Breaking the last line of performance border

Michal Mrozek

IWOCL 2019

Legal Notice and Disclaimers

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at intel.com, or from the OEM or retailer.

No computer system can be absolutely secure.

Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase. For more complete information about performance and benchmark results, visit **http://www.intel.com/benchmarks**.

OpenCL and the OpenCL logo are trademarks of Apple Inc. used by permission by Khronos.

** Other names and brands may be claimed as the property of others.*

Legal Disclaimer and Optimization Notice

INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Optimization Notice

Intel’s compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

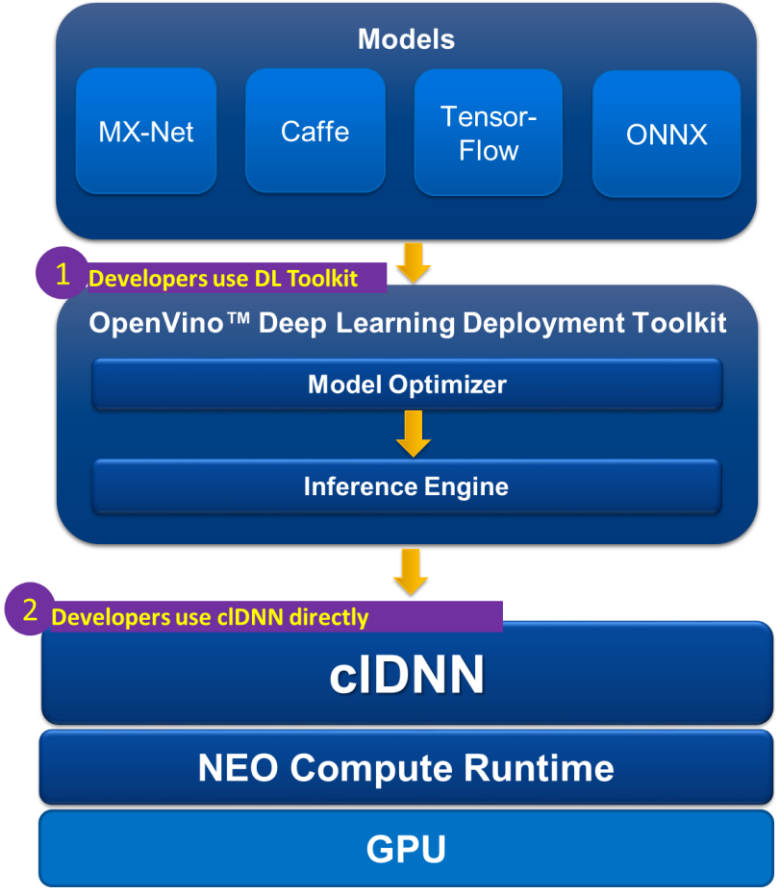
Quick introduction

clDNN:

- Contains kernels-primitives optimized for DNN inference acceleration on Intel® SKL+ GPU devices
- Supports most of commonly known latest neural network topologies
- Delivered with Intel® OpenVino™ Deep Learning Deployment Toolkit which supports Caffe, Tensor-Flow, ONNX and MX-Net models.
- Check out here https://github.com/opencv/dldt

Neo Compute Runtime:

- Unified OpenCL driver supporting BDW+ Intel® GPU devices
- Check out here https://github.com/intel/compute-runtime

