ONNC–CIM | An End-to-End Crossbar-Based CIM Simulator

Jen-ho Kuo & Yu-Jie Wang | May 2020

Skymizer Taiwan Inc.

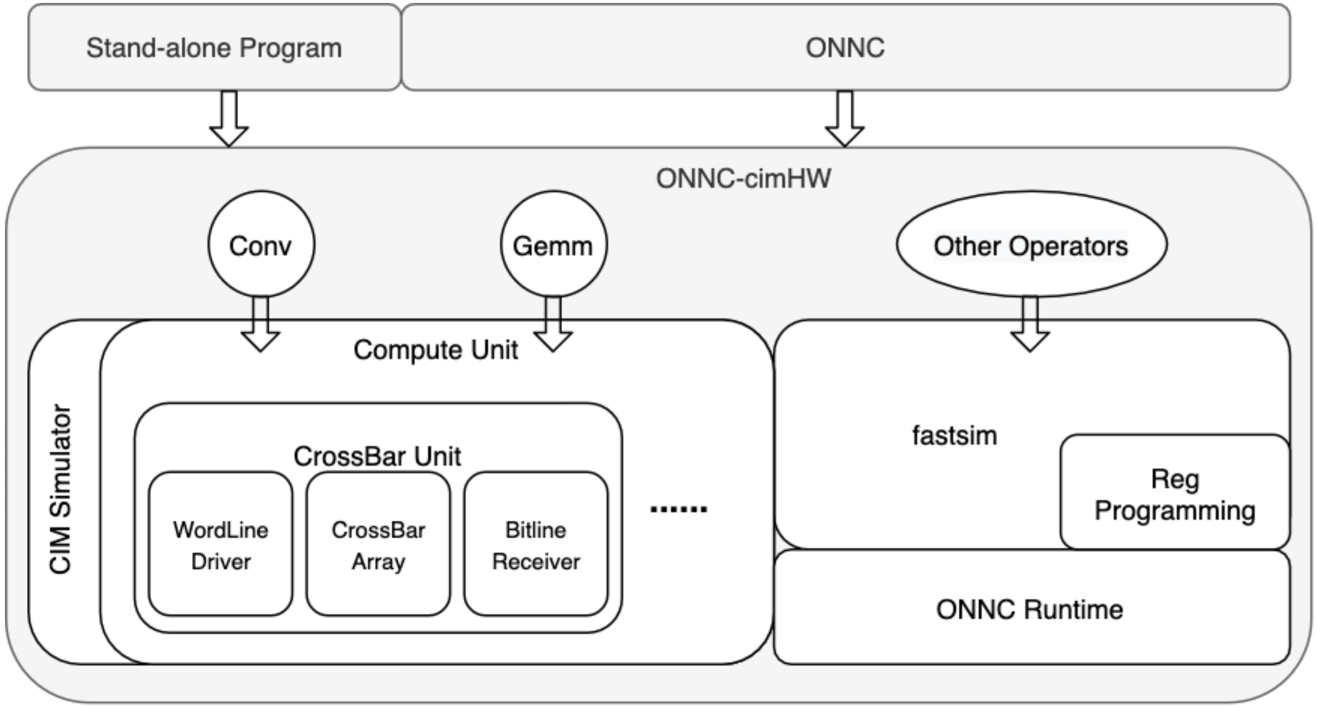
A Joint Work with the Embedded Computing Lab @ CSIE, National Taiwan University

Meng-Yao Lin, Wei-Ting Lin, Tzu-Hsien Yang, I-Ching Tseng, Hsiang-Yun Cheng, Chia-Lin Yang

Outline

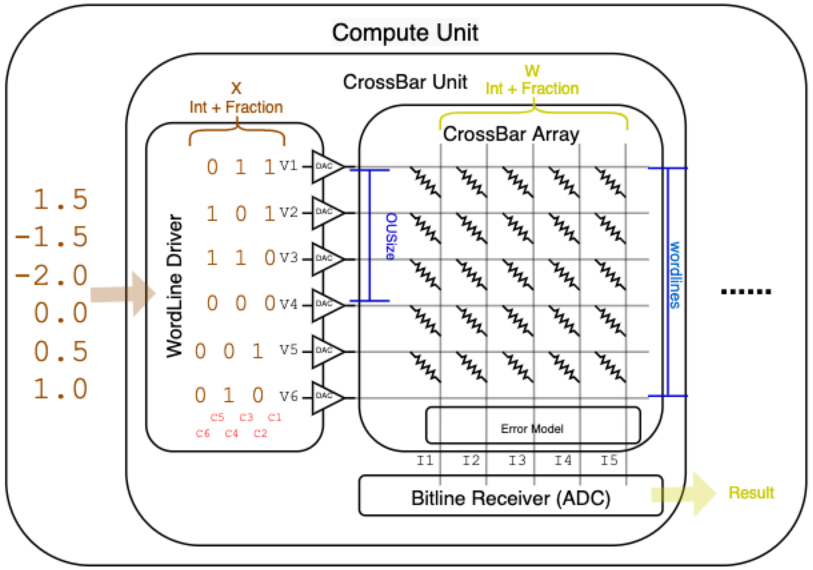
- Project overview
- Device Error Model
- How to build and run ONNC-CIM
- How to create a user application in the stand-alone mode
- How to support a new operator in ONNC-CIM

Project Overview



- Support ONNX Operator v1.3.0
- Support full-model simulation
  - CrossBar device model for **Conv** and **Gemm**
  - Fastsim (behavior model) for **Other operators**
- Usages
  - Build as a library in the **Stand-alone** mode
  - Build with an ONNC frontend in the **ONNC** mode
- Extendibility
  - Support adding new operators that use crossbar devices
  - A unified JSON format for device model

CrossBar Unit



- CIM simulator
  - Compute Unit has several Crossbar Units
  - Crossbar Unit has
    - WordLine Driver (DAC)
    - Crossbar Array
    - Bitline Receiver (ADC)
- CrossBar Array
  - The resistor could be configured by vector W with given quantized parameters
- WordLine Driver
  - Repeatedly accepts vector X and quantizes into given length of bits
- Bitline Receiver
  - For each cycle, wordline driver electrifies maximum OUSize of wordlines of bits into crossbar array
  - Amplification will be sensed with error model simulation
  - Final result is then added and shifted cycle-by-cycle

How to Build and Run ONNC-CIM

- Prerequisite
  - Docker
  - ONNC Community Docker image

- Clone the onnc-cim project from GitHub

```
$ git clone https://github.com/onnc/onnc-cim.git && cd onnc-cim
```

- Run the container with

```
$ docker run -it -v `pwd`:/onnc/onnc --cap-add=SYS_PTRACE --security-opt seccomp=unconfined
onnc/onnc-community:latest
```

Step to Build onnc-cim in Stand-alone mode

1. Create build folder and compile onnc-cim

```
$ mkdir -p /onnc/onnc-cimHW.build

$ cd /onnc/onnc-cimHW.build
$ cmake -DCMAKE_BUILD_TYPE=Release /onnc/onnc/skysim/onnc-cimHW
$ make -j8
```

2. Install the built library

```
$ sudo make install
$ sudo ldconfig
```

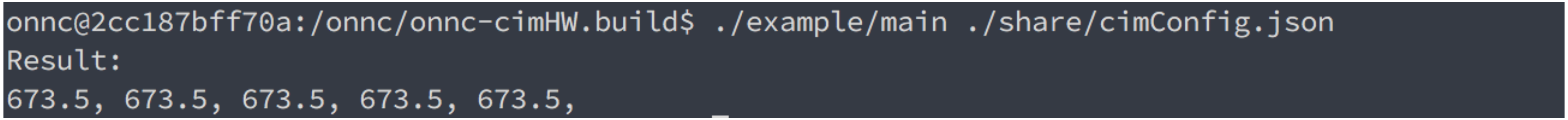
Build Options

- DCMAKE_BUILD_TYPE
  - DCMAKE_BUILD_TYPE=Release
  - DCMAKE_BUILD_TYPE=Debug
- DBUILD_TESTS=ON
  - Test cases will be built to a single test_all program in tests folder.
  - ./tests/test_all share/cimConfig.json

