

Agenda

Time (CET)	Time (ET)	Topic	Presenter
15:00 – 16:00	9:00 – 10:00	Introduction to Distributed Deep Learning Training Platforms	Tushar Krishna
16:00 – 17:00	10:00 – 11:00	ASTRA-sim	Saeed Rashidi
17:00 – 17:10	11:00 – 11:10	Break	
17:10 – 17:50	11:10 – 11:50	Demo and Exercises	William Won and Taekyung Heo
17:50 – 18:00	11:50 – 12:00	Extensions and Future Development	Tushar Krishna and Saeed Rashidi

Tutorial Website

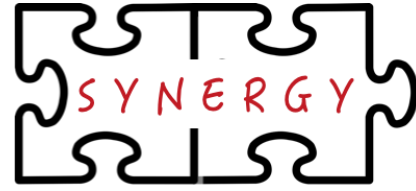
includes agenda, slides, ASTRA-sim installation instructions (via source + docker image)

<https://astra-sim.github.io/tutorials/asplos-2022>

Attention: Tutorial is being recorded



Georgia Tech School of Electrical and
Computer Engineering
College of Engineering



<http://synergy.ece.gatech.edu>



Exercise 4: Implementing a New Training Loop



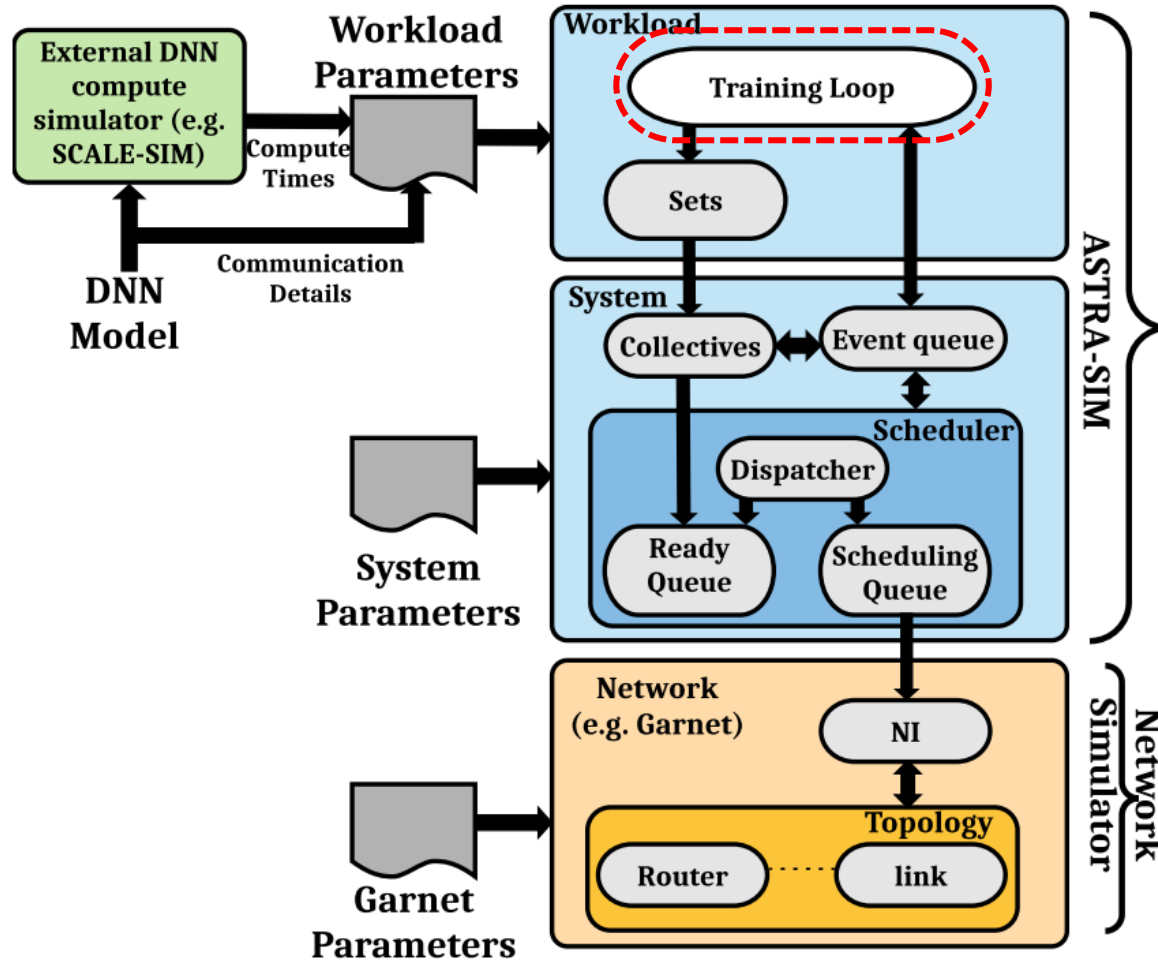
Taekyung Heo

Postdoctoral Fellow, School of ECE
Georgia Institute of Technology

tkheo@casys.kaist.ac.kr

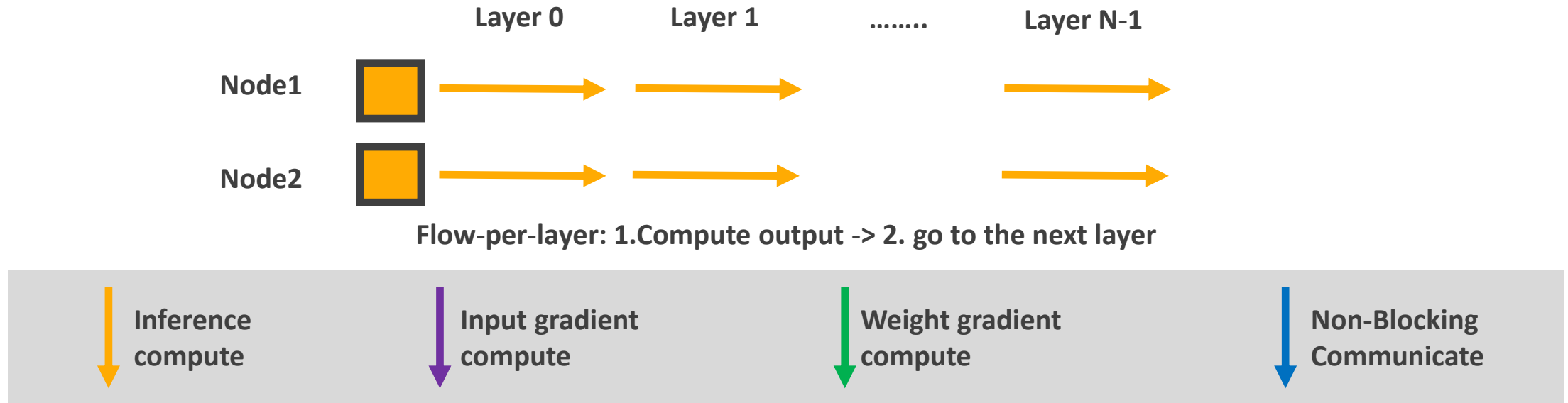
Acknowledgments: William Won (GT), Srinivas Sridharan (Facebook), Sudarshan Srinivasan (Intel)

Training Loops



- Training loop determines the behavior of a workload
 - Parallelization strategy
 - Computation order
 - Communication order
- Supported training loops
 - Data parallel ← Goal: tweak this loop
 - Model parallel
 - DLRM
 - Transformer
- You can implement a new training loop to support other models