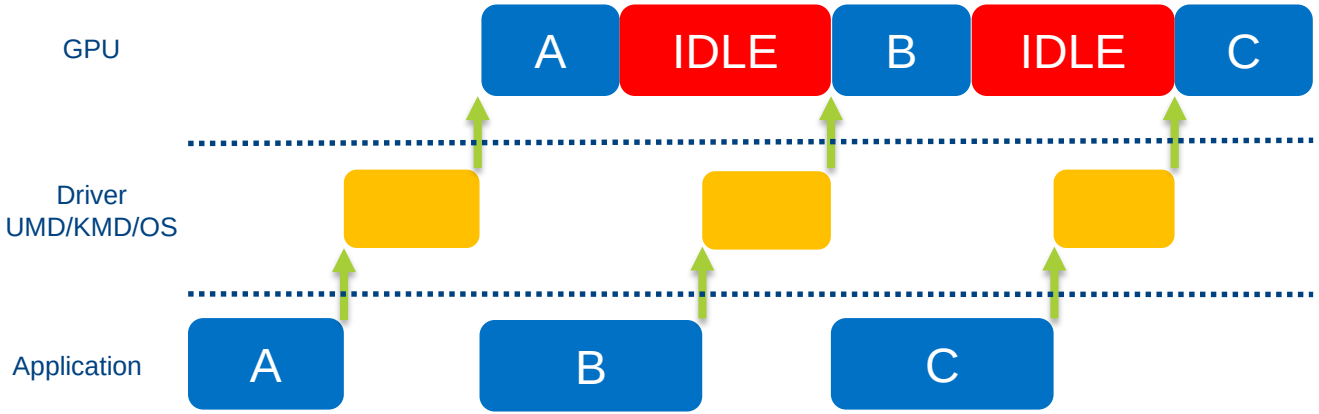


Agenda

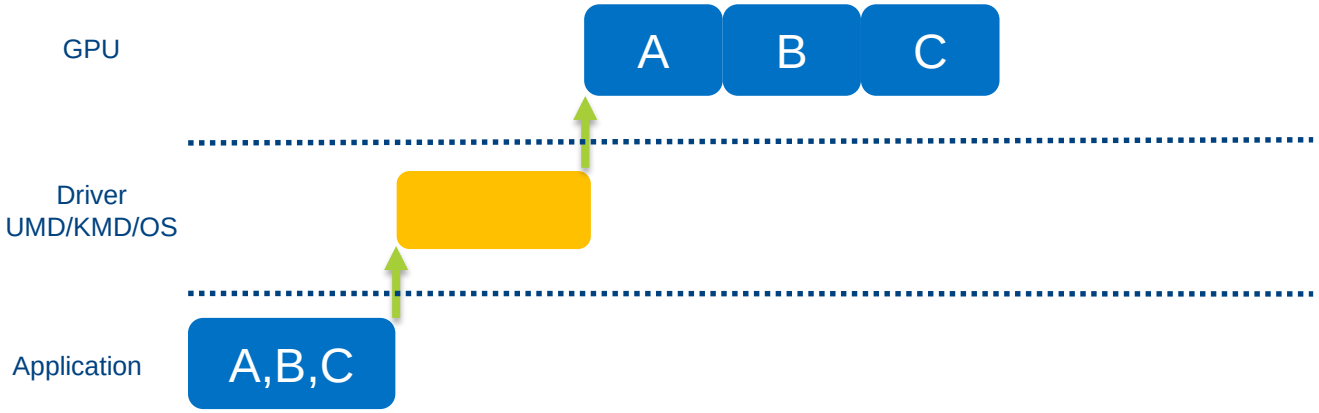
- Primitive vs Graph
- Let's optimize graphs!
  - Offload execution
  - Utilizing padding
  - Data Fusing
  - Primitive Fusing
  - Kernel Selection
  - Optimizing execution order
  - Kernel level optimizations
- Performance Results

Primitive vs Graph

Primitive vs Graph

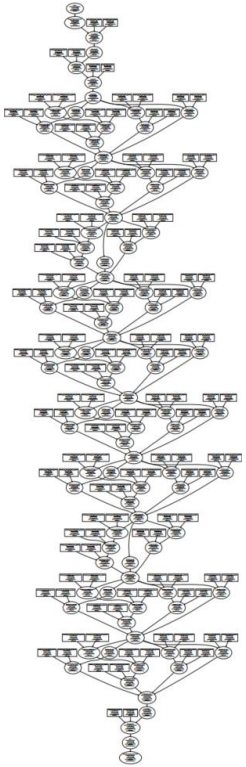


Primitive based:
- No locality
- Bubbles
- Lot of driver calls
- Many command buffers



Graph based:
- No bubbles
- Good locality
- One Command Buffer
- Good GPU utilization

Graph compilation



Graph Compilation

Offline execution

Layout selection

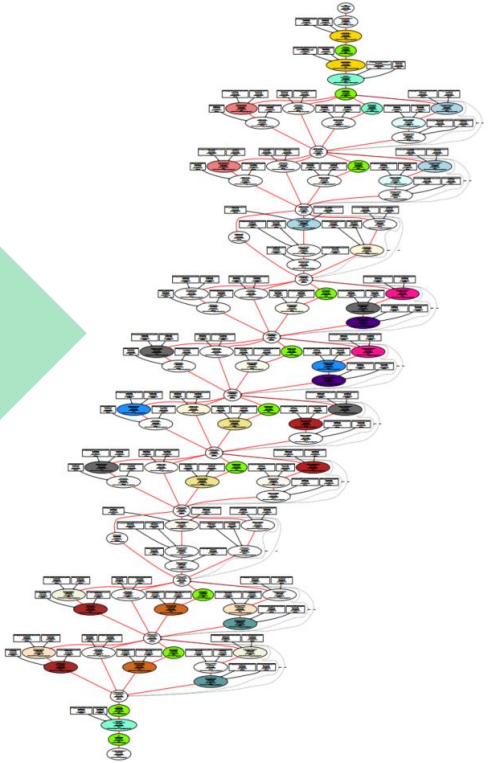
Padding

Fusing

Execution order

Kernel selection

Kernel optimizations



Let’s optimize graphs!

