codeplay®

Enable AI & HPC to be Open, Safe and Accessible to All

User-Driven Online Kernel Fusion for SYCL

Víctor Pérez, Lukas Sommer, Victor Lomüller

27th July 2022

Agenda

- Motivation
- SYCL extension design
- Implementation
- Case study 1: SYCL-DNN
- Case study 2: ONNX Runtime
- Outlook on port to DPC++
- Conclusions

Motivation

Motivation: Why use fusion?

- **Short-running kernels** can affect performance.
- **Manual fusion** is already used in some domains.
  - Too much work.
  - Hurts composability.
  - Error-prone.
  - Domain-specific.
- Instead, **extend** the **SYCL API** to enable **user-driven, automatic, online kernel fusion.**

SYCL Extension Design

Extension Requirements

- **Legality:** The fused kernel must compute an equivalent result to the original one.
- **Profitability:** Kernel fusion should improve overall performance.
- **Non-Invasive:** Require minimal changes in existing code bases.
- Whether or not to fuse is up to the user (human or other, e.g., the ONNX runtime), although the final decision on whether to fuse will be made by the SYCL runtime.
- Use of runtime-available information for optimization, e.g., identical arguments, scalar values...

Extension in 1 slide

```
// New queue member functions
void queue::start_fusion();
void queue::cancel_fusion();
event queue::complete_fusion(const property_list &props = {});
// Queue properties
class property::queue::enable_fusion;
class property::no_barriers;
// Buffers/accessors properties
class property::promote_local;
class property::promote_private;
```

Kernel Fusion: Minimal Overhead

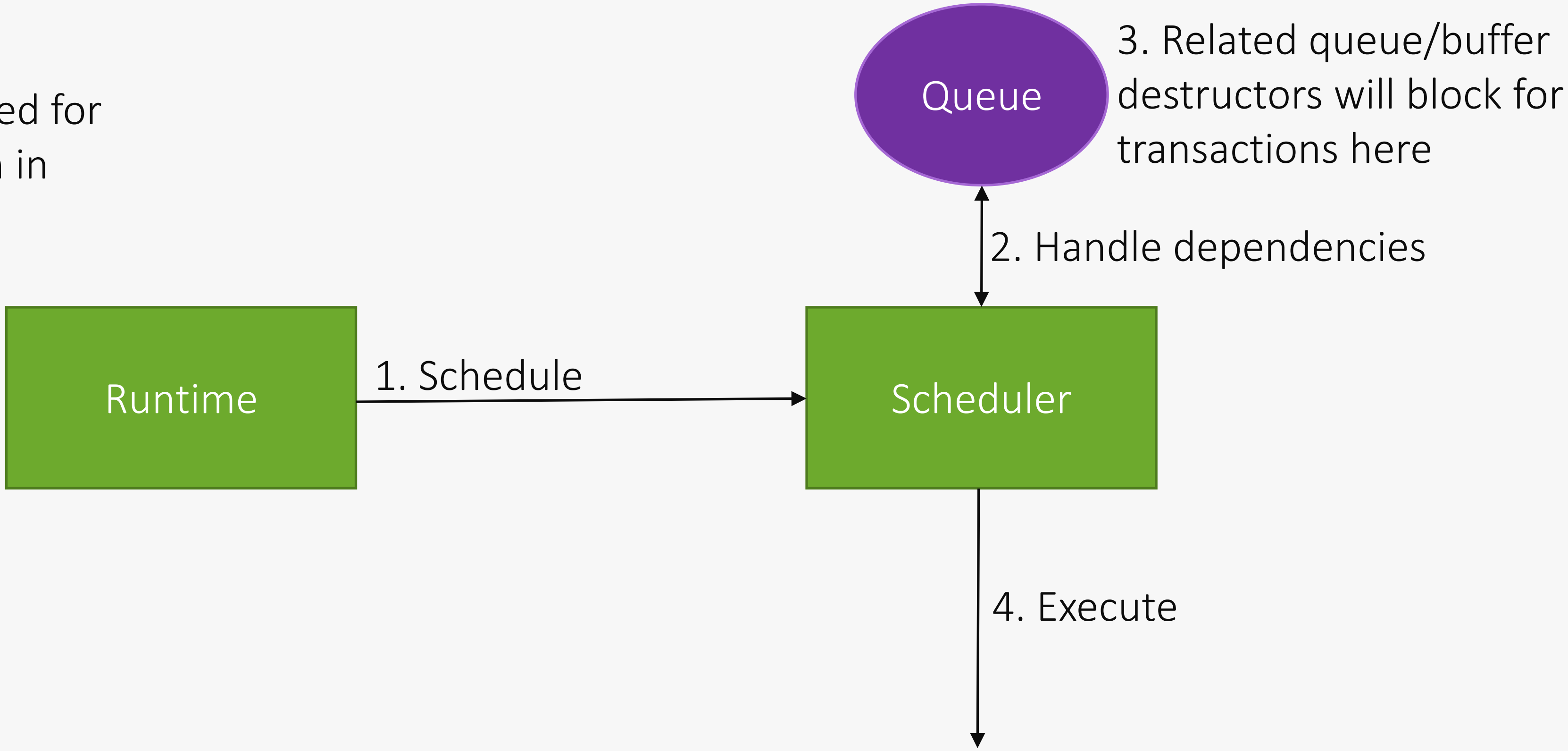
```
queue q{gpu_selector{}, {property::queue::enable_fusion{}}};
{
  buffer<float> buffer1{data1, range};
  buffer<float> buffer2{data2, range,
                                {property::promote_private{}}};
  q.start_fusion();
  q.submit(…);
  q.submit(…);
  …
  q.complete_fusion();
}
```

Kernel Fusion: Minimal Overhead

```
queue q{gpu_selector{}, {property::queue::enable_fusion{}}};
{
  buffer<float> buffer1{data1, range};
  buffer<float> buffer2{data2, range,
                        {property::promote_private{}}};
  q.start_fusion();
  q.submit(…);
  q.submit(…);
  …
  q.complete_fusion();
}
```

Kernel Scheduling

This process is repeated for every kernel to be run in the program.

