EUROPEAN AI FOR FUNDAMENTAL PHYSICS CONFERENCE
EuCAIFCon 2025

Inference optimization with Memory Management and GPU Acceleration in TMVA SOFIE

17th June 2025

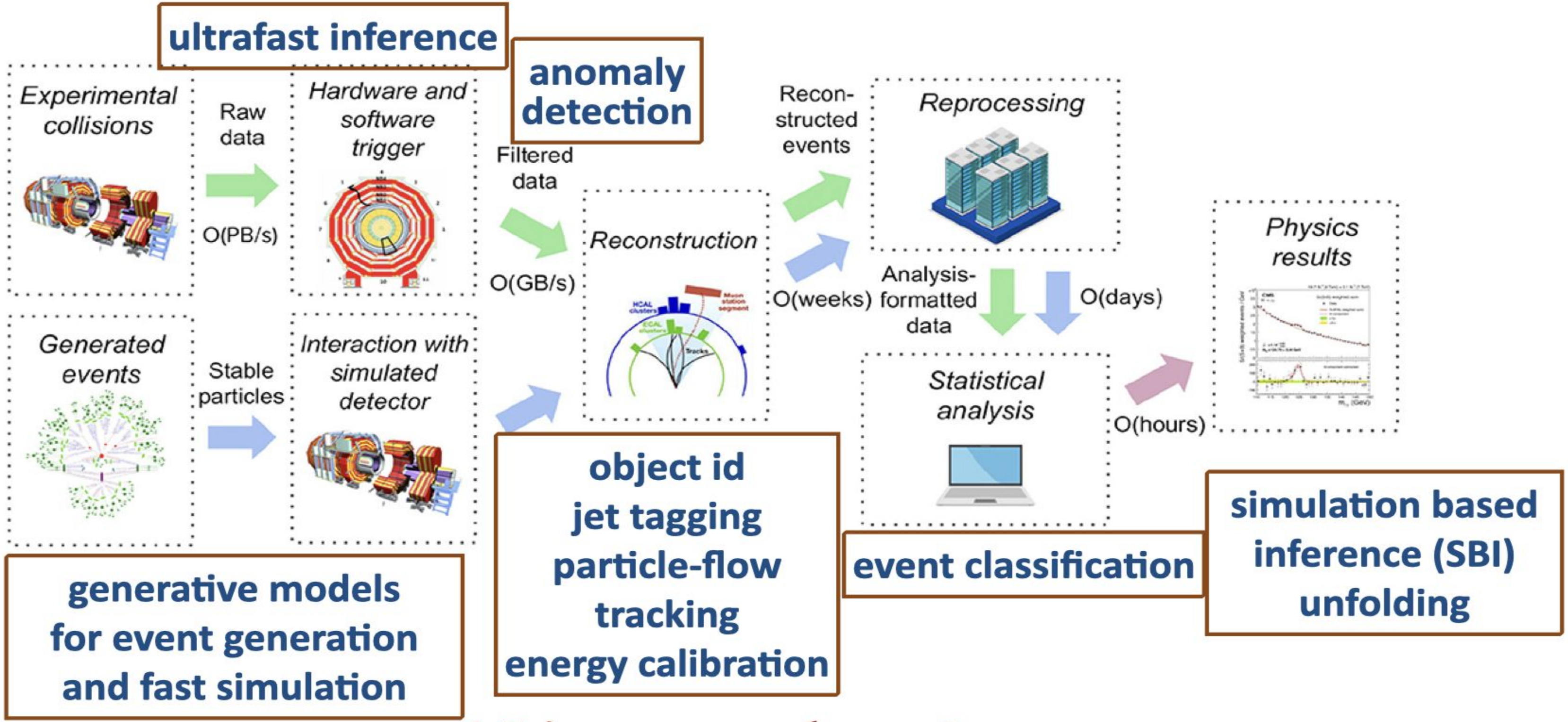
Sanjiban Sengupta[1,2] and Lorenzo Moneta[1]

1 CERN, Switzerland
2 The University of Manchester, United Kingdom

CERN | NexTGen Next Generation Triggers

MANCHESTER 1824
The University of Manchester

AI in Experiment Data Analysis



AI is everywhere !

fdata.2021.661501

Next-Gen Triggers

- **LHC processes data for collisions at 40MHz**
  - ~ 100TB/s data generated that needs to be filtered
  - Expected to rise significantly in the High-Luminosity phase of LHC
- **Next-Gen Triggers Project**
  - New initiative for cross-collaboration among teams at CERN to develop computing technologies for data acquisition and processing in preparation for HL-LHC

Motivation

Machine Learning Development Workflow

Case study → Model architecture development → Training → Testing and Evaluation → Inference!

- **Now the challenge is, how can we use and integrate them in experiments' production?**
  - Inference engines
    - Loads a trained ML model
    - Accepts data => produces results

    - Why so difficult then?
      - Experiments at CERN produces data at a tremendously high rate
      - Requires the engine to be efficient!

Motivation

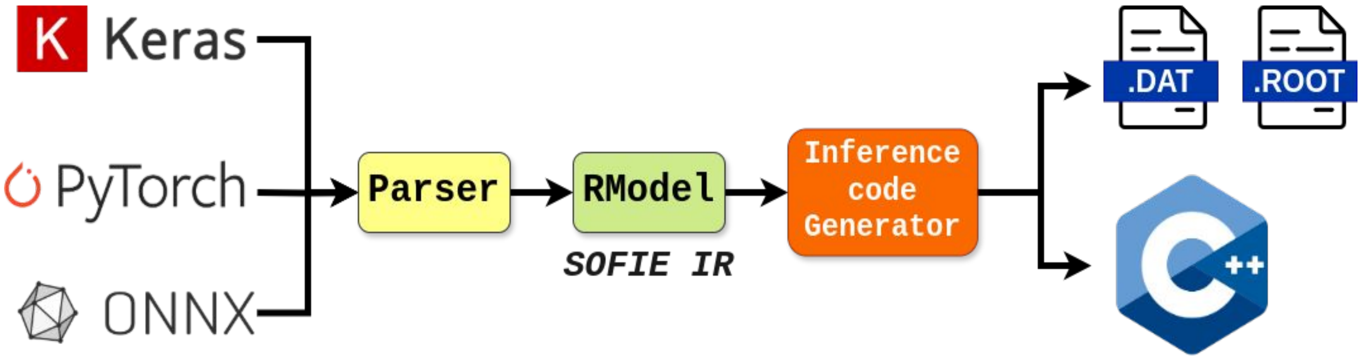
Machine Learning Development Workflow

Case study → Model architecture development → Training → Testing and Evaluation → Inference!

Keras PyTorch ONNX

SOFIE

- **SOFIE - System for Optimized Fast Inference code Emit**
- **Tool for optimized ML Inference - developed at CERN, that**
  - **Parses a trained model in ONNX, Keras or PyTorch format to its IR (based on ONNX standard)**
  - **Generates inference code in the form of C++ functions (only dependent on BLAS)**
  - **Supports several ONNX operators, Transformers, GNNs from Deepmind Graphnets.**



SOFIE

- **SOFIE - System for Optimized Fast Inference code Emit**
- **Tool for optimized ML Inference - developed at CERN, that**
  - **Parses a trained model in ONNX, Keras or PyTorch format to its IR (based on ONNX standard)**
  - **Generates inference code in the form of C++ functions (only dependent on BLAS)**
  - **Supports several ONNX operators, Transformers, GNNs from Deepmind Graphnets.**

GEMM
ReLU
Conv
RNN
Gather
Unsqueeze
BatchNorm
LayerNorm
...

SOFIE

- **SOFIE - System for Optimized Fast Inference code Emit**
- **Tool for optimized ML Inference - developed at CERN, that**
  - **Parses a trained model in ONNX, Keras or PyTorch format to its IR (based on ONNX standard)**
  - **Generates inference code in the form of C++ functions (only dependent on BLAS)**
  - **Supports several ONNX operators, Transformers, GNNs from Deepmind Graphnets.**

GEMM
ReLU
Conv
RNN
Gather
Unsqueeze
BatchNorm
LayerNorm
...

ParticleNet

SOFIE

- **SOFIE - System for Optimized Fast Inference code Emit**
- **Tool for optimized ML Inference - developed at CERN, that**
  - **Parses a trained model in ONNX, Keras or PyTorch format to its IR (based on ONNX standard)**
  - **Generates inference code in the form of C++ functions (only dependent on BLAS)**
  - **Supports several ONNX operators, Transformers, GNNs from Deepmind Graphnets.**

GEMM
ReLU
Conv
RNN
Gather
Unsqueeze
BatchNorm
LayerNorm
...

ParticleNet

ATLAS InteractionGNN

SOFIE

- **SOFIE - System for Optimized Fast Inference code Emit**
- **Tool for optimized ML Inference - developed at CERN, that**
  - **Parses a trained model in ONNX, Keras or PyTorch format to its IR (based on ONNX standard)**
  - **Generates inference code in the form of C++ functions (only dependent on BLAS)**
  - **Supports several ONNX operators, Transformers, GNNs from Deepmind Graphnets.**

GEMM
ReLU
Conv
RNN
Gather
Unsqueeze
BatchNorm
LayerNorm
...

