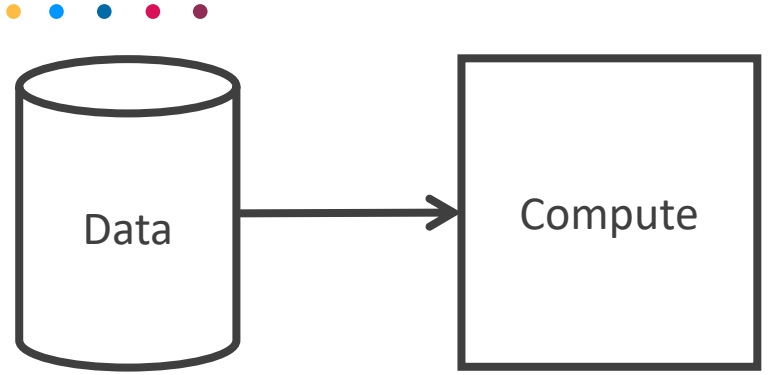
How most systems store ML data today:



MLWeaving:

1st row A

MLWeaving Memory Layout



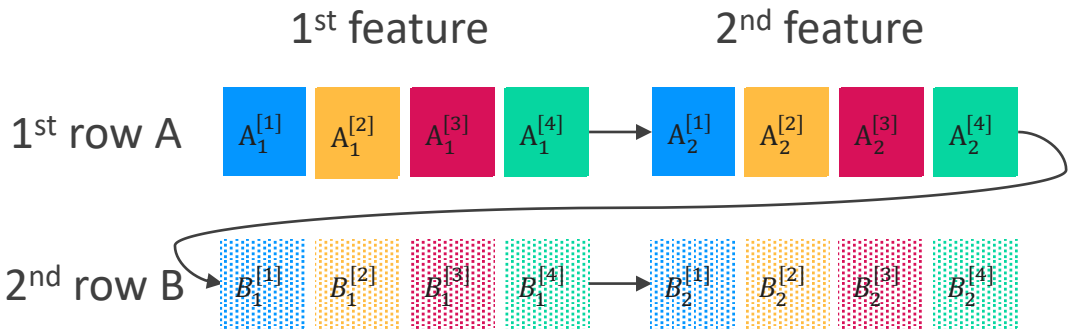
Observation 1:
Often memory bandwidth bound

Observation 2: Low precision (e.g., 8 bit fixed point) often provides reasonable quality

Observation 3: Different training task might need different precision level even on the same dataset

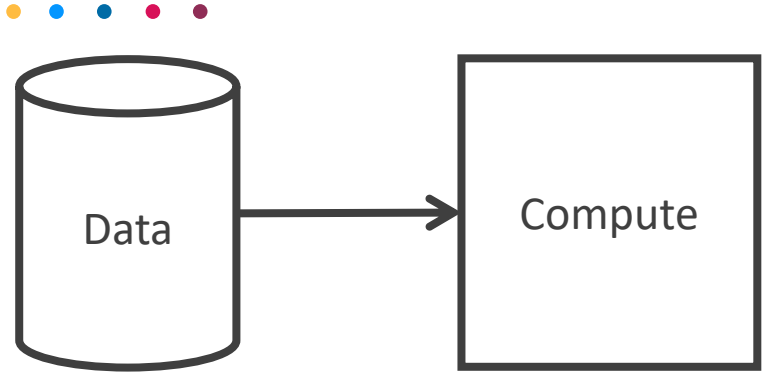
Can we store the data in a new data structure that supports arbitrary precision data movement?

How most systems store ML data today:



MLWeaving:

MLWeaving Memory Layout



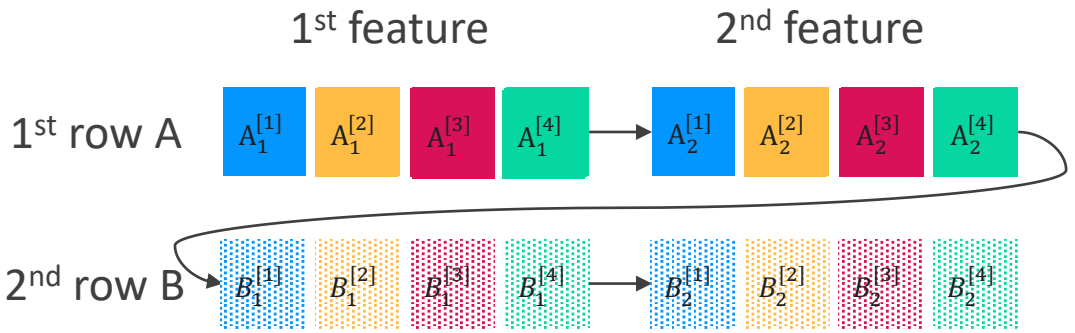
Observation 1:
Often memory bandwidth bound

Observation 2: Low precision (e.g., 8 bit fixed point) often provides reasonable quality

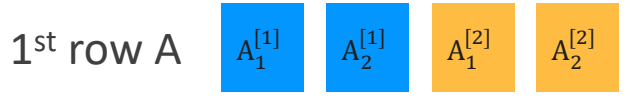
Observation 3: Different training task might need different precision level even on the same dataset

Can we store the data in a new data structure that supports arbitrary precision data movement?

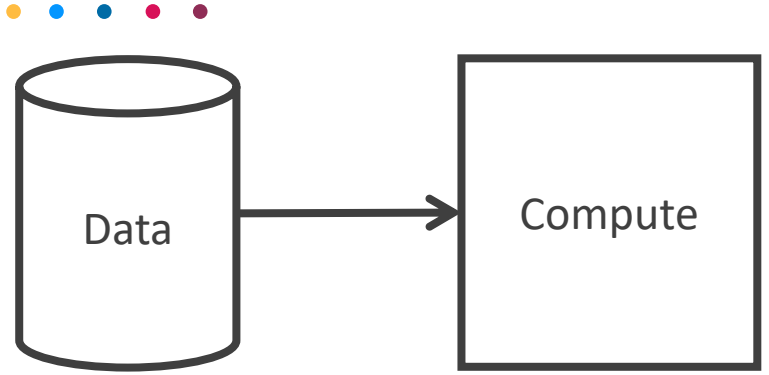
How most systems store ML data today:



MLWeaving:



MLWeaving Memory Layout



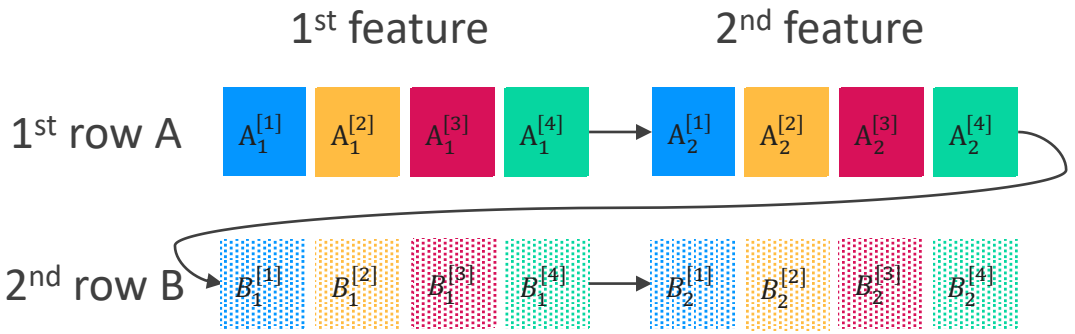
Observation 1:
Often memory bandwidth bound

Observation 2: Low precision (e.g., 8 bit fixed point) often provides reasonable quality

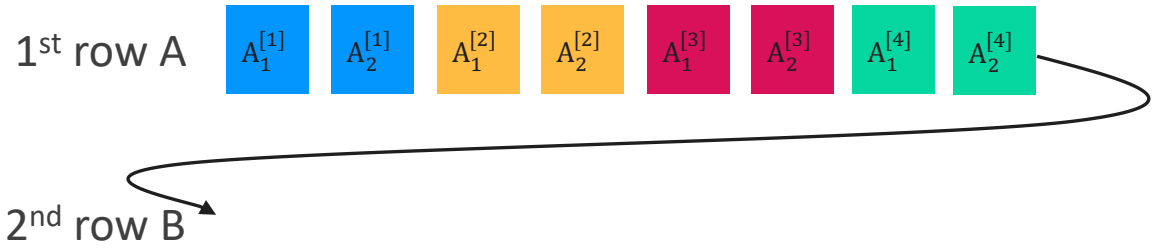
Observation 3: Different training task might need different precision level even on the same dataset

Can we store the data in a new data structure that supports arbitrary precision data movement?

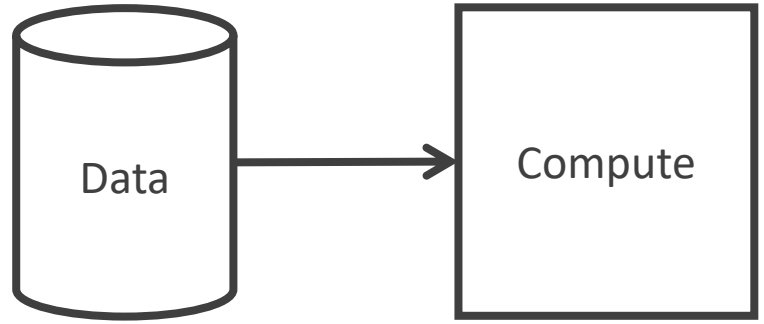
How most systems store ML data today:



MLWeaving:



MLWeaving Memory Layout



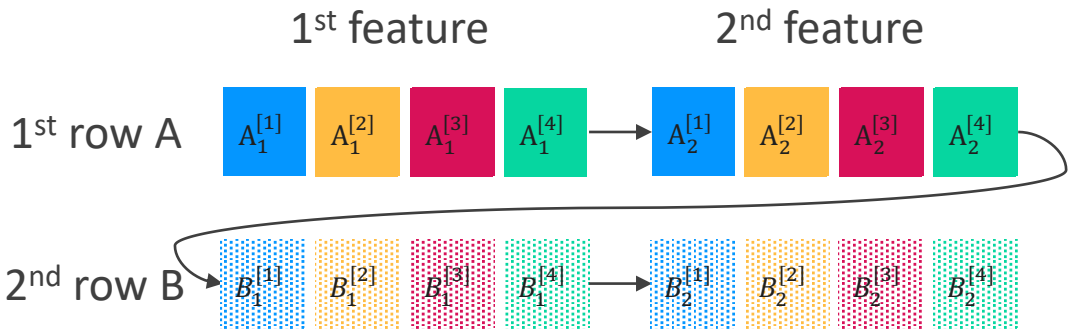
Observation 1:
Often memory bandwidth bound