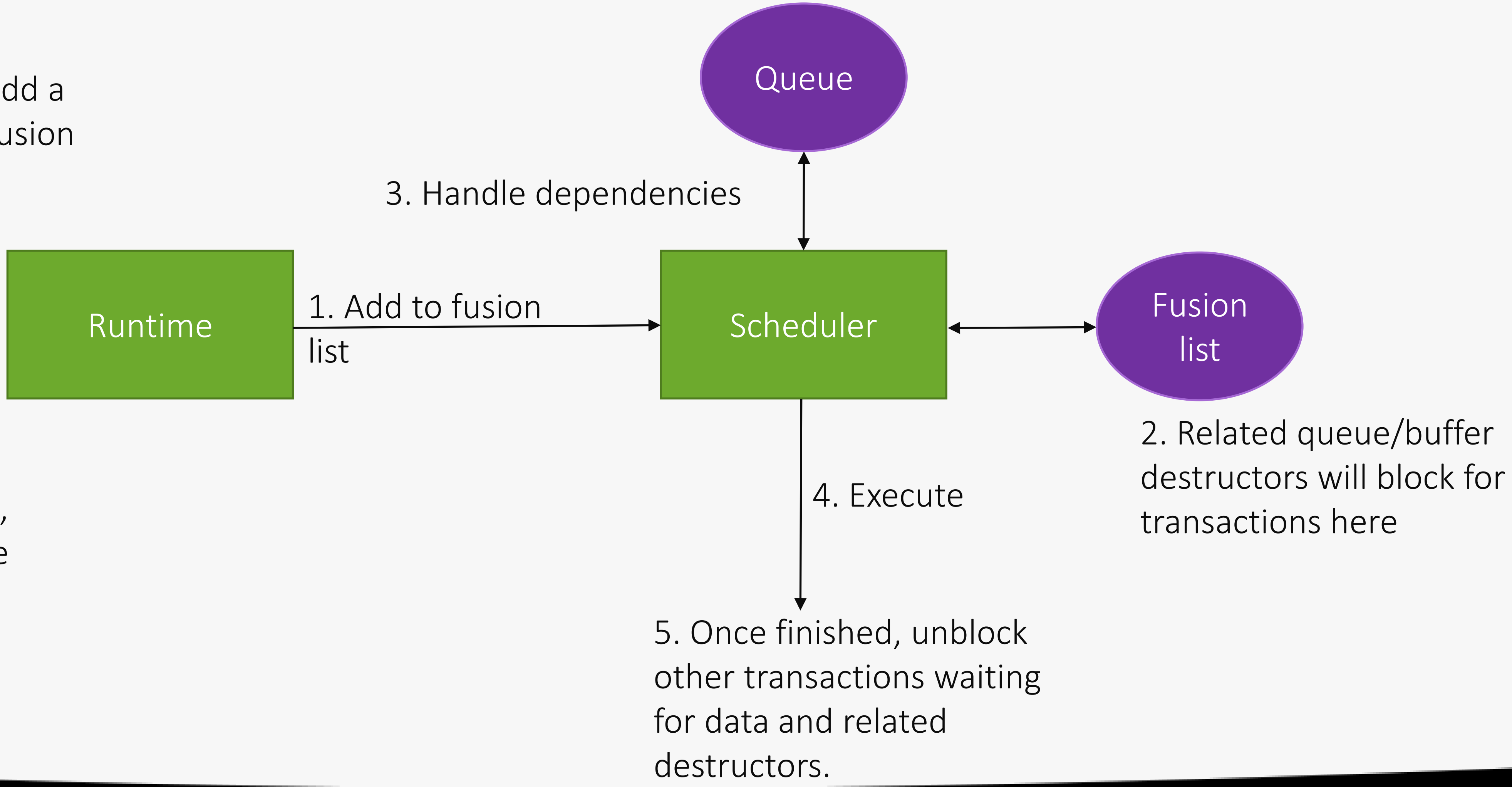


5. Once finished, unblock other transactions waiting for data and related destructors.

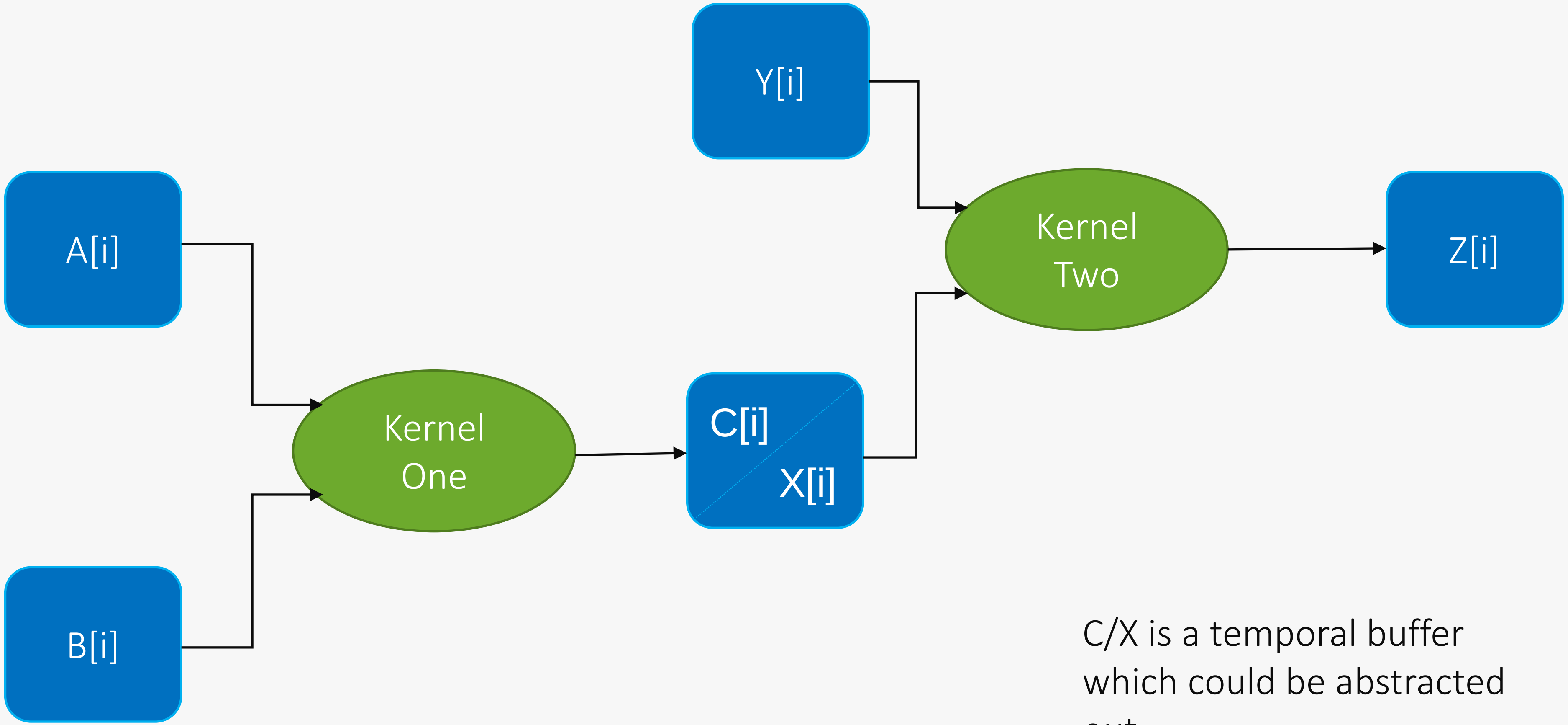
Kernel Fusion

For kernel fusion, we add a new component: the fusion list.

If we call complete_fusion, the fused kernel will be produced and added to the queue; if we call cancel_fusion, each transaction will be added to the queue in submission order.

