STONNE Hands-on

Francisco Muñoz-Martinez

francisco.munoz2@um.es

Universidad de Murcia (UM)

Agenda

Attention: Tutorial is being recorded

Time (CET)	Time (ET)	Topic	Presenter
14:00 – 14:40	8:00 – 8:40	**Flexible Accelerators**	Tushar Krishna
14:40 – 15:10	8:40 – 9:10	**Cycle accurate simulation and Overview of STONNE**	José Luis Abellán
15:10 – 16:10	9:10 – 10:10	**(Hands-on) STONNE Deep-Dive**	Francisco Muñoz-Martínez
16:10 – 16:40	10:10 – 10:40	**Coffee Break**	
16:40 – 17:10	10:40 – 11:10	**(Hands-on) STONNE Deep-Dive**	Francisco Muñoz-Martínez
17:10 – 17:40	11:10 – 11:40	**Dataflow exploration for Graph Neural Networks**	Raveesh Garg
17:50 – 18:00	11:50 – 12:00	**Roadmap for Future Development**	Manuel Acacio

Tutorial Website https://stonne-simulator.github.io/ASPLOSTUT.html

includes agenda and STONNE/OMEGA installation instructions

Acknowledgements

Tushar Krishna

José L. Abellán

Francisco Muñoz-Martínez

Raveesh Garg

Manuel E. Acacio

Outline

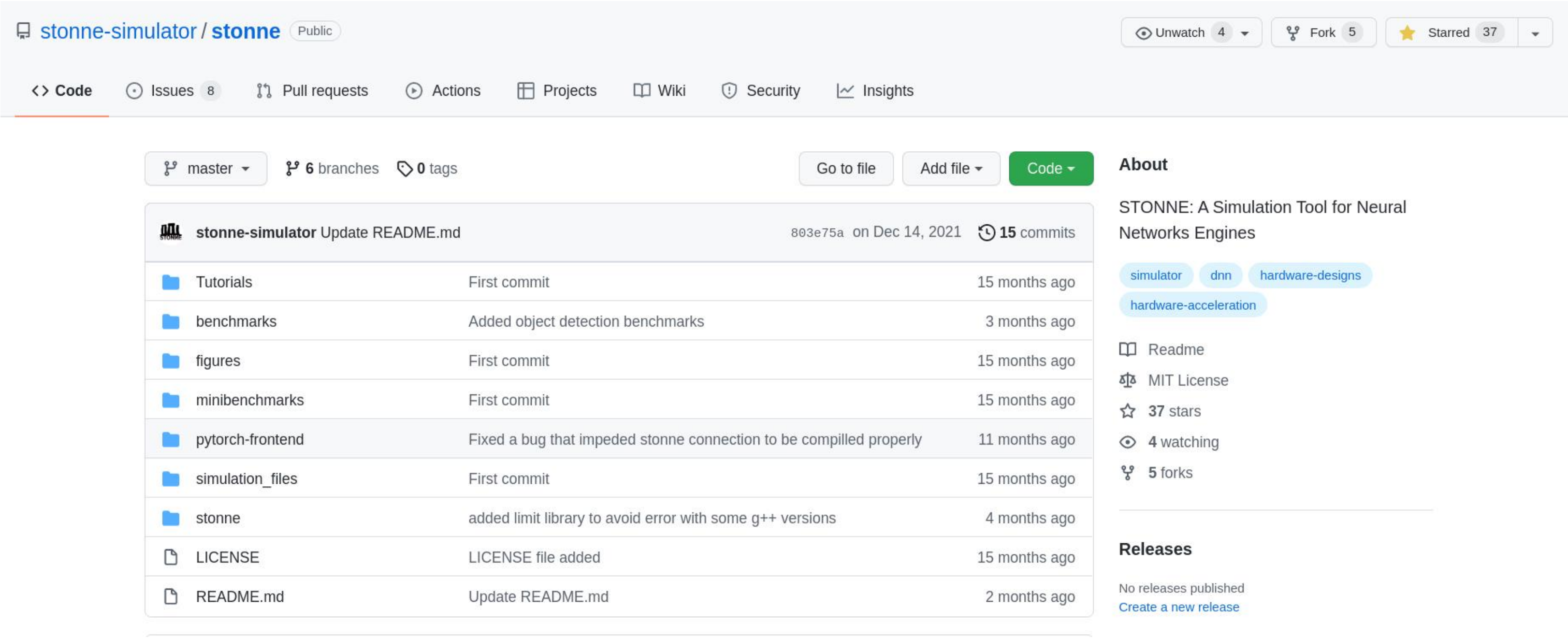
- Docker Image, Installation and overview of STONNE.
- Hands-on #1: Simulating a real DNN on Flexible DNN Accelerators.
- Hands-on #2: Using STONNE to evaluate a research use case – Exploiting sparsity.
- Hands-on #3: Extending new operations in STONNE – Adding the Conv1d operation.
- Research use cases.
- Conclusions.

Outline

- Docker Image, Installation and overview of STONNE.
- Hands-on #1: Simulating a real DNN on Flexible DNN Accelerators.
- Hands-on #2: Using STONNE to evaluate a research use case – Exploiting sparsity.
- Hands-on #3: Extending new operations in STONNE – Adding the Conv1d operation.
- Research use cases.
- Conclusions.

STONNE Repository

- STONNE available under the MIT license on Github:
  - https://github.com/stonne-simulator/stonne



STONNE Requirements

- STONNE dependencies:
  - Python 3.6 >= version< 3.9
  - C++ 14 or later.
  - Anaconda version 2021.05 or older.
  - .Linux OS (Tested on Ubuntu 18.05 and Manjaro 21.2.1).
  - Not tested in MAC OS.

STONNE Installation

- STONNE can be easily installed following a few steps:
  - STONNE User interface

```
(base) root@ae0d97074e84:/home/stonne_omega/stonne# cd stonne/
(base) root@ae0d97074e84:/home/stonne_omega/stonne/stonne# make
mkdir -p objs
g++ -O3 -Iinclude/ -Iexternal/   -c src/DNNLayer.cpp -o objs/DNNLayer.o  #-ltcmalloc
g++ -O3 -Iinclude/ -Iexternal/   -c src/Stats.cpp -o objs/Stats.o  #-ltcmalloc
g++ -O3 -Iinclude/ -Iexternal/   -c src/OSMeshMN.cpp -o objs/OSMeshMN.o  #-ltcmalloc
g++ -O3 -Iinclude/ -Iexternal/   -c src/CompilerMultiplierMesh.cpp -o objs/CompilerMultiplierMesh.o  #-ltcmalloc
g++ -O3 -Iinclude/ -Iexternal/   -c src/Accumulator.cpp -o objs/Accumulator.o  #-ltcmalloc
g++ -O3 -Iinclude/ -Iexternal/   -c src/CompilerFEN.cpp -o objs/CompilerFEN.o  #-ltcmalloc
g++ -O3 -Iinclude/ -Iexternal/   -c src/Fifo.cpp -o objs/Fifo.o  #-ltcmalloc
g++ -O3 -Iinclude/ -Iexternal/   -c src/DataPackage.cpp -o objs/DataPackage.o  #-ltcmalloc
g++ -O3 -Iinclude/ -Iexternal/   -c src/TemporalRN.cpp -o objs/TemporalRN.o  #-ltcmalloc
g++ -O3 -Iinclude/ -Iexternal/   -c src/CollectionBusLine.cpp -o objs/CollectionBusLine.o  #-ltcmalloc
g++ -O3 -Iinclude/ -Iexternal/   -c src/MSwitch.cpp -o objs/MSwitch.o  #-ltcmalloc
g++ -O3 -Iinclude/ -Iexternal/   -c src/MSNetwork.cpp -o objs/MSNetwork.o  #-ltcmalloc
g++ -O3 -Iinclude/ -Iexternal/   -c src/ASwitch.cpp -o objs/ASwitch.o  #-ltcmalloc
g++ -O3 -Iinclude/ -Iexternal/   -c src/Tile.cpp -o objs/Tile.o  #-ltcmalloc
^CMakefile:23: recipe for target 'objs/Tile.o' failed
make: *** [objs/Tile.o] Interrupt

(base) root@ae0d97074e84:/home/stonne_omega/stonne/stonne# 
```

STONNE Installation

- STONNE can be easily installed following a few steps:
  - Pytorch-Frontend

```
(base) root@ae0d97074e84:/home/stonne_omega/stonne# cd pytorch-frontend/
(base) root@ae0d97074e84:/home/stonne_omega/stonne/pytorch-frontend# python setup.py install
Building wheel torch-1.7.0a0
-- Building version 1.7.0a0
cmake --build . --target install --config Release -- -j 8
[  0%] Built target clog
[  0%] Built target gtest
[  0%] Built target defs.bzl
[  0%] Built target pthreadpool
[  0%] Built target benchmark
[  1%] Built target libprotobuf-lite
[  2%] Built target asmjit
[  2%] Built target fmt
[  3%] Built target c10
[  3%] Built target foxi_loader
[  3%] Built target ATEN_CPU_FILES_GEN_TARGET
[  4%] Built target dnnl_common
[  6%] Built target libprotobuf
[  6%] Built target mkrename
[  6%] Built target common
[  6%] Built target mkdisp
[  7%] Built target mkalias
[ 22%] Built target python_copy_files
[ 23%] Built target dnnl_cpu
[ 23%] Built target arraymap
[ 23%] Built target mkrename_gnuabi
[ 23%] Built target mkmasked_gnuabi
[ 23%] Built target torch_global_deps
[ 23%] Built target torch_python_stubs
```

STONNE Installation

- STONNE can be easily installed following a few steps:
  - Pytorch-stonne

```
(base) root@ae0d97074e84:/home/stonne_omega/stonne/pytorch-frontend# cd stonne_connection/
(base) root@ae0d97074e84:/home/stonne_omega/stonne/pytorch-frontend/stonne_connection# python setup.py install
['torch_stonne.cpp', '../../stonne/stonne_linker_src/stonne_linker.cpp', '../../stonne/src/Fifo.cpp', '../../st
nne/src/CollectionBusLine.cpp', '../../stonne/src/Connection.cpp', '../../stonne/src/FENetwork.cpp', '../../sto
ne/src/Accumulator.cpp', '../../stonne/src/MSwitch.cpp', '../../stonne/src/CompilerMultiplierMesh.cpp', '../../
tonne/src/DSNetworkTop.cpp', '../../stonne/src/DNNModel.cpp', '../../stonne/src/CompilerMSN.cpp', '../../stonne
src/OSMeshSDMemory.cpp', '../../stonne/src/DSwitch.cpp', '../../stonne/src/AccumulationBuffer.cpp', '../../ston
e/src/Config.cpp', '../../stonne/src/Tile.cpp', '../../stonne/src/Stats.cpp', '../../stonne/src/CollectionBus.c
p', '../../stonne/src/DNNLayer.cpp', '../../stonne/src/testbench.cpp', '../../stonne/src/STONNEModel.cpp', '../
./stonne/src/SDMemory.cpp', '../../stonne/src/MSNetwork.cpp', '../../stonne/src/LookupTable.cpp', '../../stonne
src/ASNetwork.cpp', '../../stonne/src/TemporalRN.cpp', '../../stonne/src/CompilerFEN.cpp', '../../stonne/src/Mu
tiplierOS.cpp', '../../stonne/src/DSNetwork.cpp', '../../stonne/src/DataPackage.cpp', '../../stonne/src/ASwitch
cpp', '../../stonne/src/OSMeshMN.cpp', '../../stonne/src/CompilerART.cpp', '../../stonne/src/SparseSDMemory.cpp
, '../../stonne/src/utility.cpp', '../../stonne/src/FEASwitch.cpp']
running install
running bdist_egg
running egg_info
writing torch_stonne.egg-info/PKG-INFO
writing dependency_links to torch_stonne.egg-info/dependency_links.txt
writing top-level names to torch_stonne.egg-info/top_level.txt
/opt/conda/lib/python3.8/site-packages/torch/utils/cpp_extension.py:340: UserWarning: Attempted to use ninja as
the BuildExtension backend but we could not find ninja.. Falling back to using the slow distutils backend.
  warnings.warn(msg.format('we could not find ninja.'))
reading manifest file 'torch_stonne.egg-info/SOURCES.txt'
writing manifest file 'torch_stonne.egg-info/SOURCES.txt'
installing library code to build/bdist.linux-x86_64/egg
running install_lib
```

Docker image

- To avoid having to install the software, we have set up a docker image which can be directly downloaded and used:

sudo docker run -it franciscomunoz/stonne_omega_img /bin/bash

```
[paco@paco-pc stonne]$ sudo docker run -it franciscomunoz/stonne_omega_img /bin/bash
(base) root@6e865a302d42:/home/stonne_omega/omega#
```

Docker image

- Project organization:
  - /home/stonne_omega:
    - omega
    - stonne:
      - ASPLOS22
      - pytorch-frontend
        - stonne_connection
      - stonne:
        - stonne.elf
        - Include
        - src
        - external
        - stonne_linker_src

Docker image

- Project organization:
  - /home/stonne_omega:
    - **omega**
    - stonne:
      - ASPLOS22
      - pytorch-frontend
        - stonne_connection
      - stonne:
        - stonne.elf
        - Include
        - src
        - external
        - stonne_linker_src

Docker image

- Project organization:
  - /home/stonne_omega:
    - omega
    - **stonne**:
      - ASPLOS22
      - pytorch-frontend
        - stonne_connection
      - stonne:
        - stonne.elf
        - Include
        - src
        - external
        - stonne_linker_src

Docker image

- Project organization:
    - /home/stonne_omega:
        - omega
        - stonne:
            - **ASPLOS22**
            - pytorch-frontend
                - stonne_connection
            - stonne:
                - stonne.elf
                - Include
                - src
                - external
                - stonne_linker_src

Docker image

- Project organization:
  - /home/stonne_omega:
    - omega
    - stonne:
      - ASPLOS22
      - **pytorch-frontend**
        - stonne_connection
      - stonne:
        - stonne.elf
        - Include
        - src
        - external
        - stonne_linker_src