Observation 2: Low precision (e.g., 8 bit fixed point) often provides reasonable quality

Observation 3: Different training task might need different precision level even on the same dataset

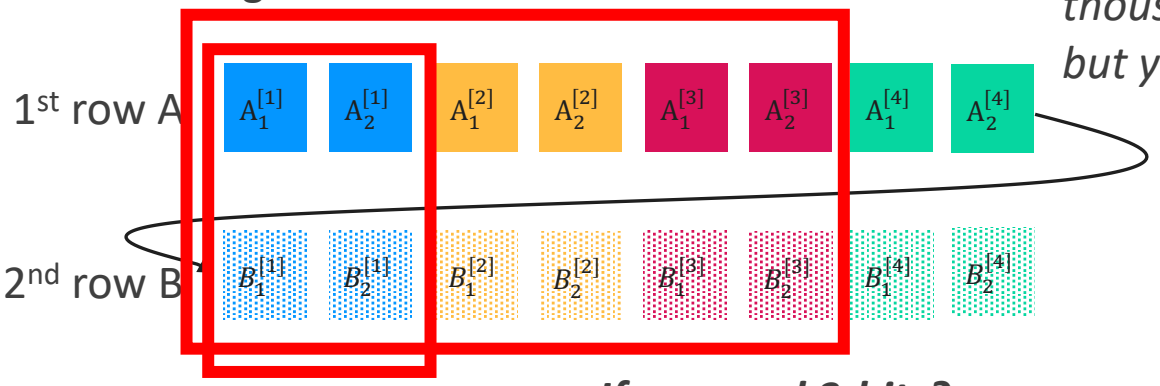
Can we store the data in a new data structure that supports arbitrary precision data movement?

How most systems store ML data today:



More complicated when a row has thousands of features, but you get the idea.

MLWeaving:



If we need 1-bit?

If we need 3-bits?

MLWeaving does not work out on CPUs*. CPU does not have custom instruction for MLWeaving memory layout and then we have to group bits from different memory locations before the further computing.*

Outline

Quick Background (5mins)

Stochastic Gradient Descent (SGD)

Synchronous vs. Asynchronous

Low Precision

MLWeaving (10mins)

Arbitrary-precision Training

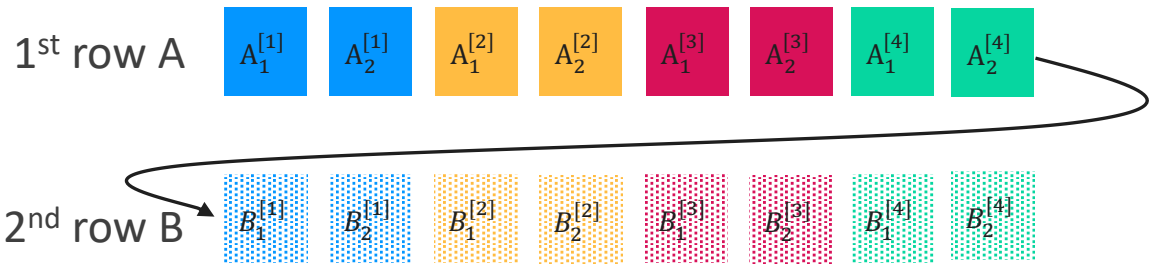
MLWeaving Memory Layout

MLWeaving Hardware Design

Efficient Synchronous Design

MLWeaving Hardware Design: Key Idea

MLWeaving memory layout:



Key idea of MLWeaving hardware design:

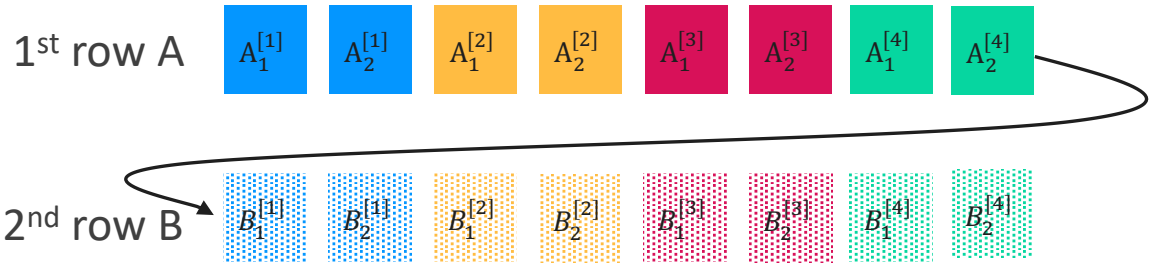
To use bit-serial multiplier to enable efficient data processing from the MLWeaving memory layout.

The modern CPU does not support bit-serial multiplier.

Please read our paper about how bit-serial multiplier works.

Custom Computation for MLWeaving

MLWeaving memory layout:

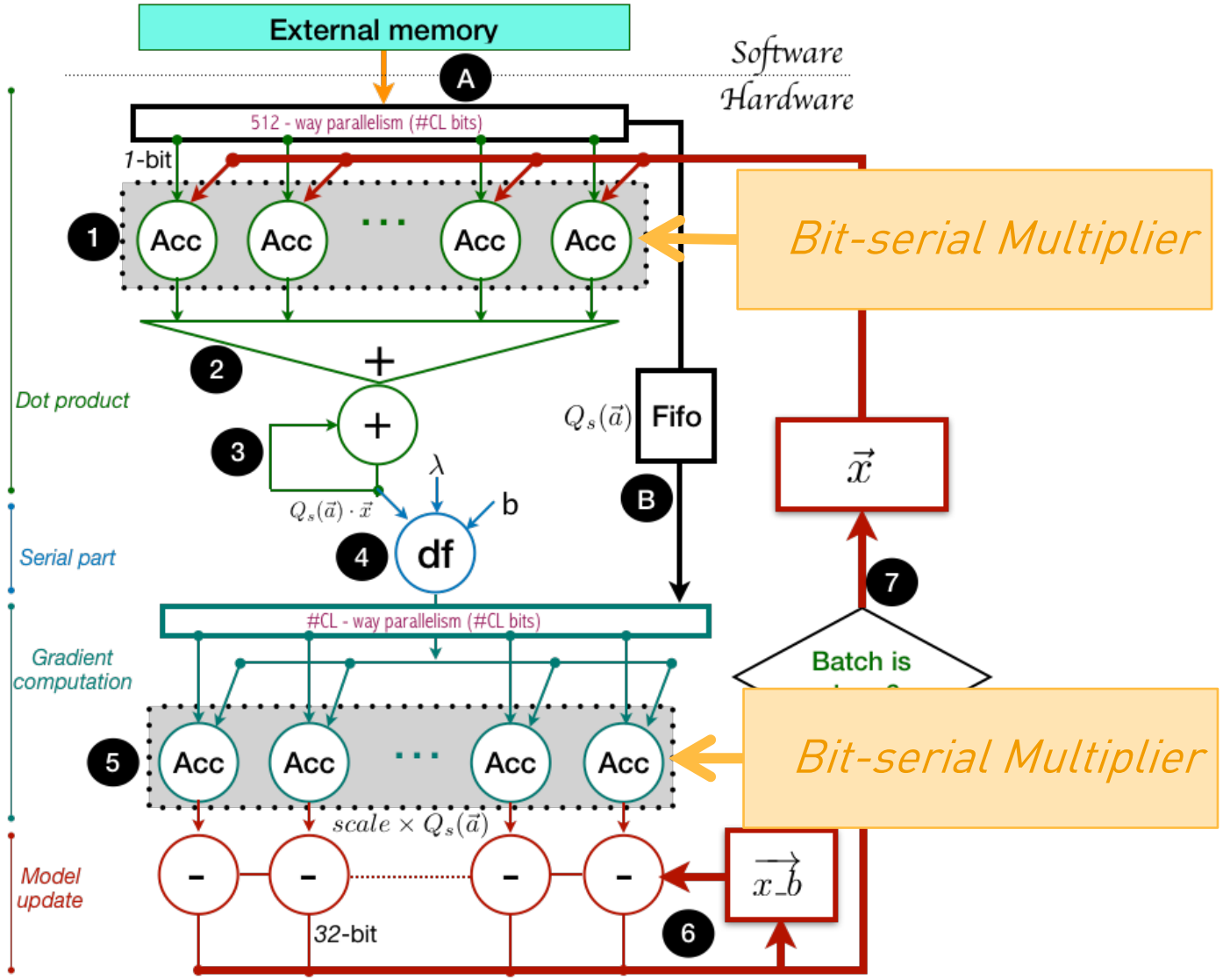


Bit-serial | Bit-parallel

Dot product: $A_r * x$

Gradient: $A_r * (A_r * x - br)$

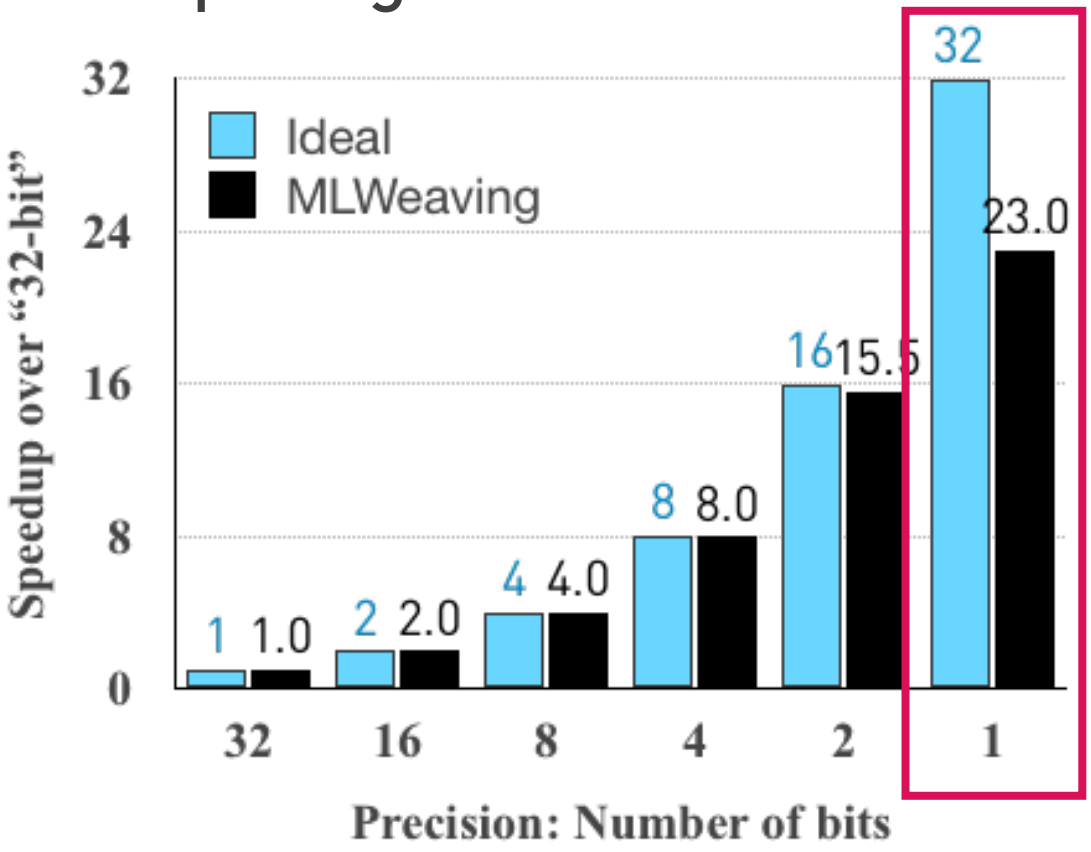
MLWeaving hardware design:



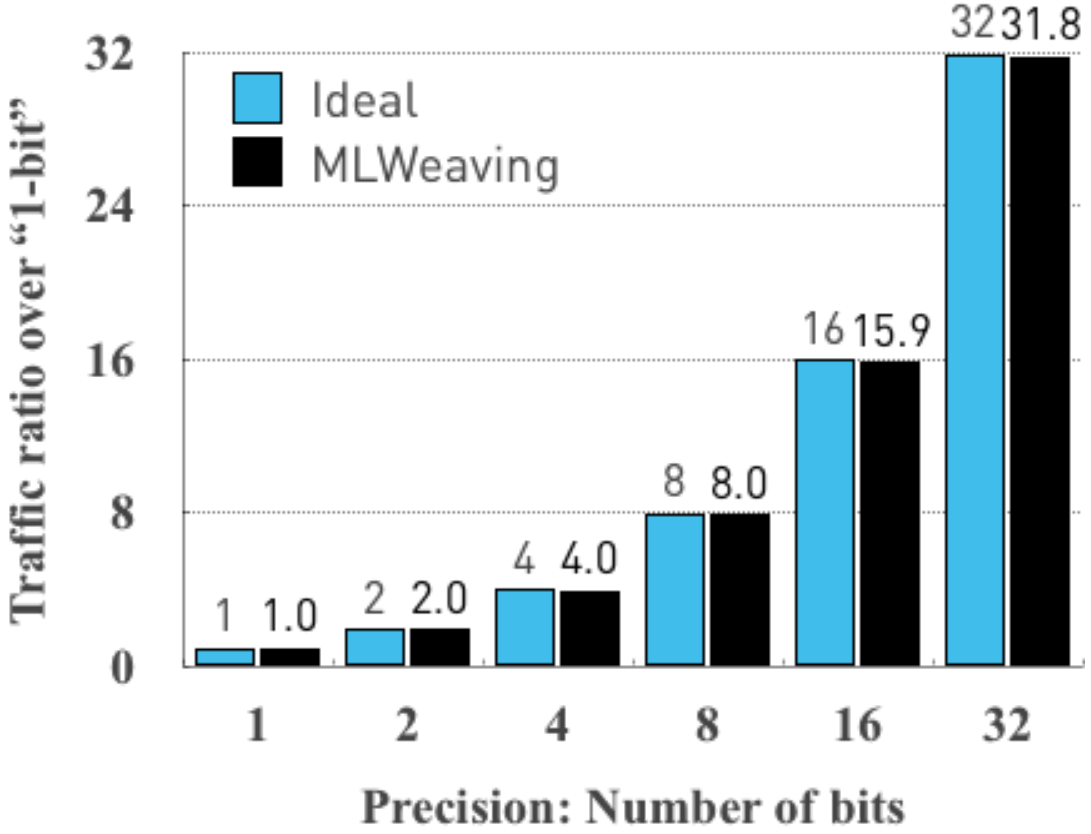
Bit-serial multiplier + MLWeaving memory layout enable any-precision ML training.

MLWeaving's Performance: Almost Linear Speedup with Lower Precision

Computing time vs. Precision



Memory traffic vs. Precision



Outline

Quick Background (5mins)

Stochastic Gradient Descent (SGD)

Synchronous vs. Asynchronous

Low Precision

MLWeaving (10mins)

Arbitrary-precision Training

MLWeaving Memory Layout

MLWeaving Hardware Design

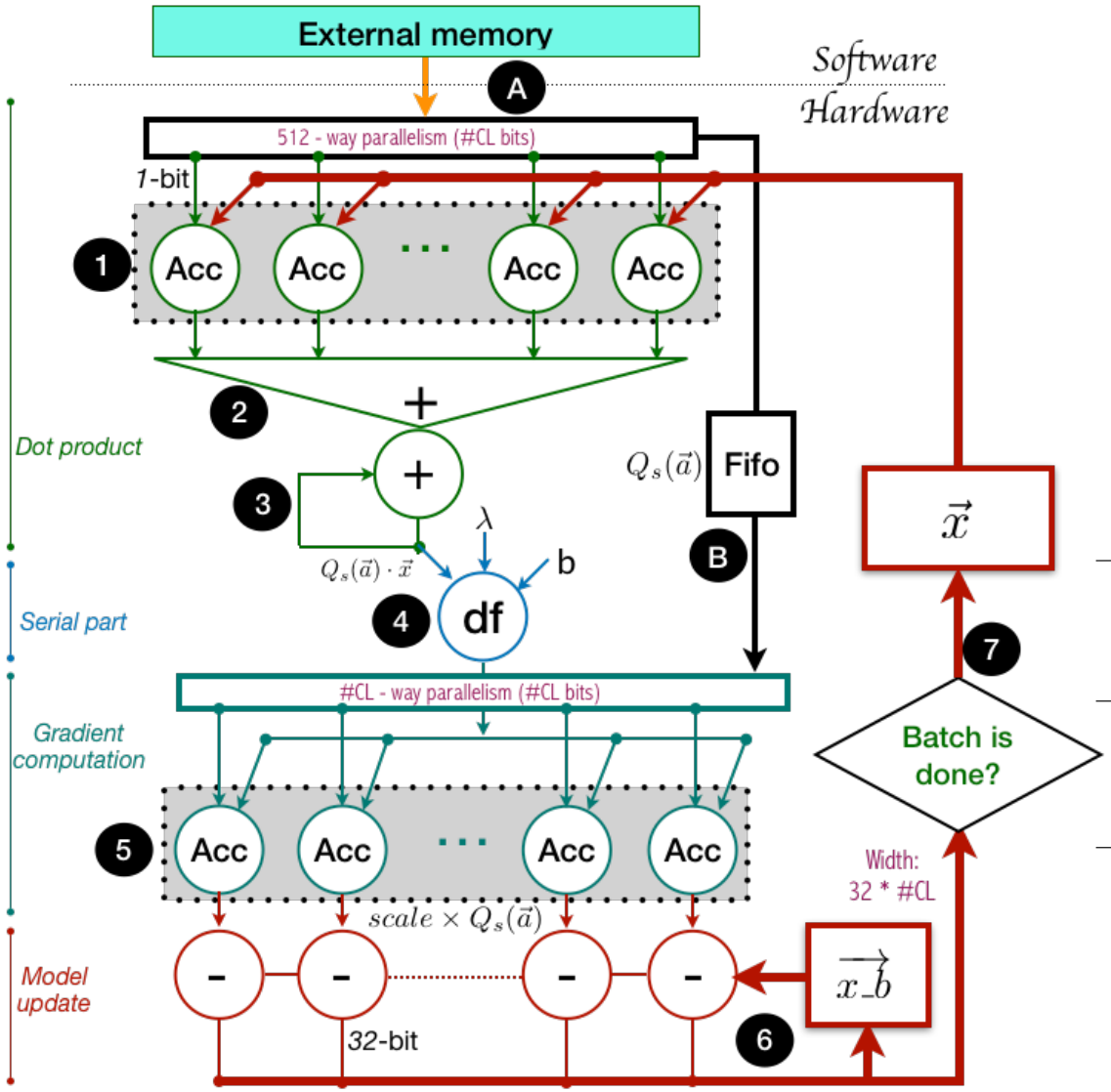
Efficient Synchronous Design

Synchronous SGD or asynchronous SGD on custom hardware?