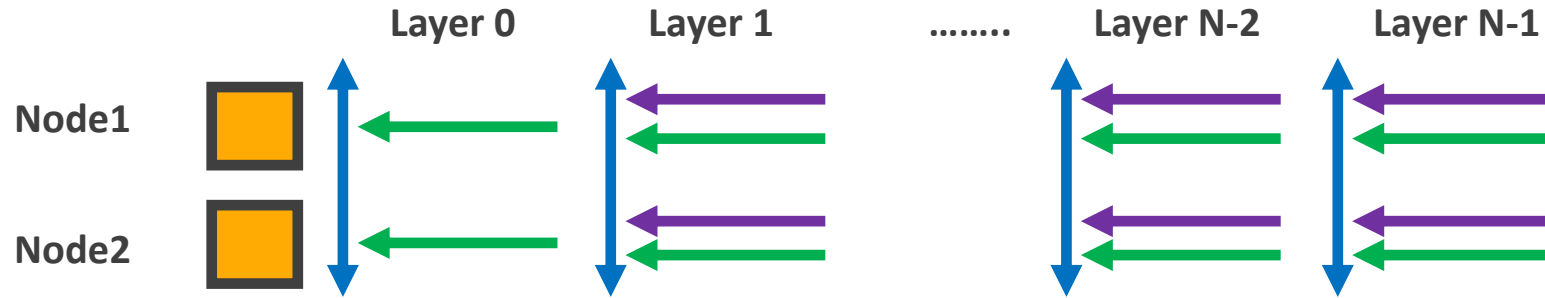
Vanilla Data-parallel Training Loop (FWD)



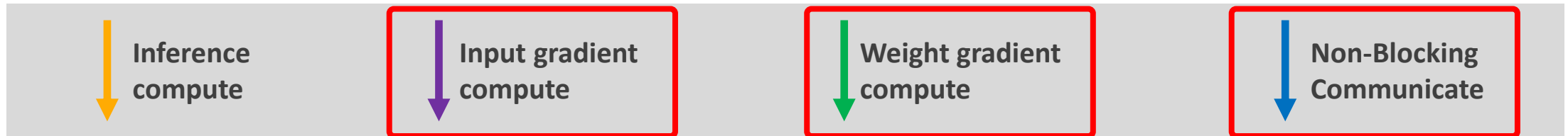
Vanilla Data-parallel Training Schedule



Vanilla Data-parallel Training Loop (BWD)



Flow-per-layer: 1. Compute weight gradient -> 2. issue weight gradient comm -> 3. compute input gradient -> 4. go to previous layer