Docker image

- Project organization:
  - /home/stonne_omega:
    - omega
    - stonne:
      - ASPLOS22
      - pytorch-frontend
        - stonne_connection
      - **stonne**:
        - **stonne.elf**
        - Include
        - src
        - external
        - stonne_linker_src

Docker image

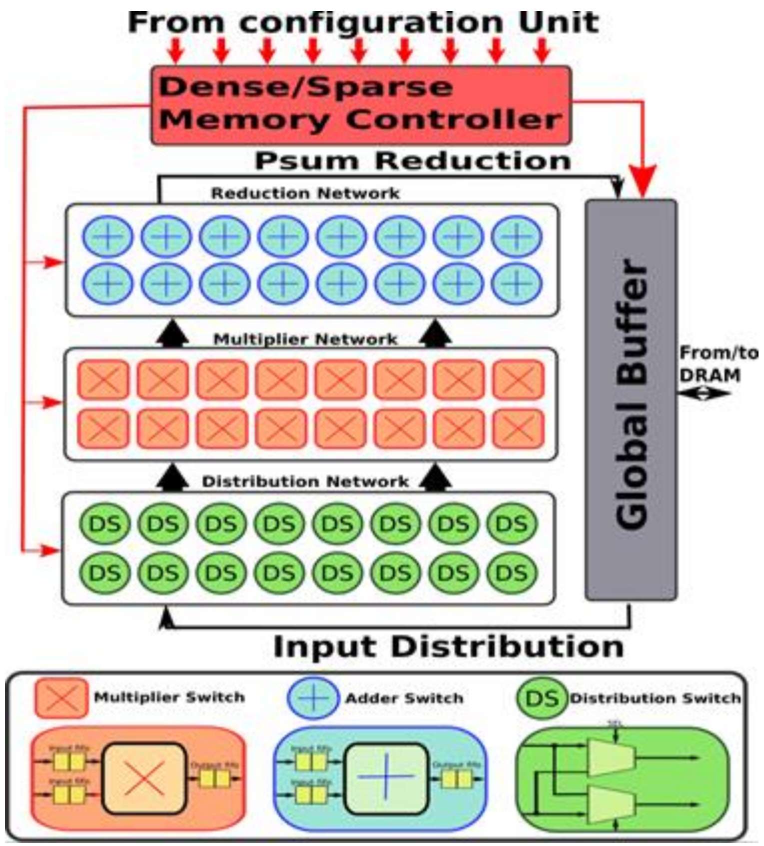
- Project organization:
  - /home/stonne_omega:
    - omega
    - stonne:
      - ASPLOS22
      - pytorch-frontend
        - stonne_connection
      - **stonne**:
        - stonne.elf
        - **Include**
        - src
        - external
        - stonne_linker_src

STONNE Source files

- */home/stonne_omega/stonne/stonne/include*: Contains the headers
  - AccumulationBuffer.h
  - Accumulator.h
  - ReductionNetwork.h
  - ASNetwork.h
  - ASwitch.h
  - CollectionBus.h
  - CollectionBusLine.h
  - DistributionNetwork.h
  - DSNetworkTop.h
  - DSNetwork.h
  - DSwitch.h
  - MultiplierNetwork.h
  - MSNetwork.h
  - MultiplierOS.h
  - MSwitch.h
  - MemoryController.h
  - SDMemory.h
  - SparseSDMemory.h
  - OSMeshSDMemory.h
  - Connection.h
  - DataPackage.h
  - Fifo.h
  - STONNEModel.h

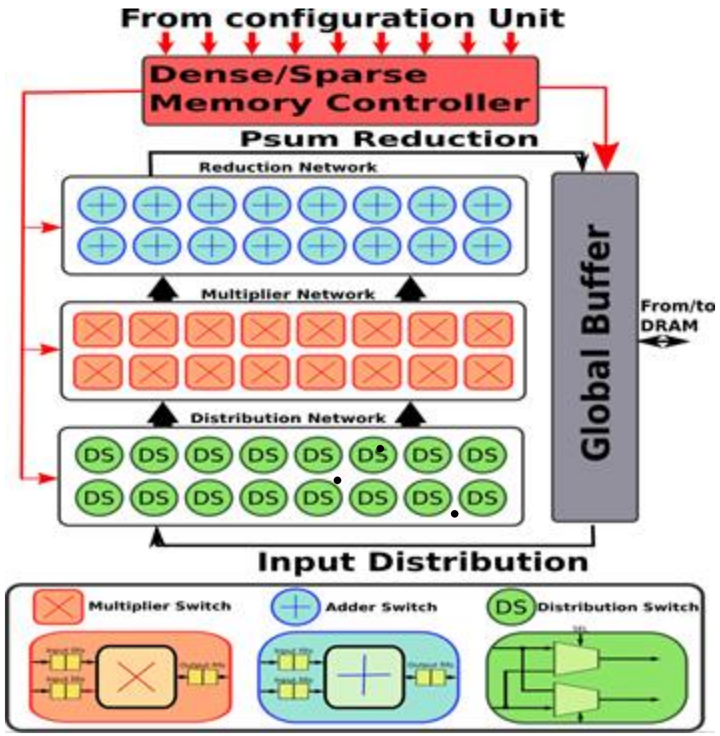
STONNE Source files

- STONNE organization is able to abstract the main components of the architecture



STONNE Source files

- Abstract classes allow to abtract the 4 main components.
  - AccumulationBuffer.h
  - Accumulator.h
  - **ReductionNetwork.h**
  - ASNetwork.h
  - ASwitch.h
  - CollectionBus.h
  - CollectionBusLine.h
  - **DistributionNetwork.h**
  - DSNetworkTop.h
  - DSNetwork.h
  - DSwitch.h
  - **MultiplierNetwork.h**
  - MSNetwork.h
  - MultiplierOS.h
  - MSwitch.h
  - **MemoryController.h**
  - SDMemory.h
  - SparseSDMemory.h
  - OSMeshSDMemory.h
  - Connection.h
  - DataPackage.h
  - Fifo.h
  - STONNEModel.h



STONNE Source files

- The real hardware comonents implement the abstract classes.
  - AccumulationBuffer.h
  - Accumulator.h
  - **ReductionNetwork.h**
  - ASNetwork.h
  - ASwitch.h
  - CollectionBus.h
  - CollectionBusLine.h
  - **DistributionNetwork.h**
  - DSNetworkTop.h
  - DSNetwork.h
  - DSwitch.h
  - **MultiplierNetwork.h**
  - MSNetwork.h
  - MultiplierOS.h
  - MSwitch.h
  - **MemoryController.h**
  - SDMemory.h
  - SparseSDMemory.h
  - OSMeshSDMemory.h
  - Connection.h
  - DataPackage.h
  - Fifo.h
  - STONNEModel.h

Reduction Networks

Distribution Networks

Multiplier Networks

Memory Controllers

Other Useful components

STONNE Source files

- Every component implements its **void cycle()** method which defines the behaviour of the component.

```
class MultiplierNetwork : public Unit{
public:
    /*
      By the default the implementation of the MS just receives a single element, calculate a single psum and/or send a single input activation to the neighbour. This way, the parameters
      input_ports, output_ports and forwarding_ports will be set as the single data size. If this implementation change for future tests, this can be change easily bu mofifying these three parameters.
     */
    MultiplierNetwork(id_t id, std::string name) : Unit(id, name){}
    //set connections from the distribution network to the multiplier network
    virtual void setInputConnections(std::map<int, Connection*> input_connections) {assert(false);}
    //Set connections from the Multiplier Network to the Reduction Network
    virtual void setOutputConnections(std::map<int, Connection*> output_connections) {assert(false);}
    virtual void cycle() {assert(false);}
    virtual void configureSignals(Tile* current_tile, DNNLayer* dnn_layer, unsigned int ms_size, unsigned int n_folding) {assert(false);}
    virtual void configureSparseSignals(std::vector<SparseVN> sparseVNs, DNNLayer* dnn_layer, unsigned int ms_size) {assert(false);}

    virtual void resetSignals() {assert(false);}
    virtual void printConfiguration(std::ofstream& out, unsigned int indent) {assert(false);}
    virtual void printStats(std::ofstream &out, unsigned int indent) {assert(false);}
    virtual void printEnergy(std::ofstream& out, unsigned int indent) {assert(false);}
};
```

STONNE Source files

- */home/stonne_omega/stonne/stonne/src*: Contains the main code
    - AccumulationBuffer.cpp
    - Accumulator.cpp
    - ASNetwork.cpp
    - ASwitch.cpp
    - CollectionBus.cpp
    - CollectionBusLine.cpp
    - DSNetworkTop.cpp
    - DSNetwork.cpp
    - DSwitch.cpp
    - MSNetwork.cpp
    - MultiplierOS.cpp
    - MSwitch.cpp
    - MemoryController.cpp
    - SDMemory.cpp
    - SparseSDMemory.cpp
    - OSMeshSDMemory.cpp
    - Connection.cpp
    - DataPackage.cpp
    - Fifo.cpp
    - STONNEModel.cpp

STONNE Source files

- *STONNEModel.cpp drives the simulation based on the user configuration file*
  - AccumulationBuffer.cpp
  - Accumulator.cpp
  - ASNetwork.cpp
  - ASwitch.cpp
  - CollectionBus.cpp
  - CollectionBusLine.cpp
  - DSNetworkTop.cpp
  - DSNetwork.cpp
  - DSwitch.cpp
  - MSNetwork.cpp
  - MultiplierOS.cpp
  - MSwitch.cpp
  - MemoryController.cpp
  - SDMemory.cpp
  - SparseSDMemory.cpp
  - OSMeshSDMemory.cpp
  - Connection.cpp
  - DataPackage.cpp
  - Fifo.cpp
  - **STONNEModel.cpp**

STONNE Source files

- *STONNEModel.cpp drives the simulation based on the user configuration file*
  - *1. The code reads the input file and selects the correct modules:*

```
Stonne::Stonne(Config stonne_cfg) {
    this->stonne_cfg=stonne_cfg;
    this->ms_size = stonne_cfg.m_MSNetworkCfg.ms_size;
    this->layer_loaded=false;
    this->tile_loaded=false;
    this->outputASConnection = new Connection(stonne_cfg.m_SDMemoryCfg.port_width);
    this->outputLTConnection = new Connection(stonne_cfg.m_LookUpTableCfg.port_width);
    switch(stonne_cfg.m_MSNetworkCfg.multiplier_network_type) {
        case LINEAR:
            this->msnet = new MSNetwork(2, "MSNetwork", stonne_cfg);
            break;
        case OS_MESH:
            this->msnet = new OSMeshMN(2, "OSMesh", stonne_cfg);
            break;
        default:
            assert(false);
    }
    //switch(DistributionNetwork). It is possible to create instances of other Distributi
    this->dsnet = new DSNetworkTop(1, "DSNetworkTop", stonne_cfg);

    //Creating the ReduceNetwork according to the parameter specified by the user
    switch(stonne_cfg.m_ASNetworkCfg.reduce_network_type) {
    case ASNETWORK:
        this->asnet = new ASNetwork(3, "ASNetwork", stonne_cfg, outputASConnection);
        break;
    case FENETWORK:
        this->asnet = new FENetwork(3, "FENetwork", stonne_cfg, outputASConnection);
        break;
    case TEMPORALRN:
        this->asnet = new TemporalRN(3, "TemporalRN", stonne_cfg, outputASConnection);
        break;
    default:
        assert(false);
    }
```

STONNE Source files

- *STONNEModel.cpp drives the simulation based on the user configuration file*
  - *2. After this, the class defines methods to load the layer, tile parameters, memory addresses and run the simulation.* ***This is used to interface the simulator***

```
   void loadDNNLayer(Layer_t layer_type, std::string layer_name, unsigned int R, unsigned int S, unsigned int C, unsigned int K, unsigned int G,  unsigned in
t N, unsigned int X, unsigned int Y, unsigned int strides, address_t input_address, address_t filter_address, address_t output_address, Dataflow dataflow); //
General constructor

   //Load CONV Layer. At the end this calls to the general constructor  with all the parameters
    void loadCONVLayer(std::string layer_name, unsigned int R, unsigned int S, unsigned int C, unsigned int K, unsigned int G, unsigned int N, unsigned int X,
 unsigned int Y, unsigned int strides, address_t input_address, address_t filter_address, address_t output_address);

    //Load FC layer just with the appropiate parameters
    //N = batch size (i.e., number of rows in input matrix); S=number of inputs per batch (i.e., column size in input matrix and weight matrix); K=number of o
utputs neurons (i.e, number of rows weight matrix)
    void loadFCLayer(std::string layer_name, unsigned int N, unsigned int S, unsigned int K, address_t input_address, address_t filter_address, address_t outp
ut_address);

    //Load Sparse GEMM onto STONNE according to SIGMA parameter taxonomy.
    void loadGEMM(std::string layer_name, unsigned int N, unsigned int K, unsigned int M, address_t MK_matrix, address_t KN_matrix, metadata_address_t MK_meta
data, metadata_address_t KN_metadata, address_t output_matrix, metadata_address_t output_metadata, Dataflow dataflow);

   //Load Dense GEMM onto STONNE according to SIGMA parameter taxonomy and tiling according to T_N, T_K and T_M
   void loadDenseGEMM(std::string layer_name, unsigned int N, unsigned int K, unsigned int M, address_t MK_matrix, address_t KN_matrix, address_t output_matri
x, Dataflow dataflow);

    //Load sparse-dense GEMM onto STONNE
    void loadSparseDense(std::string layer_name, unsigned int N, unsigned int K, unsigned int M, address_t MK_matrix, address_t KN_matrix, metadata_address_t
MK_metadata_id, metadata_address_t MK_metadata_pointer, address_t output_matrix, unsigned int T_N, unsigned int T_K);

   //Load a Dense GEMM tile to run it using the loadDenseGEMM function
   void loadGEMMTile(unsigned int T_N, unsigned int T_K, unsigned int T_M);

    void loadTile(unsigned int T_R, unsigned int T_S, unsigned int T_C, unsigned int T_K, unsigned int T_G, unsigned int T_N, unsigned int T_X_, unsigned int
T_Y_); //Load general and CONV tile
    void loadFCTile(unsigned int T_S, unsigned int T_N, unsigned int T_K); //VNSize = T_S, NumVNs= T_N*T_K
    void run();
```