SGD on Custom Hardware: The Best of Two Worlds

	Hardware Efficiency (Throughput)	Statistical Efficiency (Convergence Rate)
Single-core (Synchrony)	Low 	High 
Multi-core (Asynchrony)	High 	Low 
Custom hardware (Synchrony)	High 	High 

Original Synchronous Implementation: Compute-Bound



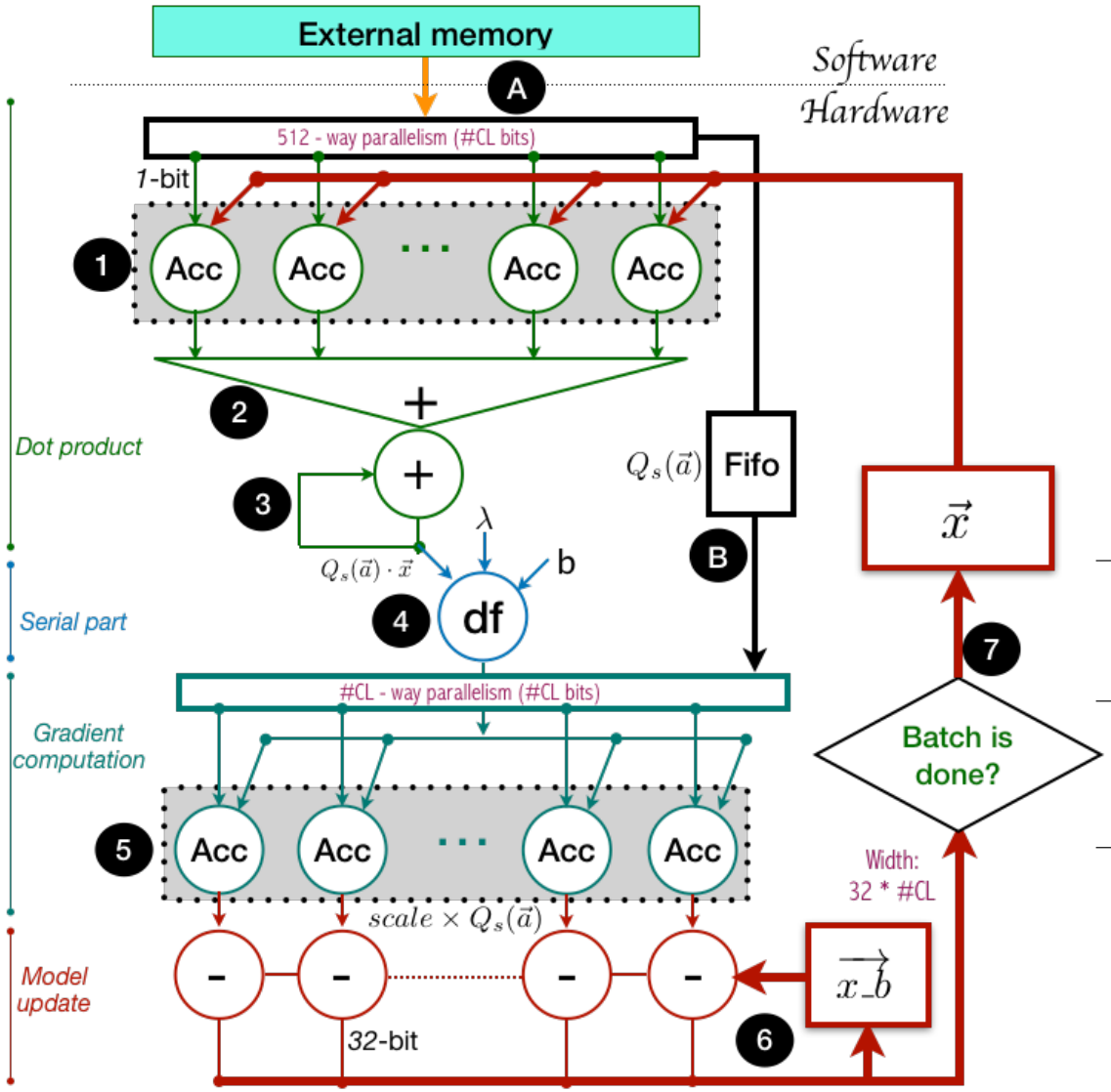
***Key idea**: to keep the RAW dependency, regarding the model x.*