Vanilla Data-parallel Training Schedule

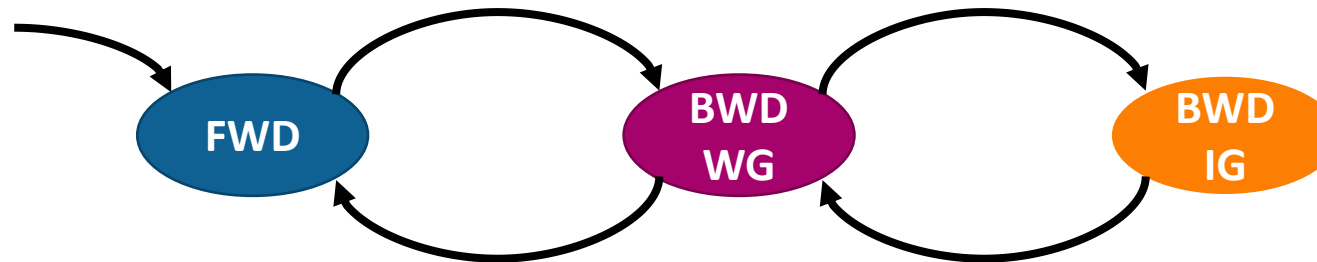


Vanilla Data-parallel Training Loop

Vanilla Data-parallel Training Schedule



FSM Diagram

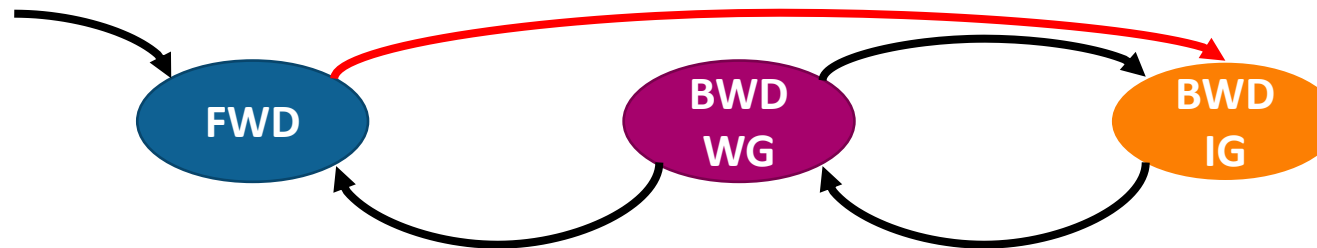


Exercise: Reorder Data-parallel Training Loop

Reordered Data-parallel Training Loop



FSM Diagram



Adding a New Training Loop

- See `astra-sim/workload/Workload.cc`
- Vanilla data-parallel loop is implemented in `iterate_data_parallel()`
- Add a reordered version, `iterate_data_parallel_reorder()`

```
void Workload::call(EventType event, CallData* data) {  
    if (counter > 0) {  
        generator->try_register_event(  
            this, EventType::Workload_Wait, NULL, counter);  
        return;  
    }  
    if (parallelismPolicy == ParallelismPolicy::Data) {  
        iterate_data_parallel();  
    } else if (parallelismPolicy == ParallelismPolicy::DataReorder) {  
        iterate_data_parallel_reorder();  
    } else if (parallelismPolicy == ParallelismPolicy::Transformer) {  
        iterate_hybrid_parallel_Transformer();  
    }  
}
```


Vanilla Training Loop (iterate_data_parallel)

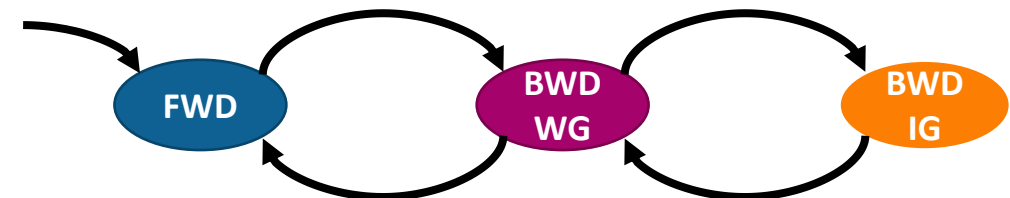
```
void Workload::iterate_data_parallel() {
    assert(index >= 0);
    assert(index < SIZE);
    check_for_sim_end();
    if (current_state == LoopState::Forward_Pass) {
+-- 31 lines: if (!layers[index]->is_weight_grad_comm_finished_block) {
        if (index >= SIZE) {
            current_state = LoopState::Weight_Gradient;
            index--;
        }
        generator->register_event(this, EventType::General, NULL, 1);
        return;
    } else if (current_state == LoopState::Weight_Gradient) {
+-- 14 lines: if (delay_loaded == false) {-----
        if (index == 0) {
            pass_counter++;
            current_state = LoopState::Forward_Pass;
        } else {
            current_state = LoopState::Input_Gradient;
        }
        generator->register_event(this, EventType::General, NULL, 1);
        return;
    } else if (current_state == LoopState::Input_Gradient) {
+-- 11 lines: if (delay_loaded == false) {-----
        delay_loaded = false;
        index--;
        current_state = LoopState::Weight_Gradient;
        generator->register_event(this, EventType::General, NULL, 1);
        return;
    }
}
```

- Training loop is implemented as a FSM
- index presents the current layer index
- current_state holds the current state