STONNE Source files

- *STONNEModel.cpp drives the simulation based on the user configuration file*
  - *3. The code runs the **cycle()** method of each component until the execution is completed.*

```
bool execution_finished=false;
while(!execution_finished) {
    this->mem->cycle();
    this->collectionBus->cycle();
    this->asnet->cycle();
    this->lt->cycle();
      this->collectionBus->cycle(); //This order since these are connecti

    this->msnet->cycle();
    this->dsnet->cycle();
    execution_finished = this->mem->isExecutionFinished();
    this->n_cycles++;
}

if(this->stonne_cfg.print_stats_enabled) { //If sats printing is enable
    this->printStats();
    this->printEnergy();
}
```

Outline

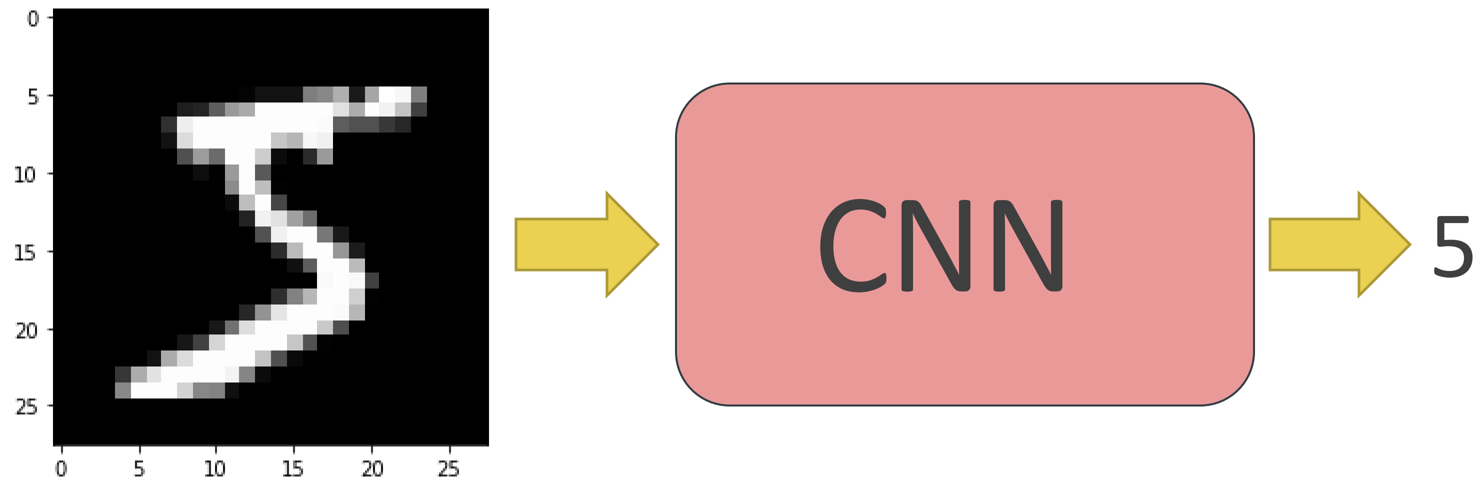
- Docker Image, Installation and overview of STONNE.
- Hands-on #1: Simulating a real DNN on Flexible DNN Accelerators.
- Hands-on #2: Using STONNE to evaluate a research use case – Exploiting sparsity.
- Hands-on #3: Extending new operations in STONNE – Adding the Conv1d operation.
- Research use cases.
- Conclusions.

Objectives

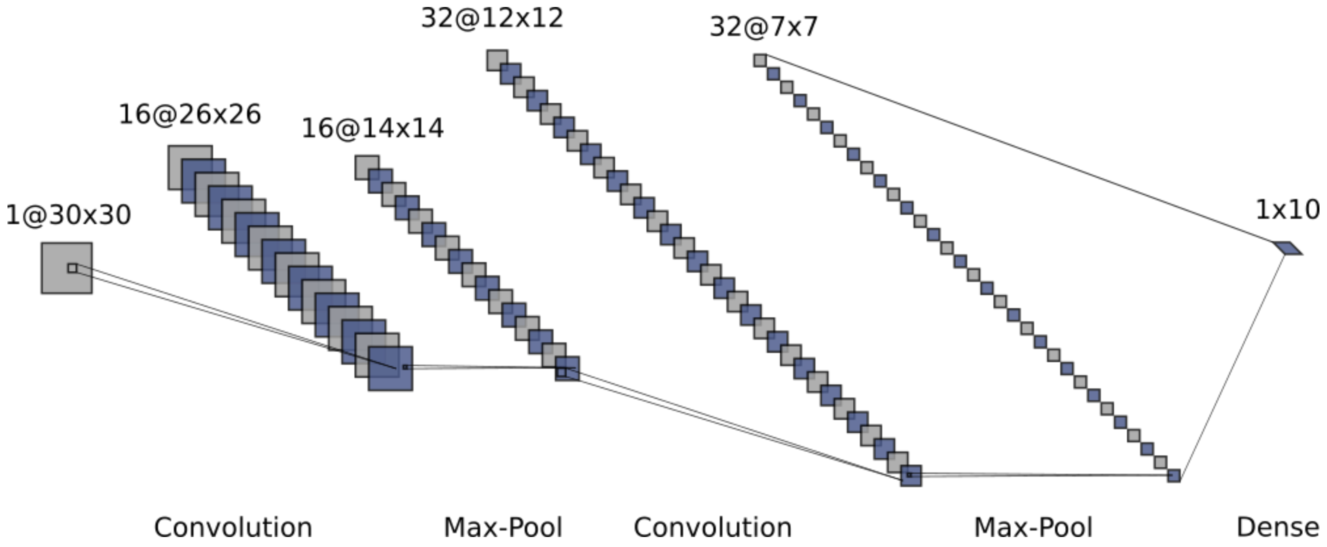
- 1. Comprehensively understand how do flexible DNN architectures work.
- 2. Understand how to run simulations in STONNE using its STONNE User interface.
- Understand how to run a real DNN in STONNE using the framework Pytorch.

DNN benchmark

- We will go through a real-life example to motivate our use case.
- **Application task**: Image digit recognition
- **Benchmark**: A Convolutional Neural Network (CNN).
- **Dataset:** MNIST (60000 28x28-size images)

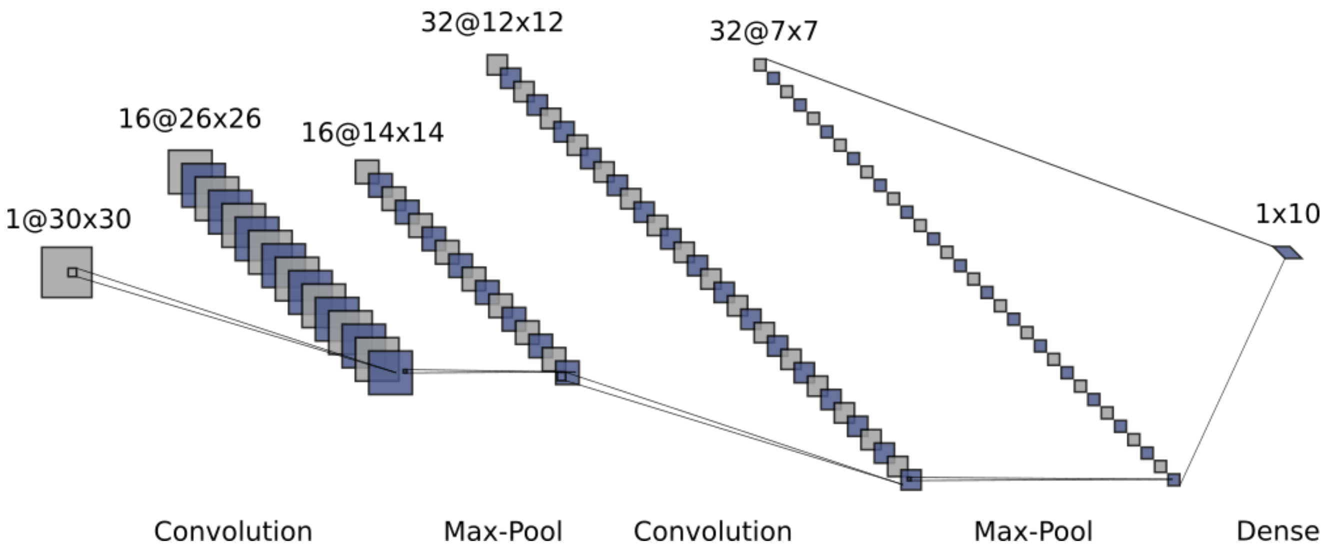


CNN model



Layer	Operation	R	S	C	K	G	N	X	Y
0	CONV	5	5	1	16	1	1	32	32
1	MaxPool	2	2	16	16	1	1	28	28
2	CONV	5	5	16	32	1	1	18	18
3	MaxPool	2	2	32	32	1	1	14	14
4	Dense	-	-	1568	10	-	-	-	-

CNN model

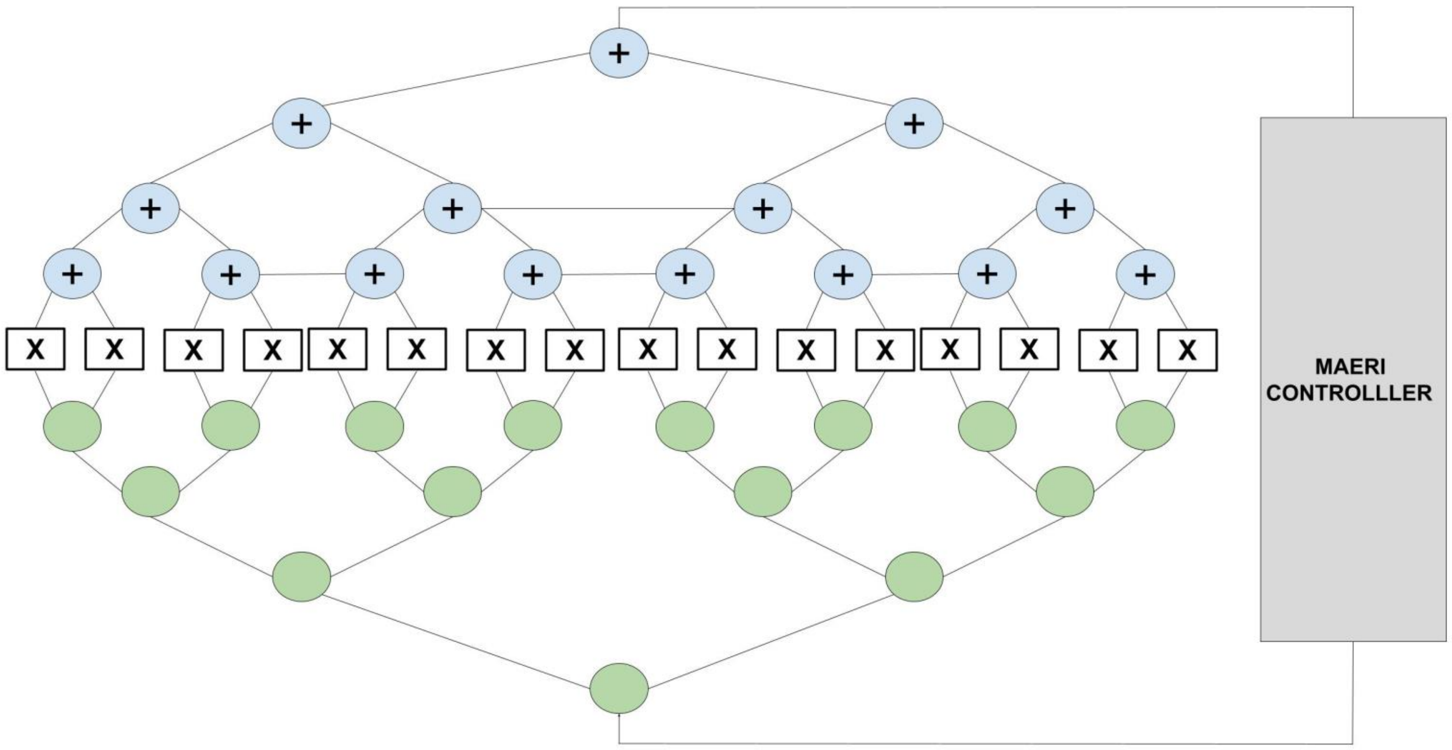


Layer	Operation	R	S	C	K	G	N	X	Y	
0	CONV	5	5	1	16	1	1	30	30	**Simulated**
1	MaxPool	2	2	16	16	1	1	26	26	
2	CONV	5	5	16	32	1	1	14	14	**Simulated**
3	MaxPool	2	2	32	32	1	1	12	12	
4	Dense	-	-	1568	10	-	-	-	-	**Simulated**

Simulated architecture

- We will simulate our benchmark in a 64-MS MAERI architecture.

ID	Number of Multipliers	Distribution Bandwidth	Reduction Bandwidth
1	64	64	64



Mapping the CNN onto the architectures

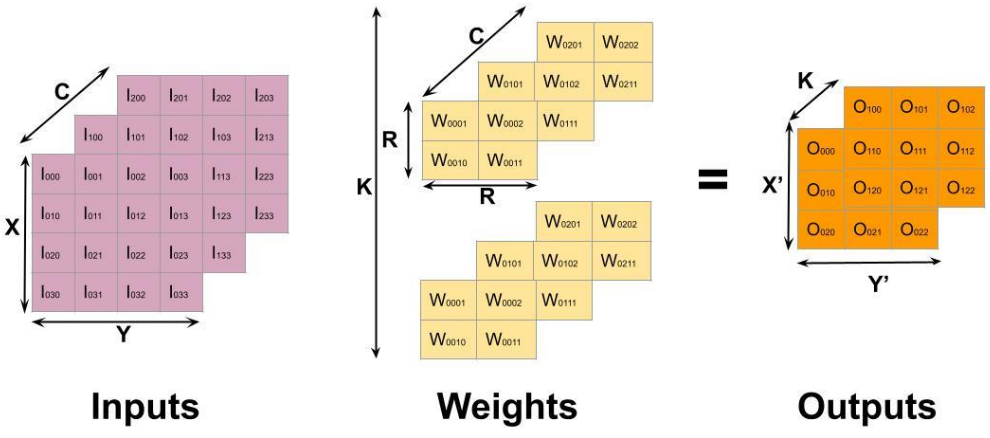
- Given our CNN and our two architectures, how do we map the computation?