Testing onnc-cimHW in stand-alone mode

- You may find a test program, ”main”, in the example folder
- Run simulation with a device configuration file, ”cimConfig.json”

  ```
  $ ./example/main ./share/cimConfig.json
  ```



```
onnc@2cc187bff70a:/onnc/onnc-cimHW.build$ ./example/main ./share/cimConfig.json
Result:
673.5, 673.5, 673.5, 673.5, 673.5,
```

Step to Build onnc-cim in onnc mode

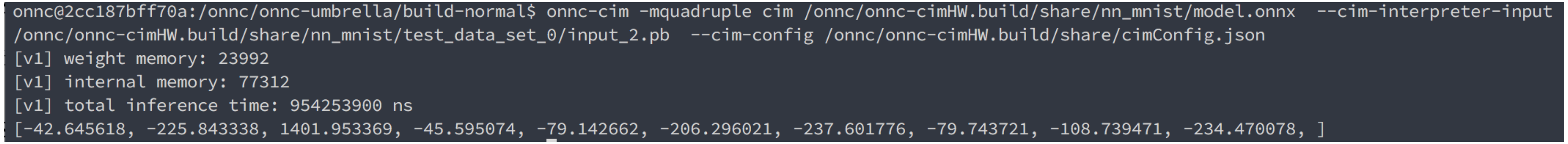
1. build onnc-cim and install the library in the docker container (page 10)
2. build onnc with the onnc-cim library

```
$ cd /onnc/onnc-umbrella/build-normal
$ smake -j8 install
```

Testing onnc-cim in onnc mode

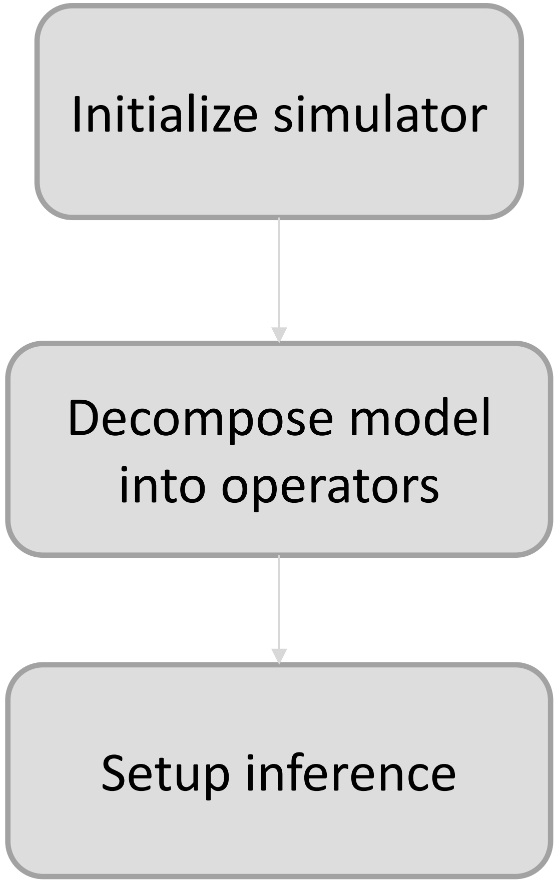
Example to use nn on MNIST

```
$ onnc-cim -mquadruple cim /onnc/onnc-cimHW.build/share/nn_mnist/model.onnx  --cim-interpreter-input /onnc/onnc-cimHW.build/share/nn_mnist/test_data_set_0/input_2.pb  --cim-config /onnc/onnc-cimHW.build/share/cimConfig.json
```



```
onnc@2cc187bff70a:/onnc/onnc-umbrella/build-normal$ onnc-cim -mquadruple cim /onnc/onnc-cimHW.build/share/nn_mnist/model.onnx  --cim-interpreter-input
/onnc/onnc-cimHW.build/share/nn_mnist/test_data_set_0/input_2.pb  --cim-config /onnc/onnc-cimHW.build/share/cimConfig.json
[v1] weight memory: 23992
[v1] internal memory: 77312
[v1] total inference time: 954253900 ns
[-42.645618, -225.843338, 1401.953369, -45.595074, -79.142662, -206.296021, -237.601776, -79.743721, -108.739471, -234.470078, ]
```

How to write a program for using the stand-alone library



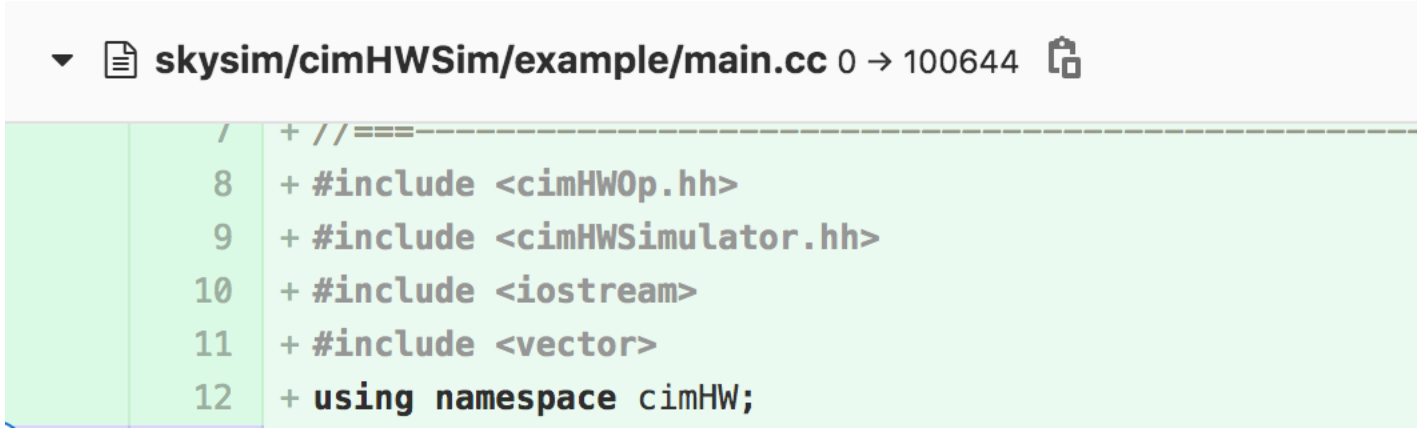
Initialize simulator

Decompose model into operators

Setup inference

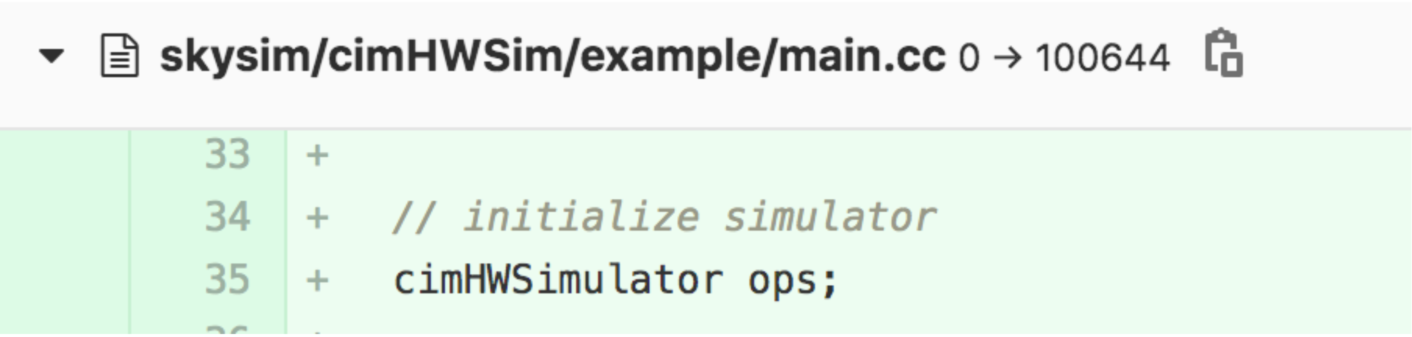
Initialize simulator

- /onnc/onnc/skysim/onnc-cimHW/example/main.cc



skysim/cimHWSim/example/main.cc 0 → 100644

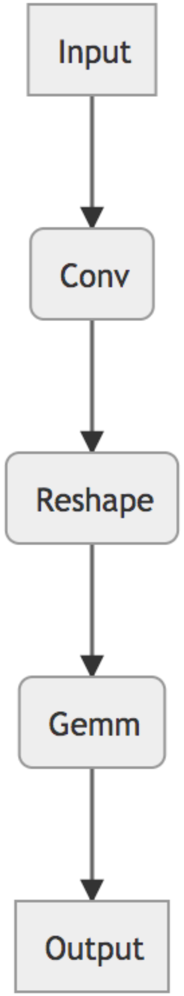
```
+ //===------------------------------------------------------
+ #include <cimHWOp.hh>
+ #include <cimHWSimulator.hh>
+ #include <iostream>
+ #include <vector>
+ using namespace cimHW;
```



skysim/cimHWSim/example/main.cc 0 → 100644

```
+
+   // initialize simulator
+   cimHWSimulator ops;
```

