User-driven Online Kernel Fusion For SYCL

VÍCTOR PÉREZ, LUKAS SOMMER, VICTOR LOMÜLLER, KUMUDHA NARASIMHAN, MEHDI GOLI, Codeplay Software Ltd., UK

梁恒中　2023.11.2

Background

Short-running kernels hurt overall performance

- transfer data
- execute kernels

Merge small kernels

- A tedious and error-prone task
- Limited to specific domain

A SYCL Extension

Automatically fuse multiple SYCL kernels:

- Users or software frameworks select kernels to be fused
- The extension creates fused kernel at runtime

Criteria:

- Legality: compute a equivalent result
- Profitability: improve the overall performance

SYCL Overview

Platform Model, Backend Model

Platform, Backend, Context, Device

Memory Model

Buffer, USM, Accessor

Execution Model

Context, Device, Queue, Event, Kernel

```
int main(){
  sycl::device device{ sycl::gpu_selector_v };

  int* A = new int[16];
  int* B = new int[16];

  for(int i = 0; i < 16; i++){
    A[i] = i;
    B[i] = i;
  }

  sycl::buffer<int, 1> bufferA(A, {16});
  sycl::buffer<int, 1> bufferB(B, {16});
  sycl::buffer<int, 1> bufferC(16);

  sycl::queue queue(device);
  queue.submit([&](sycl::handler& h){
    sycl::accessor aA(bufferA, h, sycl::read_only);
    sycl::accessor aB(bufferB, h, sycl::read_only);
    sycl::accessor aC(bufferC, h, sycl::write_only,
                              sycl::no_init);
    h.parallel_for({16}, [=](auto& item){
      sycl::id<1> id = item.get_id();
      aC[id] = aA[id] + aB[id];
    });
  });

  queue.wait();

  sycl::host_accessor hC(bufferC);
  for(int i = 0; i < 16; i++){
    std::cout << hC[i] << std::endl;
  }


  delete [] A;
  delete [] B;
}
```

Kernel Fusion

```
void sycl::queue::start_fusion();

void sycl::queue::cancel_fusion();

sycl::event sycl::queue::complete_fusion(const sycl::property_list &props = {});
```

- Few changes are required
  Some frameworks require a queue instance to be passed as argument
- Abstract away the actual submission of kernels
- Asynchronous kernel submission

Kernel Fusion

- A new property is required for queue to perform a fusion
- Kernels are enqueued for fusion when queue is in fusion mode
- If fusion is aborted, kernels are submitted to the queue for executtion
- If fusion is complete, collected kernels are removed from fusion list and a new kernel is submitted to the queue

```
sycl::queue queue{ gpu_selector_v,
                   {property::queue::enable_fusion{}}};
...
queue.start_fusion();
...
queue.submit(...);
...
queue.complete_fusion();
```

Synchronization

- Host Synchronization

  The host can still synchronize with the execution of submitted kernels or the fused kernel

- Between Kernels

  Introduce a group barrier between each of the fused kernels

  Can be avoided by passing property::no_barriers to complete_fusion()

Dataflow Internalization

Eliminate unnecessary global memory accesses.

Data accesses can be internalized if:

- Two kernels refer to the same memory object
- The memory object is not used by any other kernels(need a global view of the application)

Dataflow Internalization

```
q.submit([&](handler& cgh){
  auto A =  buffer1.get_access<access_mode::read>(cgh);
  auto B =  buffer2.get_access<access_mode::read>(cgh);
  auto C =  buffer3.get_access<access_mode::write>(cgh);
  cgh.parallel_for<class KernelOne>(dataSize,
    [=](id<1> i){
      C[i] = A[i] + B[i];
    });
});

q.submit([&](handler& cgh){
  auto X =  buffer3.get_access<access_mode::read>(cgh);
  auto Y =  buffer4.get_access<access_mode::read>(cgh);
  auto Z =  buffer5.get_access<access_mode::write>(cgh);
  cgh.parallel_for<class KernelTwo>(dataSize,
    [=](id<1> i){
      Z[i] = X[i] * Y[i];
    });
});
```