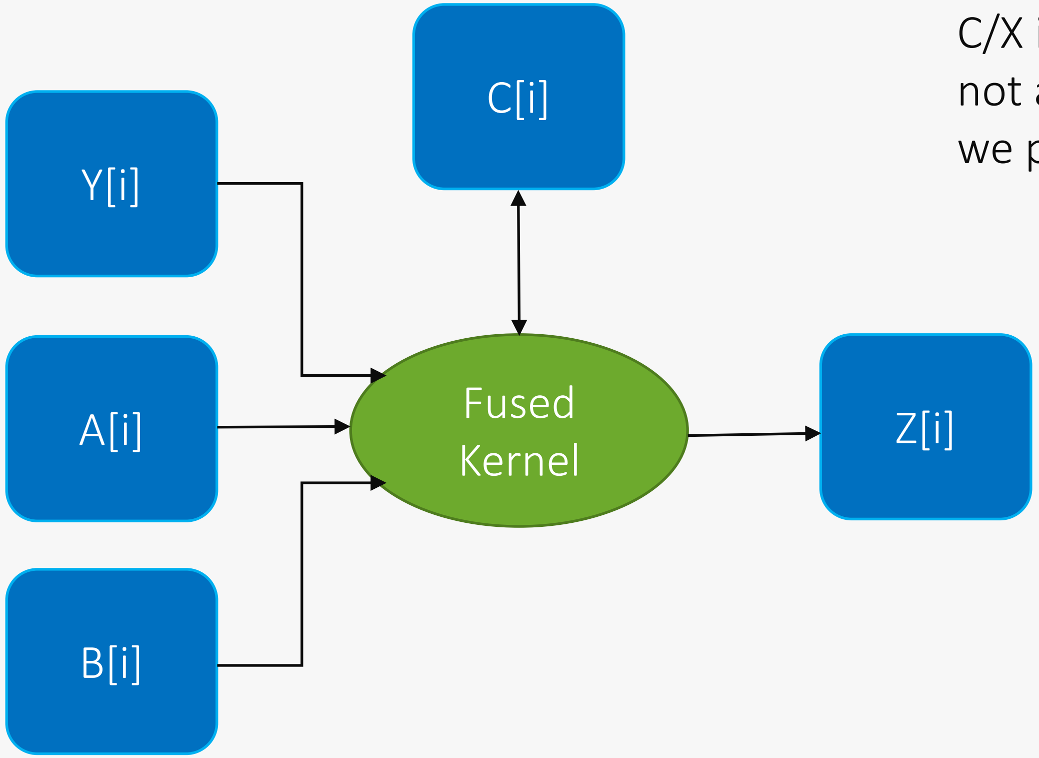


Unfused Kernels Dataflow



C/X is a temporal buffer which could be abstracted out.

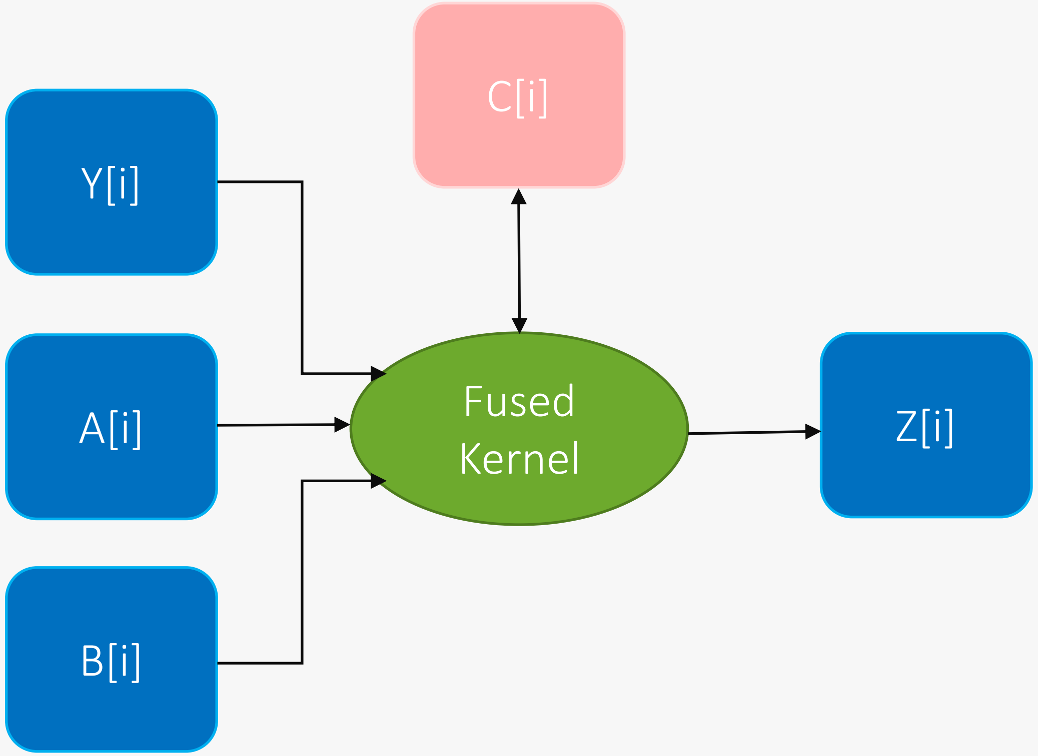
Fused Kernel Dataflow



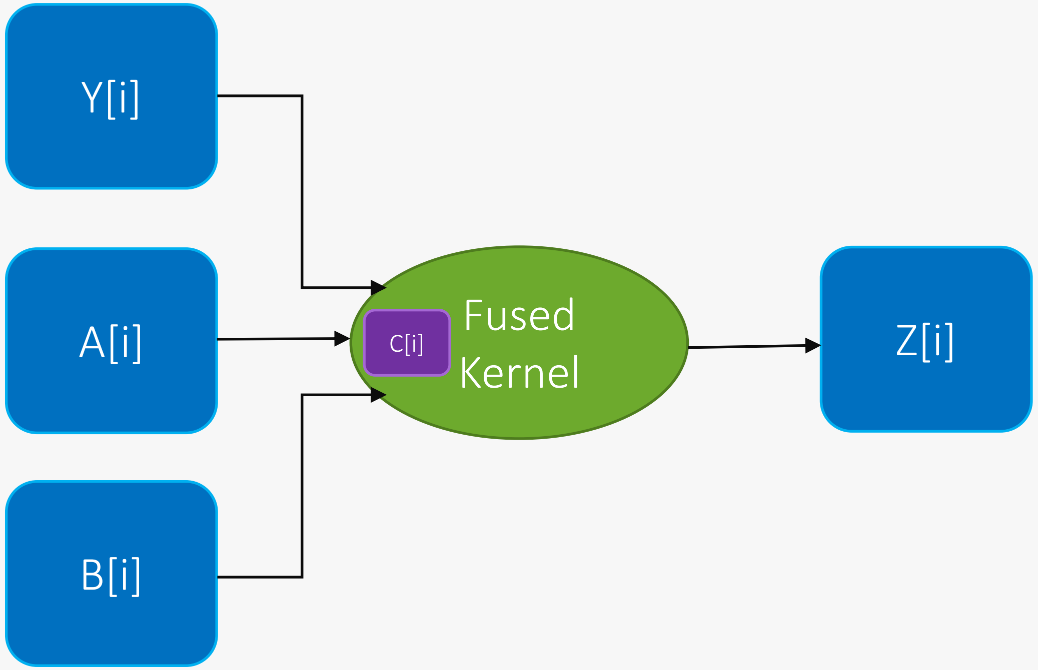
C/X is called C here. We did not abstract it even though we performed fusion.