Decompose a model into operators



1x1x5x5

(1x3x5x5)
kernel=3x1x3x3, bias=1x3, pads=[1,1,1,1], strides=[1,1])

(1x75)
shape=[1,75]

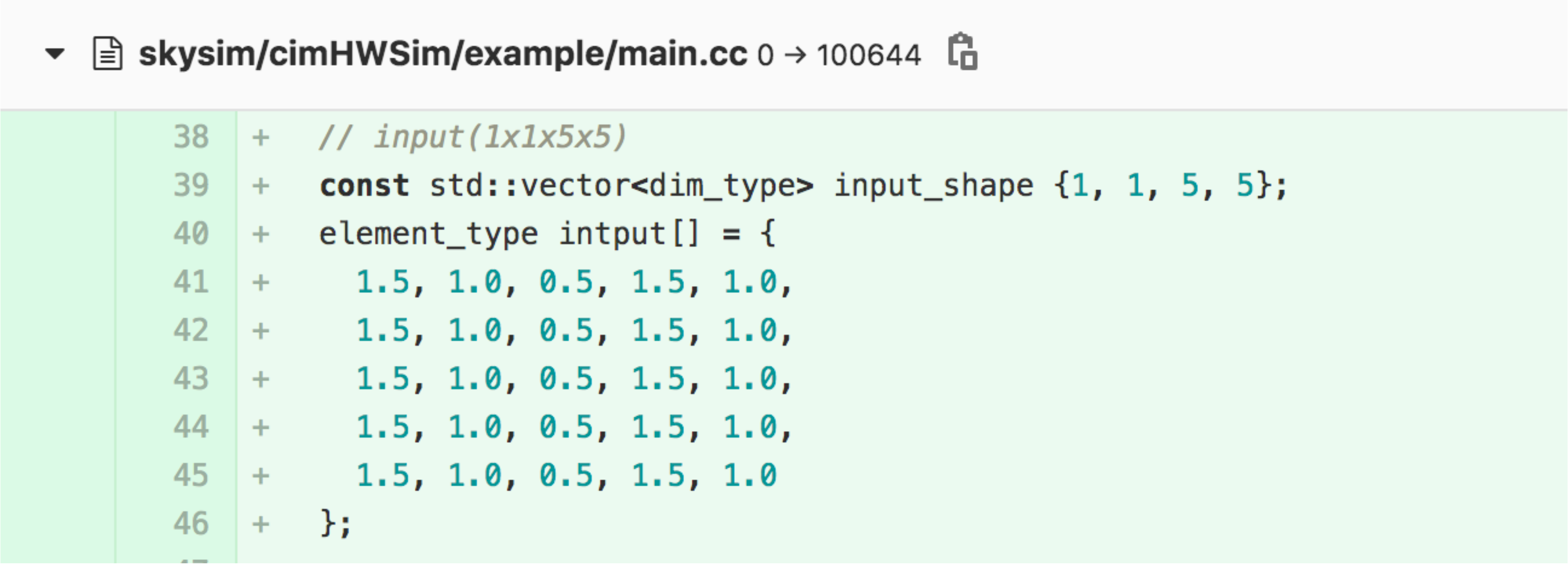
(1x5)
gemmB=5x75, transA=0, transB=1

Inference Setup

1. Prepare input, output, and activation tensors
2. Specify operator attributes
3. Call onnc-cimHW runtime API and get output tensors

Inference Setup

Prepare input, output, and activation tensors

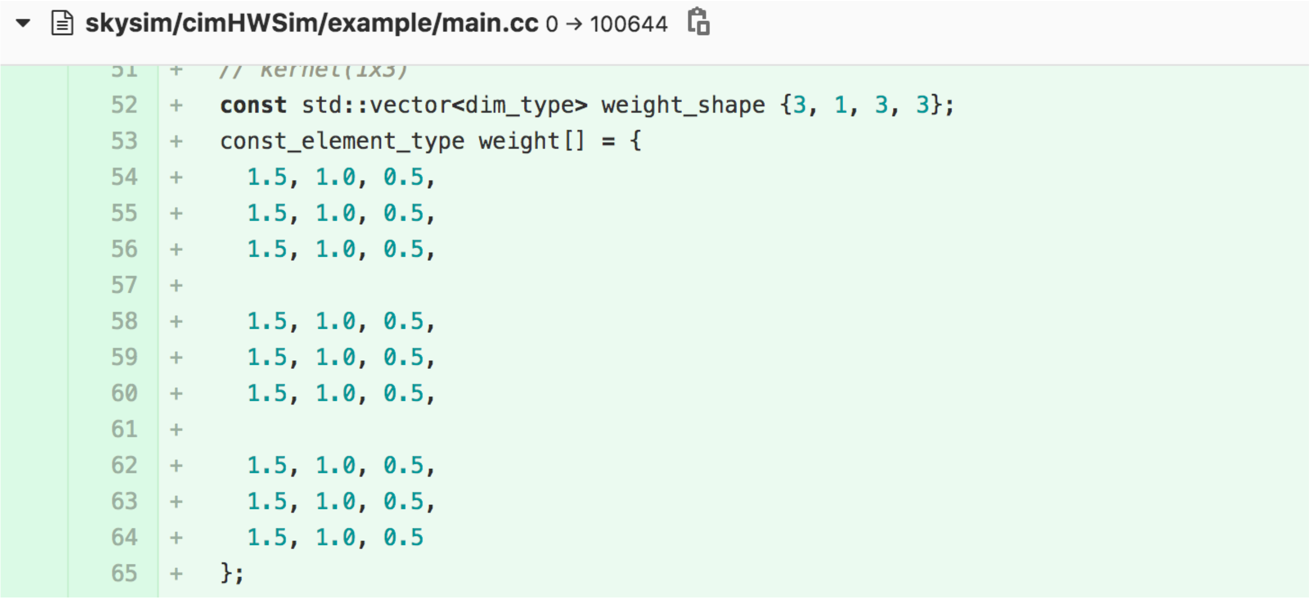


skysim/cimHWSim/example/main.cc 0 → 100644

```
+ // input(1x1x5x5)
+ const std::vector<dim_type> input_shape {1, 1, 5, 5};
+ element_type intput[] = {
+   1.5, 1.0, 0.5, 1.5, 1.0,
+   1.5, 1.0, 0.5, 1.5, 1.0,
+   1.5, 1.0, 0.5, 1.5, 1.0,
+   1.5, 1.0, 0.5, 1.5, 1.0,
+   1.5, 1.0, 0.5, 1.5, 1.0
+ };
```