Original Implementation

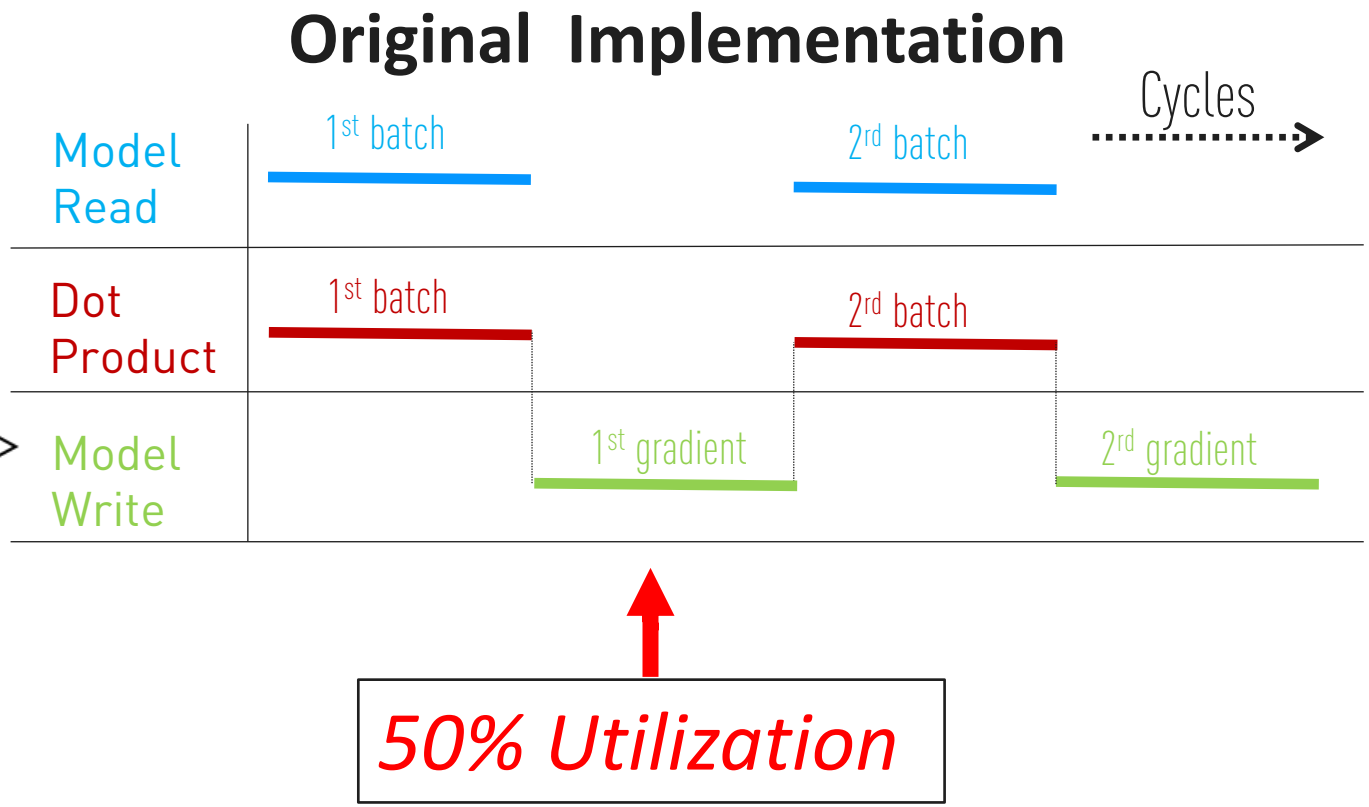
Cycles →

Model Read	
Dot Product	
Model Write	

Original Synchronous Implementation: Compute-Bound



***Key idea**: to keep the RAW dependency, regarding the model x.*

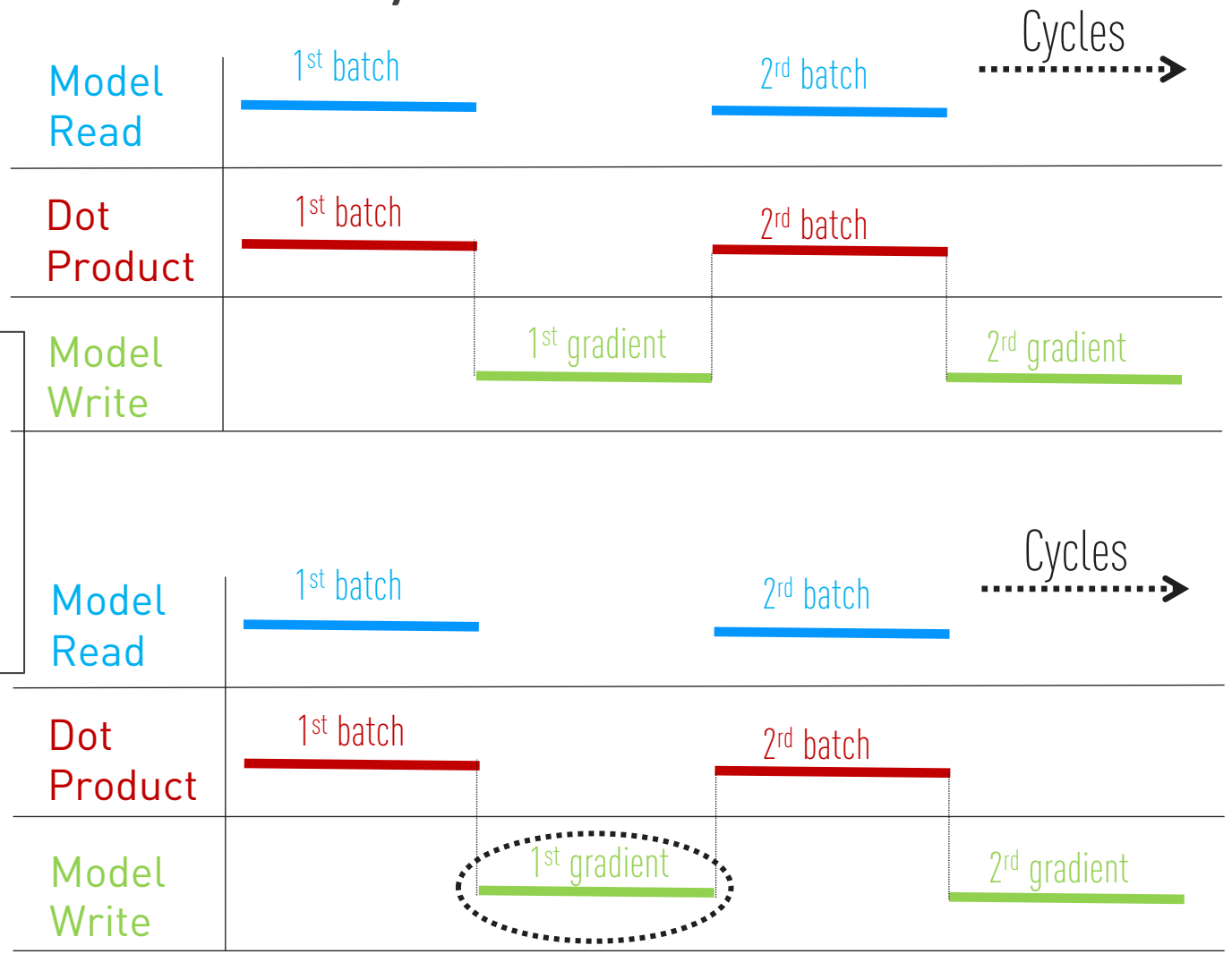


Optimal Synchronous Implementation: Memory-Bound

Original: Compute-bound

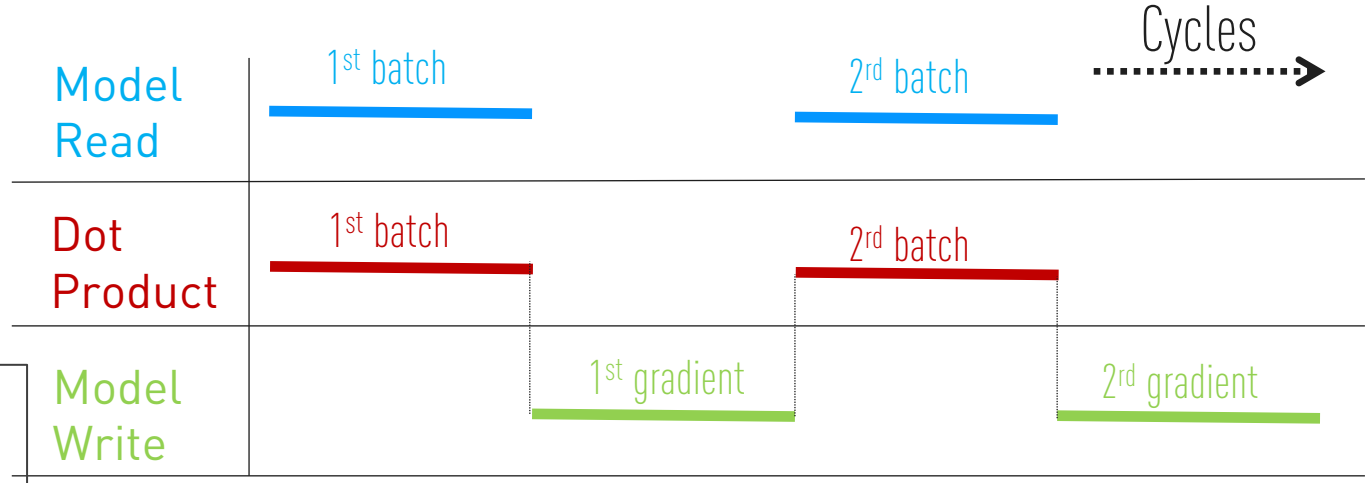
Observation: Custom hardware can update the model (thousands of weights) at the granularity level: 64 weights, not the whole model.

With Chaining: Memory-bound



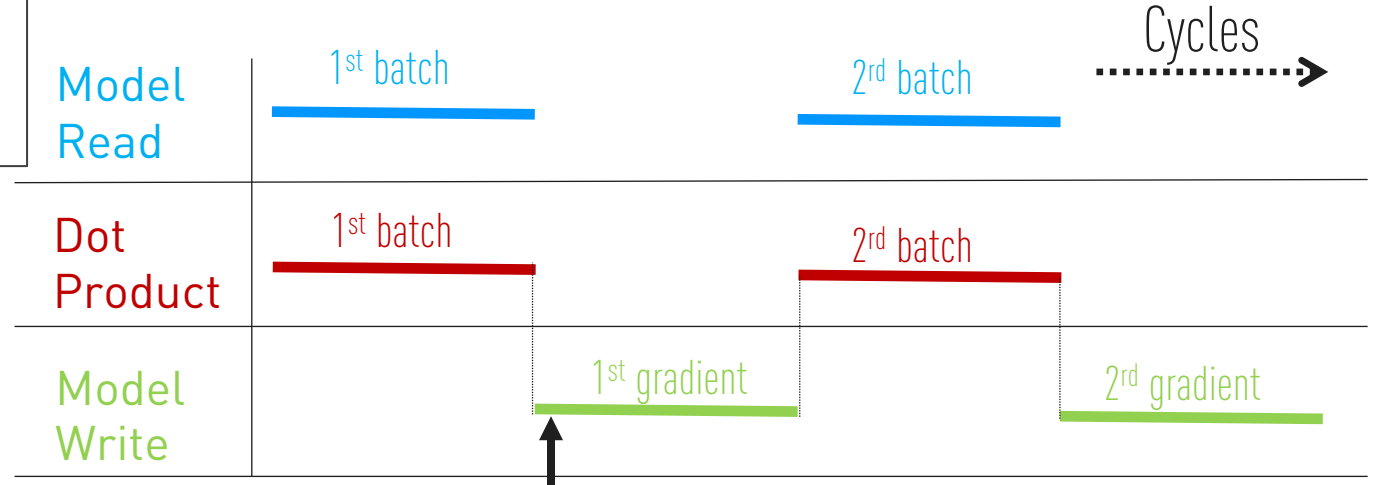
Optimal Synchronous Implementation: Memory-Bound

Original: Compute-bound



Observation: Custom hardware can update the model (thousands of weights) at the granularity level: 64 weights, not the whole model.

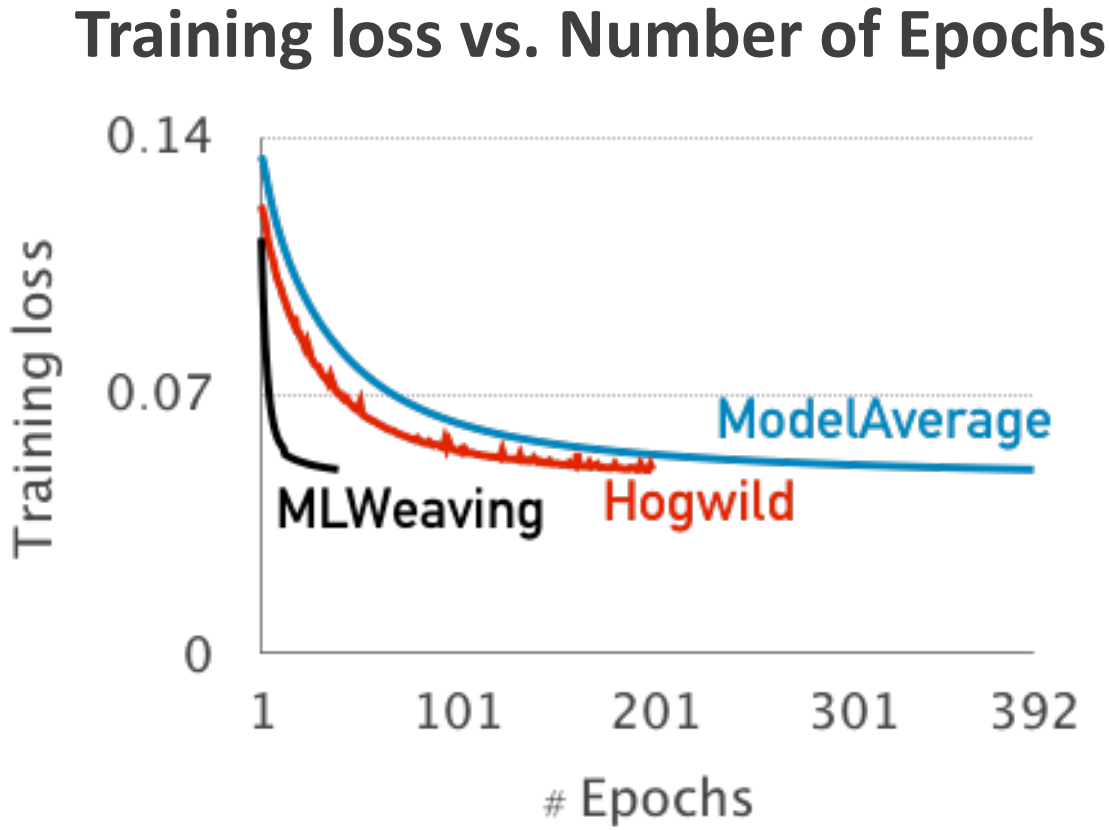
With Chaining: Memory-bound



High throughput: "sync" is as fast as "async".

***Gap**: gradient from 64 weights*

Effect of Sync. Design



Our sync. design needs fewer epochs to converge.

ModelAverage *and* ***Hogwild*** on the multi-core CPU: **Async.**
MLWeaving on the *custom hardware* : **Sync.**

Outline

Quick Background (5mins)

Stochastic Gradient Descent (SGD)

Synchronous vs. Asynchronous

Low Precision

MLWeaving (10mins)

Arbitrary-precision Training

MLWeaving Memory Layout

MLWeaving Hardware Design

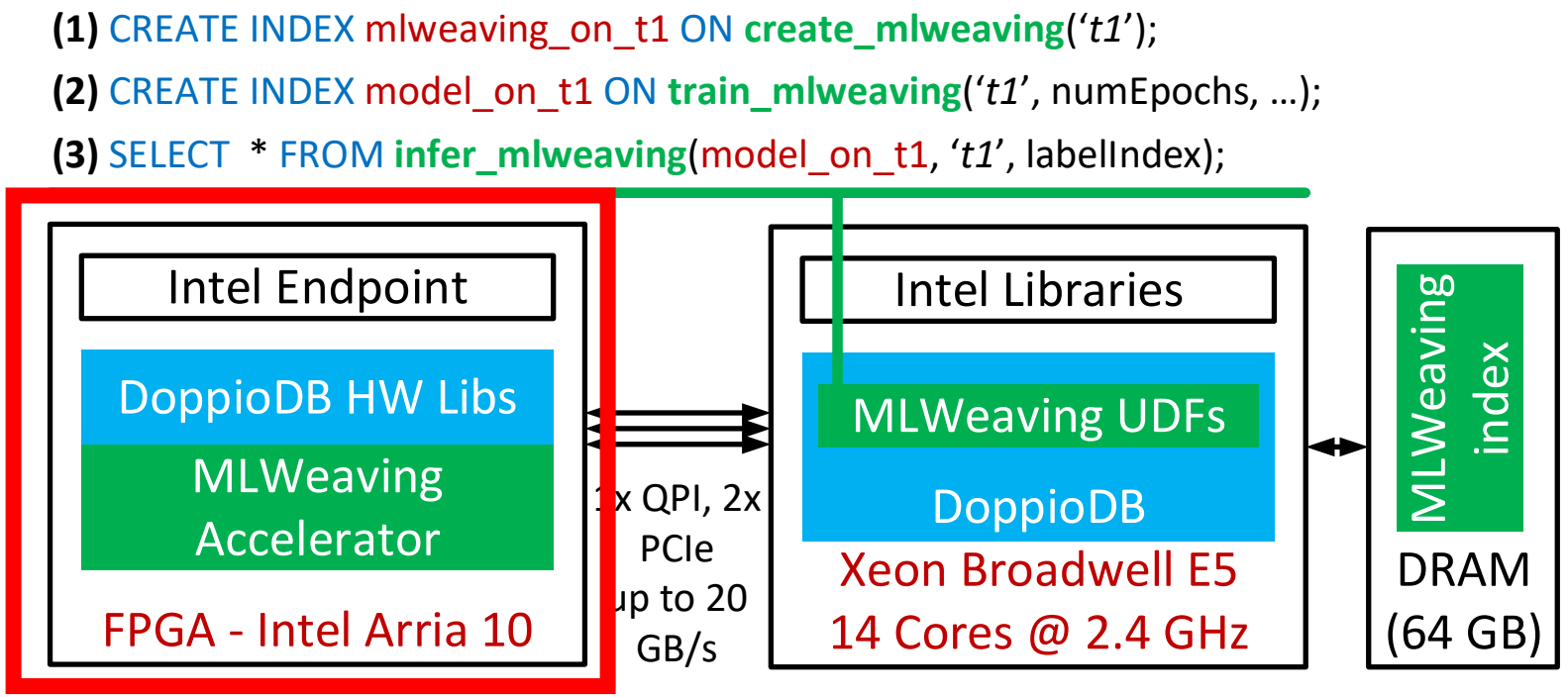
Efficient Synchronous Design

System Integration

Intel HARP2 Platform:

Intel Broadwell 14-core CPU
Intel Arria 10 FPGA

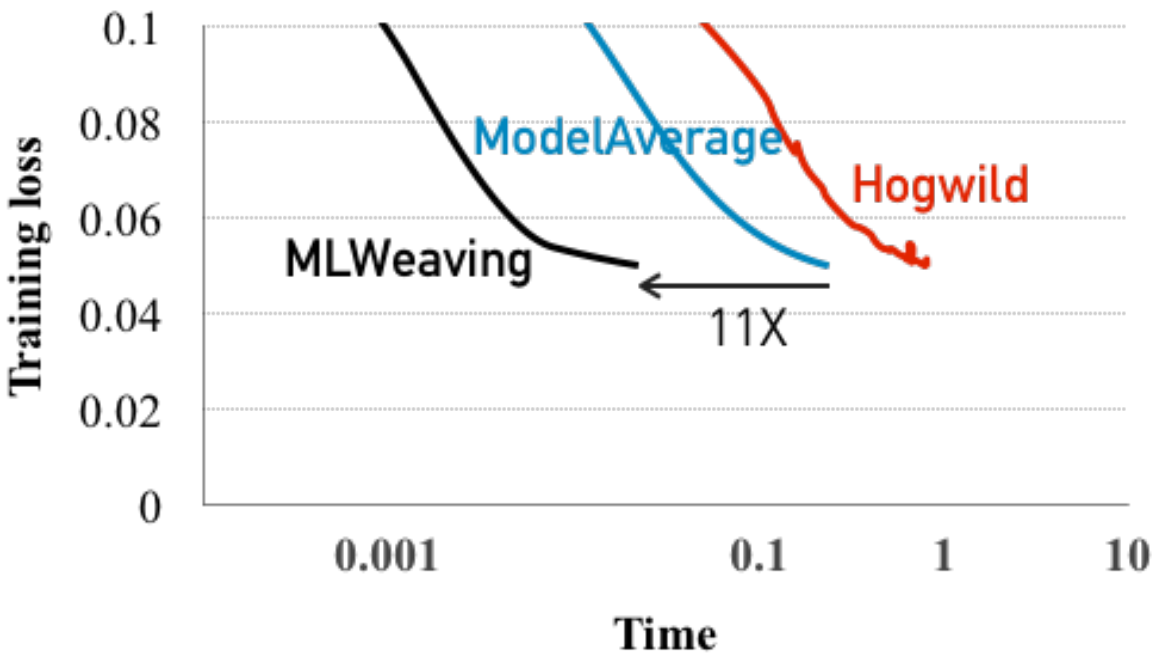
We integrate MLWeaving into DoppioDB that is an open source solution for FPGA-enhanced databases based on MonetDB.



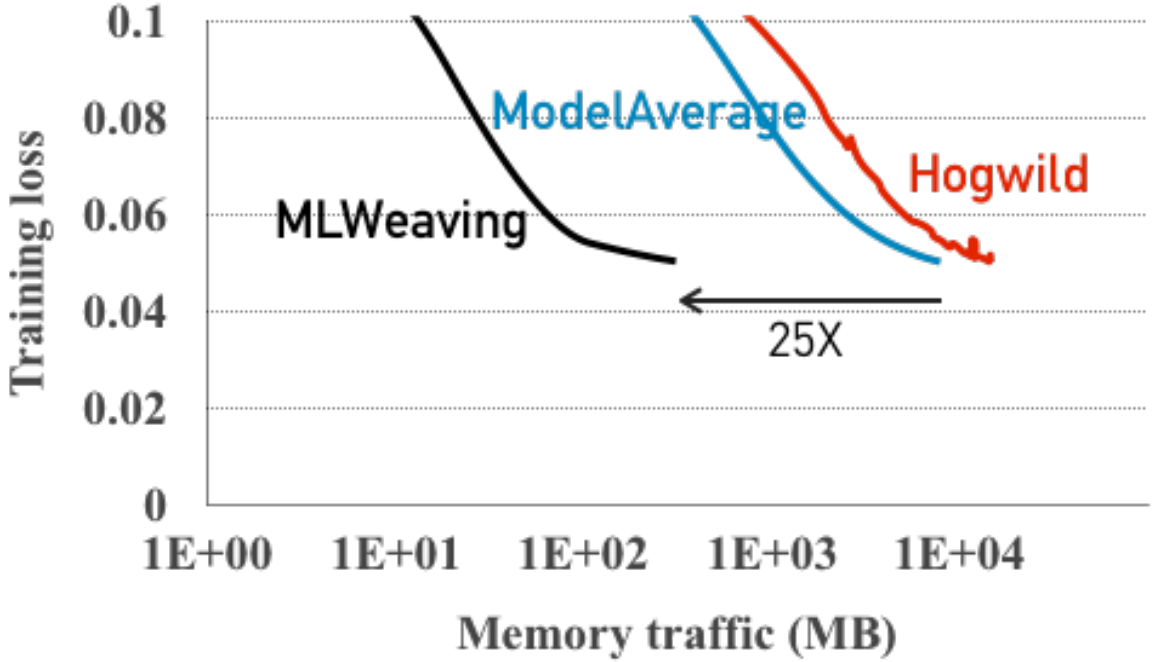
If you are interested to play with MLWeaving, please visit our demo DoppioDB 2.0 in the Hollywood ballroom, 11:00-12:30, 27-29 Aug, 2019.

End-to-End Performance: MLWeaving

Training loss vs. Time



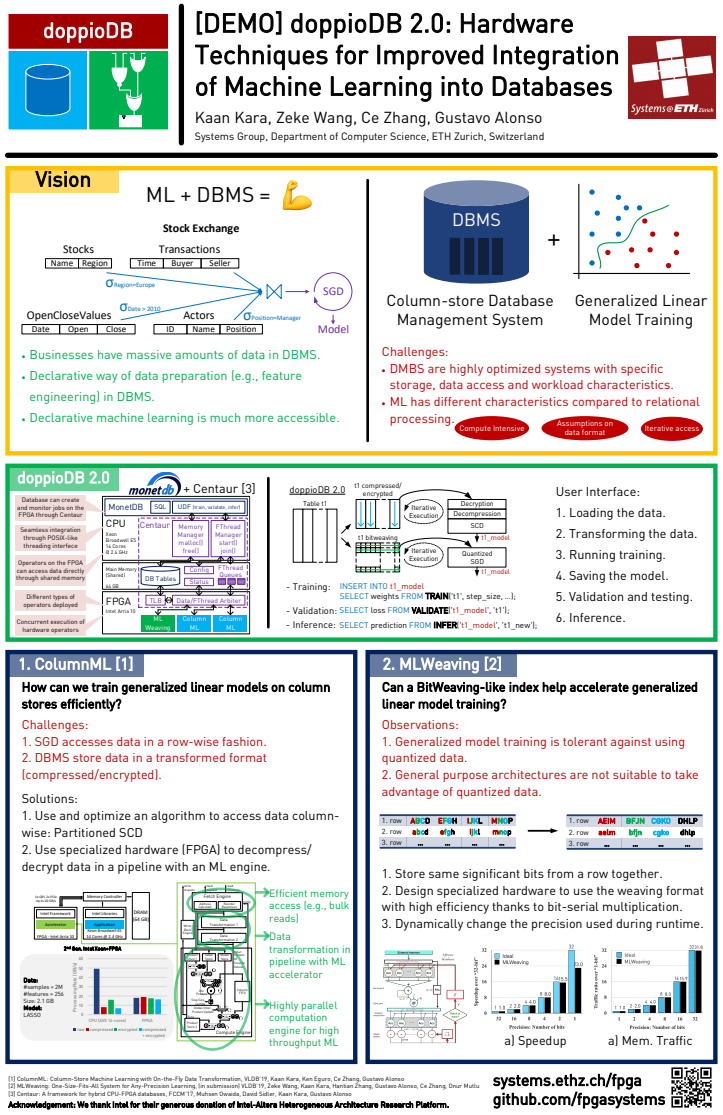
Training loss vs. Memory



ModelAverage ***and*** ***Hogwild*** **on an Intel CPU: 14 cores, AVX2-enhanced, 8-bit dataset.**

MLWeaving **on an FPGA: 3-bit dataset.**

If you are interested to play with MLWeaving, please visit our demo DoppioDB 2.0 in the Hollywood ballroom, 11:00-12:30, 27-29 Aug, 2019.



Any Questions?

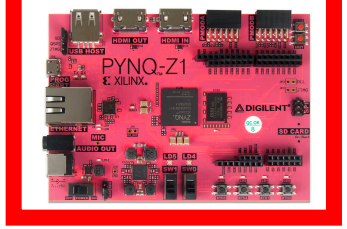
Motivations

Database

Generalized Linear Model

FPGA

Linear Model, Logistic Regression, SVM



?

For GLM training, can we enable things that cannot be well done on CPUs/GPUs?