ParticleNet

ATLAS InteractionGNN

DeepMind Graph Nets

SOFIE

- **SOFIE - System for Optimized Fast Inference code Emit**
- **Tool for optimized ML Inference - developed at CERN, that**
  - **Parses a trained model in ONNX, Keras or PyTorch format to its IR (based on ONNX standard)**
  - **Generates inference code in the form of C++ functions (only dependent on BLAS)**
  - **Supports several ONNX operators, Transformers, GNNs from Deepmind Graphnets.**

GEMM
ReLU
Conv
RNN
Gather
Unsqueeze
BatchNorm
LayerNorm
...

ParticleNet

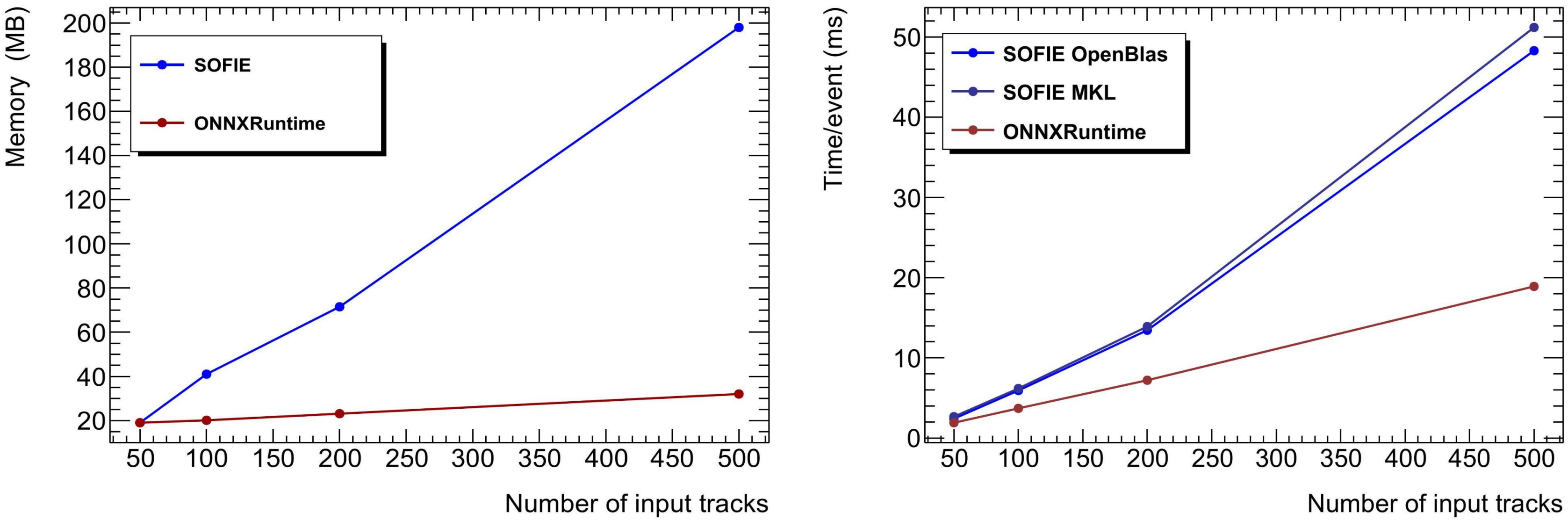
ATLAS InteractionGNN

DeepMind Graph Nets

SmartPixels

SOFIE

- **Benchmarking SOFIE against ONNXRuntime**
  - **System configuration**
    - Linux desktop
    - AMD Ryzen processor (24 threads, 4.4 GHz)
    - NVIDIA RTX 4090 GPU



- **Results**
  - **SOFIE performs better for smaller models and single event evaluation in time and memory**
  - **SOFIE takes longer time and intensive memory in inference of large models (eg. ResNet, ParticleNet)**
  - **Reason: SOFIE didn't have any optimization yet!**

SOFIE

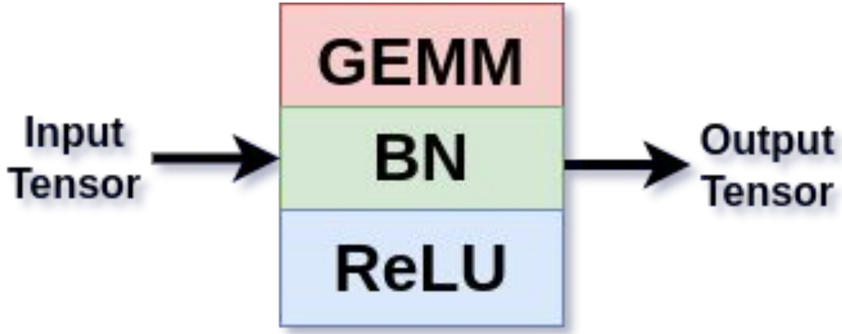
- **SOFIE needs optimization mechanisms to reduce memory usage and inference latency**
- **Optimization methods**
  - **Multi-Layer Fusion**
  - **Memory reuse**
  - **Efficient tensor broadcasting**
  - **Operator elimination**
  - **Sparse-tensor handling**
  - **Kernel-level optimization**

SOFIE

- **Multi-layer Fusion**
  - **Operators which are weightless, involve element-wise operation can be fused with the preceding operation**

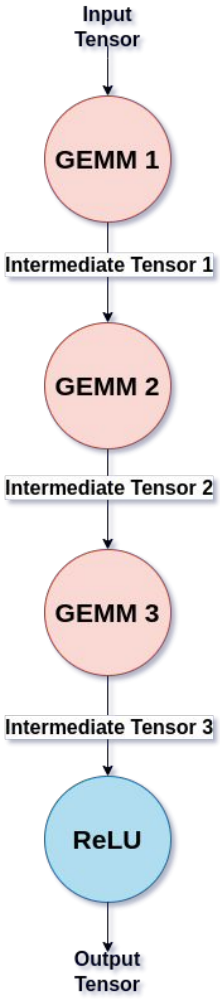
SOFIE

- **Multi-layer Fusion**
  - **Operators which are weightless, involve element-wise operation can be fused with the preceding operation**



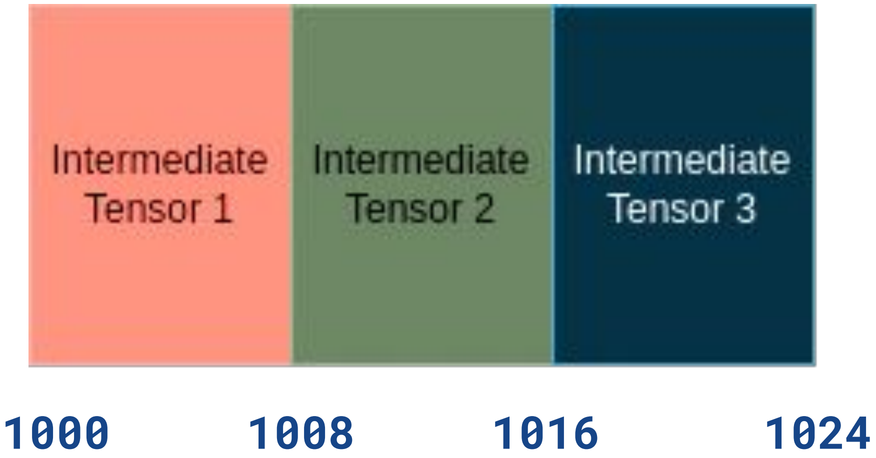
SOFIE

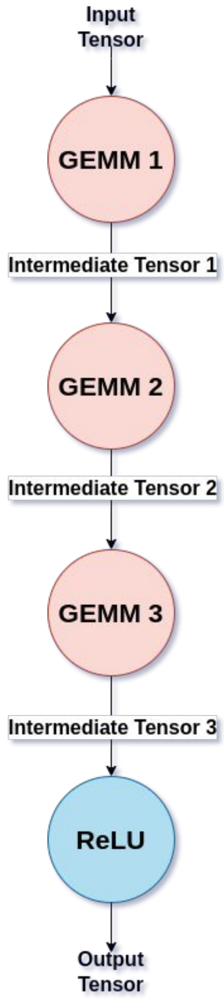
- **Memory Reuse**
  - **Initial iteration of SOFIE allocated memory of all the intermediate tensors without any memory reuse**



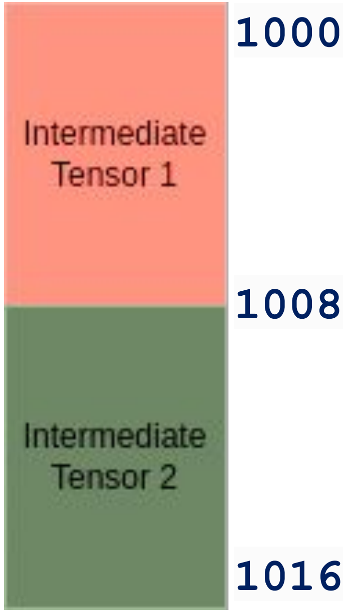
Memory Organization for Intermediate tensors