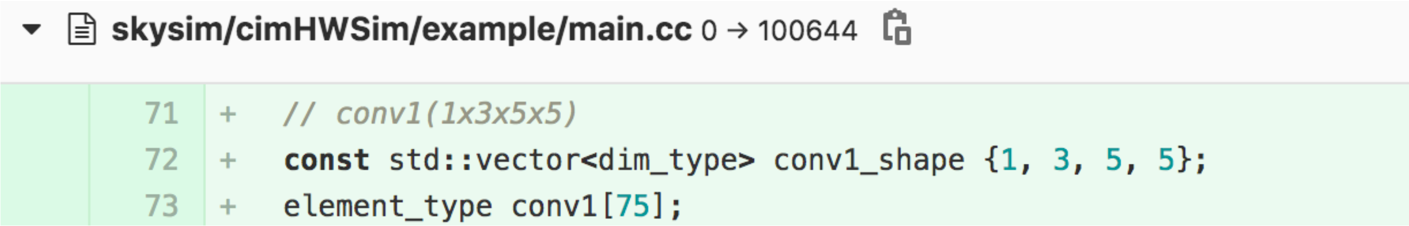
Inference Setup

Prepare input, output, and activation tensors



skysim/cimHWSim/example/main.cc 0 → 100644

```
+ // kernel(1x3)
+ const std::vector<dim_type> weight_shape {3, 1, 3, 3};
+ const_element_type weight[] = {
+   1.5, 1.0, 0.5,
+   1.5, 1.0, 0.5,
+   1.5, 1.0, 0.5,
+
+   1.5, 1.0, 0.5,
+   1.5, 1.0, 0.5,
+   1.5, 1.0, 0.5,
+
+   1.5, 1.0, 0.5,
+   1.5, 1.0, 0.5,
+   1.5, 1.0, 0.5
+ };
```

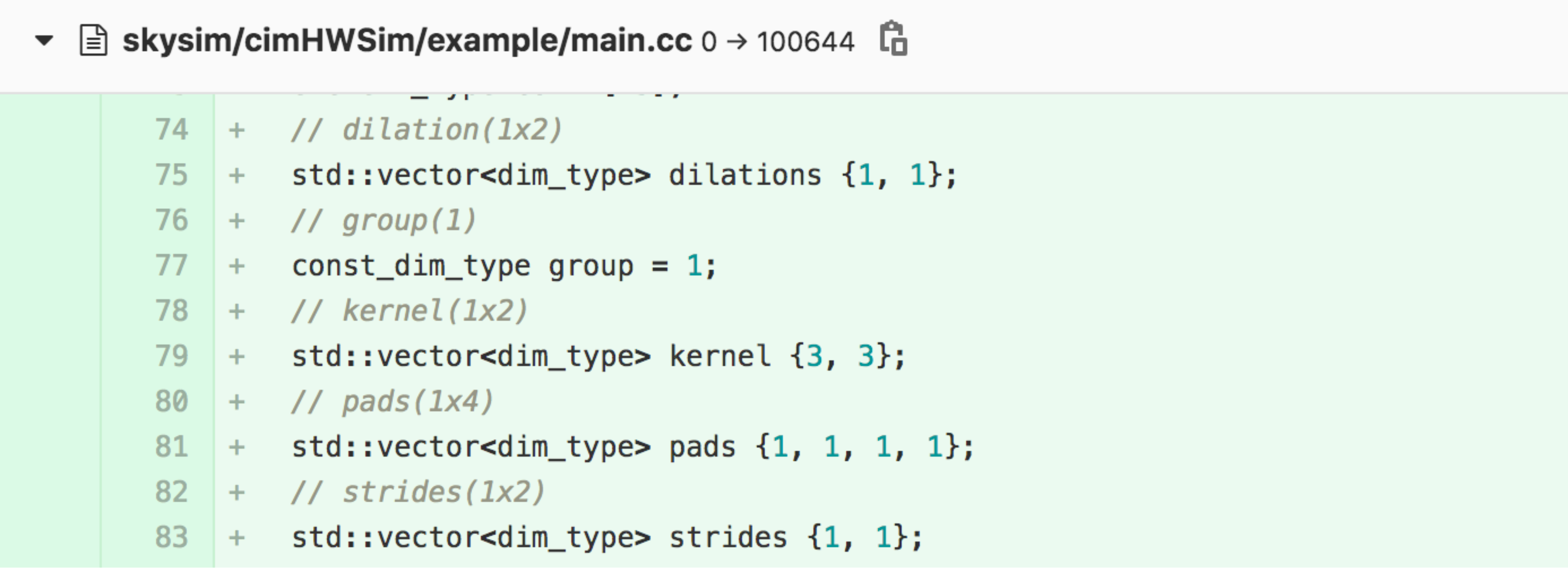


skysim/cimHWSim/example/main.cc 0 → 100644

```
+ // conv1(1x3x5x5)
+ const std::vector<dim_type> conv1_shape {1, 3, 5, 5};
+ element_type conv1[75];
```

Inference Setup

Specify operator attributes

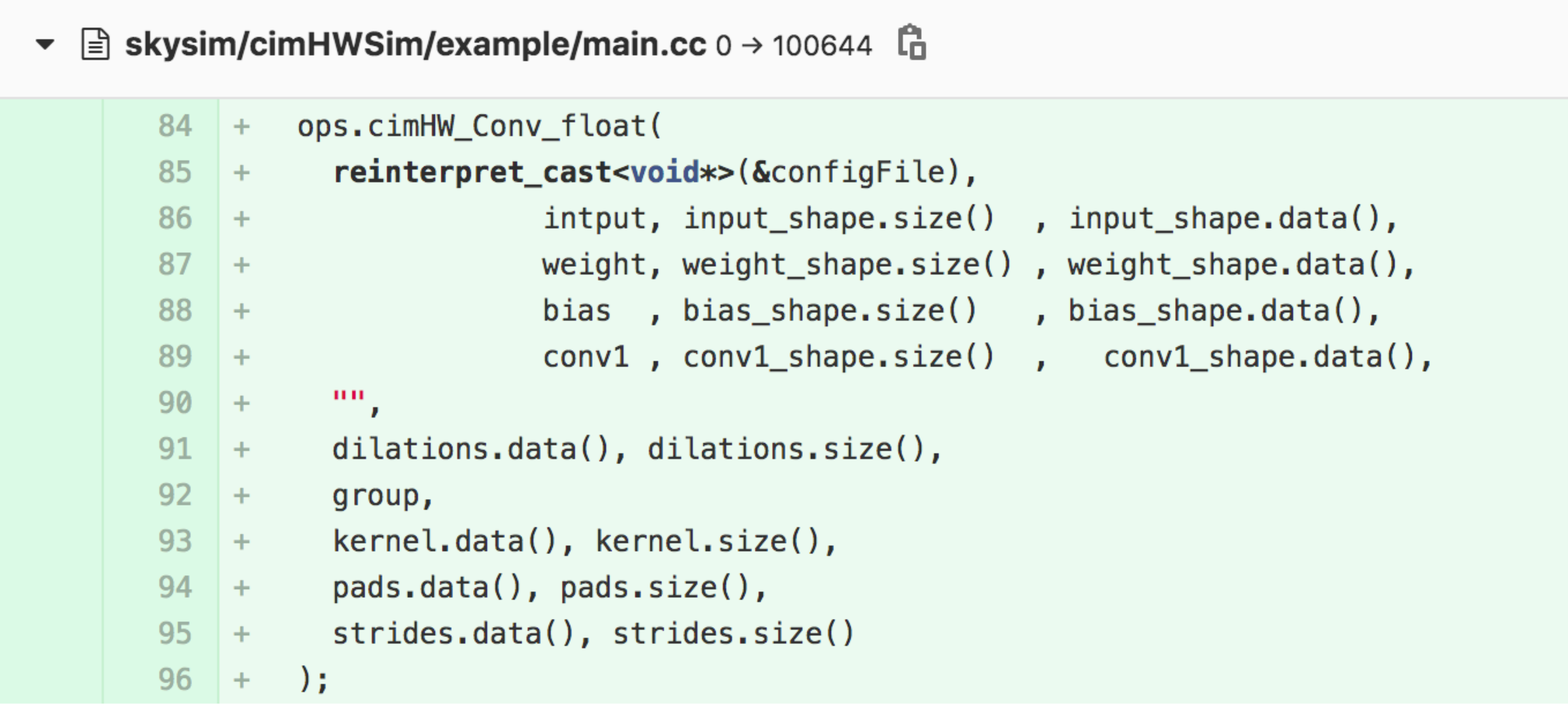


skysim/cimHWSim/example/main.cc 0 → 100644

```
+   // dilation(1x2)
+   std::vector<dim_type> dilations {1, 1};
+   // group(1)
+   const_dim_type group = 1;
+   // kernel(1x2)
+   std::vector<dim_type> kernel {3, 3};
+   // pads(1x4)
+   std::vector<dim_type> pads {1, 1, 1, 1};
+   // strides(1x2)
+   std::vector<dim_type> strides {1, 1};
```