Dataflow Internalization

Internalization property

- promote_local
- promote_private

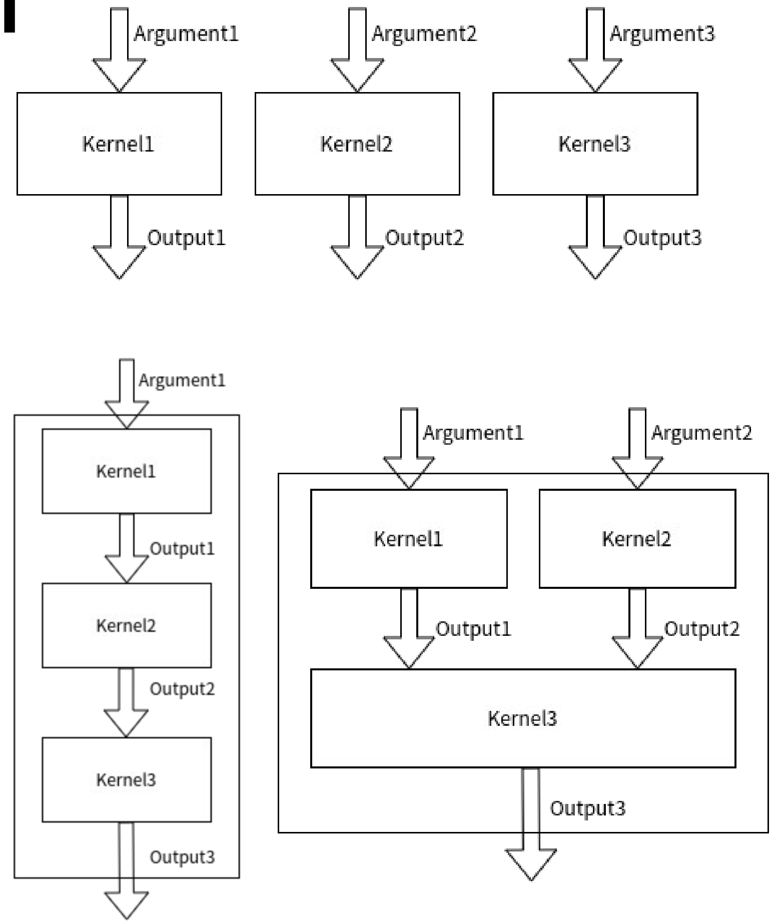
Specified when constructing buffers and accessors.
If a buffer is constructed using internalization property, all accessors refering to the buffer will inherit the property.

```
buffer<float> buffer1{data1 , range<1>{dataSize}};
buffer<float> buffer2{data2 , range<1>{dataSize}};
buffer<float> buffer3{data3 , range<1>{dataSize}, {property::promote_private{}}};
```

Vertical Internalization

For a fused kernel of N kernels K_i:

- The output of the K_i serves as the input of K_{i+1}
- Input of the fused kernel is the input of K_0
- Output of the fused kernel is the output of K_{N-1}