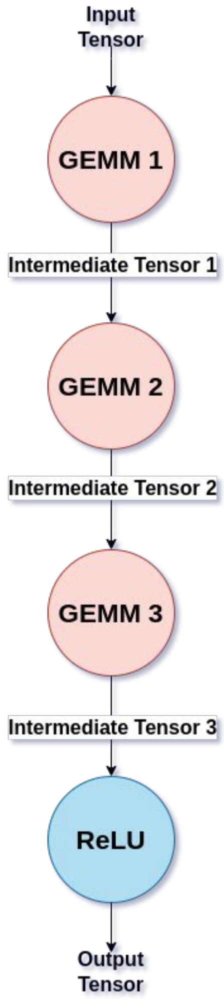
(each are a float tensor of length 2)





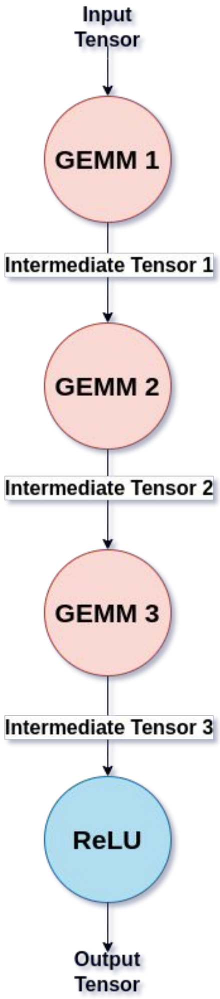
Memory Organization for Intermediate tensors
(each are a float tensor of length 2)





Memory Organization for Intermediate tensors

(each are a float tensor of length 2)



Memory Organization for Intermediate tensors

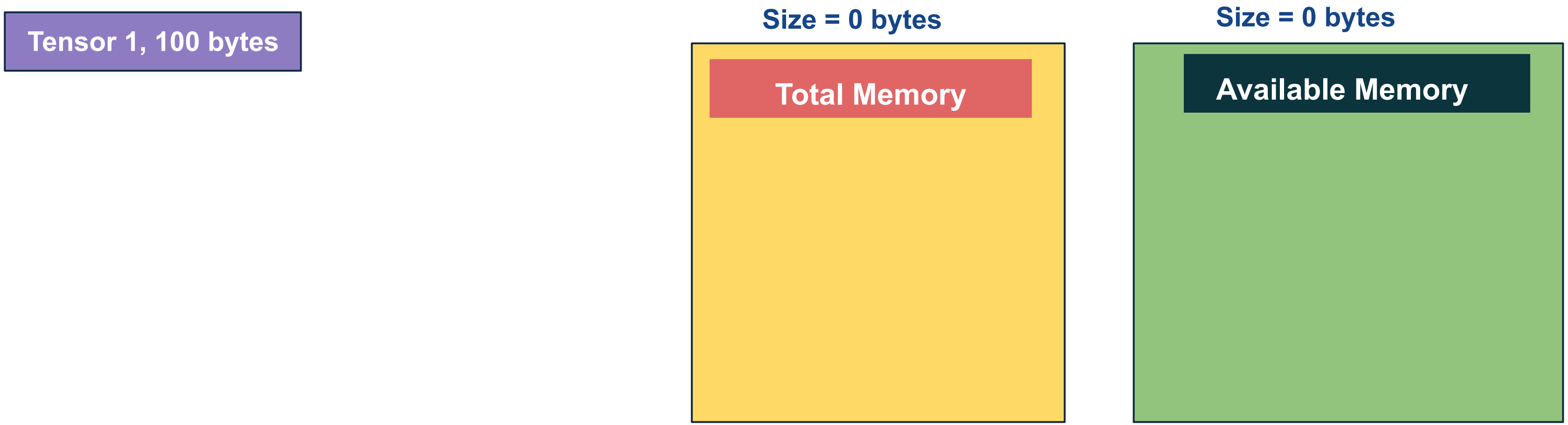
(each are a float tensor of length 2)