Mapping the CNN onto the architectures

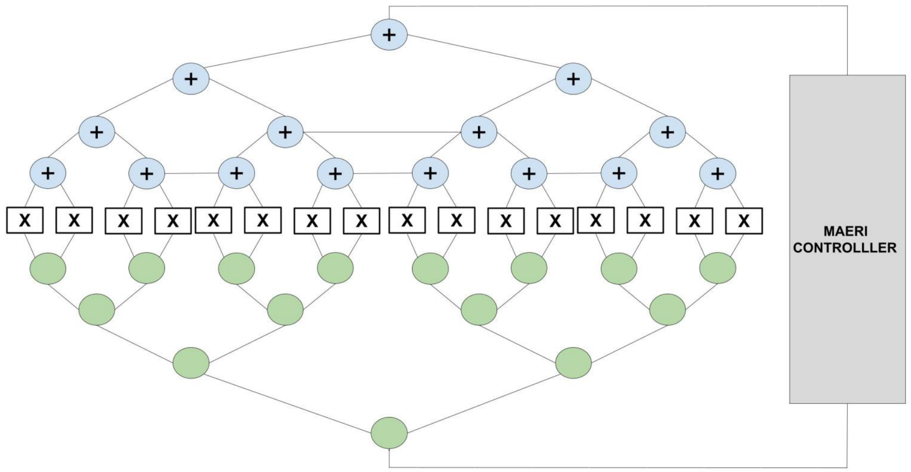
- Given our CNN and our two architectures, how do we map the computation?
  - Answer: Tiling the layer.

Mapping the CNN onto the architecture

- Given our CNN and our two architectures, how do we map the computation?
- Example: How do we map this layer?



Mapping?

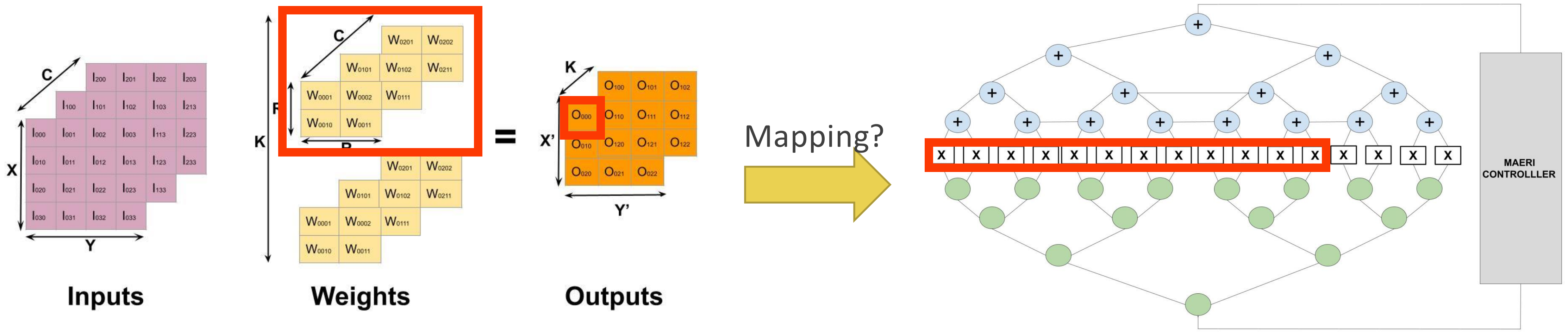


R	S	C	K	G	N	X	Y	X'	Y'
2	2	3	2	1	1	4	4	3	3

T_R	T_S	T_C	T_K	T_G	T_N	T_X'	T_Y'
?	?	?	?	?	?	?	?

Mapping the CNN onto the architecture

- Given our CNN and our two architectures, how do we map the computation?
- Example: How do we map this layer?



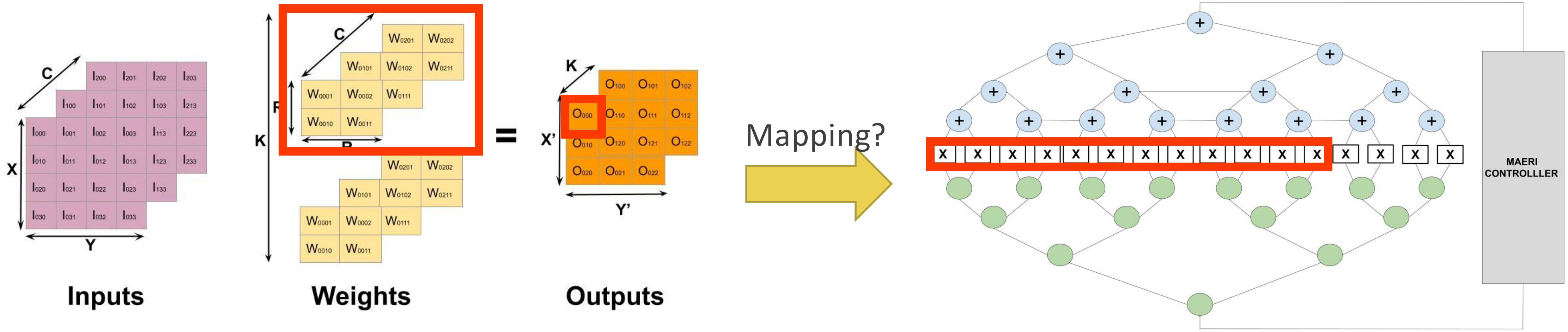
R	S	C	K	G	N	X	Y	X'	Y'
2	2	3	2	1	1	4	4	3	3

T_R	T_S	T_C	T_K	T_G	T_N	T_X'	T_Y'
2	2	3	1	1	1	1	1

Virtual Neuron Size: ?
Number of Virtual Neurons: ?

Mapping the CNN onto the architecture

- Given our CNN and our two architectures, how do we map the computation?
- Example: How do we map this layer?



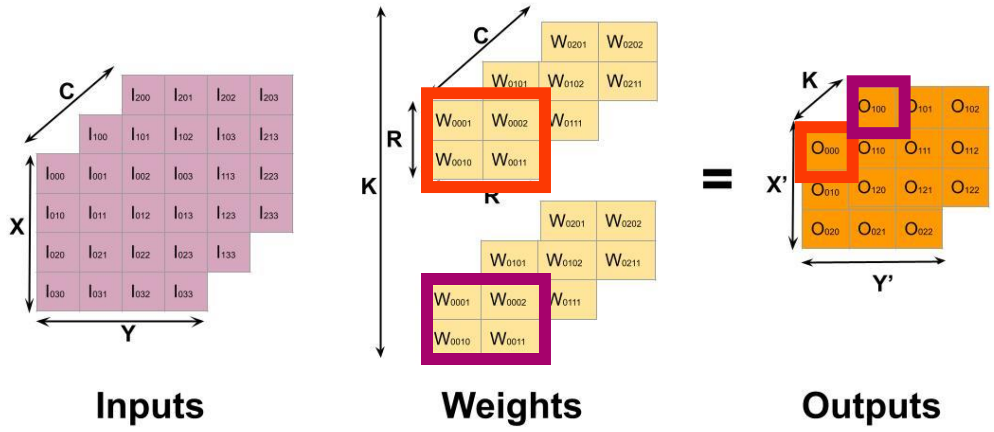
R	S	C	K	G	N	X	Y	X'	Y'
2	2	3	2	1	1	4	4	3	3

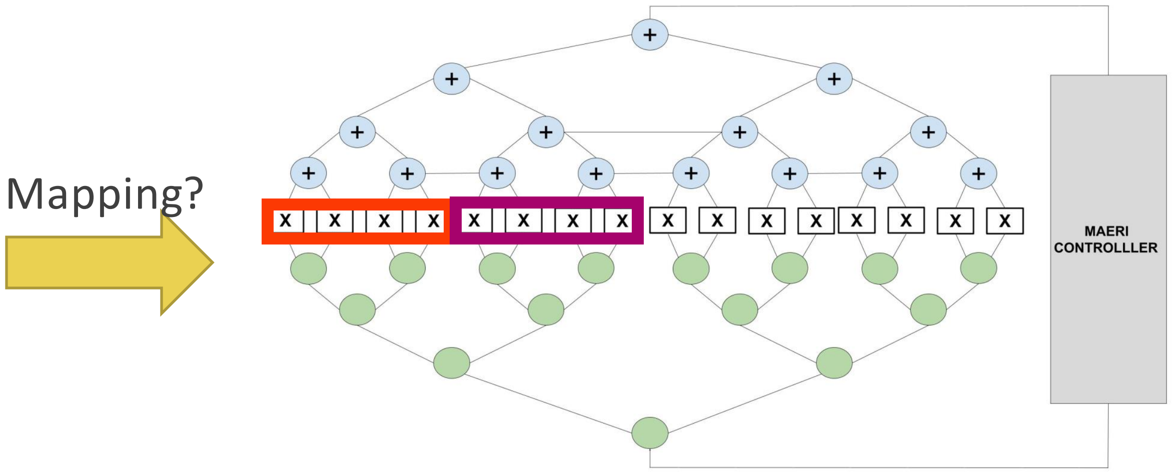
T_R	T_S	T_C	T_K	T_G	T_N	T_X'	T_Y'
2	2	3	1	1	1	1	1

Virtual Neuron Size: 2*2*3=12
Number of Virtual Neurons: 1*1*1*1*1=1

Mapping the CNN onto the architecture

- Given our CNN and our two architectures, how do we map the computation?
- Example: How do we map this layer?





R	S	C	K	G	N	X	Y	X'	Y'
2	2	3	2	1	1	4	4	3	3

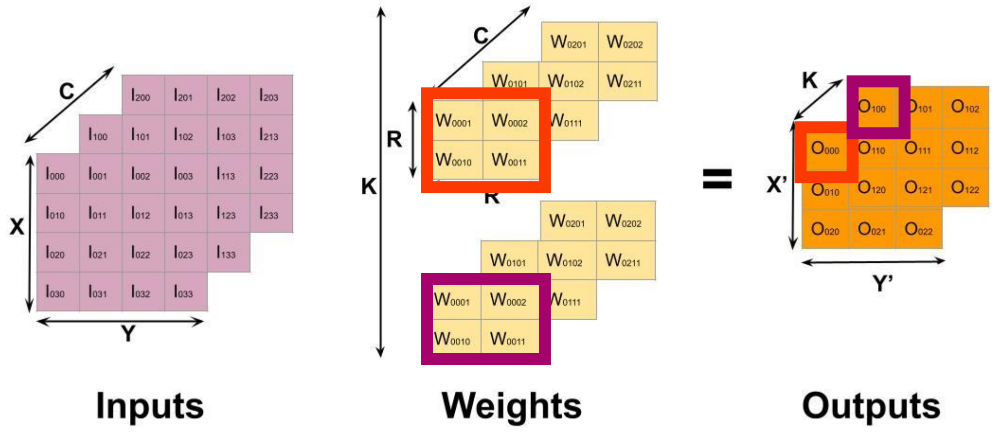
T_R	T_S	T_C	T_K	T_G	T_N	T_X'	T_Y'
?	?	?	?	?	?	?	?

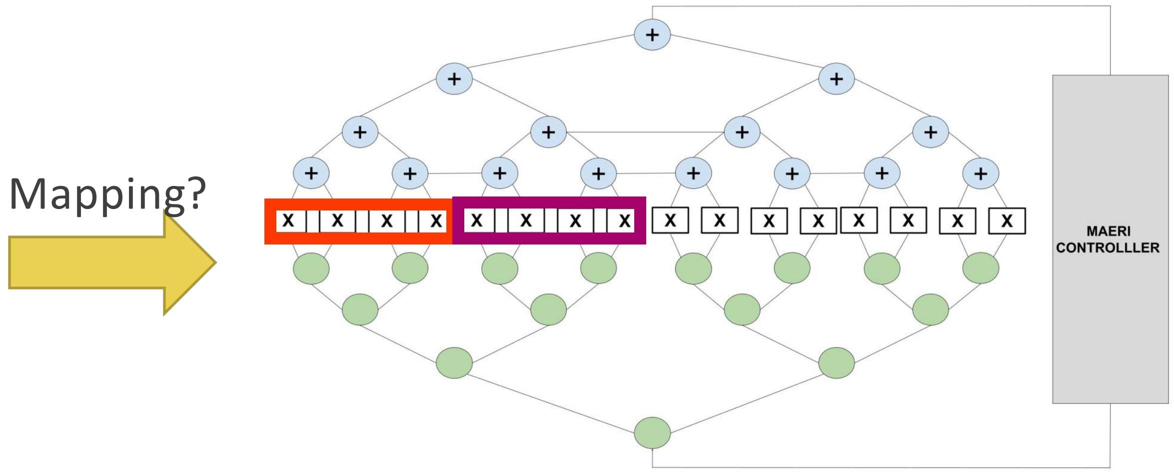
Virtual Neuron Size: ?

Number of Virtual Neurons: ?

Mapping the CNN onto the architecture

- Given our CNN and our two architectures, how do we map the computation?
- Example: How do we map this layer?