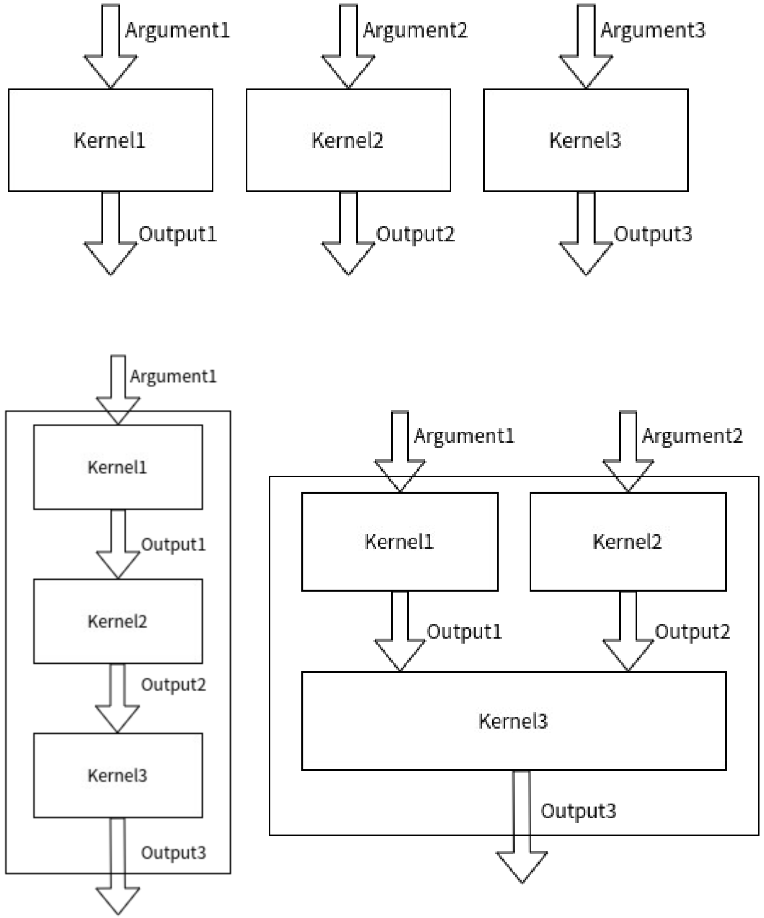


Horizontal Internalization

For a fused kernel of N kernels K_i:

- The output of the K_i, $i \in (0, N\text{-}1)$, serves as the input of K_{N-1}
- Input of the fused kernel is the input of K_i, $i \in (0, N\text{-}1)$
- Output of the fused kernel is the output of K_{N-1}

May require more memory region/registers to hold arguments.



Implementation

JIT compiler

- Translate SPIR-V modules to LLVM IR modules
- Link LLVM IR modules into single module
- Generate a kernel with calls to the input kernels
- Inline the called kernels
- Perform internalization and optimizations
- Translate LLVM IR module back to SPIR-V module

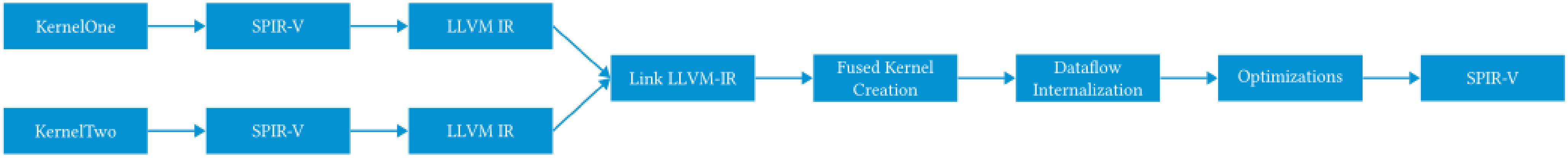


Fig. 4. Outline of the JIT compiler compilation flow.

Implementation

Dataflow Internalization

- Cast argument to appropriate LLVM address space
- Memory access remapping

Other Optimization

- Constants propagation
- Other passes

Evaluation

Table 1. Environment Setup

Device Type	Model	OpenCL driver Version	OS	SYCL Compiler Version
CPU	Intel i7-6700K	2022.13.3.0.16_160000	Ubuntu 18.04.6 Kernel 4.15.0	ComputeCpp-PE 2.10.0
GPU	Intel Gen9 HD Graphics NEO	21.38.21026		