SOFIE

- **Memory Reuse**

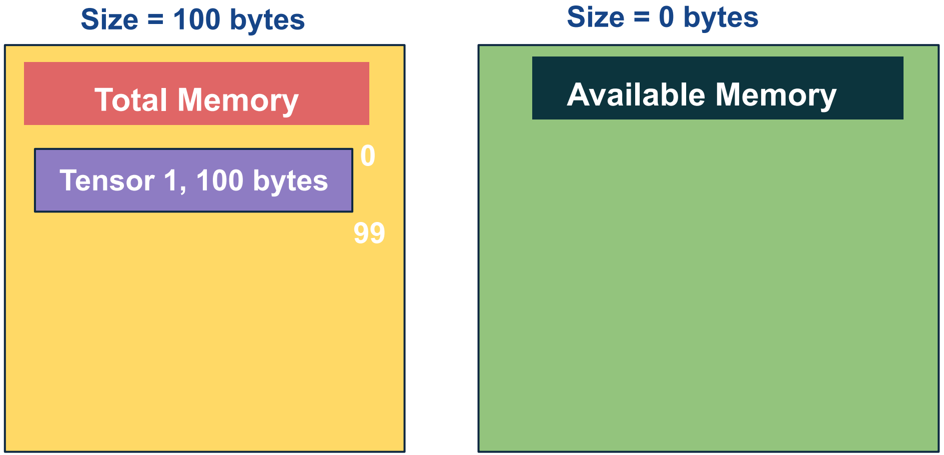
SOFIE

- **Memory Reuse**



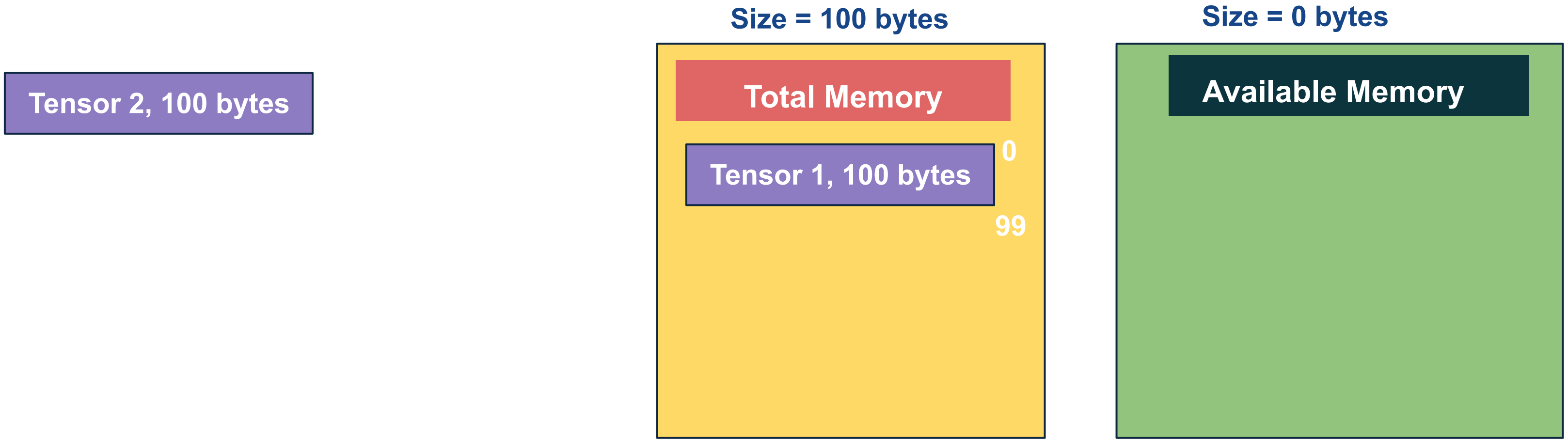
SOFIE

- **Memory Reuse**



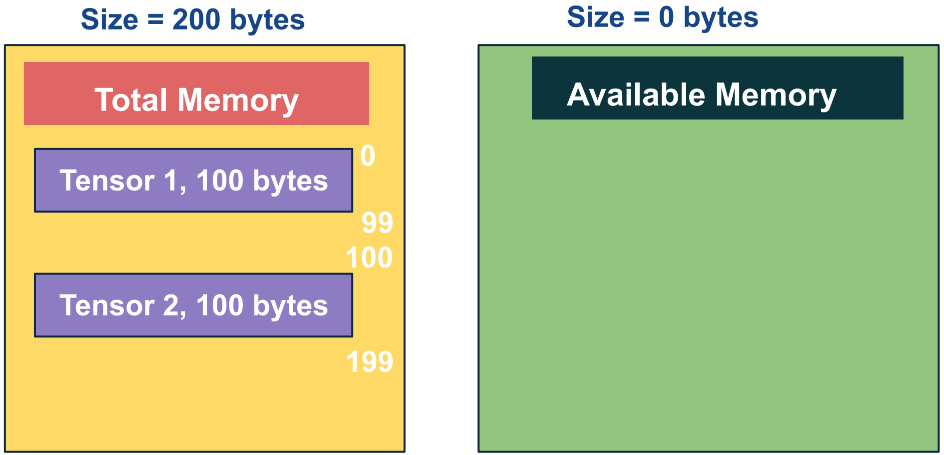
SOFIE

- **Memory Reuse**



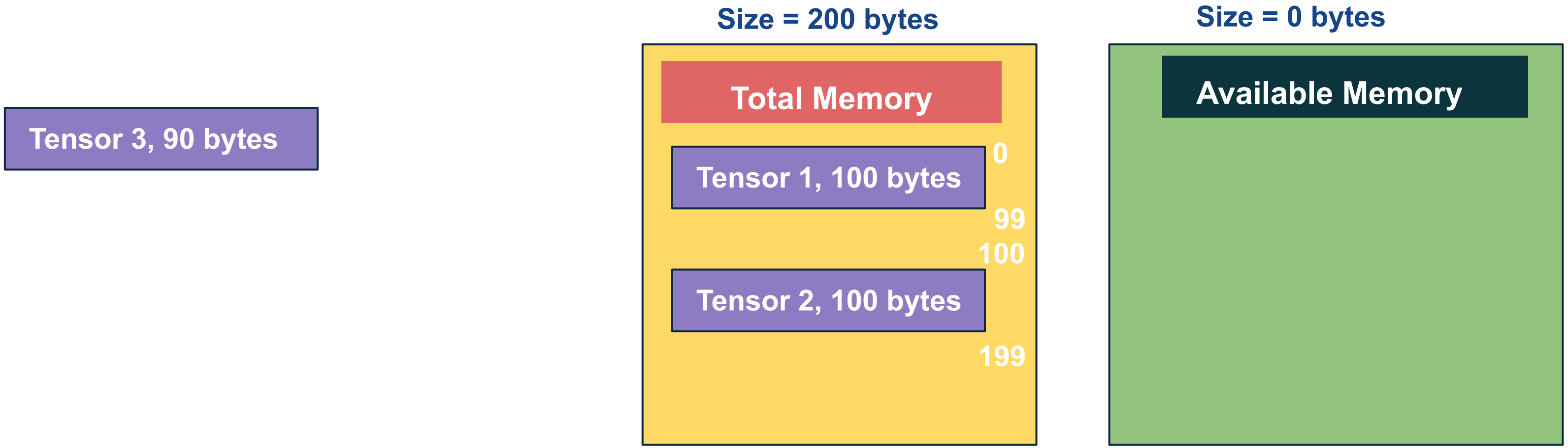
SOFIE

- **Memory Reuse**



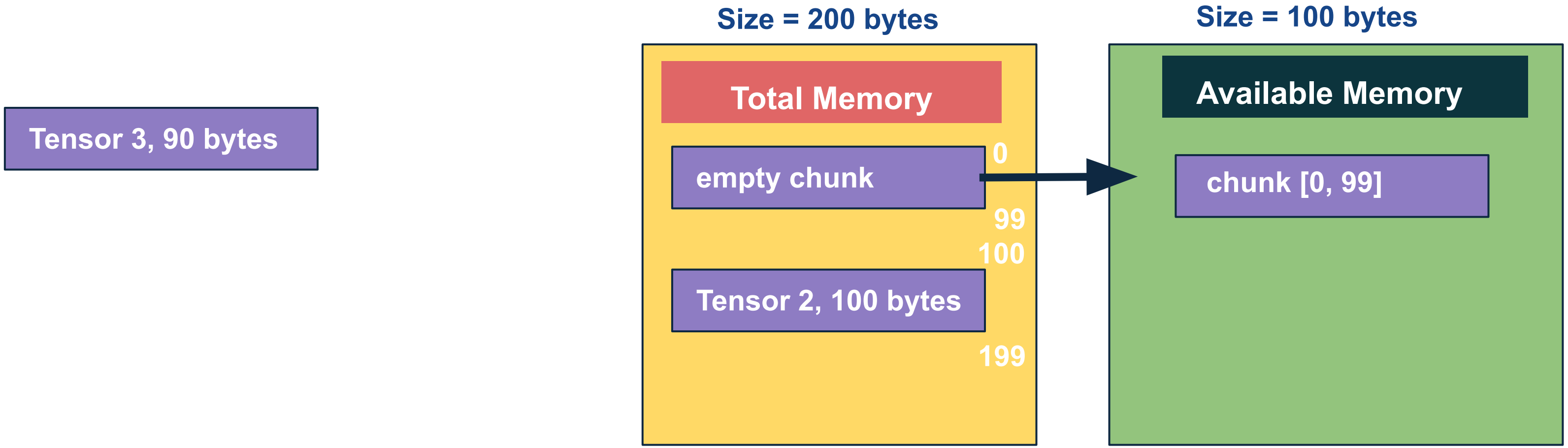
SOFIE

- **Memory Reuse**



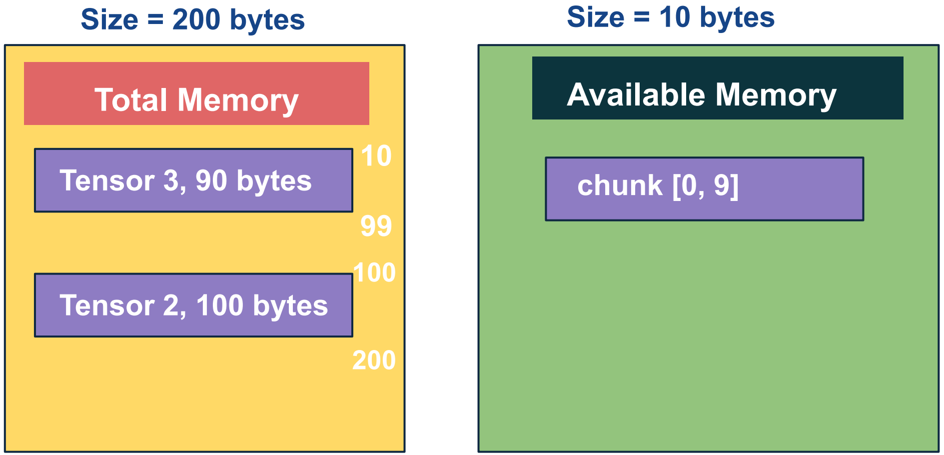
SOFIE

- **Memory Reuse**



SOFIE

- **Memory Reuse**



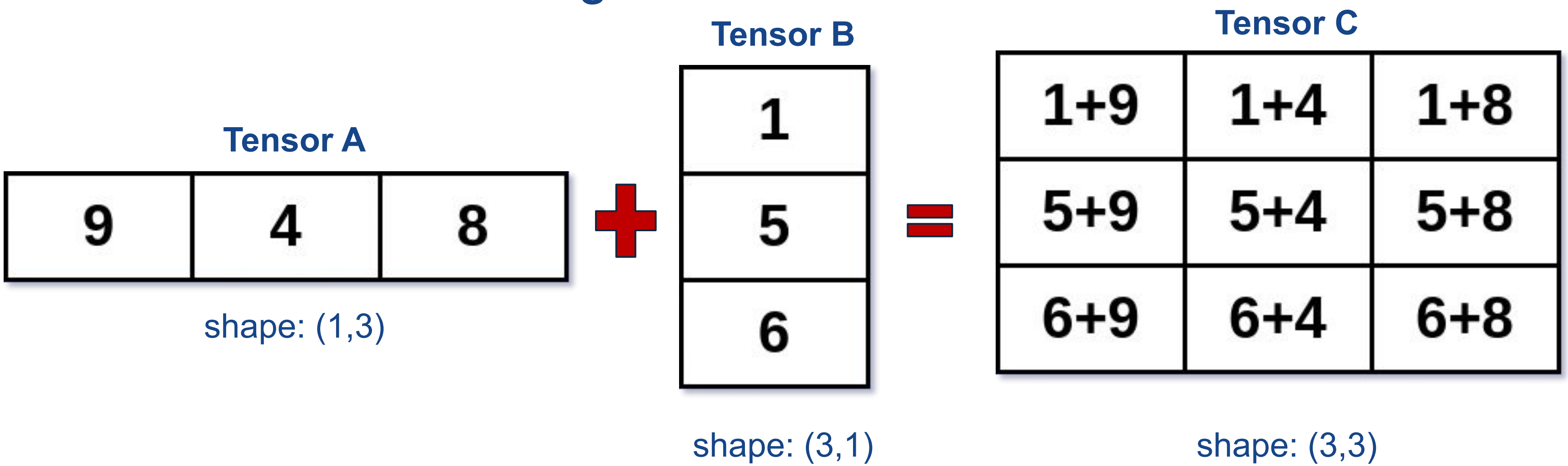
SOFIE

- ## Efficient Broadcasting
  - **NumPy defines**

> *"The term broadcasting describes how NumPy treats arrays with different shapes during arithmetic operations. Subject to certain constraints, the smaller array is "broadcast" across the larger array so that they have compatible shapes. Broadcasting provides a means of vectorizing array operations so that looping occurs in C instead of Python. It does this without making needless copies of data and usually leads to efficient algorithm implementations. There are, however, cases where broadcasting is a bad idea because it leads to inefficient use of memory that slows computation."*