Fused Kernel Dataflow Using Internalization

Local Internalization
`sycl::property::promote_local`



Private Internalization
`sycl::property::promote_private`

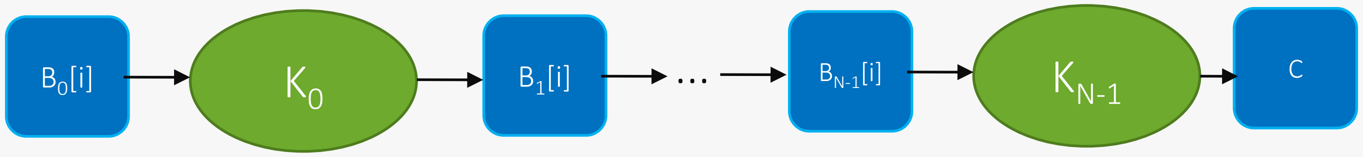


Internalization Requirements

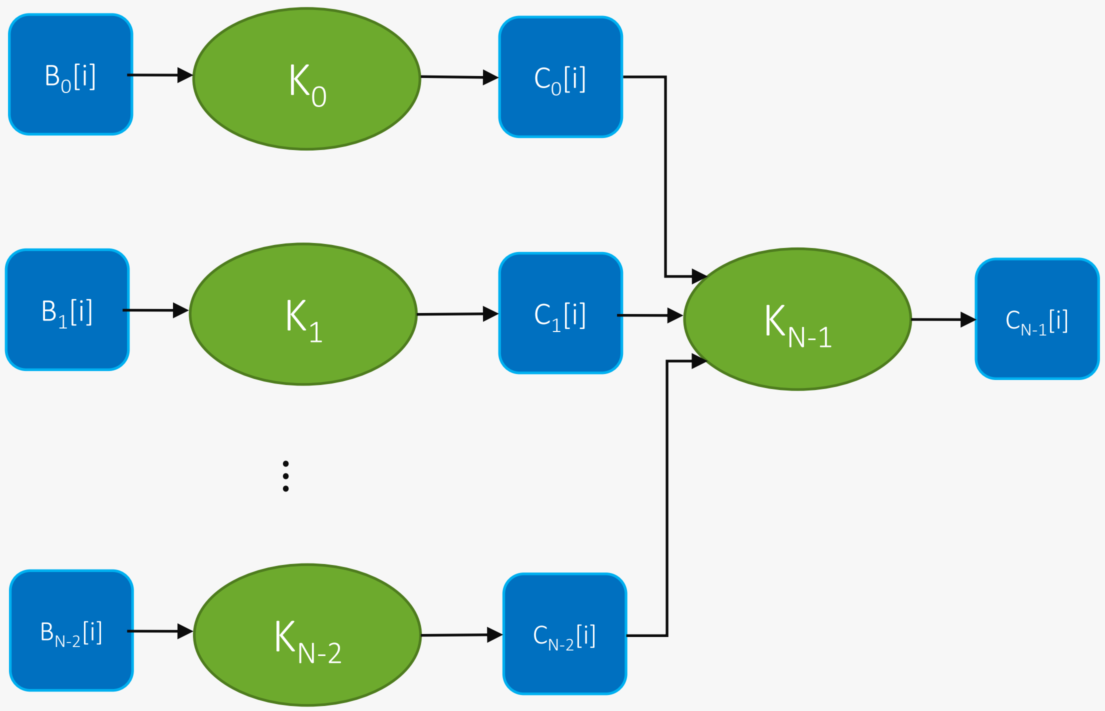
- The overlapped access mode to a promoted accessor must be `read_write`.
- The first kernel using a promoted accessor must have `(read_)write` access to it.

Two Kinds of Data Internalization

Vertical Internalization



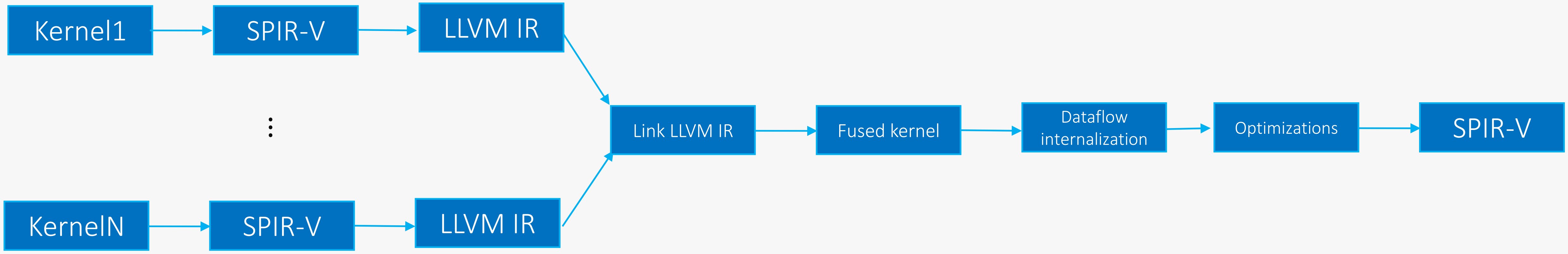
Horizontal Internalization



In horizontal internalization, more resources are needed, as results from different kernels must coexist.

Implementation

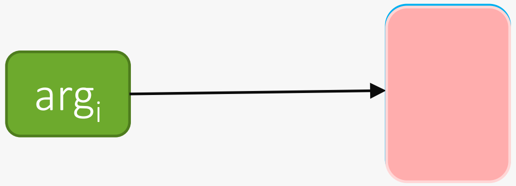
JIT Compilation



Kernels resulting from this process are cached to avoid running the JIT compiler every time, saving between 746 and 23 ms (avg. 112 ms) in our benchmarks.

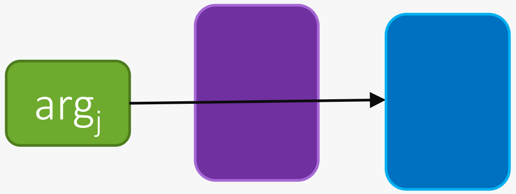
Dataflow Internalization: Overview

1.