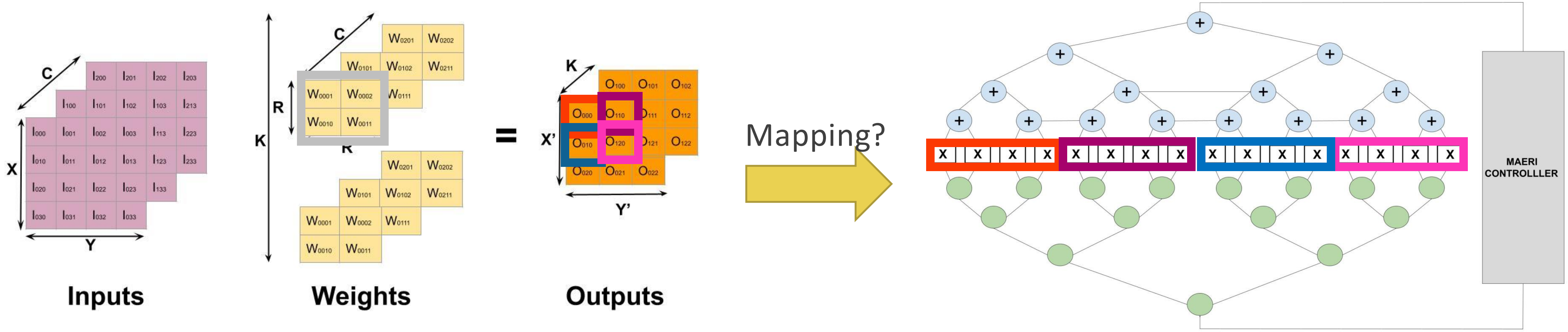
R	S	C	K	G	N	X	Y	X'	Y'
2	2	3	2	1	1	4	4	3	3

T_R	T_S	T_C	T_K	T_G	T_N	T_X'	T_Y'
2	2	1	2	1	1	1	1

Virtual Neuron Size: 2*2*1=4
Number of Virtual Neurons: 2*1*1*1*1=2

Mapping the CNN onto the architecture

- Given our CNN and our two architectures, how do we map the computation?
- Example: How do we map this layer?



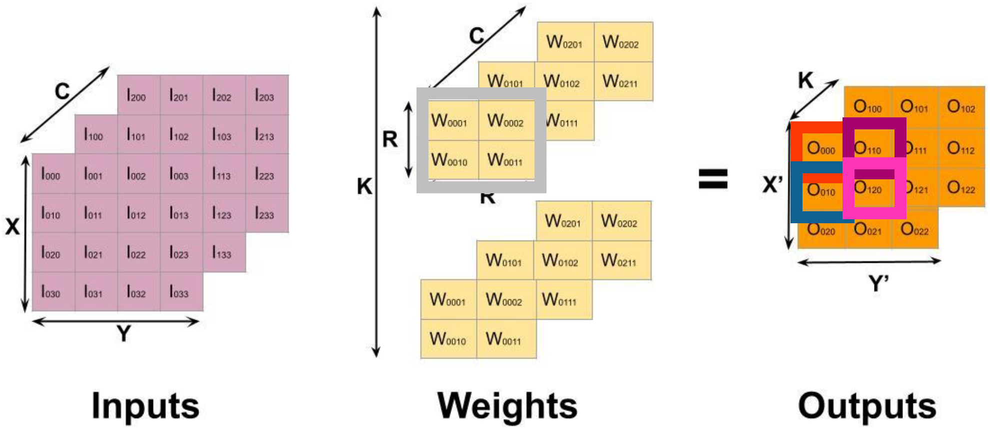
R	S	C	K	G	N	X	Y	X'	Y'
2	2	3	2	1	1	4	4	3	3

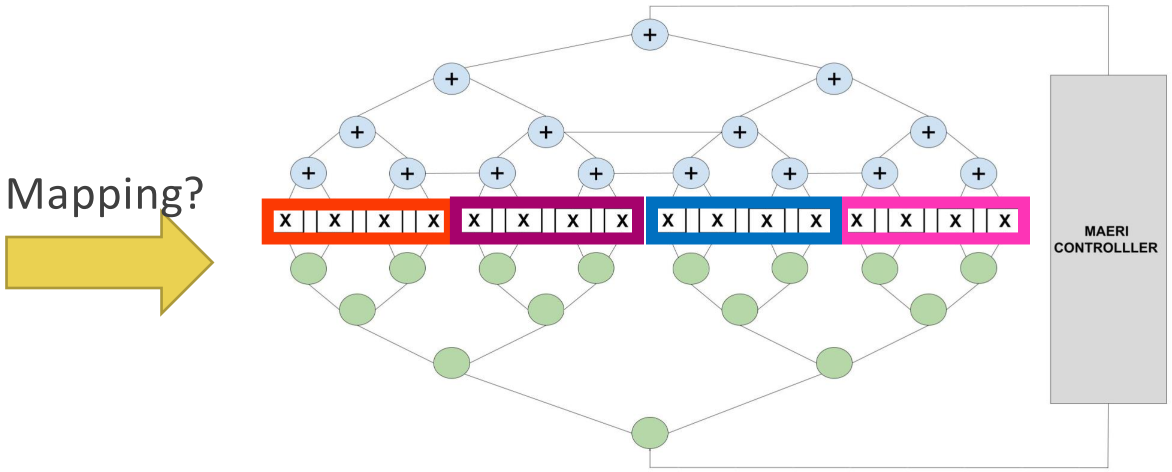
T_R	T_S	T_C	T_K	T_G	T_N	T_X'	T_Y'
?	?	?	?	?	?	?	?

Virtual Neuron Size: ?
Number of Virtual Neurons: ?

Mapping the CNN onto the architecture

- Given our CNN and our two architectures, how do we map the computation?
- Example: How do we map this layer?





R	S	C	K	G	N	X	Y	X'	Y'
2	2	3	2	1	1	4	4	3	3

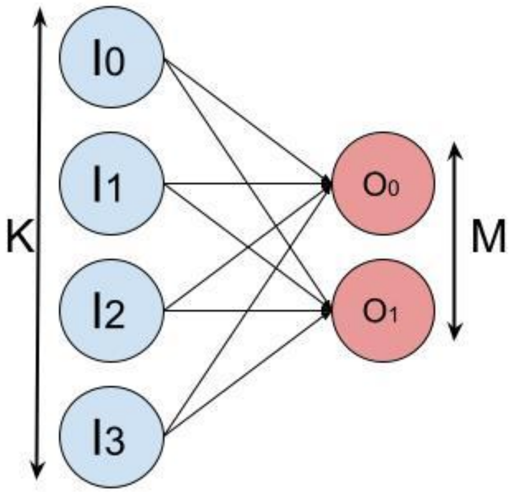
T_R	T_S	T_C	T_K	T_G	T_N	T_X'	T_Y'
2	2	1	1	1	1	2	2

Virtual Neuron Size: 2*2*1=4

Number of Virtual Neurons: 1*1*1*2*2=4

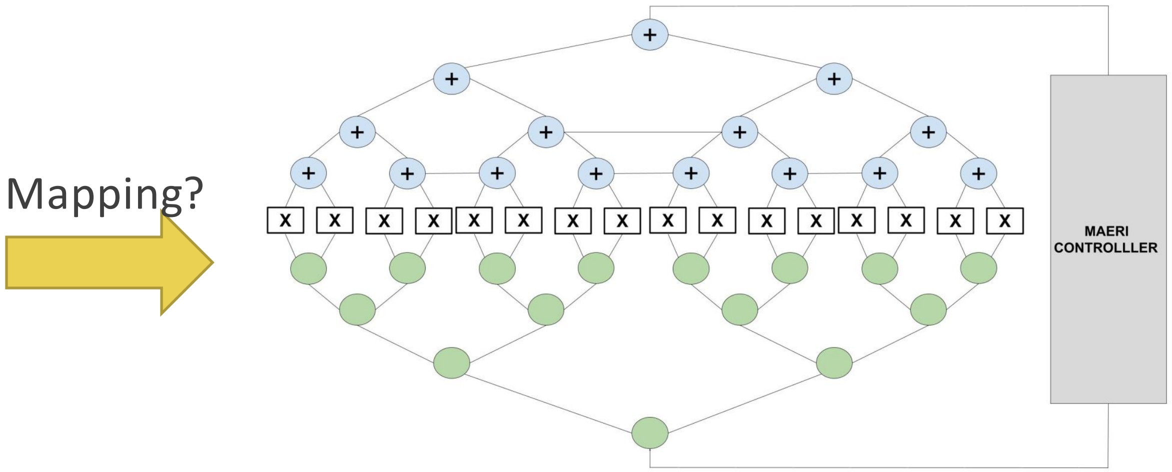
Mapping the CNN onto the architecture

- Given our CNN and our two architectures, how do we map the computation?
- Example: How do we map this layer?



K	M	N
4	2	1

Mapping?



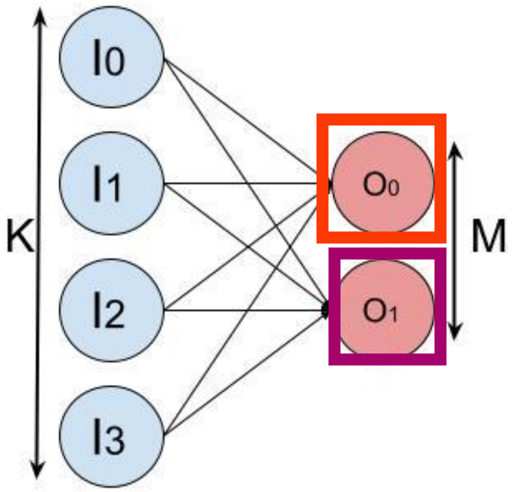
T_K	T_M	T_N
?	?	?

Virtual Neuron Size: ?

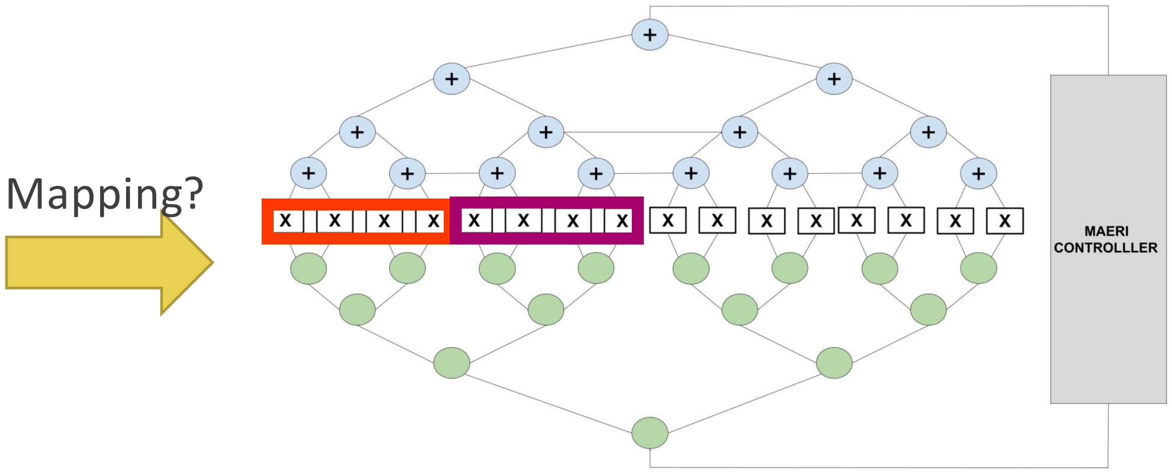
Number of Virtual Neurons: ?

Mapping the CNN onto the architecture

- Given our CNN and our two architectures, how do we map the computation?
- Example: How do we map this layer?



K	M	N
4	2	1



T_K	T_M	T_N
4	2	1

Virtual Neuron Size: 4

Number of Virtual Neurons: 2*1=2

Mapping the CNN onto the architecture

- Given our CNN and our architecture, how do we map the computation?

Layer	Operation	R	S	C	K	G	N	X	Y	
0	CONV	5	5	1	16	1	1	32	32	**Simulated**
1	MaxPool	2	2	16	16	1	1	28	28	
2	CONV	5	5	16	32	1	1	18	18	**Simulated**
3	MaxPool	2	2	32	32	1	1	14	14	
Layer	**Operation**	**K**	**M**	**N**						
4	Dense	1568	10	1		-	-	-	-	**Simulated**

Number of Multipliers	DN BW	RN BW
64	64	64

Mapping the CNN onto the architecture

Layer	Operation	R	S	C	K	G	N	X	Y	
0	CONV	5	5	1	16	1	1	32	32	**Simulated**
1	MaxPool	2	2	16	16	1	1	28	28	
2	CONV	5	5	16	32	1	1	18	18	**Simulated**
3	MaxPool	2	2	32	32	1	1	14	14	
Layer	**Operation**	**K**	**M**	**N**						
4	Dense	1568	10	1		-	-	-	-	**Simulated**

Number of Multipliers	DN BW	RN BW
64	64	64

Layer	Operation	T_R	T_S	T_C	T_K	T_G	T_N	T_X'	T_Y'	VN Size	# VNs	MSs used
0	CONV	?	?	?	?	?	?	?	?	?	?	?
2	CONV	?	?	?	?	?	?	?	?	?	?	?
Layer	**Operation**	**T_K**	**T_M**	**T_N**								
4	Dense	?	?	?	?	?	?	?	?	?	?	?

Mapping the CNN onto the architecture

Layer	Operation	R	S	C	K	G	N	X	Y	
0	CONV	5	5	1	16	1	1	32	32	**Simulated**
1	MaxPool	2	2	16	16	1	1	28	28	
2	CONV	5	5	16	32	1	1	18	18	**Simulated**
3	MaxPool	2	2	32	32	1	1	14	14	
Layer	**Operation**	**K**	**M**	**N**						
4	Dense	1568	10	1		-	-	-	-	**Simulated**

Number of Multipliers	DN BW	RN BW
64	64	64

Layer	Operation	T_R	T_S	T_C	T_K	T_G	T_N	T_X'	T_Y'	VN Size	# VNs	MSs used
0	CONV	5	5	1	2	1	1	1	1	25	2	50
2	CONV	?	?	?	?	?	?	?	?	?	?	?
Layer	**Operation**	**T_K**	**T_M**	**T_N**								
4	Dense	?	?	?	?	?	?	?	?	?	?	?