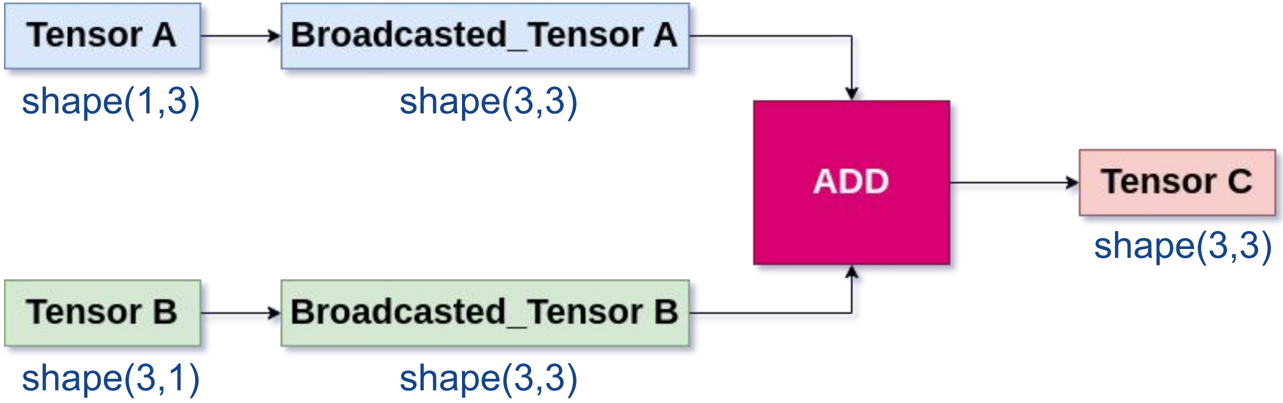
SOFIE

- **Efficient Broadcasting**



SOFIE

- **Efficient Broadcasting**

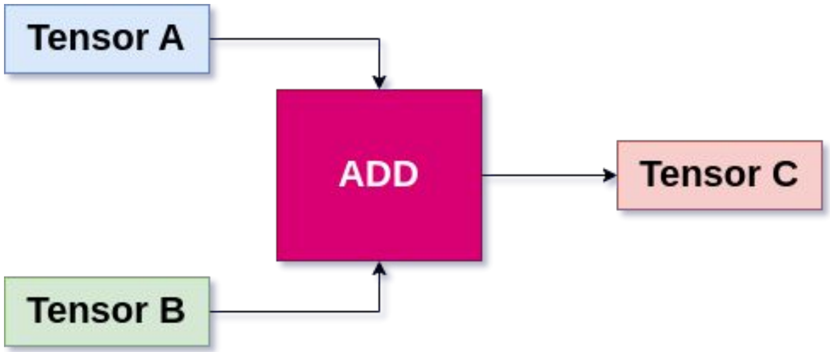


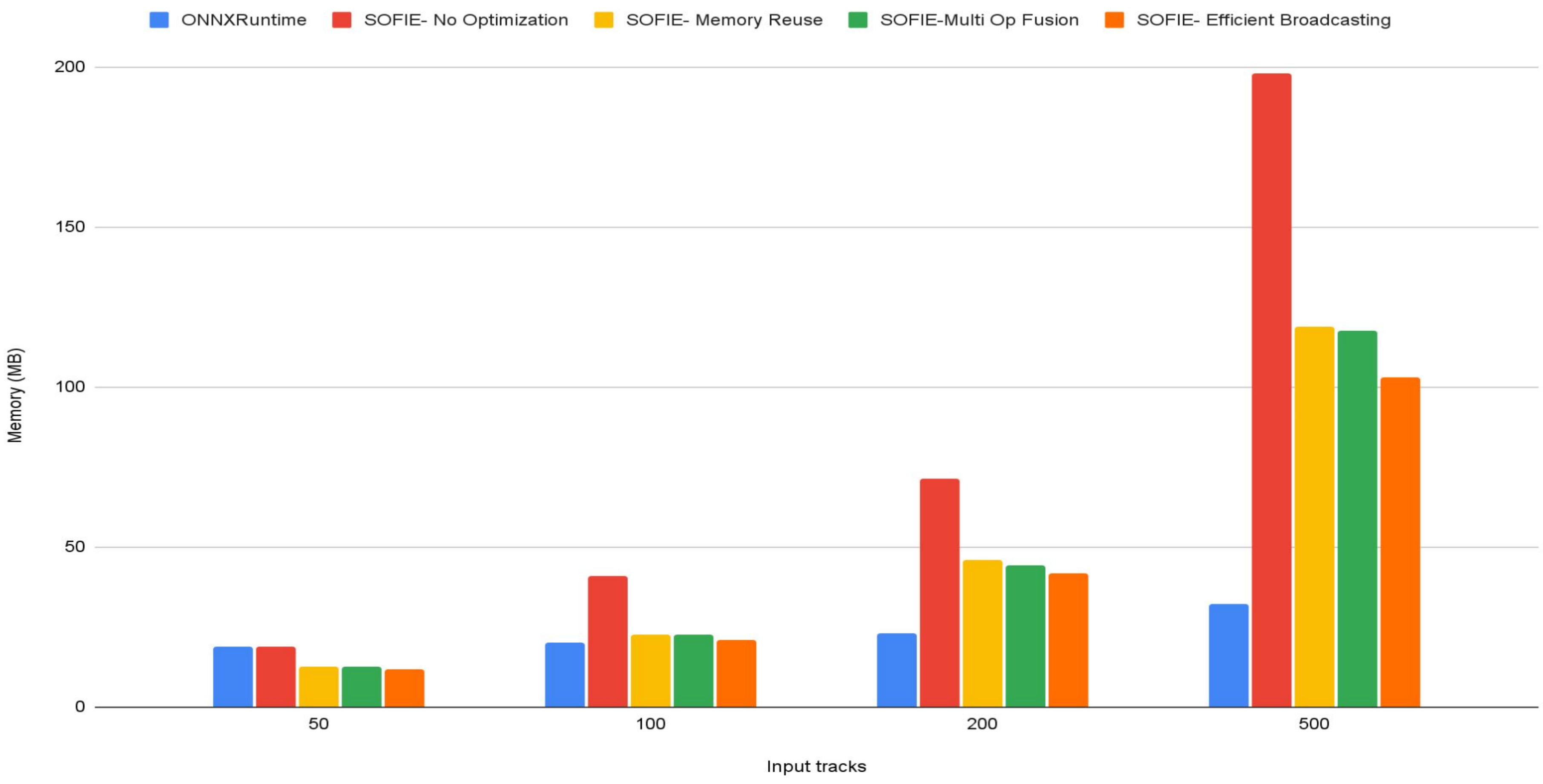
SOFIE

- **Efficient Broadcasting**
  I. **Extend unequal dimensions toward the broadcasted dimension on-the-fly**

output_shape = (3, 2, 1)
A_shape = (1, 2, 2)
B_shape = (3, 1, 2)

```
output[i*strides[0] +k*strides[1]]

= A[j*strides[1] + k*strides[2]] +
  B[i*strides[0] + k*strides[2]]
```




ONNXRuntime
SOFIE- No Optimization
SOFIE- Memory Reuse
SOFIE-Multi Op Fusion
SOFIE- Efficient Broadcasting
Memory (MB)
200
150
100
50
0
50
100
200
500
Input tracks

NGT 1.7

- **Efficient interfaces to Machine Learning inference engines to minimize data movements and execution latencies**

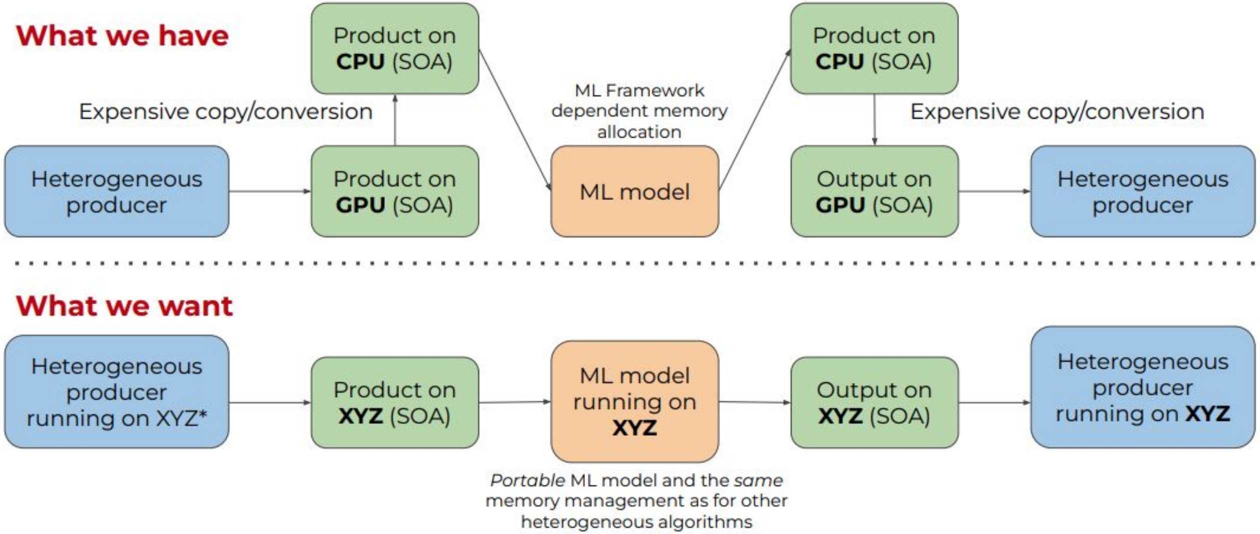
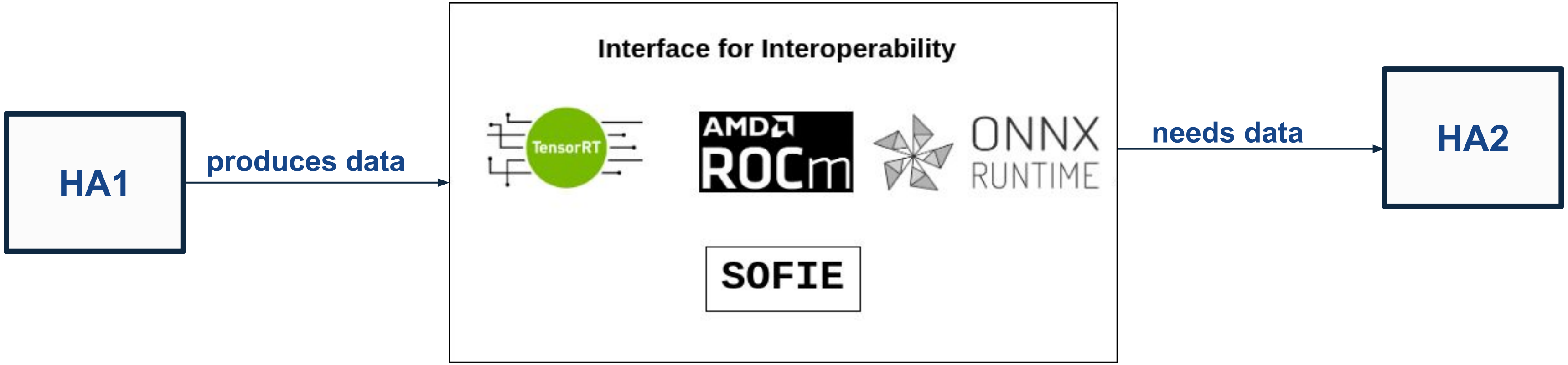