Vanilla Data-parallel Training Schedule



FSM Diagram



Reordered Training Loop (iterate_data_reorder)

```

void Workload::iterate_data_parallel_reorder() {
    assert(index >= 0);
    assert(index < SIZE);
    check_for_sim_end();
    if (current_state == LoopState::Forward_Pass) {
+-- 16 lines: if (!layers[index]->is_weight_grad_comm_finished_block) {
        if (index >= SIZE) {
            current_state = LoopState::Input_Gradient;
            index--;
        }
        generator->register_event(this, EventType::General, NULL, 1);
        return;
    } else if (current_state == LoopState::Weight_Gradient) {
+-- 15 lines: if (delay_loaded == false) {-----
        if (index > 1) {
            index--;
            current_state = LoopState::Input_Gradient;
        } else if (index == 1) {
            index--;
            current_state = LoopState::Weight_Gradient;
        } else if (index == 0) {
            pass_counter++;
            current_state = LoopState::Forward_Pass;
        }
        generator->register_event(this, EventType::General, NULL, 1);
        return;
    } else if (current_state == LoopState::Input_Gradient) {
+-- 11 lines: if (delay_loaded == false) {-----
        delay_loaded = false;
        current_state = LoopState::Weight_Gradient;
        generator->register_event(this, EventType::General, NULL, 1);
        return;
    }
}

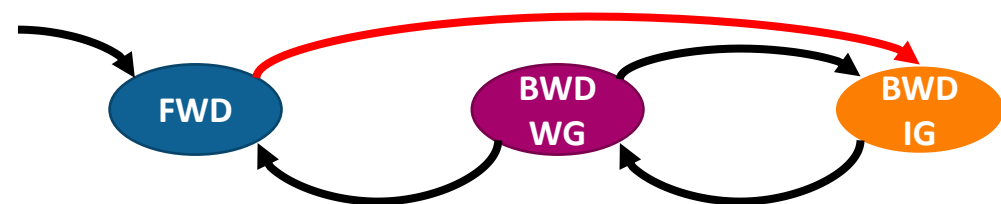
```

- You can reorder the computation schedule by tweaking the index and current_state

Reordered Data-parallel Training Schedule



FSM Diagram



Adding Debugging Messages

```
void Workload::iterate_data_parallel_reorder() {
    assert(index >= 0);
    assert(index < SIZE);
    check_for_sim_end();
    if (current_state == LoopState::Forward_Pass) {
+-- 3 lines: if (!layers[index]->is_weight_grad_comm_finished_blocking)
        if (delay_loaded == false) {
            counter = layers[index]->get_fwd_pass_compute();
            delay_loaded = true;
            if (generator->id == 0)
                std::cout << "[TUTORIAL] FWD[" << index << "]" << std::endl;
        }
+-- 13 lines: if (counter > 0) {-----
    } else if (current_state == LoopState::Weight_Gradient) {
        if (delay_loaded == false) {
            counter = layers[index]->get_weight_grad_compute();
            delay_loaded = true;
            if (generator->id == 0)
                std::cout << "[TUTORIAL] BWD_WG[" << index << "]" << std::endl;
        }
+-- 25 lines: if (counter > 0) {-----
    } else if (current_state == LoopState::Input_Gradient) {
        if (delay_loaded == false) {
            counter = layers[index]->get_input_grad_compute();
            delay_loaded = true;
            if (generator->id == 0)
                std::cout << "[TUTORIAL] BWD_IG[" << index << "]" << std::endl;
        }
+-- 9 lines: if (counter > 0) {-----
    }
}
```

- You can add debugging messages to make sure that the training loop works as expected
- Make sure to print debugging messages only when (generator->id == 0)
 - Each processing element is a generator
 - If you don't filter the ID, you will see debugging messages from all PEs

Adding Debugging Messages

Vanilla Data-parallel Loop

`./exercise_4/exercise_4_vanilla.sh | grep TUTORIAL`

```
[TUTORIAL] FWD[0]
[TUTORIAL] FWD[1]
[TUTORIAL] FWD[2]
[TUTORIAL] FWD[3]
[TUTORIAL] BWD_WG[3]
[TUTORIAL] BWD_IG[3]
[TUTORIAL] BWD_WG[2]
[TUTORIAL] BWD_IG[2]
[TUTORIAL] BWD_WG[1]
[TUTORIAL] BWD_IG[1]
[TUTORIAL] BWD_WG[0]
....
```

Reordered Data-parallel Loop

`./exercise_4/exercise_4_reorder.sh | grep TUTORIAL`

```
[TUTORIAL] FWD[0]
[TUTORIAL] FWD[1]
[TUTORIAL] FWD[2]
[TUTORIAL] FWD[3]
[TUTORIAL] BWD_IG[3]
[TUTORIAL] BWD_WG[3]
[TUTORIAL] BWD_IG[2]
[TUTORIAL] BWD_WG[2]
[TUTORIAL] BWD_IG[1]
[TUTORIAL] BWD_WG[1]
[TUTORIAL] BWD_WG[0]
....
```