Mapping the CNN onto the architecture

Layer	Operation	R	S	C	K	G	N	X	Y	
0	CONV	**5**	**5**	1	**16**	1	1	32	32	**Simulated**
1	MaxPool	2	2	16	16	1	1	28	28	
2	CONV	5	5	**16**	**32**	1	1	18	18	**Simulated**
3	MaxPool	2	2	32	32	1	1	14	14	
Layer	**Operation**	**K**	**M**	**N**						
4	Dense	1568	10	1		-	-	-	-	**Simulated**

Number of Multipliers	DN BW	RN BW
64	64	64

Layer	Operation	T_R	T_S	T_C	T_K	T_G	T_N	T_X'	T_Y'	VN Size	# VNs	MSs used
0	CONV	**5**	**5**	1	**2**	1	1	1	1	25	2	50
2	CONV	1	1	**16**	**4**	1	1	1	1	16	4	64
Layer	**Operation**	**T_K**	**T_M**	**T_N**								
4	Dense	?	?	?	?	?	?	?	?	?	?	?

Mapping the CNN onto the architecture

Layer	Operation	R	S	C	K	G	N	X	Y	
0	CONV	**5**	**5**	1	**16**	1	1	32	32	**Simulated**
1	MaxPool	2	2	16	16	1	1	28	28	
2	CONV	5	5	**16**	**32**	1	1	18	18	**Simulated**
3	MaxPool	2	2	32	32	1	1	14	14	
Layer	**Operation**	**K**	**M**	**N**						
4	Dense	**1568**	**10**	1		-	-	-	-	**Simulated**

Number of Multipliers	DN BW	RN BW
64	64	64

Layer	Operation	T_R	T_S	T_C	T_K	T_G	T_N	T_X'	T_Y'	VN Size	# VNs	MSs used
0	CONV	**5**	**5**	1	**2**	1	1	1	1	25	2	50
2	CONV	1	1	**16**	**4**	1	1	1	1	16	4	64
Layer	**Operation**	**T_K**	**T_M**	**T_N**								
4	Dense	**32**	**2**	1	-	-	-	-	-	32	2	64

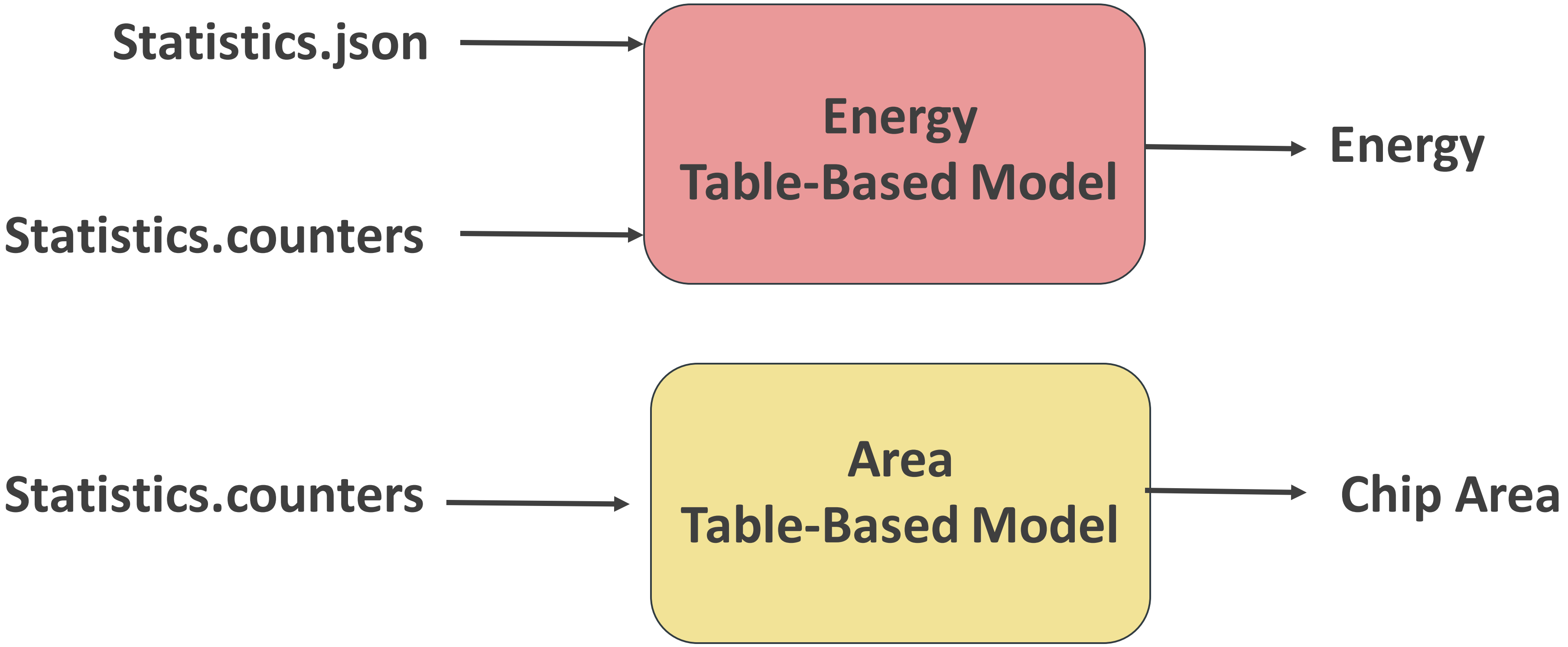
Simulating using the STONNE User interface

- Before running the real benchmark we will use the STONNE User interface to simulate our layers.
- This will run the layers with the correct dimensions but using random data.

```
[PC@User $] ./stonne -CONV -R=3 -S=3 -C=1 -G=1 -K=1 -N=1 -X=4 -Y=4 -T_R=3 -T_S=3 -T_C=1
-T_G=1 -T_K=1 -T_N=1 -T_X_=1 -T_Y_=1 -num_ms=64 -dn_bw=64 -rn_bw=64
[PC@User $] Running CONV layer in STONNE Simulator...
[PC@User $] Output files generated correctly.
```

Simulating using the STONNE User interface

- Once the simulation has finished we can get the energy and area numbers:



Simulating using the STONNE User interface

- Script to calculate the energy and area numbers:
  - /home/stonne_omega:
    - omega
    - stonne:
      - ASPLOS22
      - pytorch-frontend
        - stonne_connection
      - **stonne**:
        - stonne.elf
        - Include
        - src
        - external
        - stonne_linker_src
        - **energy_tables**
          - **calculate_energy.py**
          - **energy_model.txt**

Simulating using the STONNE User interface

- Syntax: **./calculate_energy** [-v] **-table_file**=<*Energy&Area table*> **-counter_file**=<*counter stats*> **-out_file**=<*output file name*>

Simulating using the Pytorch interface

- Our image digit recognition benchmark is located in */home/stonne_omega/stonne/ASPLOS22/handson-1*
  - **train_dnn.py**: Used to train our CNN and get the trained weights.
  - **evaluate_dnn.py**: Used to evaluate the accuracy of our CNN.
  - **dnn.py**: Defines our CNN.
  - **deployment.py**: This is the final file which is used to predict real images.

Simulating using the Pytorch interface

- Steps towards CPU execution:
  - 1. Run train_dnn.py script to obtain the weights.
  - 2. Run evaluate_dnn.py to measure the accuracy of the network.
  - 3. Run deployment.py

Simulating using the Pytorch interface

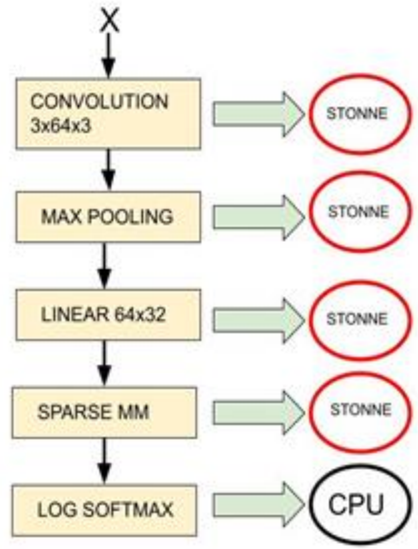
- Steps towards CPU execution:
  - 1. Run train_dnn.py script to obtain the weights.
  - 2. Run evaluate_dnn.py to measure the accuracy of the network.
  - 3. Run deployment.py

Simulating using the Pytorch interface

- Once we have tested the inference procedure on the CPU, let's run it in our simulated accelerator.
- Steps towards simulation:
  - 1. Set up the hardware configuration file.
  - 2. Set up the tile configuration files
  - 3. Change the CONV and Linear operations in order to simulate them.

```
import torch
import torch.nn as nn
import torch.nn.functional as F

class My_DNN(nn.Module):
  def __init__(self):
    super().__init__()
    self.conv = nn.Simulatedconv2d(3, 64, 3, conf='stonne_hw.cfg')
    self.max = nn.SimulatedMaxPool2d(kernel_size=3, conf='stonne_hw.cfg')
    self.fc = nn.SimulatedLinear(64, 32, conf='stonne_hw.cfg')
  def forward(self, x, y):
    x = self.conv(x) #simulated
    x = self.max(x) #simulated
    x = self.fc(x) #simulated
    x = F.simulated_sparse_mm(x,y,conf='stonne_hw.cfg')#simulated
    x = F.log_softmax(x, dim=1) #on cpu
    return x
```



Outline

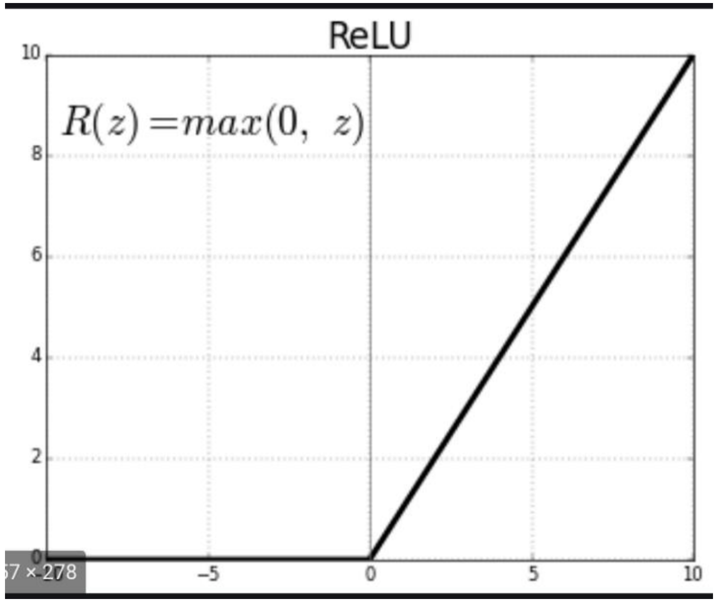
- Docker Image, Installation and overview of STONNE.
- Hands-on #1: Simulating a real DNN on Flexible DNN Accelerators.
- Hands-on #2: Using STONNE to evaluate a research use case – Exploiting sparsity.
- Hands-on #3: Extending new operations in STONNE – Adding the Conv1d operation.
- Research use cases.
- Conclusions.

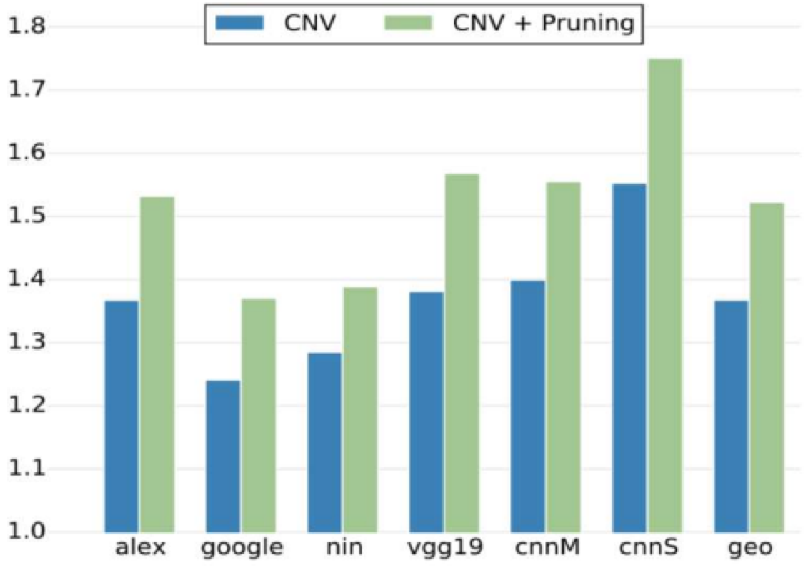
Objectives

- 1. Understand why running the simulation with the real numbers is of paramount importance.
- 2. Understand how to modify the hardware components in the STONNE simulator.
- Understand why STONNE can be useful to research.

Presence of zeros

- DNNs present a large number of zeros due to both prunning (weights) and the ReLU activation function (activations).