Image source: Presentation on Task 1.7: Common Software Developments for Heterogeneous Architectures, by Jolly Chen - NGT Workshop - 25th Nov 2024

NGT 1.7

- **Efficient interfaces to Machine Learning inference engines to minimize data movements and execution latencies**



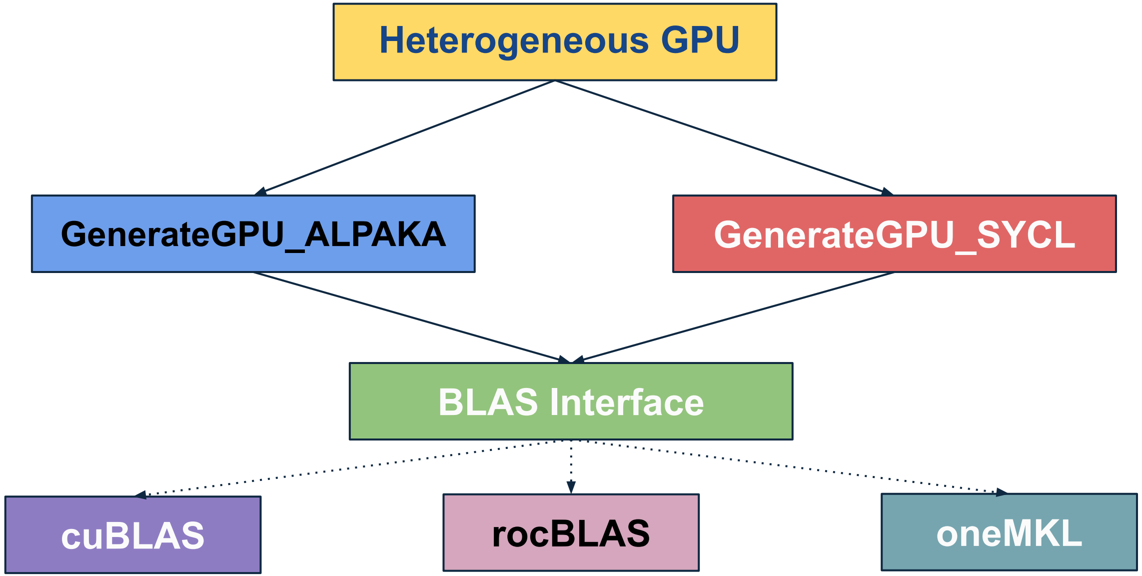
*** HA: Heterogeneous Architecture**

SOFIE

- **Inference on Heterogeneous Architecture**
  - **via SYCL and ALPAKA**
  - **through cuBLAS, rocBLAS, oneMKL**

SOFIE

- **Inference on Heterogeneous Architecture**



SOFIE

- **Inference on Heterogeneous Architecture**
  - **Architecture**
    - **Using buffer-accessor model**
    - **Initializes buffers during session instantiation**
    - **Accepts buffers as inputs**
    - **Returns buffers as outputs**
    - **Abstract Infer function**
      - **user has the control of running it on Intel, NVIDIA, or AMD GPUs**
      - **BLAS methods chosen automatically as per the chosen execution architecture**

    - **Current Status**
      - **SYCL Prototype developed -> under testing**
      - **ALPAKA Prototype for NVIDIA CUDA under testing**
        - **GEMM and ReLU operations**

Future plans

- R&D on further optimization methods
- Continue extending support for inference on Hetergeneous architectures
- Benchmarking Heterogeneous support with other frameworks (ONNX-GPU)
- Interoperability with hls4ml
- Interfaces to TensorRT and ROCm
- Experimenting with AITemplate

Conclusion

- **GitHub Repository:** https://github.com/root-project/root/tree/master/tmva/sofie

Conclusion

- **Acknowledgement**

This work has been funded by the Eric & Wendy Schmidt Fund for Strategic Innovation through the CERN Next Generation Triggers project under grant agreement number SIF-2023-004

- **For more information**

Lorenzo Moneta
Senior Applied Physicist
<lorenzo.moneta@cern.ch>

Sanjiban Sengupta
Doctoral Student
<sanjiban.sengupta@cern.ch>

CERN

NexTGen

Next Generation Triggers

backup

What we need that ONNXRuntime cannot do?

- **Known issues with OnnxRuntime:**
  - **Certain ML operations are not supported by ONNXRuntime**
  - **Users simply cannot convert their ML model to ONNX.**
  - **ONNXRuntime inference values may vary in the last digits for different runs [GH issue].**

- **What we need but ONNX cannot do**
  - **Access to remote co-processors (GPUs, FPGAs, and so on)**
  - **Support non-ML algorithms that can enjoy speedups by running in co-processors**

Source: AthenaTriton: A Tool for running Machine Learning Inference as a Service in Athena. Talk by Yuan-Tang Chou on behalf of the ATLAS Computing Facility. CHEP2024

SOFIE

```
using namespace TMVA::Experimental;
SOFIE::RModelParser_ONNX parser;
SOFIE::RModel model = parser.Parse("Linear_16.onnx");
model.Generate();
model.OutputGenerated("Linear_16_FromONNX.hxx");
```

SOFIE

```cpp
#include "Linear_16_FromONNX.hxx"

int main() {
     std::vector<float> input(1600);
     std::fill_n(input.data(), input.size(), 1.0f);
     TMVA_SOFIE_Linear_16::Session s("Linear_16_FromONNX.dat");
     std::vector<float> output = s.infer(input.data());
     return 0;
}
```

```
//Code generated automatically by TMVA for Inference of Model file [Linear_16.onnx] at [Sat Apr  5 21:25:43 2025]

#ifndef ROOT_TMVA_SOFIE_LINEAR_16
#define ROOT_TMVA_SOFIE_LINEAR_16

#include <algorithm>
#include <vector>
#include "TMVA/SOFIE_common.hxx"
#include <fstream>

namespace TMVA_SOFIE_Linear_16{
namespace BLAS{
    extern "C" void sgemv_(const char * trans, const int * m, const int * n, const float * alpha, const float * A,
                           const int * lda, const float * X, const int * incx, const float * beta, const float * Y, const int * incy);
    extern "C" void sgemm_(const char * transa, const char * transb, const int * m, const int * n, const int * k,
                           const float * alpha, const float * A, const int * lda, const float * B, const int * ldb,
                           const float * beta, float * C, const int * ldc);
}//BLAS

struct Session {
// initialized tensors
std::vector<float> fTensor_8weight = std::vector<float>(2500);
float * tensor_8weight = fTensor_8weight.data();
std::vector<float> fTensor_8bias = std::vector<float>(50);
float * tensor_8bias = fTensor_8bias.data();
...

//--- Allocating session memory pool to be used for allocating intermediate tensors
char* fIntermediateMemoryPool = new char[29440];


// --- Positioning intermediate tensor memory --
 // Allocating memory for intermediate tensor 22 with size 3200 bytes
float* tensor_22= reinterpret_cast<float*>(fIntermediateMemoryPool + 0);

 // Allocating memory for intermediate tensor 24 with size 3200 bytes
float* tensor_24= reinterpret_cast<float*>(fIntermediateMemoryPool + 3200);

 // Allocating memory for intermediate tensor 26 with size 3200 bytes
float* tensor_26= reinterpret_cast<float*>(fIntermediateMemoryPool + 0);
...


//--- declare and allocate the intermediate tensors
std::vector<float> fTensor_18biasbcast = std::vector<float>(160);
float * tensor_18biasbcast = fTensor_18biasbcast.data();
std::vector<float> fTensor_14biasbcast = std::vector<float>(800);
float * tensor_14biasbcast = fTensor_14biasbcast.data();
```