Inference Setup

Call onnc-cimHW function and get output



skysim/cimHWSim/example/main.cc 0 → 100644

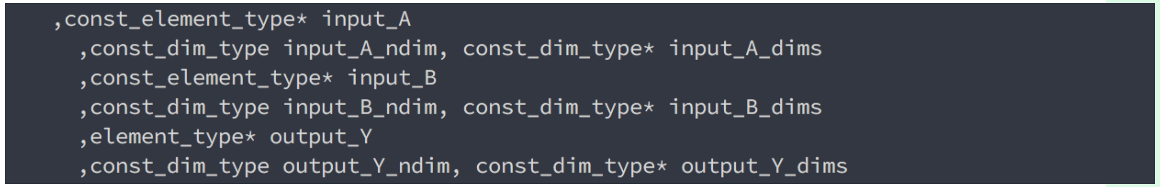
```
+ ops.cimHW_Conv_float(
+   reinterpret_cast<void*>(&configFile),
+               intput, input_shape.size()  , input_shape.data(),
+               weight, weight_shape.size() , weight_shape.data(),
+               bias  , bias_shape.size()   , bias_shape.data(),
+               conv1 , conv1_shape.size()  ,   conv1_shape.data(),
+   "",
+   dilations.data(), dilations.size(),
+   group,
+   kernel.data(), kernel.size(),
+   pads.data(), pads.size(),
+   strides.data(), strides.size()
+ );
```

How to implement a new operator

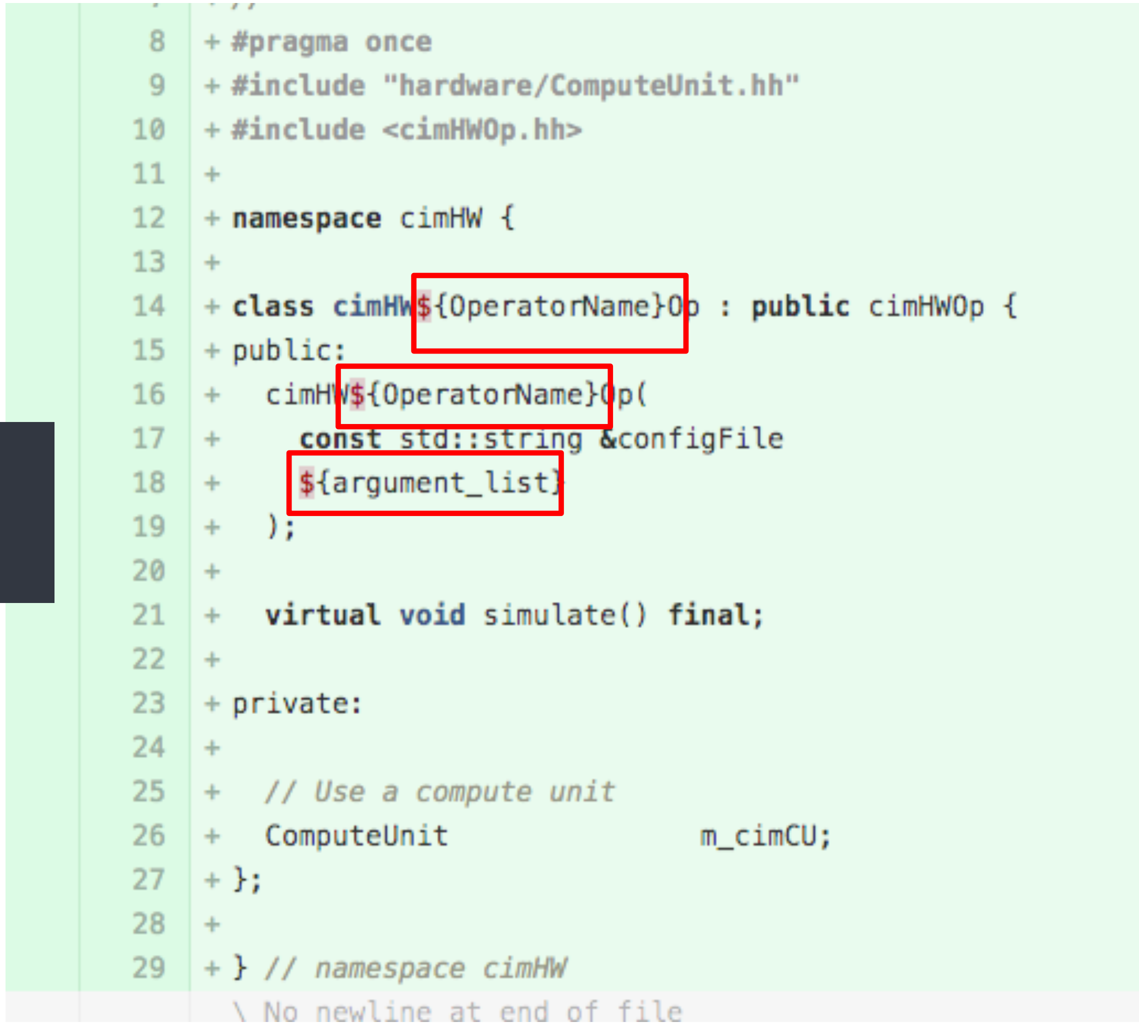
1. Create a new operator class from the template
2. Register to desired factory class
3. Add the new file to the building system
4. Rebuild

Create a new operator class from the template

- `$ cp lib/op.template.hh lib/matmul.hh`
- replace ${OperatorName} => MatMul
- replace ${argument_list}
  - skysim/onnc-cimHW/include/abstractOpFactory.hh
    - 2nd parameter of create${OperatorName}Op function



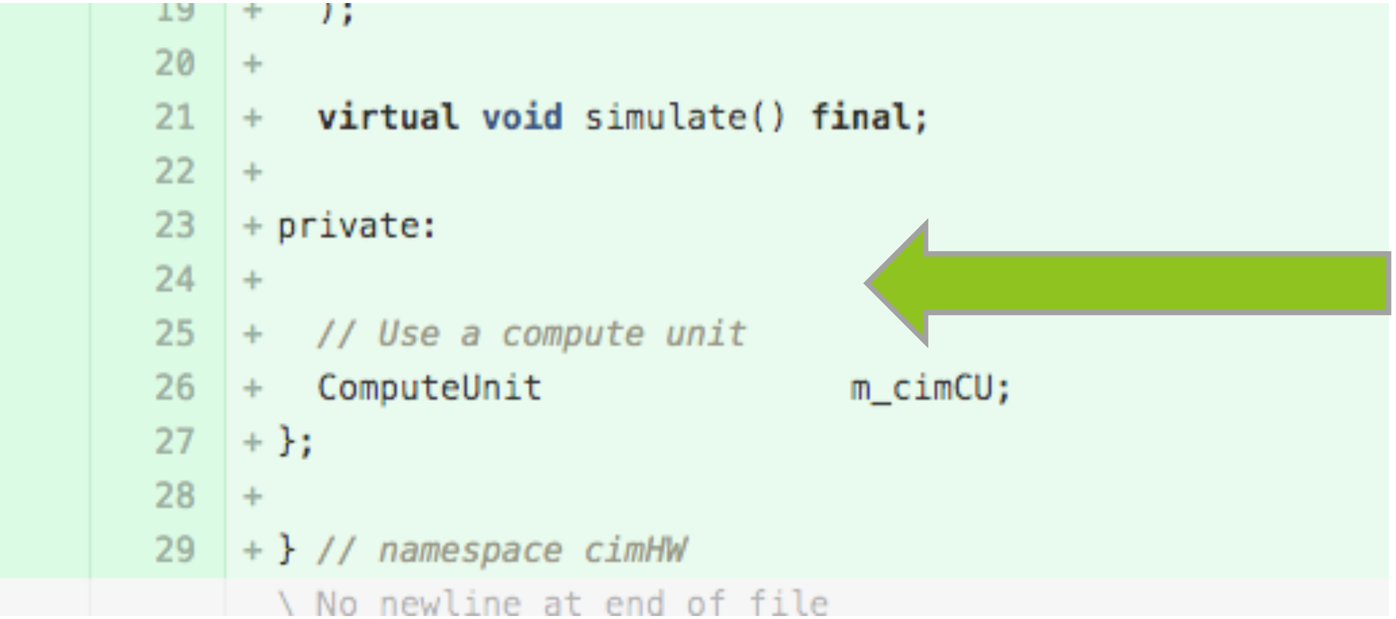
```
  ,const_element_type* input_A
    ,const_dim_type input_A_ndim, const_dim_type* input_A_dims
    ,const_element_type* input_B
    ,const_dim_type input_B_ndim, const_dim_type* input_B_dims
    ,element_type* output_Y
    ,const_dim_type output_Y_ndim, const_dim_type* output_Y_dims
```