Evaluation:SYCL-DNN

Benchmark name	Source Network(s)	SYCL-DNN Operators
winograd transformation fusion[a]	VGG16	input/weight/output transformation
conv+relu	GoogleNet	Conv + ReLU
convadd	ResNet-50	Conv + Add
batchnorm+arith	ResNet-50	BatchNorm. + Add
batchnorm+arith+arith	DenseNet	BatchNorm. + Mul + Add
batchnormx2+arith	ResNet-50	(BatchNorm. x 2) + Add
batchnorm+arith+batchnorm	ArcFace	BatchNorm. + Add + BatchNorm.
batchnormx2+arith+batchnorm	ArcFace	(BatchNorm. x 2) + Add + BatchNorm.
subsquare	BERT SQuAD-8	Sub + Mul
addadd	BERT SQuAD-8	Add + Add
bertsquad	BERT SQuAD-8	Mul + Mul + Sub + Mul + Add
bidaf_0	BiDAF	Sub + Mul + Add
bidaf_1	BiDAF	Add + Add + Add + Add + Mul
gpt2_0	GPT-2	Div + Mul + Add
gpt2_1	GPT-2	Mul + Sub + Mul
faster_rcnn	Faster R-CNN	Mul + Add + Mul + Add + Add

Evaluation

Table 3. Kernel Fusion Microbenchmarks Evaluation with No Internalization

		GPU			CPU		
Benchmark name	Input size	Unfused (ms)	Fused (ms)	Speedup	Unfused (ms)	Fused (ms)	Speedup
winograd transformation fusion	1.5×10^5	24.49	24.52	1.00	6.51	6.49	1.00
	1.5×10^5	4.39	4.16	1.06	10.90	5.66	1.92
conv+relu	1.0×10^7	433.03	442.81	0.98	785.95	706.72	1.11
	1.3×10^7	616.45	617.06	1.00	1,160.41	1,033.86	1.12
	1.3×10^6	13.34	14.66	0.91	36.14	21.96	1.65
convadd	1.5×10^7	433.41	442.49	0.98	779.92	722.44	1.08
	1.8×10^7	618.01	620.39	1.00	1,171.08	1,035.20	1.13
batchnorm+arith	2.0×10^5	0.59	0.47	1.26	0.31	0.34	0.90
	4.0×10^5	0.73	0.38	1.91	0.48	0.51	0.94
batchnorm+arith+arith	1.9×10^7	13.29	11.45	1.16	14.15	11.51	1.23
	9.1×10^7	73.15	61.73	1.18	67.94	52.67	1.29
batchnormx2+arith	4.0×10^5	0.92	0.58	1.58	0.40	0.41	0.96
	1.6×10^6	1.13	0.45	2.54	0.61	0.59	1.03
batchnorm+arith+batchnorm	2.0×10^6	2.80	2.74	1.02	2.75	3.09	0.89
	8.3×10^6	9.13	8.11	1.13	10.44	10.54	0.99
	3.3×10^7	43.34	41.44	1.05	40.52	39.12	1.04
batchnormx2+arith+batchnorm	2.0×10^6	4.56	3.85	1.18	3.88	4.05	0.96
	8.3×10^6	11.56	11.00	1.05	14.06	13.17	1.07
	3.3×10^7	56.16	53.31	1.05	53.11	48.89	1.09
subsquare	1.5×10^8	144.08	140.62	1.02	105.29	119.85	0.88
addadd	2.3×10^8	134.15	126.42	1.06	94.65	108.17	0.88
bertsquad	3.8×10^8	264.44	206.21	1.28	207.05	154.40	1.34
bidaf_0	8.0×10^7	52.72	44.33	1.19	39.42	37.45	1.05
bidaf_1	1.2×10^8	82.66	61.51	1.34	68.66	56.37	1.22
gpt2_0	3.0×10^8	191.12	160.39	1.19	151.36	143.07	1.06
gpt2_1	3.0×10^8	262.49	218.11	1.20	196.47	165.10	1.19
faster_rcnn	3.0×10^8	201.41	159.61	1.26	153.24	128.15	1.20