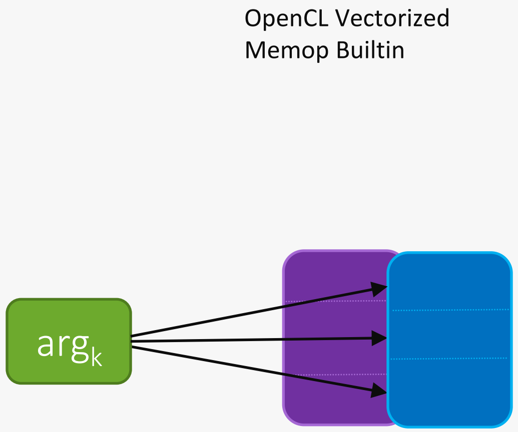
"Cast" to SPIR-V's local address space for local promotion

2.



Replace arguments with allocas in function memory for private promotion.

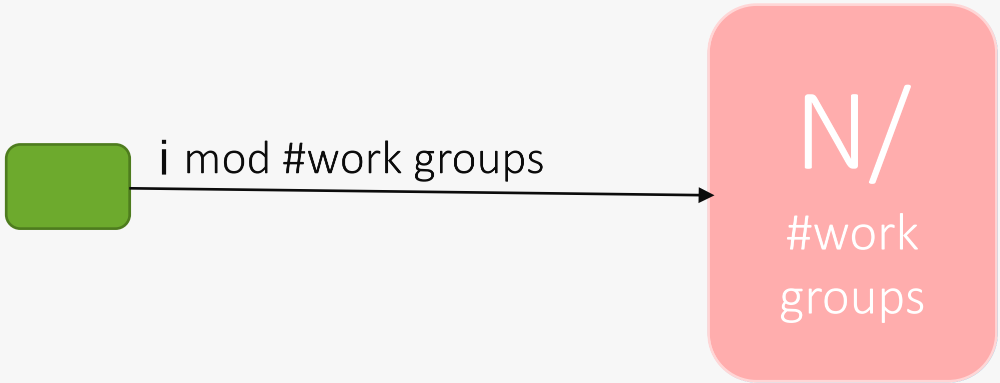
3.



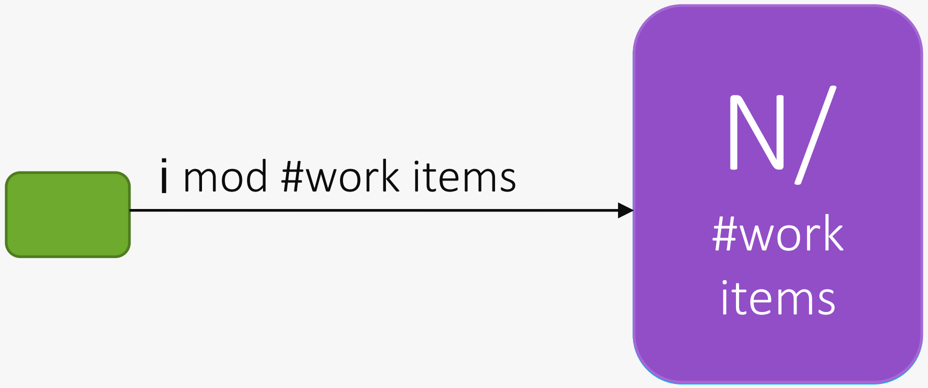
Inline OpenCL vectorized memory operations on privately-promoted arguments.

Dataflow Internalization: Remapping

Local Internalization

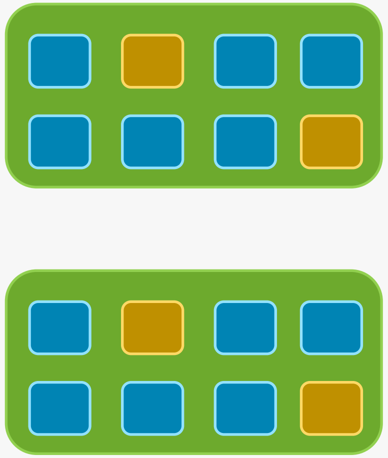


Private Internalization

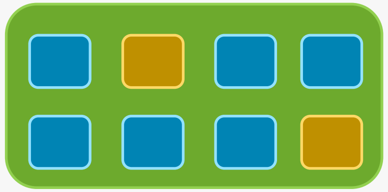


The Kernel Fusion Pipeline

0. Original kernels

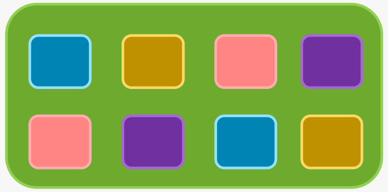


1. Fusion

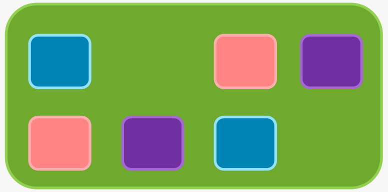


130 instructions,
11 blocks

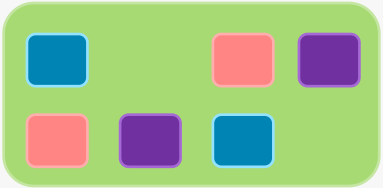
2. Dataflow internalization



3. SYCL constants propagation



4. Other passes
(SROA, Mem2Reg, jump threading, etc.)



21 instructions,
1 block

Case Study 1: SYCL-DNN

SYCL-DNN

- SYCL-DNN provides low level primitives to build Deep Neural Networks (DNN) using SYCL.
- Some of kernels comes from other libraries (like SYCL-Blas)
- SYCL-DNN lacks information for further graph - based optimization such as kernel fusion, graph partitioning, memory reusability, etc.