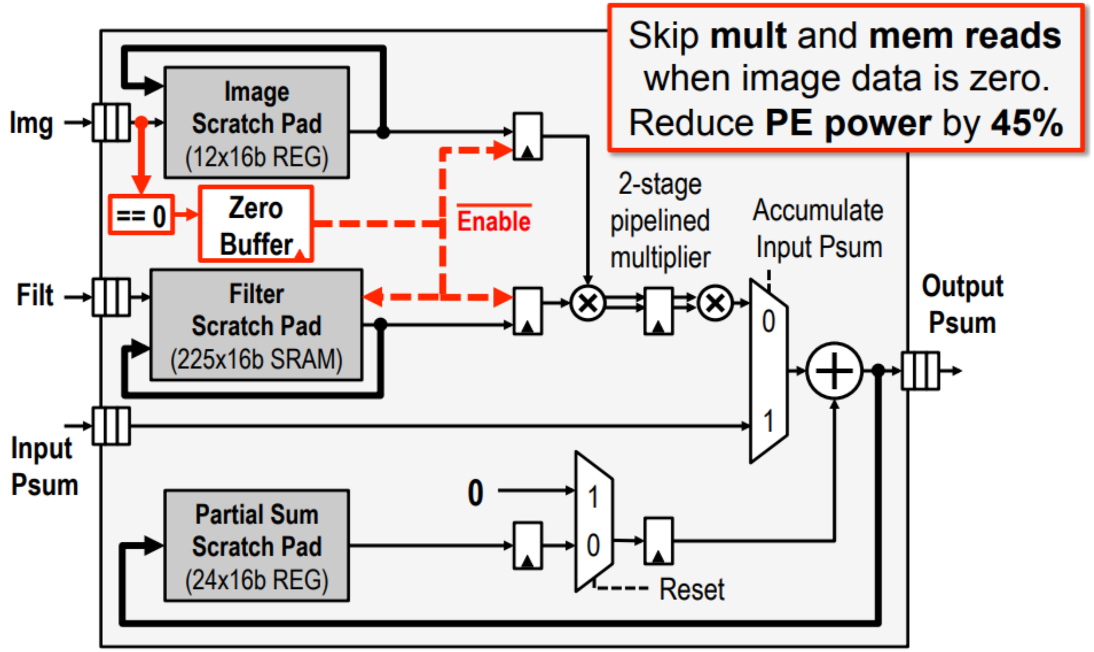




Cnvlutin [ISCA 2016]

Gate operations

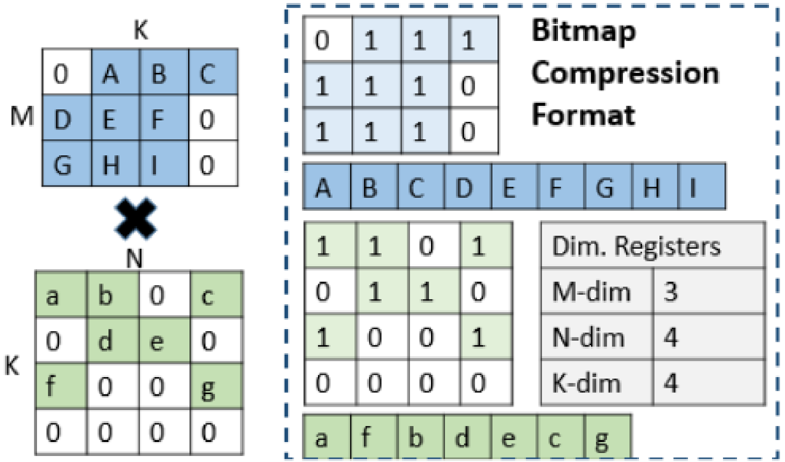
- Some accelerators leverage this feature to save energy by skiping the computation involving the zeros:
  - Any multiplication involving a zero is zero.
  - Output = i*w + p



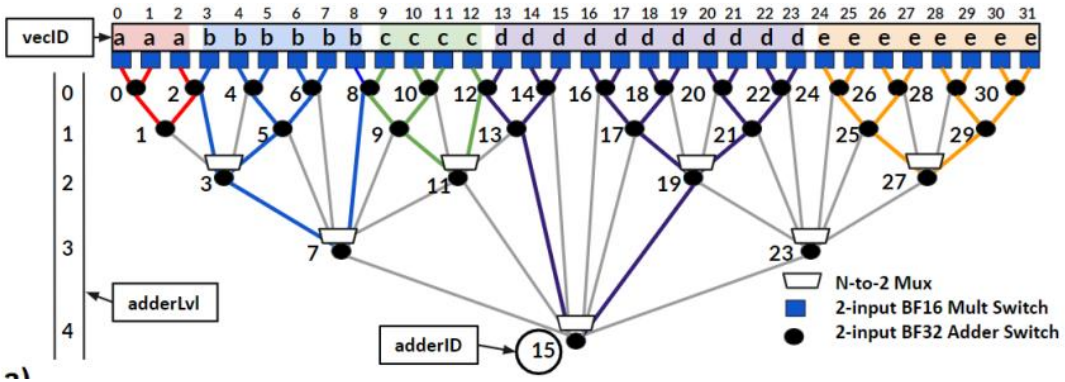
Eyeriss [ISSCC 2016]

Compression

- Other accelerators compress the zeros (e.g., using a CSR format or bitmap) and avoid useless transfers from memory.
- e.g., SIGMA:



SIGMA [HPCA 2020]



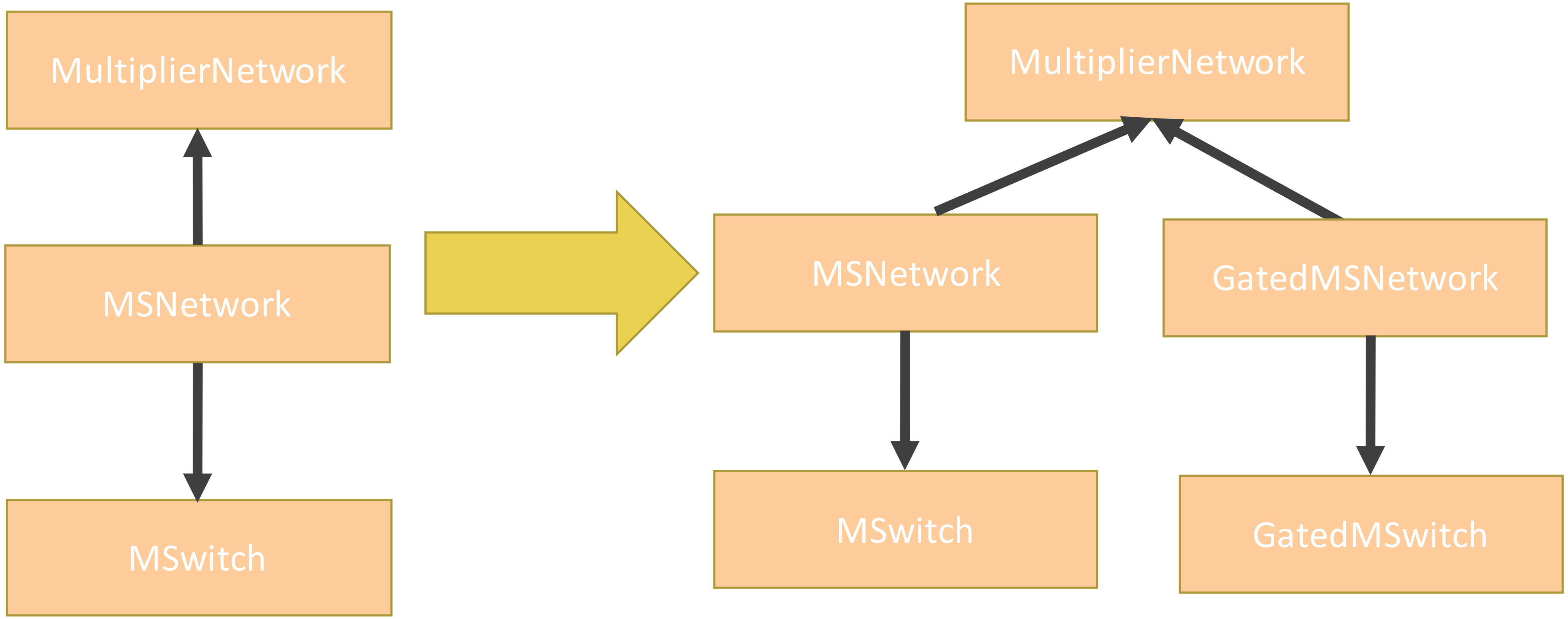
SIGMA [HPCA 2020]

Research use case

- In this use case we will evaluate the impact of the gate operation technique when running our benchmark.
- Sparse executions will be explored in OMEGA.
- Gate operations (i.e., avoid computing zeros) are not yet implemented.

Research use case

The idea is to create another multiplier network with multipliers implementing this feature.



Research use case

- ¿How many operations do we compute?
- ¿How can we add a new stat tracking the number of operations that are skiped?
- ¿What is the percentage of operations that are skiped?
- ¿What implications does it have on both performance and energy?

Outline

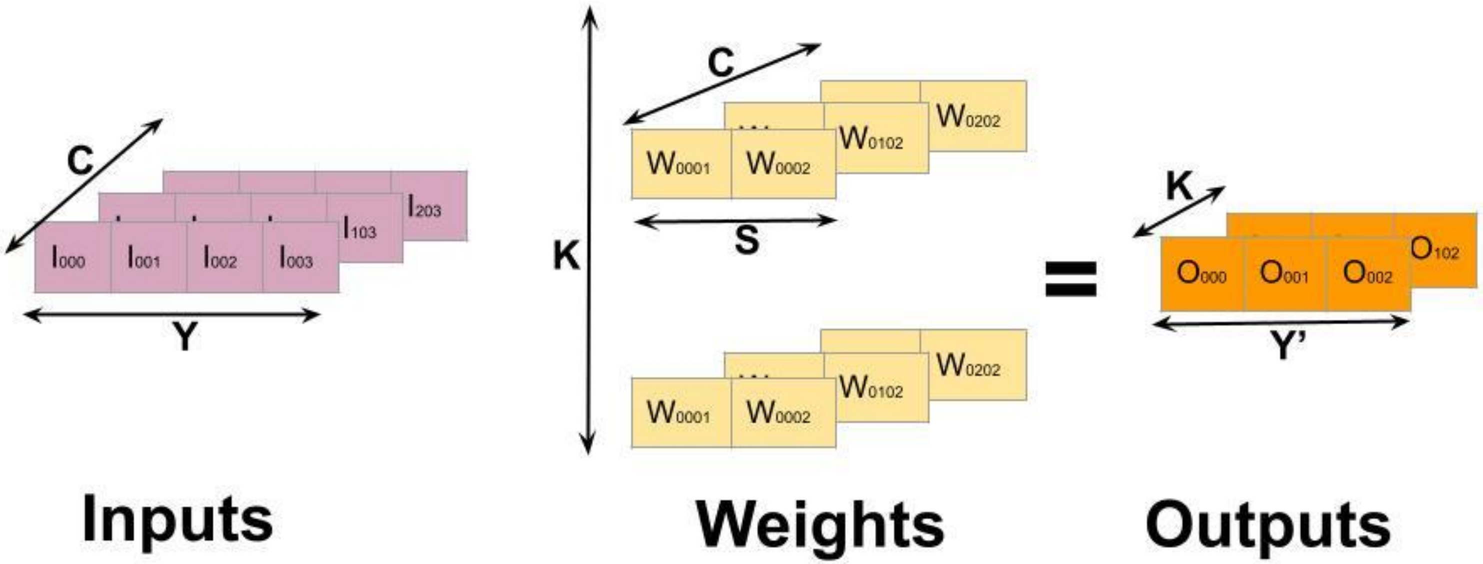
- Docker Image, Installation and overview of STONNE.
- Hands-on #1: Simulating a real DNN on Flexible DNN Accelerators.
- Hands-on #2: Using STONNE to evaluate a research use case – Exploiting sparsity.
- Hands-on #3: Extending new operations in STONNE – Adding the Conv1d operation.
- Research use cases.
- Conclusions.

Adding the conv1d operation

- At this point, we have seen how the operations nn.Linear and nn.Conv2d can be simulated in STONNE.
- What if we want to integrate a new operation and how does the Pytorch-STONNE connection work?

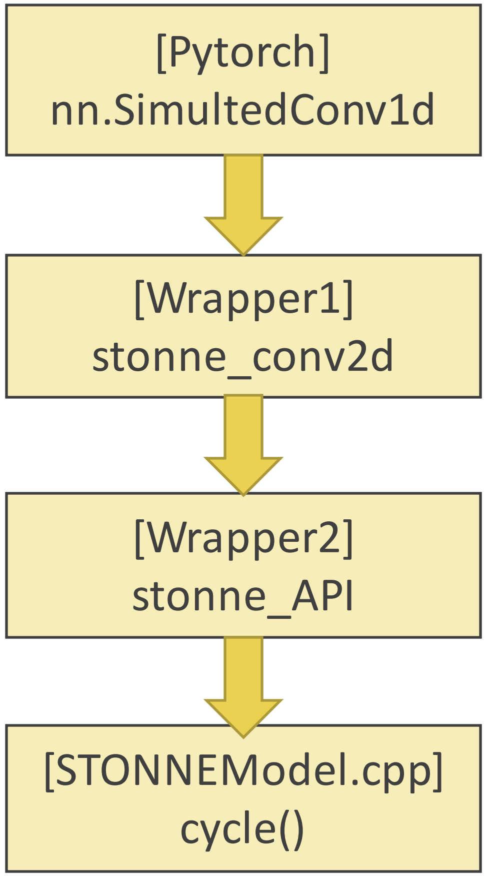
Adding the conv1d operation

- The nn.conv1d operation is not integrated with STONNE.



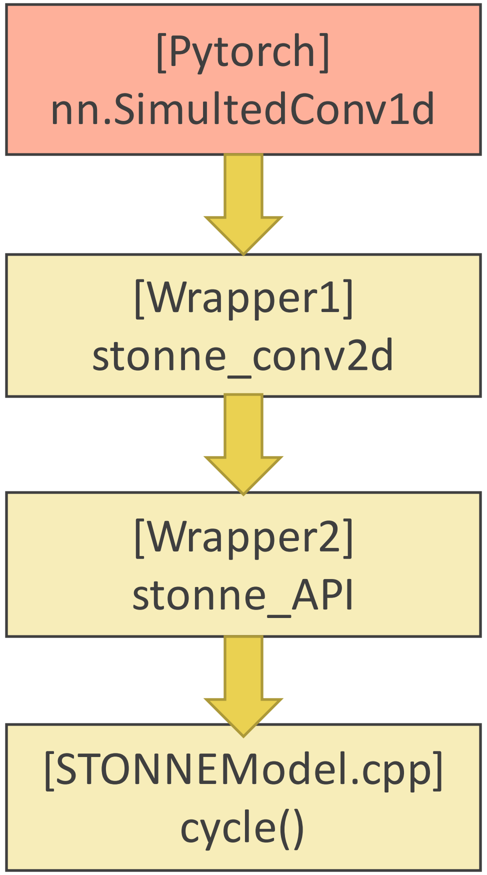
Adding the conv1d operation

- During this exercise, we will see how to connect the conv1d operation to the convolution operation implemented in STONNE.