Evaluation

Table 4. Kernel Fusion Microbenchmarks Evaluation with Private Internalization

		GPU			CPU		
Benchmark name	Input size	Unfused (ms)	Fused (ms)	Speedup	Unfused (ms)	Fused (ms)	Speedup
winograd transformation fusion	1.5×10^5	18.07	17.59	1.03	7.34	6.40	1.15
	1.5×10^5	4.56	4.17	1.10	10.20	5.11	2.00
conv+relu	1.0×10^7	432.74	438.02	0.99	746.15	706.88	1.06
	1.3×10^7	617.42	621.17	0.99	1,159.77	1,020.17	1.14
	1.3×10^6	13.21	13.58	0.97	34.52	17.89	1.93
convadd	1.5×10^7	434.30	441.69	0.98	738.90	721.64	1.02
	1.8×10^7	619.07	624.05	0.99	1,156.38	1,022.03	1.13
batchnorm+arith	2.0×10^5	0.54	0.39	1.38	0.32	0.27	1.20
	4.0×10^5	0.70	0.41	1.72	0.48	0.38	1.26
batchnorm+arith+arith	1.9×10^7	13.60	7.34	1.85	14.05	6.56	2.14
	9.1×10^7	71.40	51.78	1.38	67.68	29.40	2.30
batchnormx2+arith	4.0×10^5	0.90	0.56	1.62	0.39	0.34	1.13
	1.6×10^6	1.12	0.54	2.09	0.60	0.47	1.27
	2.0×10^6	2.53	1.78	1.42	2.19	1.98	1.11
batchnorm+arith+batchnorm	8.3×10^6	8.77	7.85	1.12	6.21	5.74	1.08
	3.3×10^7	42.65	46.92	0.91	40.34	35.91	1.12
	2.0×10^6	4.07	2.43	1.67	3.65	2.34	1.56
batchnormx2+arith+batchnorm	8.3×10^6	11.68	8.96	1.30	8.31	6.71	1.24
	3.3×10^7	55.43	56.79	0.98	54.72	40.19	1.36
subsquare	1.5×10^8	140.69	113.29	1.24	105.24	37.48	2.81
addadd	2.3×10^8	131.00	119.49	1.10	94.64	49.24	1.92
bertsquad	3.8×10^8	256.12	189.92	1.35	206.99	73.12	2.83
bidaf_0	8.0×10^7	51.74	38.55	1.34	39.36	16.28	2.42
bidaf_1	1.2×10^8	81.39	44.26	1.84	68.54	22.55	3.04
gpt2_0	3.0×10^8	187.88	136.76	1.37	151.25	61.15	2.47
gpt2_1	3.0×10^8	256.89	177.37	1.45	196.31	61.19	3.21
faster_rcnn	3.0×10^8	196.58	134.60	1.46	153.11	56.94	2.69