User-driven Fusion

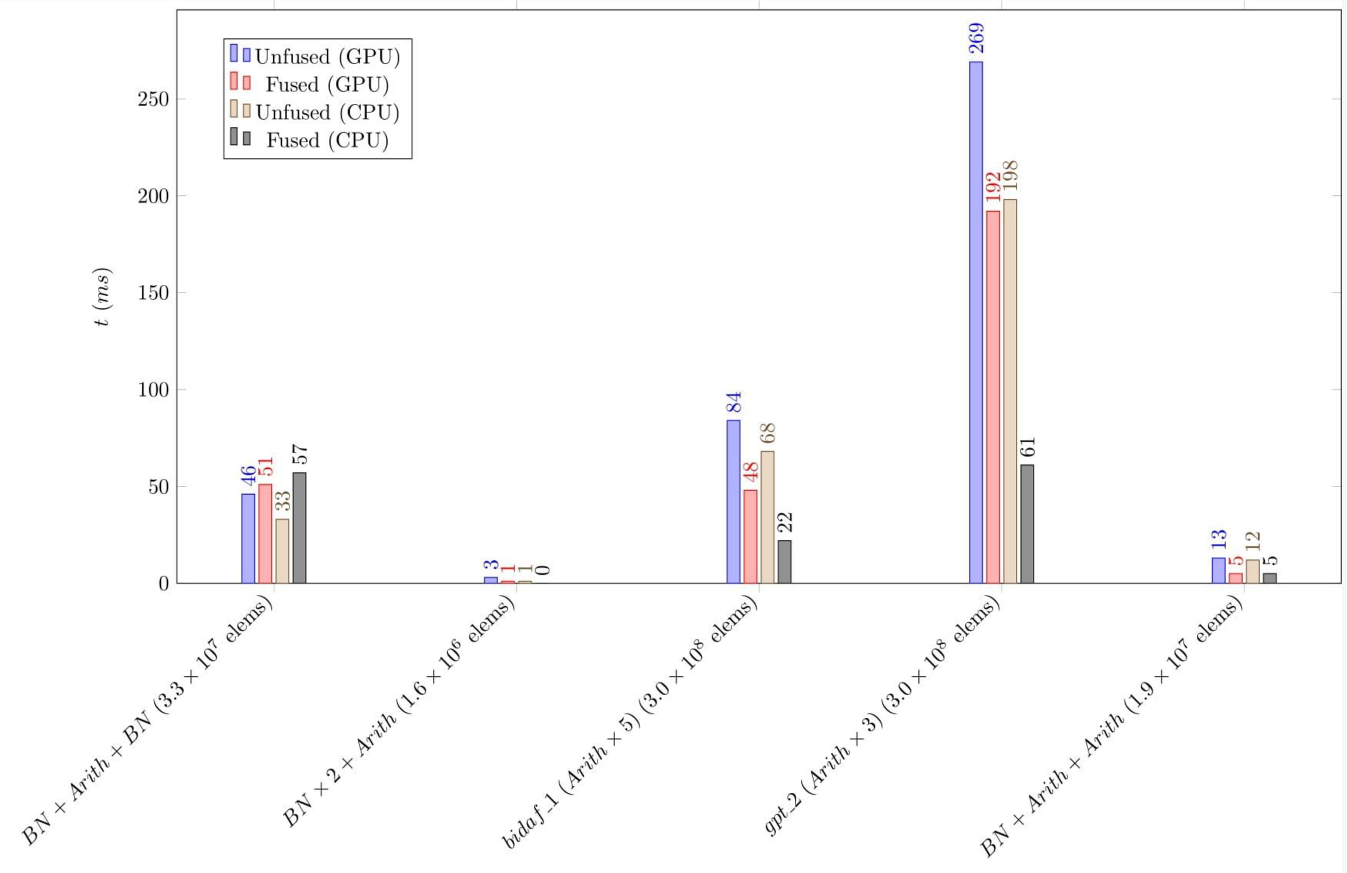
```
enum class sycldnn::backend::Internalization;

enum class sycldnn::covn2d::Fusion;


void start_fusion(Backend &backend);
void cancel_fusion(Backend &backend);
void complete_fusion(Backend &backend, bool addBarriers);
```

Evaluation setup

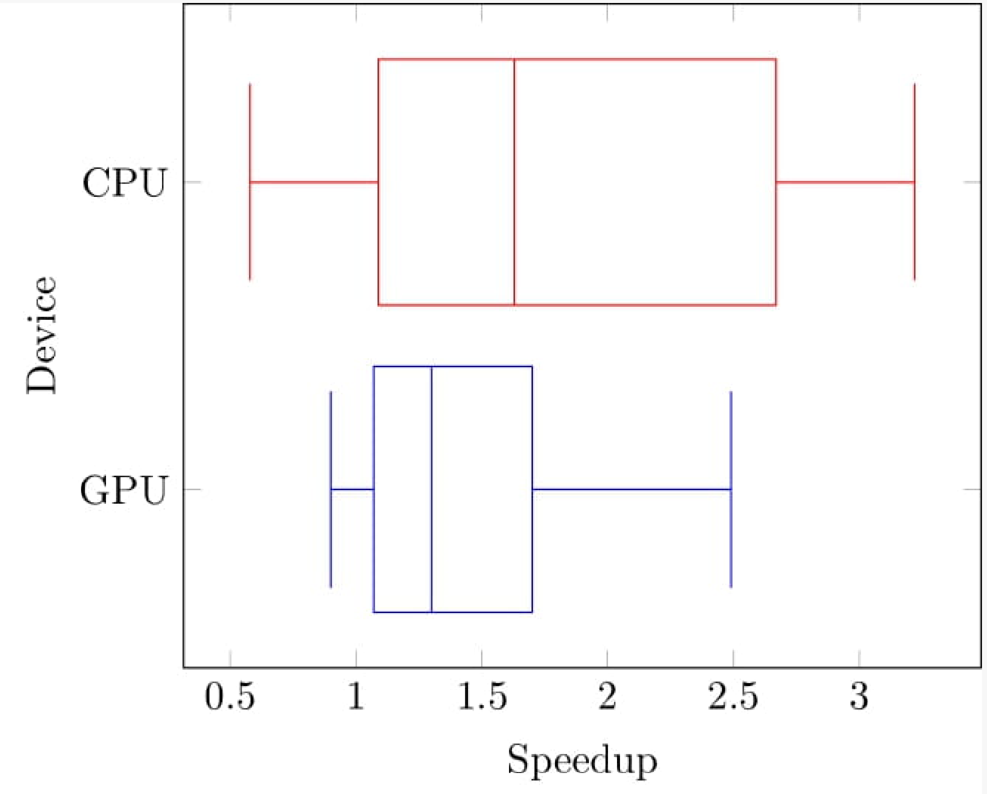
- Several microbenchmarks (arithmetic-heavy kernel fusion, BatchNormalization+Relu...).
- ResNet-50 and VGG16 complete runs (also run forcing a less efficient, more fusion friendly convolution algorithm).

Device type	Model	OpenCL driver Version	OS	SYCL Compiler Version
CPU	Intel i7-6700K	18.1.0.0920	Ubuntu 18.04.6 Kernel 4.1.50	ComputeCpp-PE 2.9.0
GPU	Intel Gen9 HD Graphics NEO	19.41.14441		