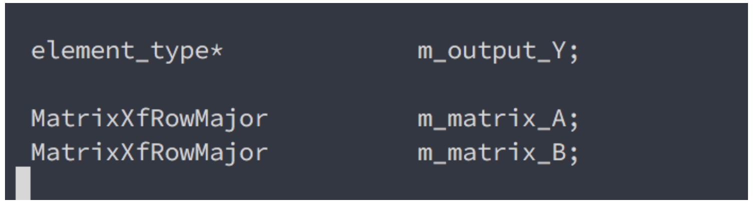
```
+ */
+ #pragma once
+ #include "hardware/ComputeUnit.hh"
+ #include <cimHWOp.hh>
+
+ namespace cimHW {
+
+ class cimHW${OperatorName}Op : public cimHWOp {
+ public:
+   cimHW${OperatorName}Op(
+     const std::string &configFile
+     ${argument_list}
+   );
+
+   virtual void simulate() final;
+
+ private:
+
+   // Use a compute unit
+   ComputeUnit              m_cimCU;
+ };
+
+ } // namespace cimHW
  \ No newline at end of file
```

Create a new operator class from the template

- op.template.hh
  - Add member for computing & output



```
+   );
+
+   virtual void simulate() final;
+
+ private:
+
+   // Use a compute unit
+   ComputeUnit                    m_cimCU;
+ };
+
+ } // namespace cimHW
  \ No newline at end of file
```

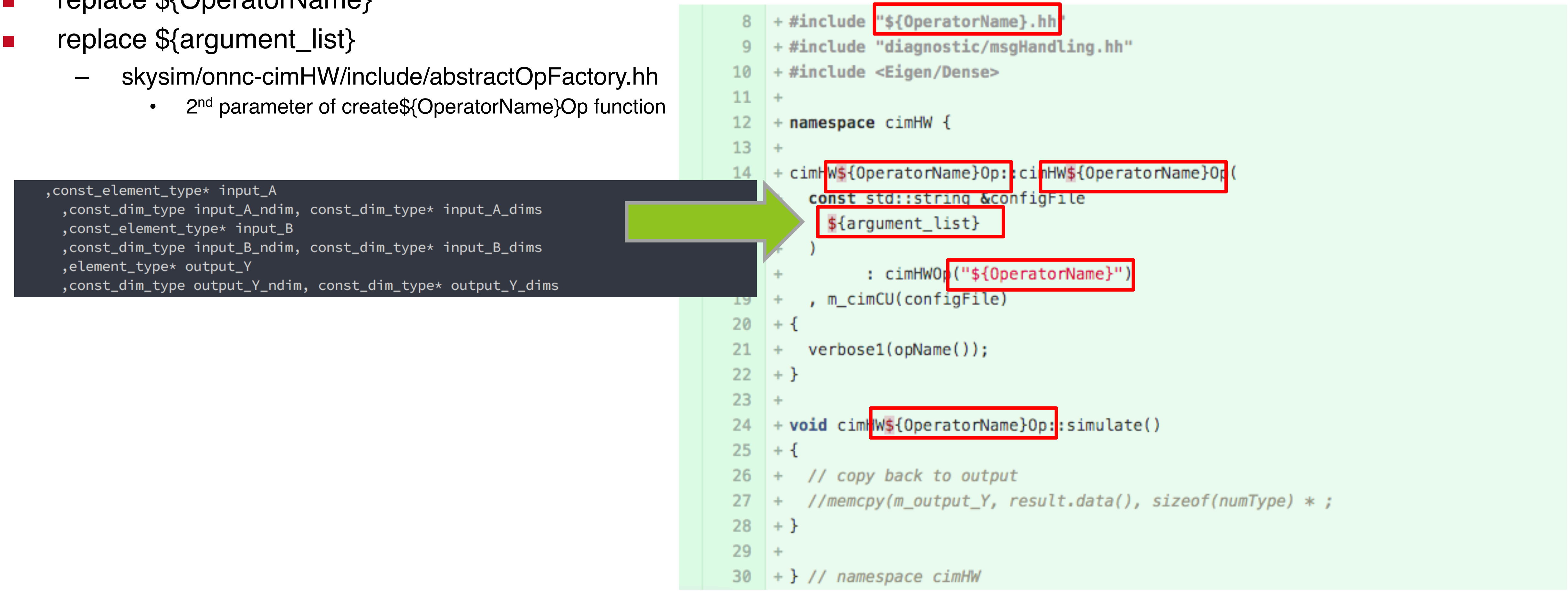


```
element_type*                 m_output_Y;

MatrixXfRowMajor              m_matrix_A;
MatrixXfRowMajor              m_matrix_B;
```

Create a new operator class from the template

- `$ cp lib/op.template.cc lib/matmul.cc`
- replace ${OperatorName}
- replace ${argument_list}
  - skysim/onnc-cimHW/include/abstractOpFactory.hh
    - 2nd parameter of create${OperatorName}Op function

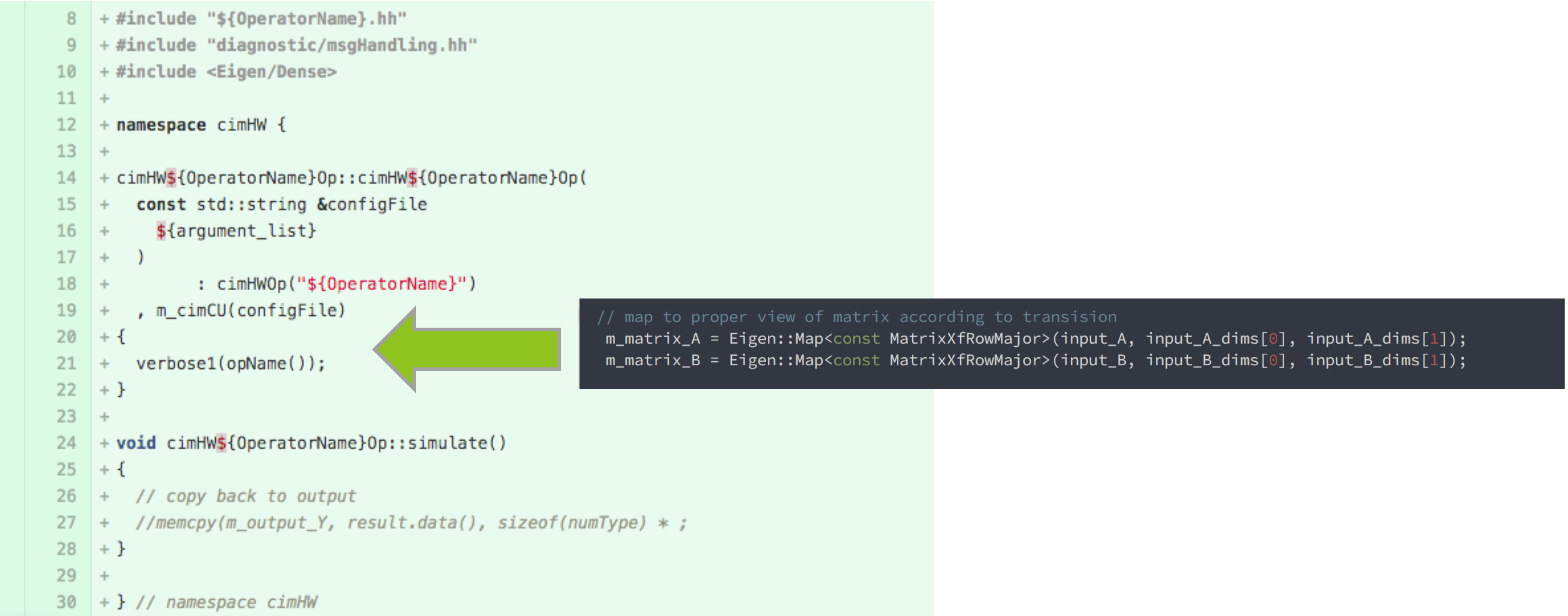


```
,const_element_type* input_A
  ,const_dim_type input_A_ndim, const_dim_type* input_A_dims
  ,const_element_type* input_B
  ,const_dim_type input_B_ndim, const_dim_type* input_B_dims
  ,element_type* output_Y
  ,const_dim_type output_Y_ndim, const_dim_type* output_Y_dims
```

```
+ #include "${OperatorName}.hh"
+ #include "diagnostic/msgHandling.hh"
+ #include <Eigen/Dense>
+
+ namespace cimHW {
+
+ cimHW${OperatorName}Op::cimHW${OperatorName}Op(
    const std::string &configFile
    ${argument_list}
    )
+         : cimHWOp("${OperatorName}")
+   , m_cimCU(configFile)
+ {
+   verbose1(opName());
+ }
+
+ void cimHW${OperatorName}Op::simulate()
+ {
+   // copy back to output
+   //memcpy(m_output_Y, result.data(), sizeof(numType) * ;
+ }
+
+ } // namespace cimHW
```