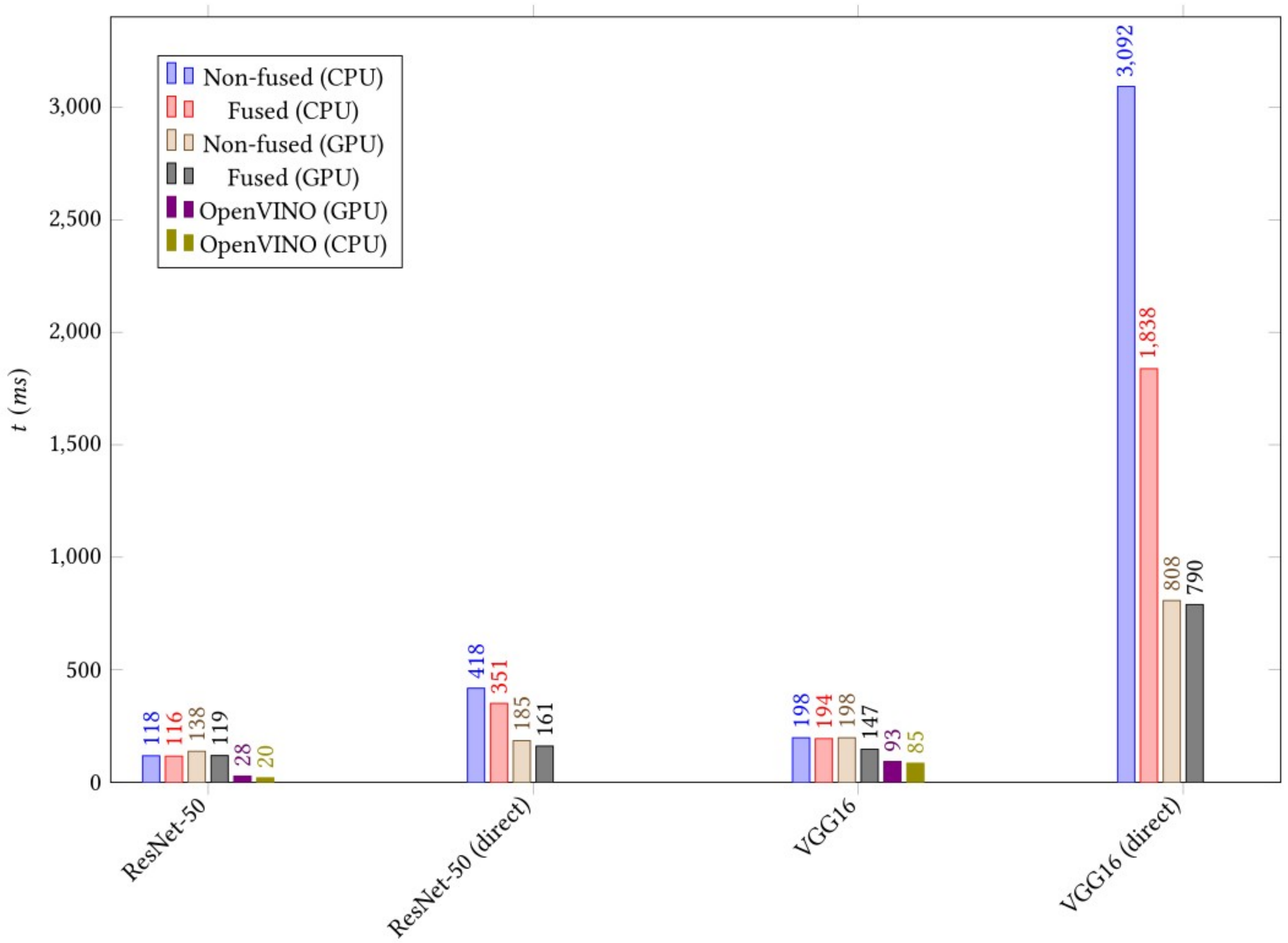
Evaluation

Default:

- more optimized,
- unable to run with internalization

Direct:

- More fusion friendly



Evaluation

Six out of nine benchmarks are able to perform fusion.

Only covariance and gramschmidt allow internalization.

Table 5. Results of Applying Kernel Fusion to the SYCL-Bench Suite

		GPU			CPU		
Benchmark name	Input size	Unfused (ms)	Fused (ms)	Speedup	Unfused (ms)	Fused (ms)	Speedup
3mm	4.0×10^6	322.04	326.43	0.99	280.79	336.95	0.83
	1.6×10^7	2,419.47	2,276.86	1.06	3,089.72	3,139.02	0.98
	3.6×10^7	9,341.02	9,343.94	1.00	10, 914.26	11,146.45	0.98
bicg	1.0×10^6	8.87	1.81	4.91	5.57	1.70	3.28
	4.0×10^6	16.99	9.59	1.77	10.49	7.34	1.43
	9.0×10^6	27.55	20.49	1.34	22.12	17.29	1.28
	1.6×10^7	41.65	34.45	1.21	33.12	30.09	1.10
	2.5×10^7	65.05	54.96	1.18	58.42	46.61	1.25
	1.0×10^8	222.23	210.10	1.06	198.82	200.61	0.99
	4.0×10^8	838.31	823.96	1.02	824.03	817.90	1.01
	9.0×10^8	1,862.21	1,836.89	1.01	1,923.53	1,956.40	0.98
	1.6×10^9	2,896.05	2,408.19	1.20	3,725.31	3,743.00	1.00
	2.5×10^9	4,509.51	3,698.19	1.22	6,091.05	6,071.58	1.00
correlation	1.0×10^6	475.03	364.74	1.30	168.00	166.09	1.01
	4.0×10^6	3,014.59	2,682.79	1.12	1,443.61	1,451.46	0.99
	9.0×10^6	9,380.68	8,795.50	1.07	5,774.76	5,869.03	0.98
covariance	1.0×10^6	478.47	366.41	1.31	172.91	166.93	1.04
	4.0×10^6	3,042.90	2,685.36	1.13	1,432.00	1,420.00	1.01
	9.0×10^6	9,376.02	8,880.75	1.06	5,593.98	5,659.49	0.99
fdtd2d	3.0×10^6	2,095.92	1,780.02	1.18	1,347.24	1,499.20	0.90
	1.2×10^7	7,275.26	6,505.02	1.12	6,146.34	5,783.75	1.06
	2.7×10^7	15,095.85	13,872.71	1.09	13,724.11	12 , 818.70	1.07
gramschmidt	3.0×10^6	1,905.71	548.06	3.48	2,412.43	2,717.39	0.89
	1.2×10^7	8,309.64	3,694.80	2.25	19,774.62	24,075.27	0.82
	2.7×10^7	23,488.17	9,972.67	2.36	67,750.92	69,185.15	0.98