STL and SOFIE header files; no other dependency

BLAS declarations

Session

```cpp
//Code generated automatically by TMVA for Inference of Model file [Linear_16.onnx] at [Sat Apr  5 21:25:43 2025]

#ifndef ROOT_TMVA_SOFIE_LINEAR_16
#define ROOT_TMVA_SOFIE_LINEAR_16

#include <algorithm>
#include <vector>
#include "TMVA/SOFIE_common.hxx"
#include <fstream>

namespace TMVA_SOFIE_Linear_16{
namespace BLAS{
    extern "C" void sgemv_(const char * trans, const int * m, const int * n, const float * alpha, const float * A,
                           const int * lda, const float * X, const int * incx, const float * beta, const float * Y, const int * incy);
    extern "C" void sgemm_(const char * transa, const char * transb, const int * m, const int * n, const int * k,
                           const float * alpha, const float * A, const int * lda, const float * B, const int * ldb,
                           const float * beta, float * C, const int * ldc);
}//BLAS

struct Session {
// initialized tensors
std::vector<float> fTensor_8weight = std::vector<float>(2500);
float * tensor_8weight = fTensor_8weight.data();
std::vector<float> fTensor_8bias = std::vector<float>(50);
float * tensor_8bias = fTensor_8bias.data();
...

//--- Allocating session memory pool to be used for allocating intermediate tensors
char* fIntermediateMemoryPool = new char[29440];


// --- Positioning intermediate tensor memory --
 // Allocating memory for intermediate tensor 22 with size 3200 bytes
float* tensor_22= reinterpret_cast<float*>(fIntermediateMemoryPool + 0);

 // Allocating memory for intermediate tensor 24 with size 3200 bytes
float* tensor_24= reinterpret_cast<float*>(fIntermediateMemoryPool + 3200);

 // Allocating memory for intermediate tensor 26 with size 3200 bytes
float* tensor_26= reinterpret_cast<float*>(fIntermediateMemoryPool + 0);
...


//--- declare and allocate the intermediate tensors
std::vector<float> fTensor_18biasbcast = std::vector<float>(160);
float * tensor_18biasbcast = fTensor_18biasbcast.data();
std::vector<float> fTensor_14biasbcast = std::vector<float>(800);
float * tensor_14biasbcast = fTensor_14biasbcast.data();
```

STL and SOFIE header files; no other dependency

BLAS declarations

Model weights

Intermediate tensors

broadcasted tensors

Session

```cpp
Session(std::string filename ="Linear_16.dat") {

//--- reading weights from file
   std::ifstream f;
   f.open(filename);
   if (!f.is_open()) {
      throw std::runtime_error("tmva-sofie failed to open file " + filename + " for input weights");
   }
   std::string tensor_name;
   size_t length;
   f >> tensor_name >> length;
   if (tensor_name != "tensor_8weight" ) {
      std::string err_msg = "TMVA-SOFIE failed to read the correct tensor name; expected name is tensor_8weight , read " + tensor_name;
      throw std::runtime_error(err_msg);
    }
  ...
}


std::vector<float> infer(float* tensor_input1){

//--------- Gemm
   char op_0_transA = 'n';
   char op_0_transB = 't';
   int op_0_m = 16;
   int op_0_n = 50;
   int op_0_k = 100;
   float op_0_alpha = 1;
   float op_0_beta = 1;
   int op_0_lda = 100;
   int op_0_ldb = 100;
   std::copy(tensor_0biasbcast, tensor_0biasbcast + 800, tensor_22);
   BLAS::sgemm_(&op_0_transB, &op_0_transA, &op_0_n, &op_0_m, &op_0_k, &op_0_alpha, tensor_0weight, &op_0_ldb, tensor_input1,
&op_0_lda, &op_0_beta, tensor_22, &op_0_n);
   for (int id = 0; id < 800 ; id++){
      tensor_22[id] = ((tensor_22[id] > 0 )? tensor_22[id] : 0);
   }

   ...

   std::vector<float> ret(tensor_39, tensor_39 + 160);
   return ret;
}
};   // end of Session
} //TMVA_SOFIE_Linear_16

#endif  // ROOT_TMVA_SOFIE_LINEAR_16
```

Session constructor

Infer function

SOFIE

- ## Multi-layer Fusion
  - ### Algorithm
    - For each operator in the computation graph:
      - Check if it is an anchor operation (e.g., `GEMM`, `Conv`, etc.):
        - If yes:
          - Check if the next operator is fusable, i.e., an in-place, weight-less operation:
            - If yes:
              - Fuse it with the preceding operation.
              - Check if this is the last fusable operation in the chain:
                - If yes: Break the fusion chain and resume the mechanism from the next operator.
                - If no: Continue to the next operator.
            - If no:
              - Fusion is not possible. Move to the next operator.
        - If it is not an anchor operation:
          - Fusion is not possible, move to the next operator.

SOFIE

- ## Memory Reuse
  - Evaluate the optimal memory using a Memory pool containing Total and Available Memory stack
  - Allocate a memory block with the total memory and position intermediate tensors to addresses which are available and/or can be reused
    - *Algorithm to determine memory addresses => allocation statically ahead of time*
      - For each operator,
        - For every output tensor
          - Check if Available Stack has any suitable memory chunk
          - If yes, reposition it for reuse
          - If no, obtain new memory from pool and track it in Total Stack
        - For every input tensor
          - Check if it is the last operator which is using this as input
          - If yes, consider it in available memory (for memory reuse)
            - Check if the newly available chunk can be coalesced with an adjoining chunk to make a larger block

SOFIE

- **Optimization Modes**
  - **Optimization can be tuned as per user requirements**
  - **Applications in automatic-differentiation**
  - **Modes (kExtended is enabled by default!)**
    - **kBasic: Operator fusion**
    - **kExtended: kBasic + Memory reuse**

SOFIE

- **Inference on Heterogeneous Architecture**

```
#include "SOFIE/RModel.hxx
#include "SOFIE/RModelParser_ONNX.hxx"

SOFIE::RModelParser_ONNX parser;
SOFIE::RModel model = parser.Parse("Linear_4.onnx");
model.GenerateGPU_SYCL();
model.OutputGenerated();
```

SOFIE

- **Inference on Heterogeneous Architecture**

```
#include "Linear_4_FromONNX_SYCL.hxx"

int main(){
    std::vector<float> input(4);
    std::fill(std::begin(input), std::end(input), 1.0f);

    sycl::buffer<float, 1> input_buffer(input.data(), range<1>(4));

    SOFIE_Linear_4::Session<EAccType::CUDA> s;

    auto output = s.infer(input_buffer);

    return 0;
}
```

Link to generated code

SOFIE

- **Inference on Heterogeneous Architecture**

```
template <EHetType HetType, EAccType AccType>
struct BLASBackend {};

template <>
struct BLASBackend<EHetType::SYCL, EAccType::CUDA> {
    void gemm(sycl::handler &h,
              sycl::accessor<float, 1, sycl::access::mode::read> d_A,
              sycl::accessor<float, 1, sycl::access::mode::read> d_B,
              sycl::accessor<float, 1, sycl::access::mode::read_write> d_C,
              int M, int N, int K,
              float alpha, float beta,
              int lda, int ldb, int ldc) {

        h.host_task([=](sycl::interop_handle ih) {
            cuCtxSetCurrent(ih.get_native_context<sycl::backend::ext_oneapi_cuda>());
            auto cuStream = ih.get_native_queue<sycl::backend::ext_oneapi_cuda>();

            cublasHandle_t handle;
            CHECK_ERROR(cublasCreate(&handle));
            cublasSetStream(handle, cuStream);

            float *cuA = reinterpret_cast<float *>(ih.get_native_mem<sycl::backend::ext_oneapi_cuda>(d_A));
            float *cuB = reinterpret_cast<float *>(ih.get_native_mem<sycl::backend::ext_oneapi_cuda>(d_B));
            float *cuC = reinterpret_cast<float *>(ih.get_native_mem<sycl::backend::ext_oneapi_cuda>(d_C));

            CHECK_ERROR(cublasSgemm(handle, CUBLAS_OP_N, CUBLAS_OP_N, N, M, K,
                                    &alpha, cuB, ldb, cuA, lda, &beta, cuC, ldc));

            cudaStreamSynchronize(cuStream);
            CHECK_ERROR(cublasDestroy(handle));
        });
    }
};
```

SOFIE

- **Inference on Heterogeneous Architecture**

```
#include "SOFIE/RModel.hxx
#include "SOFIE/RModelParser_ONNX.hxx"

SOFIE::RModelParser_ONNX parser;
SOFIE::RModel model = parser.Parse("Linear_16.onnx");
model.GenerateGPU_ALPAKA();
model.OutputGenerated();
```

SOFIE

- **Inference on Heterogeneous Architecture**

```cpp
#include "Linear_4_FromONNX_ALPAKA.hxx"

int main() {
    float input_tensor[4] = {1.0f, 2.0f, 3.0f, 4.0f};

    using AccType = typename AccFromEnum<EAccType::CUDA>::Type;
    using Queue = alpaka::Queue<AccType, alpaka::Blocking>;

    alpaka::PlatformCpu const platformHost{};
    alpaka::DevCpu const devHost = alpaka::getDevByIdx(platformHost, 0);
    auto const platformAcc = alpaka::Platform<AccType>{};
    auto const devAcc = alpaka::getDevByIdx(platformAcc, 0);
    Queue queue(devAcc);

    auto input_buffer_dev = alpaka::allocBuf<float, std::size_t>(devAcc, 4);
    alpaka::memcpy(queue, input_buffer_dev, input_tensor);

    SOFIE_Linear_4::Session<EAccType::CUDA> session;

    auto output_buffer_dev = session.infer_alpaka(input_buffer_dev);
    alpaka::wait(queue);

    return 0;
}
```