Create a new operator class from the template

- op.template.cc

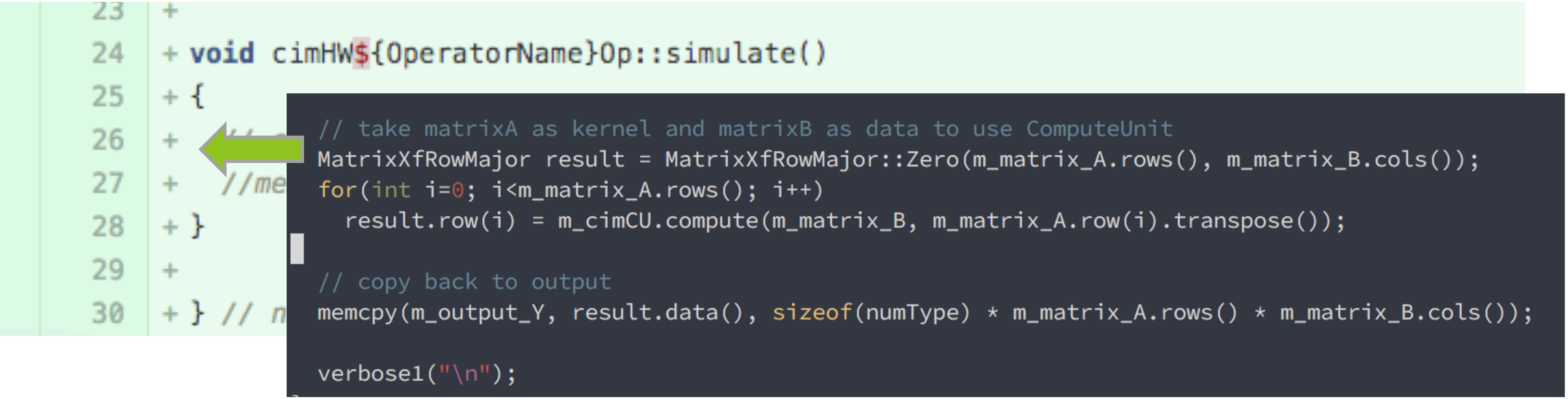


```
+ #include "${OperatorName}.hh"
+ #include "diagnostic/msgHandling.hh"
+ #include <Eigen/Dense>
+
+ namespace cimHW {
+
+ cimHW${OperatorName}Op::cimHW${OperatorName}Op(
+   const std::string &configFile
+     ${argument_list}
+   )
+         : cimHWOp("${OperatorName}")
+   , m_cimCU(configFile)
+ {
+   verbose1(opName());
+ }
+
+ void cimHW${OperatorName}Op::simulate()
+ {
+   // copy back to output
+   //memcpy(m_output_Y, result.data(), sizeof(numType) * ;
+ }
+
+ } // namespace cimHW
```

```
// map to proper view of matrix according to transision
 m_matrix_A = Eigen::Map<const MatrixXfRowMajor>(input_A, input_A_dims[0], input_A_dims[1]);
 m_matrix_B = Eigen::Map<const MatrixXfRowMajor>(input_B, input_B_dims[0], input_B_dims[1]);
```

Create a new operator class from the template

- op.template.cc



```
+
+ void cimHW${OperatorName}Op::simulate()
+ {
+
+    //me
+ }
+
+ } // n
```

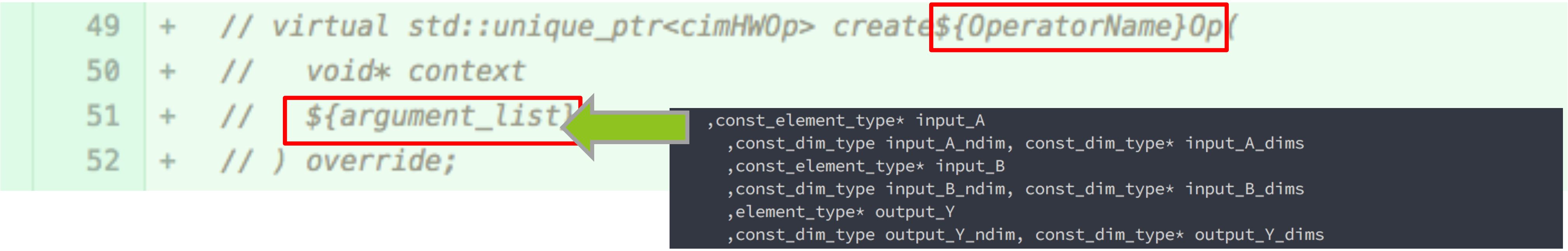
```
// take matrixA as kernel and matrixB as data to use ComputeUnit
MatrixXfRowMajor result = MatrixXfRowMajor::Zero(m_matrix_A.rows(), m_matrix_B.cols());
for(int i=0; i<m_matrix_A.rows(); i++)
  result.row(i) = m_cimCU.compute(m_matrix_B, m_matrix_A.row(i).transpose());

// copy back to output
memcpy(m_output_Y, result.data(), sizeof(numType) * m_matrix_A.rows() * m_matrix_B.cols());

verbose1("\n");
```

Register to desired factory class

- skysim/onnc-cimHW/lib/factory/cimHWOpFactory.hh
  - Add declaration
  - replace ${argument_list} skysim/onnc-cimHW/include/abstractOpFactory.hh
    - 2nd parameter of create${OperatorName}Op function