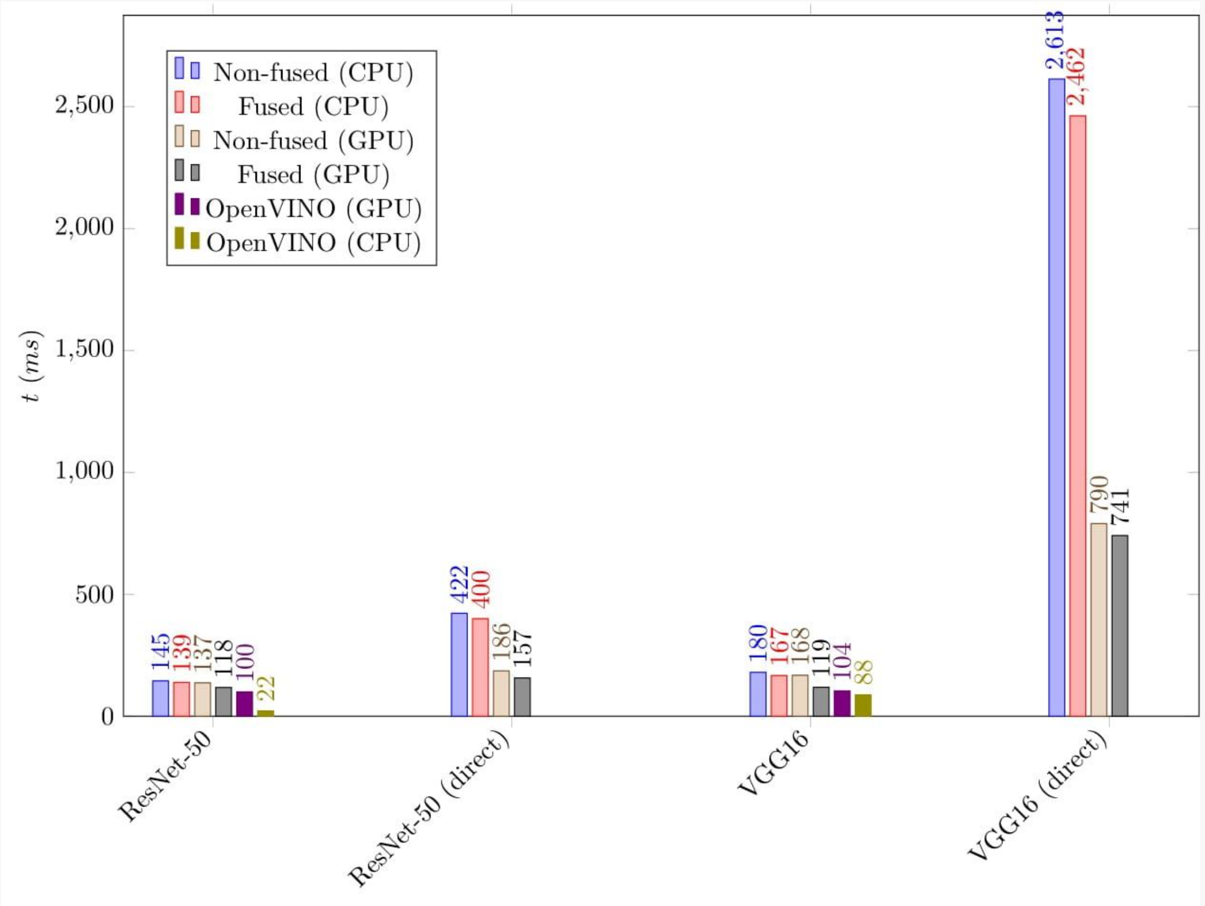
Microbenchmarks Results



Microbenchmarks Results



Full NN Results



Case Study 2: ONNX Runtime

ONNX Runtime

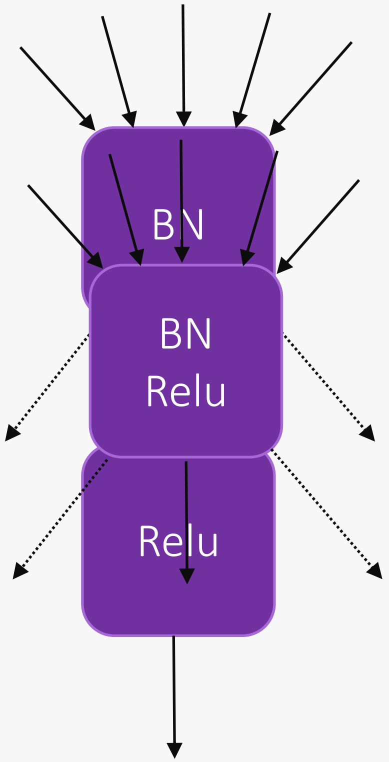
- DNN are represented in the form of a graph where the output(s) of a node are input(s) to another one(s) (ONNX format).
- Provides different execution providers (CPU, CUDA, CoreML, oneDNN, OpenVINO, TensorRT, **SYCL-DNN**, etc.).
- Graph optimizations are applied before running the models.

Graph Optimizations

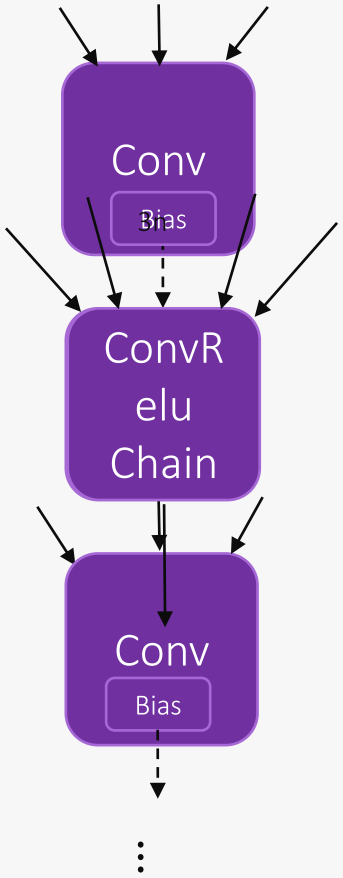
- Graph-level transformations, ranging from small graph simplifications and node eliminations to more complex node fusions and layout optimizations.
- Extended graph optimizations include complex node fusions, suiting our approach.
- We are currently capable of fusing:
  - BatchNormalization Relu Fusion.
  - Conv Relu Chain Fusion.
  - Conv Add Relu Fusion.
  - **2xConv Add Relu Fusion.**