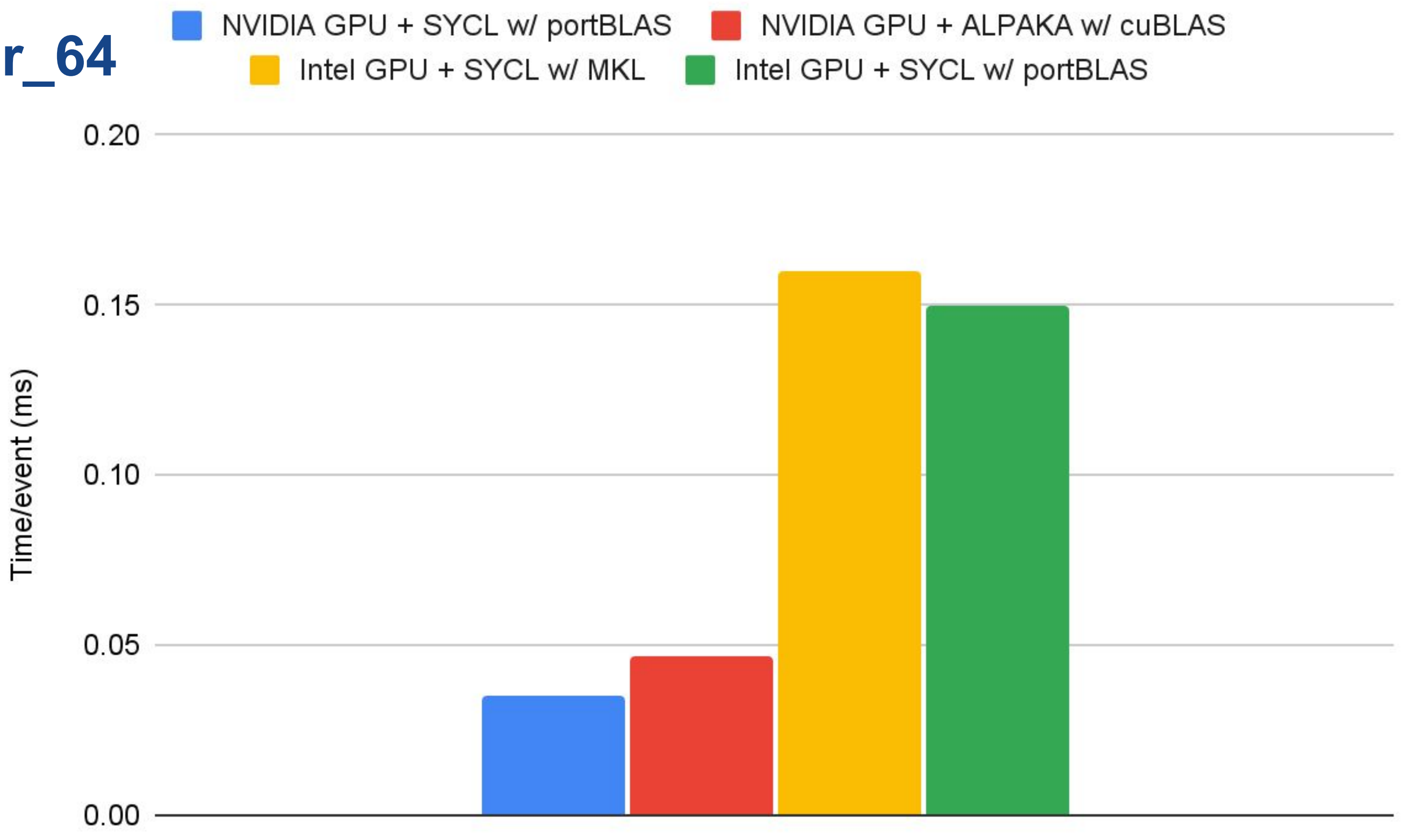
Link to generated code

SOFIE

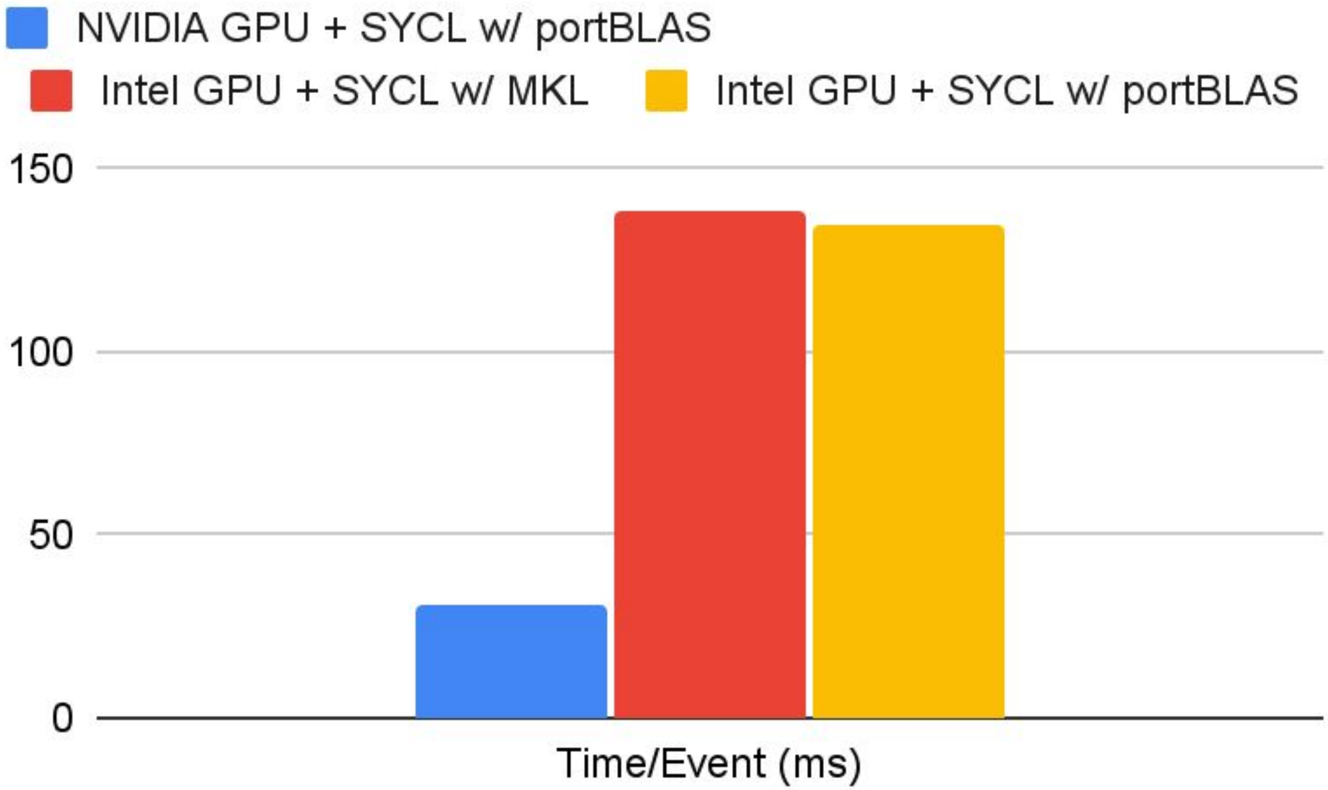
- **Inference on Heterogeneous Architecture**

```
template <EHetType HetType, EAccType AccType>
struct BLASBackend {};

template <>
struct BLASBackend<EHetType::ALPAKA, EAccType::CUDA> {
    void gemm(Queue& queue, BufA& bufA, BufB& bufB, BufC& bufC, Idx M, Idx N, Idx K,
              DataType alpha = 1.0f,DataType beta = 0.0f) {

        auto alpakaStream = alpaka::getNativeHandle(queue);

        cublasHandle_t cublasHandle;
        CHECK_CUBLAS_ERROR(cublasCreate(&cublasHandle));
        CHECK_CUBLAS_ERROR(cublasSetStream(cublasHandle, alpakaStream));

        CHECK_CUBLAS_ERROR(cublasSgemm(
            cublasHandle, CUBLAS_OP_N,
            CUBLAS_OP_N,
            M, N, K,
            &alpha, std::data(bufA), M,
            std::data(bufB), K, &beta,
            std::data(bufC), M
        ));

        alpaka::wait(queue);
        CHECK_CUBLAS_ERROR(cublasDestroy(cublasHandle));
    }
};
```

Linear_64


NVIDIA GPU + SYCL w/ portBLAS
NVIDIA GPU + ALPAKA w/ cuBLAS
Intel GPU + SYCL w/ MKL
Intel GPU + SYCL w/ portBLAS
0.20
0.15
0.10
0.05
0.00
Time/event (ms)

ResNet18


NVIDIA GPU + SYCL w/ portBLAS
Intel GPU + SYCL w/ MKL
Intel GPU + SYCL w/ portBLAS
150
100
50
0
Time/Event (ms)