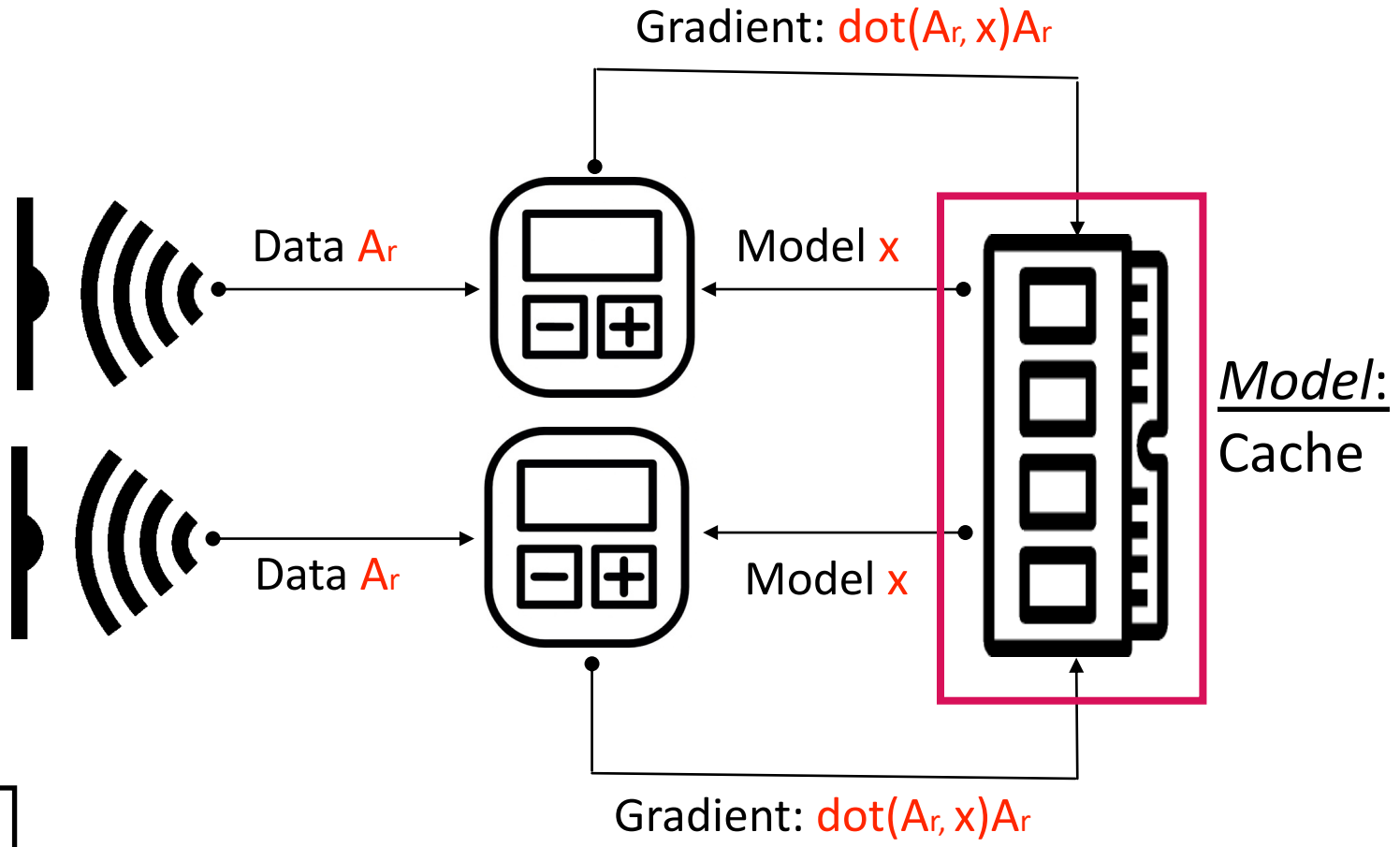
!

Any-precision Training

High-throughput Sync. Design

Hogwild: Asynchrony

```
A_r = get_data()

x = get_model()

g = comp_grad(x,A_r)

x = x - g

set_model(x)
```



Shared model x among cores

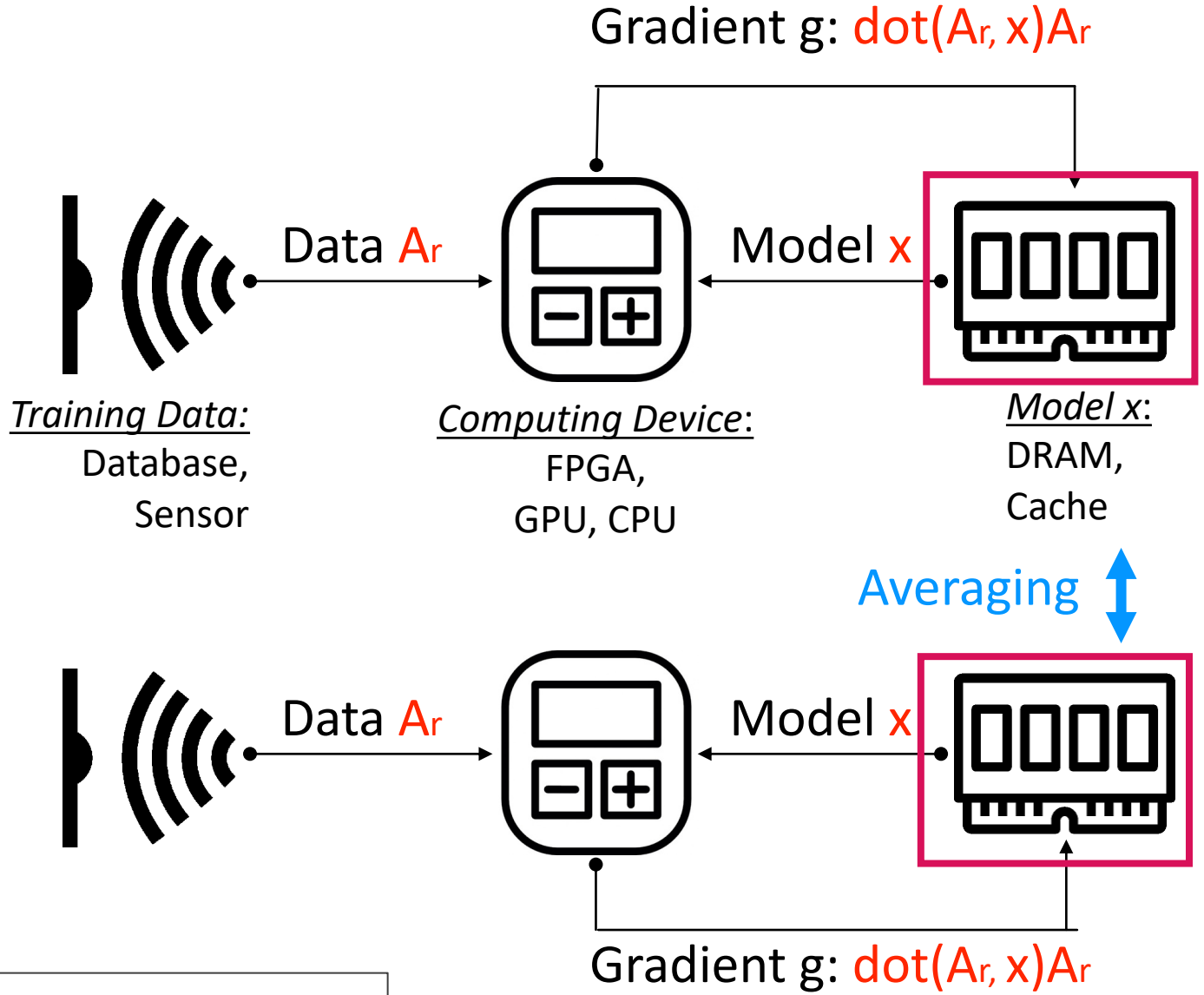
Problem? *Cache-coherence is expensive, especially for dense data!*

ModelAverage: Asynchrony

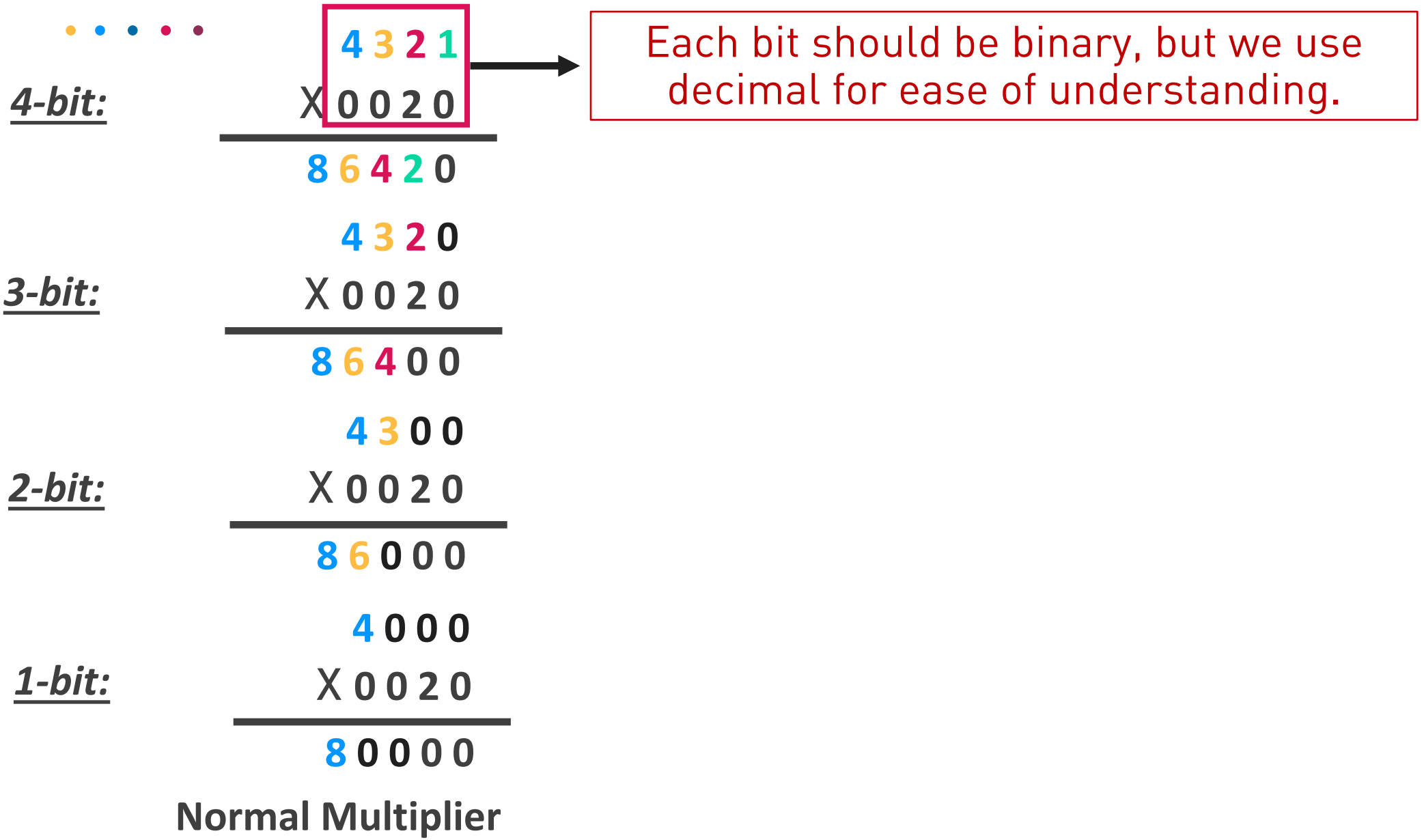
```
A_r = get_data()

x = get_model()

g = comp_grad(x, A_r)

x = x - g

set_model(x)
```