Our Subgraph Fusions

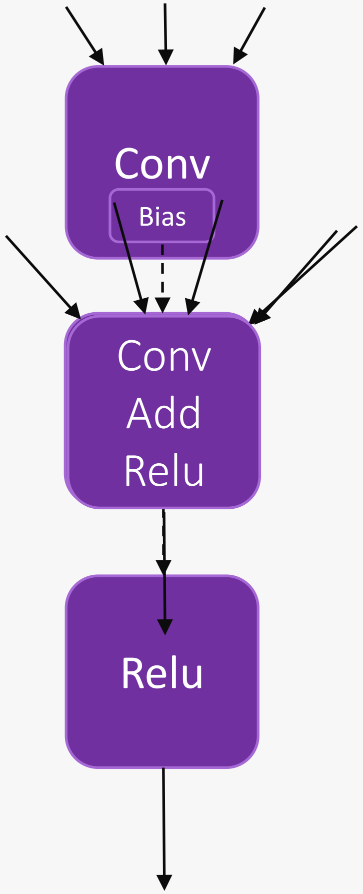
BatchNormalization Relu Fusion



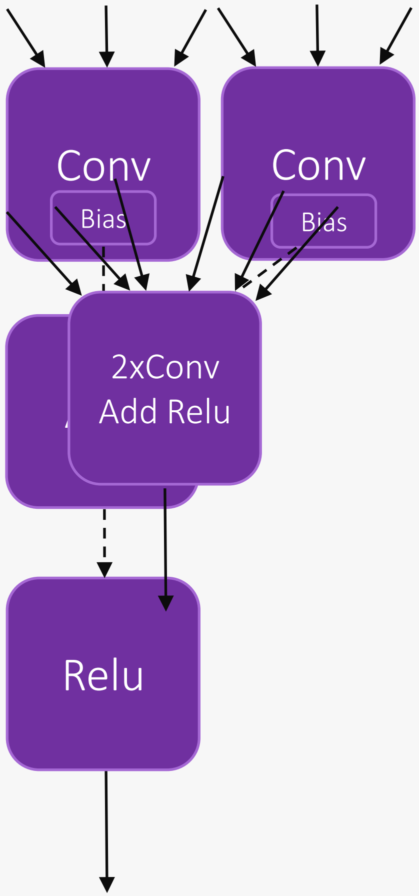
Conv Relu Chain Fusion



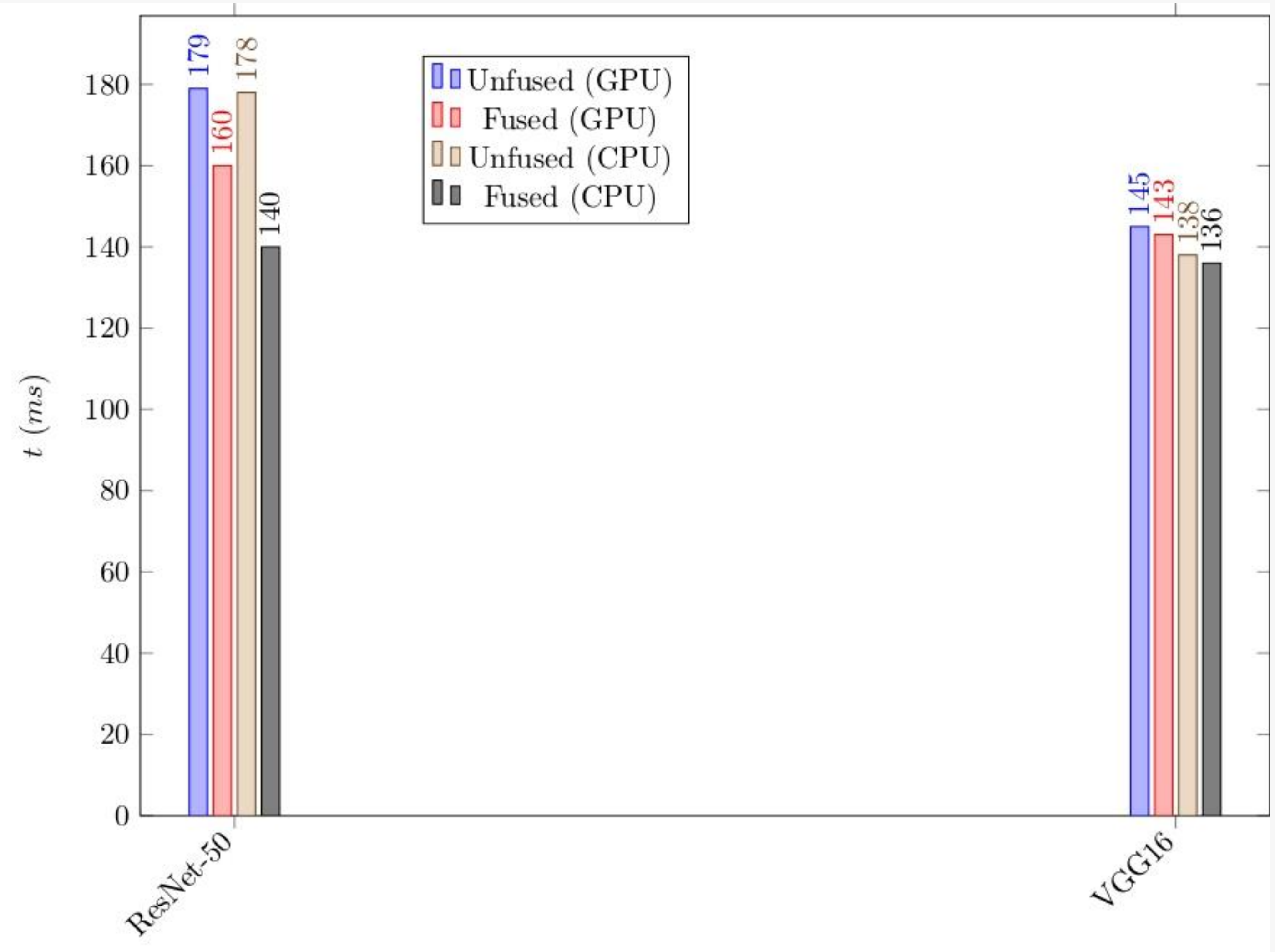
Conv Add Relu Fusion



2xConv Add Relu Fusion



Results



Unfused (GPU)
Fused (GPU)
Unfused (CPU)
Fused (CPU)
t (ms)
0
20
40
60
80
100
120
140
160
180
179
160
178
140
145
143
138
136
ResNet-50
VGG16

Porting to DPC++

Porting to DPC++

- Work on first prototype of port to DPC++ in progress
- Step 1: Port for targets using SPIR-V for device binaries
  - Intel CPU & GPU
  - Existing LLVM-based JIT-compiler can be re-used with small adaptions
  - Integration of SYCL extension API into DPC++ SYCL RT
- Step 2: Add support for targets with non-SPIR-V format
  - For example, Nvidia GPU and AMD GPU
  - In addition to device binary, attach suitable IR (LLVM IR/SPIR-V) to kernel
  - Perform fusion and create device binary from IR at runtime
    - Need PI extension as finalization is specific to each plugin

Conclusions & Future Work

Conclusions

- We implemented a first version of a user-driven kernel fusion SYCL extension.
- SYCL-DNN results are promising (x1.41 in VGG16 on GPU and more than x2.00 in microbenchmarks).
- Kernels making use of several temporal results usually give the best results when performing private internalization (speedups higher than x3.00).
- ONNX Runtime results were not as good. ResNet-50 results were better probably due to subgraphs where we could perform horizontal internalization.

Future Work

- Finalize DPC++ support
- Improve interface to make more evident to the user when fusion is legal/feasible.
- Provide a single property::internalize instead of the current properties. This would require additional analysis (memory access patterns) to decide which kind of internalization to apply.
- Extend support for fusing kernels with different ND-ranges. Currently, same number of dimensions and local range are required. This would require a remapping from original IDs to fused IDs (already explored, not implemented).
- Explore applying fusion to more arithmetic-heavy networks where fusion might really shine.

codeplay

Enable AI & HPC to be Open, Safe and Accessible to All

Any Questions?

@codeplaysoft

info@codeplay.com

codeplay.com