Case Insights

Bidaf_1

- 1.84x on GPU & 3.04x on CPU
- Internalization: 4.6×10^7st & 7.9×10^7ld → 5.4×10^6st & 3.4×10^7ld
- Reduced instructions: $3.1 \times 10^8 \rightarrow 5.8 \times 10^6$
- Reduced synchronization

3mm

- 0.99x on GPU & 0.83x on CPU
- No internalization, worse cache performance: $7.8 \times 10^5 \rightarrow 4.0 \times 10^6$ LLC misses
- 5.24x more data read from DRAM on GPU

Covariance

- 1.06x on GPU & 0.99x on CPU
- The last fused kernel takes most of the computation time

JIT Overhead

Benchmark name	Input size	Time w/o Cache (ms)	Time w/ Cache (ms)	JIT time (ms)
winograd transformation fusion	1.5×10^5	786.73	22.68	764.05
	1.5×10^5	263.98	4.45	259.53
conv+relu	1.0×10^7	578.08	439.66	138.42
	1.3×10^7	758.48	619.99	138.49
	1.3×10^6	180.47	13.54	166.93
convadd	1.5×10^7	583.38	443.60	139.79
	1.8×10^7	766.75	623.78	142.97
batchnorm+arith	2.0×10^5	67.04	0.73	66.31
	4.0×10^5	67.15	0.57	66.58
batchnorm+arith+arith	1.9×10^7	84.92	8.71	76.21
	9.1×10^7	128.24	53.57	74.67
batchnormx2+arith	4.0×10^5	71.98	0.65	71.33
	1.6×10^6	72.12	0.70	71.42
	2.0×10^6	75.92	2.92	73.00
batchnorm+arith+batchnorm	8.3×10^6	88.47	5.85	82.62
	3.3×10^7	120.81	46.61	74.21
	2.0×10^6	85.45	3.62	81.83
batchnormx2+arith+batchnorm	8.3×10^6	92.12	9.24	82.89
	3.3×10^7	134.65	57.22	77.43
subsquare	1.5×10^8	229.77	118.92	110.85
addadd	2.3×10^8	183.48	131.66	51.82
bertsquad	3.8×10^8	262.97	196.91	66.06
bidaf_0	8.0×10^7	106.08	39.93	66.15
bidaf_1	1.2×10^8	111.37	47.19	64.18
gpt2_0	3.0×10^8	257.52	138.25	119.27
gpt2_1	3.0×10^8	240.21	175.79	64.42
faster_rcnn	3.0×10^8	205.71	140.68	65.03

Conclusion

A user-driven kernel fusion extension for SYCL

- Automatic kernel fusion
- Dataflow internalization
- Performance improvement on several benchmarks