A copy of model x for each core

Problem? *Convergence might be slower.*



How Bit-serial Multiplier Deals with Low Precision?



How Bit-serial Multiplier Deals with Low Precision?

4-bit:

4 3 2 1
X 0 0 2 0
---
8 6 4 2 0

3-bit:

4 3 2 0
X 0 0 2 0
---
8 6 4 0 0

2-bit:

4 3 0 0
X 0 0 2 0
---
8 6 0 0 0

1-bit:

4 0 0 0
X 0 0 2 0
---
8 0 0 0 0

Normal Multiplier

Initialization:

4 3 2 1

X 0020 **BSM**

Sum = 0 0 0 0 0

Bit-serial Multiplier (BSM)

How Bit-serial Multiplier Deals with Low Precision?

4-bit:

4 3 2 1
X 0 0 2 0
8 6 4 2 0

3-bit:

4 3 2 0
X 0 0 2 0
8 6 4 0 0

2-bit:

4 3 0 0
X 0 0 2 0
8 6 0 0 0

1-bit:

4 0 0 0
X 0 0 2 0
8 0 0 0 0

Normal Multiplier

Initialization:

4 3 2 1 → Bit-Serial (S)

Bit-Parallel (P) → X 0020

BSM → Sum += P * $S^{[i]}$

Sum = 0 0 0 0 0

Bit-serial Multiplier (BSM)

Bit-serial Multiplier: 1-Bit Precision

4 3 2 1

4-bit: X 0 0 2 0

8 6 4 2 0

4 3 2 0

3-bit: X 0 0 2 0

8 6 4 0 0

4 3 0 0

2-bit: X 0 0 2 0

8 6 0 0 0

4 0 0 0

1-bit: X 0 0 2 0

8 0 0 0 0

Normal Multiplier

1st Cycle:

4 3 2 1

Memory

Hardware

X 0020 **BSM**

Sum = 0 0 0 0 0

Bit-serial Multiplier (BSM)

Bit-serial Multiplier: 1-Bit Precision

4 3 2 1

4-bit: X 0 0 2 0

8 6 4 2 0

4 3 2 0

3-bit: X 0 0 2 0

8 6 4 0 0

4 3 0 0

2-bit: X 0 0 2 0

8 6 0 0 0

4 0 0 0

1-bit: X 0 0 2 0

8 0 0 0 0

Normal Multiplier

1st Cycle:

4 3 2 1

4 means 4000.

Memory

Hardware

X 0020 **BSM** (4)

Sum += 20 * 4000

Sum = 0 0 0 0 0

Bit-serial Multiplier (BSM)

Bit-serial Multiplier: 1-Bit Precision

4 3 2 1

4-bit: X 0 0 2 0

8 6 4 2 0

4 3 2 0

3-bit: X 0 0 2 0

8 6 4 0 0

4 3 0 0

2-bit: X 0 0 2 0

8 6 0 0 0

4 0 0 0

1-bit: X 0 0 2 0

8 0 0 0 0

Normal Multiplier

1st Cycle:

4 3 2 1

Memory

Hardware

X 0020 **BSM**

Sum = 8 0 0 0 0

Done with 1-bit precision, or proceed to the next bit.

Bit-serial Multiplier (BSM)

Bit-serial Multiplier: 2-Bit Precision

4 3 2 1

4-bit: X 0 0 2 0

8 6 4 2 0

4 3 2 0

3-bit: X 0 0 2 0

8 6 4 0 0

4 3 0 0

2-bit: X 0 0 2 0

8 6 0 0 0

4 0 0 0

1-bit: X 0 0 2 0

8 0 0 0 0

Normal Multiplier

2nd Cycle:

4 3 2 1

X 0020 **BSM**

Sum = 8 0 0 0 0

Memory

Hardware

Bit-serial Multiplier (BSM)

Bit-serial Multiplier: 2-Bit Precision

4-bit:

```
  4 3 2 1
X 0 0 2 0
---------
  8 6 4 2 0
```

3-bit:

```
  4 3 2 0
X 0 0 2 0
---------
  8 6 4 0 0
```

2-bit:

```
  4 3 0 0
X 0 0 2 0
---------
  8 6 0 0 0
```

1-bit:

```
  4 0 0 0
X 0 0 2 0
---------
  8 0 0 0 0
```

Normal Multiplier

2nd Cycle:

4 3 2 1

3 means 300.

Memory

Hardware

```
X 0020   3
        BSM
```

Sum += 20 * 300

Sum = 8 0 0 0 0

Bit-serial Multiplier (BSM)

Bit-serial Multiplier: 2-Bit Precision

4-bit:

```
  4 3 2 1
X 0 0 2 0
---------
  8 6 4 2 0
```

3-bit:

```
  4 3 2 0
X 0 0 2 0
---------
  8 6 4 0 0
```

2-bit:

```
  4 3 0 0
X 0 0 2 0
---------
  8 6 0 0 0
```

1-bit:

```
  4 0 0 0
X 0 0 2 0
---------
  8 0 0 0 0
```

Normal Multiplier

2nd Cycle:

4 3 2 1

Memory

Hardware

X 0020 **BSM**

Sum = 8 6 0 0 0

Done with 2-bit precision, or proceed to the next bit.

Bit-serial Multiplier (BSM)

Bit-serial Multiplier: 3-Bit Precision

4-bit:

4 3 2 1
X 0 0 2 0
8 6 4 2 0

3-bit:

4 3 2 0
X 0 0 2 0
8 6 4 0 0

2-bit:

4 3 0 0
X 0 0 2 0
8 6 0 0 0

1-bit:

4 0 0 0
X 0 0 2 0
8 0 0 0 0

Normal Multiplier

3th Cycle:

4 3 2 1

Memory

Hardware

X 0020 **BSM**

Sum = 8 6 0 0 0

Bit-serial Multiplier (BSM)

Bit-serial Multiplier: 3-Bit Precision

4-bit:

4 3 2 1
X 0 0 2 0
8 6 4 2 0

3-bit:

4 3 2 0
X 0 0 2 0
8 6 4 0 0

2-bit:

4 3 0 0
X 0 0 2 0
8 6 0 0 0

1-bit:

4 0 0 0
X 0 0 2 0
8 0 0 0 0

Normal Multiplier

3th Cycle:

4 3 2 1

2 means 20.

Memory

Hardware

X 0020 **BSM** (2) Sum += 20 * 20

Sum = 8 6 0 0 0

Bit-serial Multiplier (BSM)

Bit-serial Multiplier: 3-Bit Precision

4-bit:

4 3 2 1
X 0 0 2 0
8 6 4 2 0

3-bit:

4 3 2 0
X 0 0 2 0
8 6 4 0 0

2-bit:

4 3 0 0
X 0 0 2 0
8 6 0 0 0

1-bit:

4 0 0 0
X 0 0 2 0
8 0 0 0 0

Normal Multiplier

3th Cycle:

4 3 2 1

X 0020 **BSM**

Sum = 8 6 4 0 0

Memory

Hardware

Done with 3-bit precision, or proceed to the next bit.

Bit-serial Multiplier (BSM)

Bit-serial Multiplier: 4-Bit Precision

4-bit:

4 3 2 1
X 0 0 2 0
8 6 4 2 0

3-bit:

4 3 2 0
X 0 0 2 0
8 6 4 0 0

2-bit:

4 3 0 0
X 0 0 2 0
8 6 0 0 0

1-bit:

4 0 0 0
X 0 0 2 0
8 0 0 0 0

Normal Multiplier

4th Cycle:

4 3 2 1

Memory

Hardware

X 0020 **BSM**

Sum = 8 6 4 0 0

Bit-serial Multiplier (BSM)

Bit-serial Multiplier: 4-Bit Precision

4 3 2 1

4-bit: X 0 0 2 0

8 6 4 2 0

4 3 2 0

3-bit: X 0 0 2 0

8 6 4 0 0

4 3 0 0

2-bit: X 0 0 2 0

8 6 0 0 0

4 0 0 0

1-bit: X 0 0 2 0

8 0 0 0 0

Normal Multiplier

4th Cycle:

4 3 2 1

1 means 1.

Memory

Hardware

X 0020 **BSM** (1) Sum += 20 * 1

Sum = 8 6 4 0 0

Bit-serial Multiplier (BSM)

Bit-serial Multiplier: 4-Bit Precision

4-bit:

```
  4 3 2 1
X 0 0 2 0
---------
  8 6 4 2 0
```

3-bit:

```
  4 3 2 0
X 0 0 2 0
---------
  8 6 4 0 0
```

2-bit:

```
  4 3 0 0
X 0 0 2 0
---------
  8 6 0 0 0
```

1-bit:

```
  4 0 0 0
X 0 0 2 0
---------
  8 0 0 0 0
```

Normal Multiplier

4th Cycle:

4 3 2 1

Memory

Hardware

X 0020 **BSM**

Sum = 8 6 4 2 0

Done with 4-bit precision

Bit-serial Multiplier (BSM)