Adding the conv1d operation

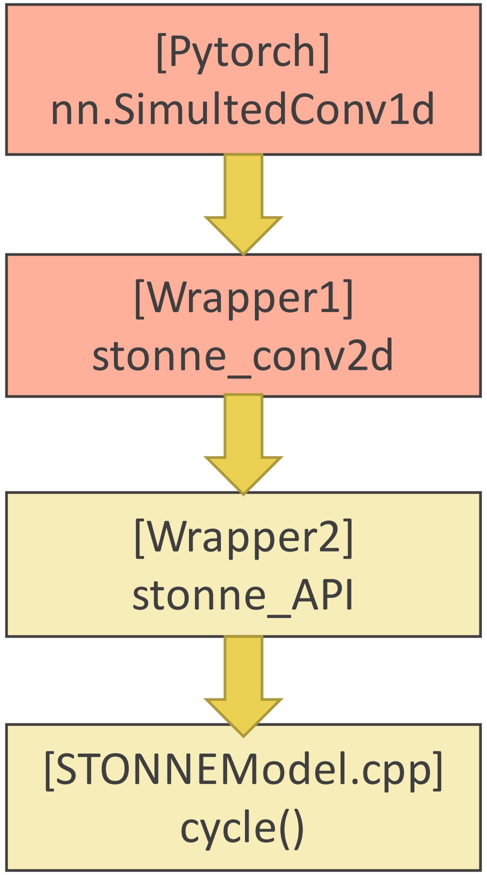
- During this exercise, we will see how to connect the conv1d operation to the convolution operation implemented in STONNE.



First we will create the operation within the **nn** package in Pytorch.

Adding the conv1d operation

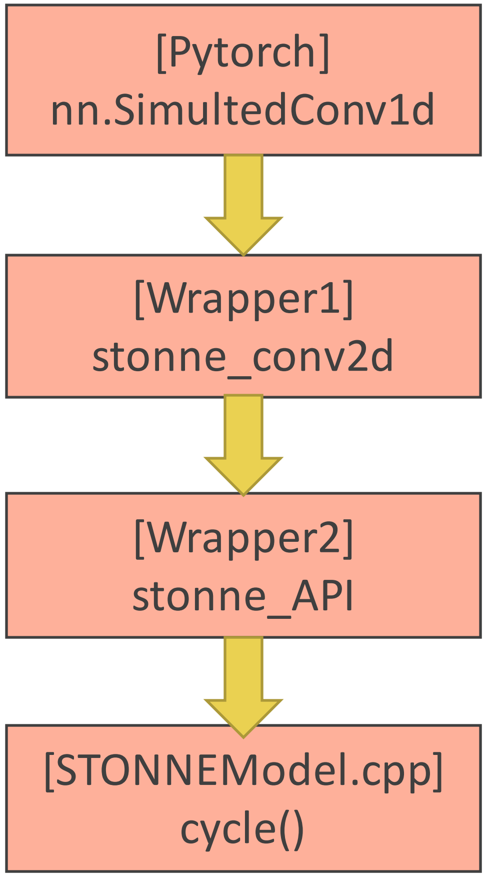
- During this exercise, we will see how to connect the conv1d operation to the convolution operation implemented in STONNE.



After this, we need to create the c++ function that connects with the simulator.

Adding the conv1d operation

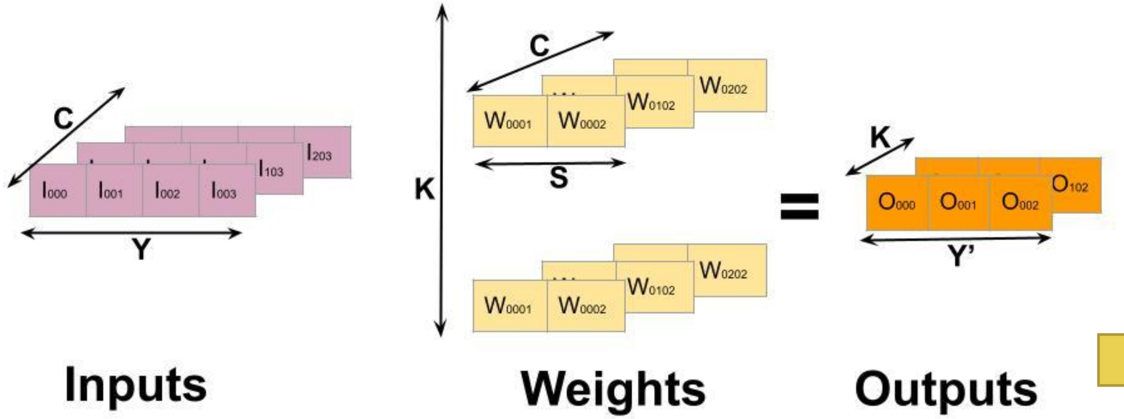
- During this exercise, we will see how to connect the conv1d operation to the convolution operation implemented in STONNE.

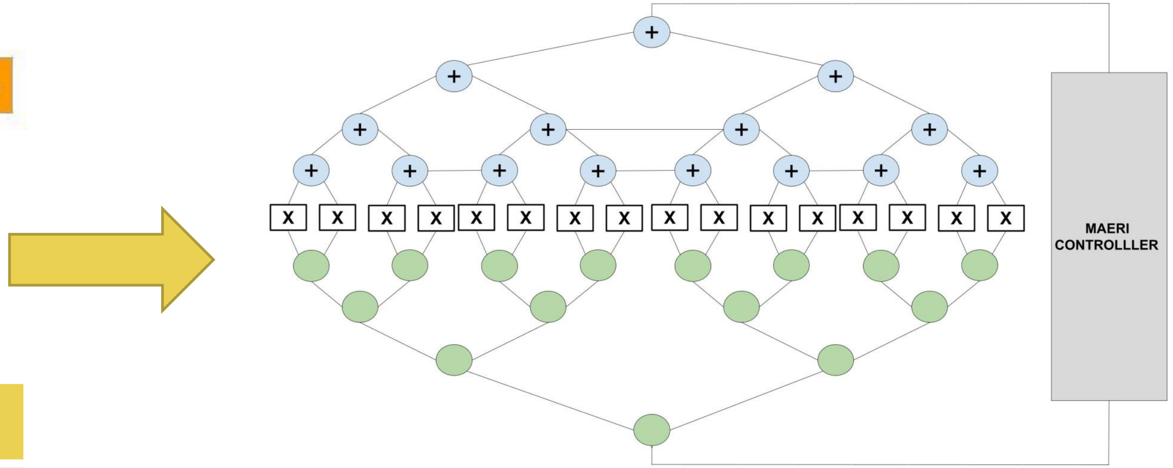


The STONNE back-end remains intact as the simulator already supports the convolution operation.

Adding the conv1d operation

- After implementing the operation we will use it to run the next CONV1D operation on our 64-MS architecture:





Layer	Operation	S	C	K	G	N	Y
0	CONV1D	2	64	1	1	1	32

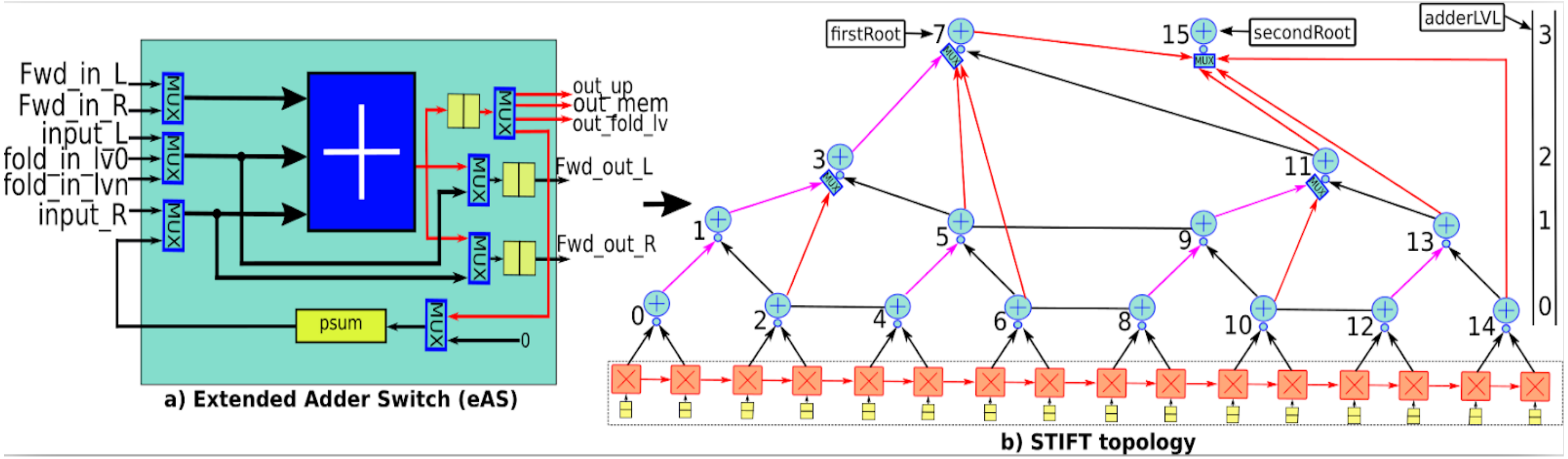
T_S	T_C	T_K	T_G	T_N	T_Y
1	64	1	1	1	1

Outline

- Docker Image, Installation and overview of STONNE.
- Hands-on #1: Simulating a real DNN on Flexible DNN Accelerators.
- Hands-on #2: Using STONNE to evaluate a research use case – Exploiting sparsity.
- Hands-on #3: Extending new operations in STONNE – Adding the Conv1d operation.
- Research use cases.
- Conclusions.

Use case #1: STIFT

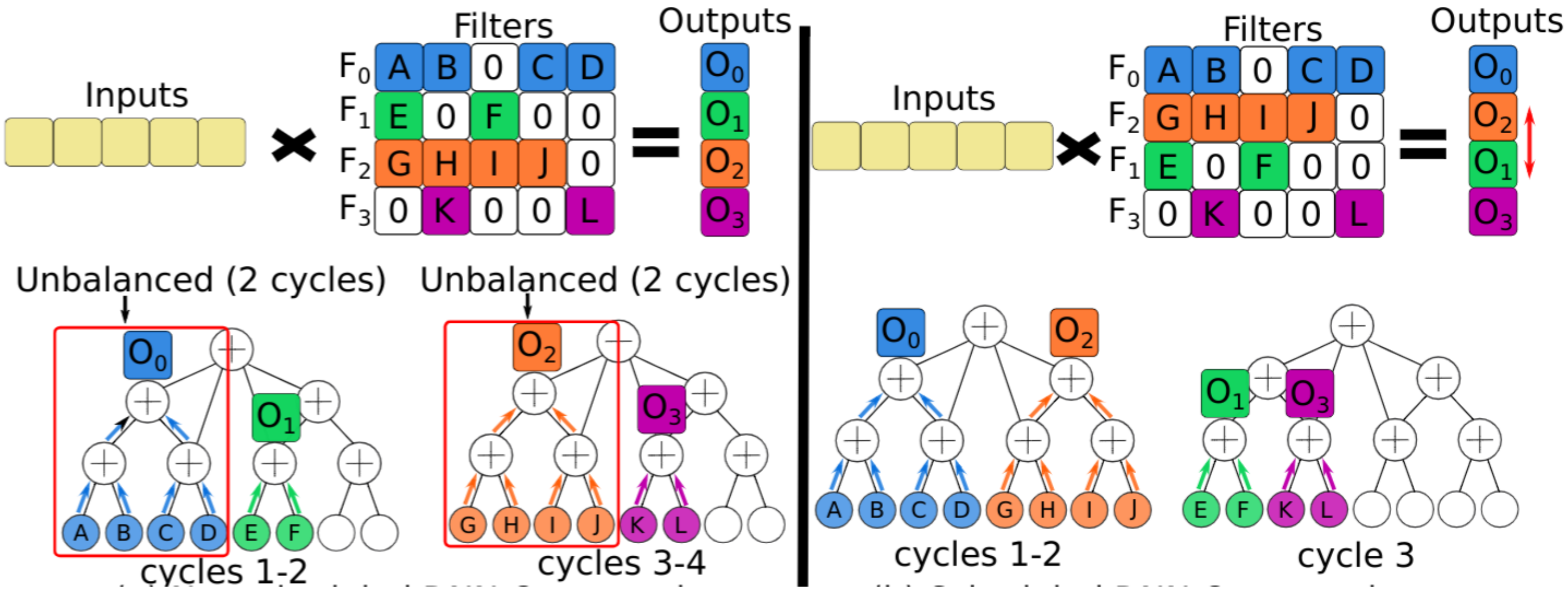
- We can use STONNE to create and evaluate new architectural modules for ML accelerators.



STIFT [Francisco Munoz-Martinez et. al., NOCS21]

Use case #2: Sparse filters scheduling

- We can use STONNE to evaluate new compilation techniques such as sparse scheduling.



STONNE [Francisco Munoz-Martinez et. al., IISWC21]

Outline

- Docker Image, Installation and overview of STONNE.
- Hands-on #1: Simulating a real DNN on Flexible DNN Accelerators.
- Hands-on #2: Using STONNE to evaluate a research use case – Exploiting sparsity.
- Hands-on #3: Extending new operations in STONNE – Adding the Conv1d operation.
- Research use cases.
- Conclusions.

Conclusions

- STONNE is a cycle-accurate simulator for ML accelerators.
- During this hands-on we have seen:
  - How to use the STONNE user interface to run synthetic operations.
  - How to use the Pytorch interface to run end-to-end models with STONNE.
  - How to extend the architecture of STONNE.
  - How to connect new operations and frameworks to the STONNE simulator.

Find STONNE on http://github.com/stonne_simulator/stonne

Georgia Tech

UNIVERSIDAD DE MURCIA

UCAM UNIVERSIDAD CATÓLICA DE MURCIA

thank you

STONNE Hands-on

Francisco Muñoz-Martínez

francisco.munoz2@um.es

Universidad de Murcia (UM)