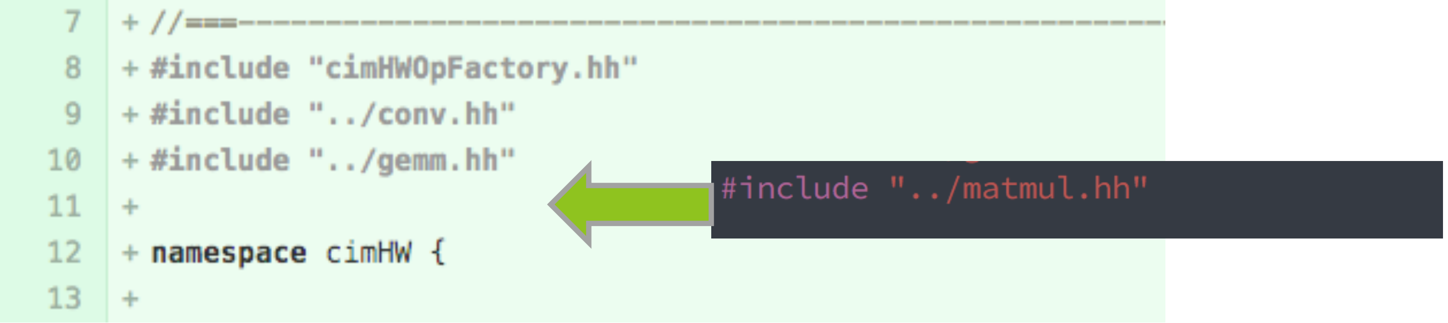


```
+   // virtual std::unique_ptr<cimHWOp> create${OperatorName}Op(
+   //    void* context
+   //    ${argument_list}
+   // ) override;
```

```
,const_element_type* input_A
  ,const_dim_type input_A_ndim, const_dim_type* input_A_dims
  ,const_element_type* input_B
  ,const_dim_type input_B_ndim, const_dim_type* input_B_dims
  ,element_type* output_Y
  ,const_dim_type output_Y_ndim, const_dim_type* output_Y_dims
```

Register to desired factory class

- skysim/onnc-cimHW/lib/factory/cimHWOpFactory.cc
  - Add include

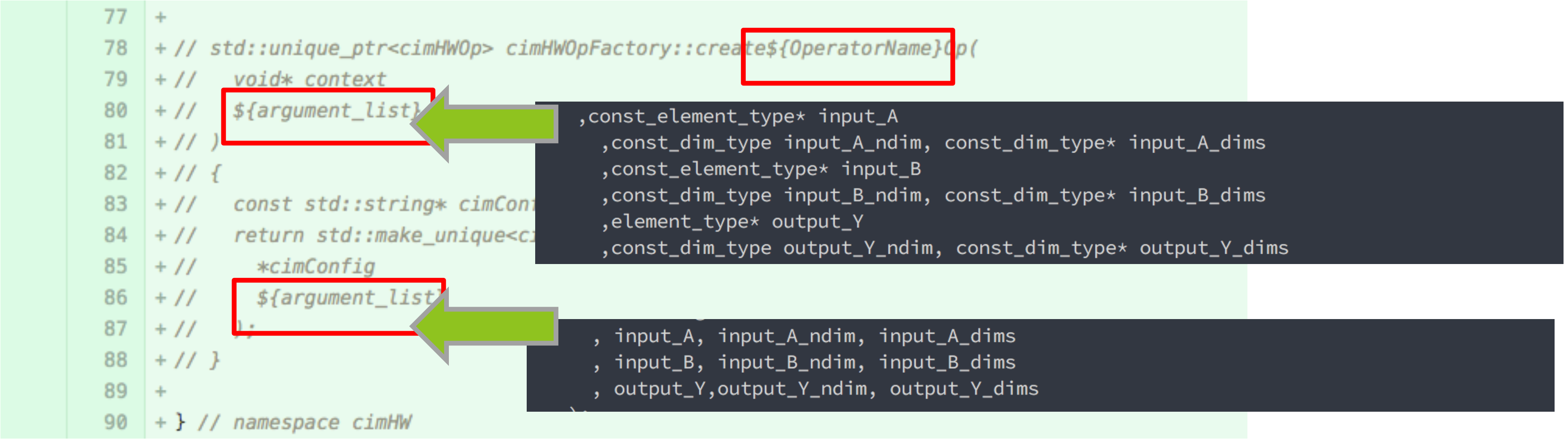


```
+ //===-----------------------------------------------------------------
+ #include "cimHWOpFactory.hh"
+ #include "../conv.hh"
+ #include "../gemm.hh"
+
+ namespace cimHW {
+
```

```
#include "../matmul.hh"
```

Register to desired factory class

- skysim/onnc-cimHW/lib/factory/cimHWOpFactory.cc
  - Add definition
    - replace ${OperatorName
    - replace ${argument_list}
    - remove type in 2nd ${argument_list}



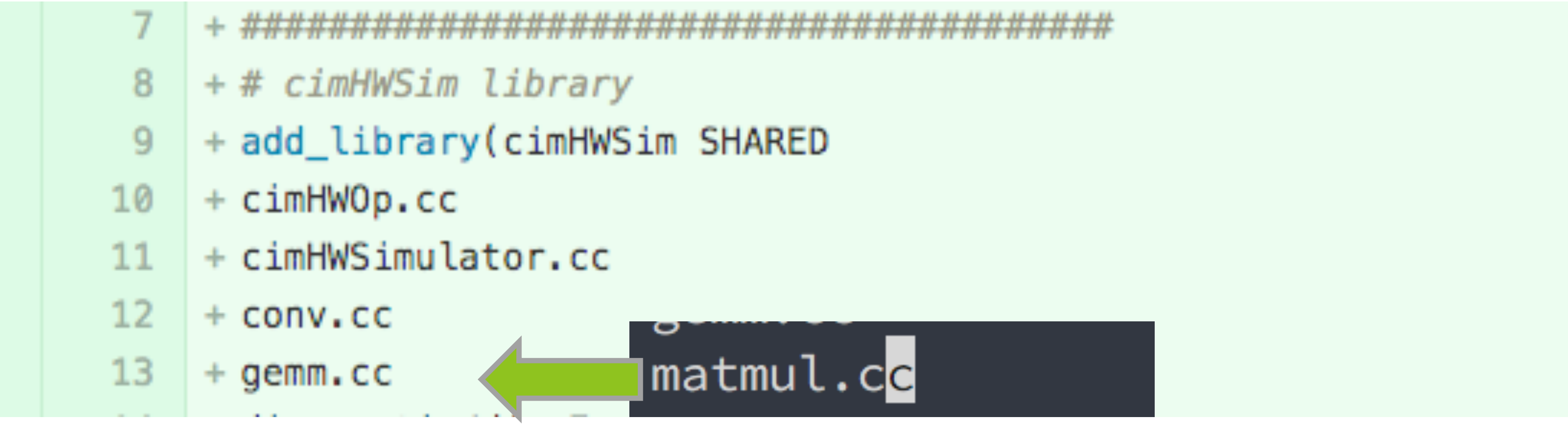
```
+
+ // std::unique_ptr<cimHWOp> cimHWOpFactory::create${OperatorName}Op(
+ //    void* context
+ //    ${argument_list}
+ // )
+ // {
+ //    const std::string* cimCon
+ //    return std::make_unique<ci
+ //      *cimConfig
+ //      ${argument_list
+ //    );
+ // }
+
+ } // namespace cimHW
```

```
 ,const_element_type* input_A
   ,const_dim_type input_A_ndim, const_dim_type* input_A_dims
   ,const_element_type* input_B
   ,const_dim_type input_B_ndim, const_dim_type* input_B_dims
   ,element_type* output_Y
   ,const_dim_type output_Y_ndim, const_dim_type* output_Y_dims
```

```
 , input_A, input_A_ndim, input_A_dims
 , input_B, input_B_ndim, input_B_dims
 , output_Y,output_Y_ndim, output_Y_dims
```

Adding the new file to building system

- Add the matmul.cc to skysim/onnc-cimHW/lib/CMakeLists.txt



```
+ ########################################
+ # cimHWSim library
+ add_library(cimHWSim SHARED
+ cimHWOp.cc
+ cimHWSimulator.cc
+ conv.cc
+ gemm.cc
```

```
matmul.cc
```

Rebuild

-
  ```
  $ cd /onnc/onnc-cimHW.build
  $ make;
  $ sudo make install
  ```

- Run a test case to verify the implementation

-
  ```
  $ ./example/matmul-layer share/cimConfig.json
  ```

Skymizer Taiwan Inc.

CONTACT US

E-mail sales@skymizer.com **Tel** +886 2 8797 8337

HQ 12F-2, No.408, Ruiguang Rd., Neihu Dist., Taipei City 11492, Taiwan
BR Center of Innovative Incubator, National Tsing Hua University, Hsinchu Taiwan

skymizer

Boost deep learning accelerator
with compiler technology

https://skymizer.com