Layout selection

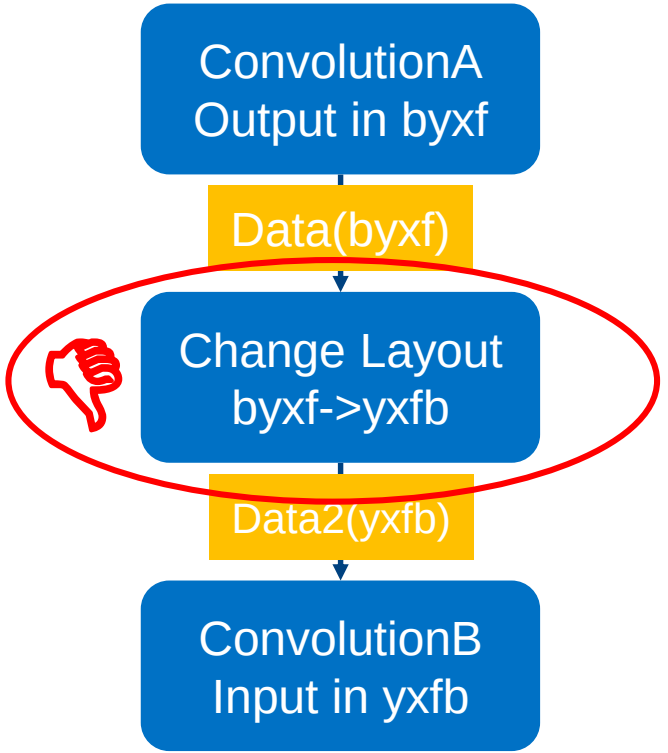
BYXF

b0f0y0x0	b0f1y0x0	b0f2y0x0	b0f0y0x1	b0f1y0x1	b0f2y0x1	b0f0y0x2	b0f1y0x2	b0f2y0x2
b0f0y1x0	b0f1y1x0	b0f2y1x0	b0f0y1x1	b0f1y1x1	b0f2y1x1	b0f0y1x2	b0f1y1x2	b0f2y1x2
b0f0y2x0	b0f1y2x0	b0f2y2x0	b0f0y2x1	b0f1y2x1	b0f2y2x1	b0f0y2x2	b0f1y2x2	b0f2y2x2
b1f0y0x0	b1f1y0x0	b1f2y0x0	b1f0y0x1	b1f1y0x1	b1f2y0x1	b1f0y0x2	b1f1y0x2	b1f2y0x2

...

YXFB

b0f0y0x0	b1f0y0x0	b0f1y0x0	b1f1y0x0	b0f2y0x0	b1f2y0x0	b0f0y0x1	b1f0y0x1	b0f1y0x1	b1f1y0x1
b0f2y0x1	b1f2y0x1	b0f0y0x2	b1f0y0x2	b0f1y0x2	b1f1y0x2	b0f2y0x2	b1f2y0x2	b0f0y1x0	b1f0y1x0



Layout selection

Convolution1
Output in byxf

Data(byxf)

Convolution2
Input in byxf

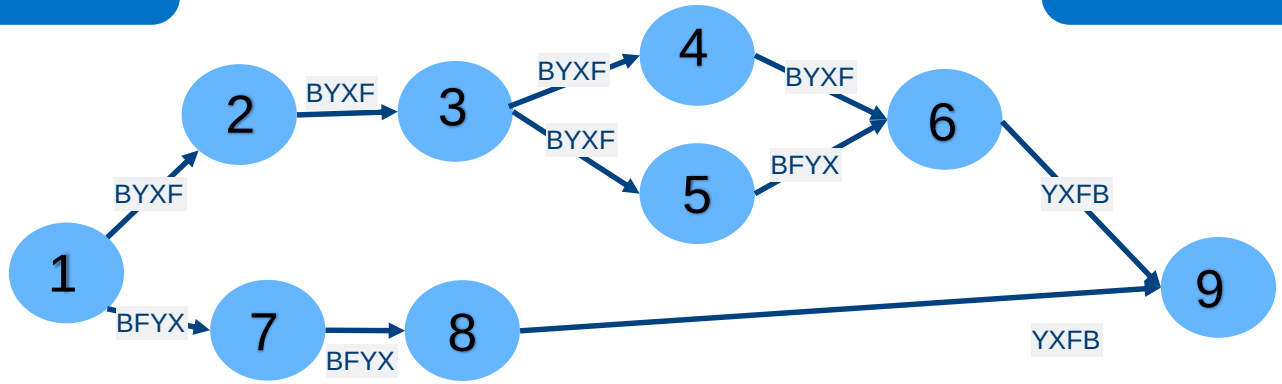
How:
Have multiple versions of kernels accepting / producing different layouts

- Layout changing kernel eliminated
- Memory footprint reduced

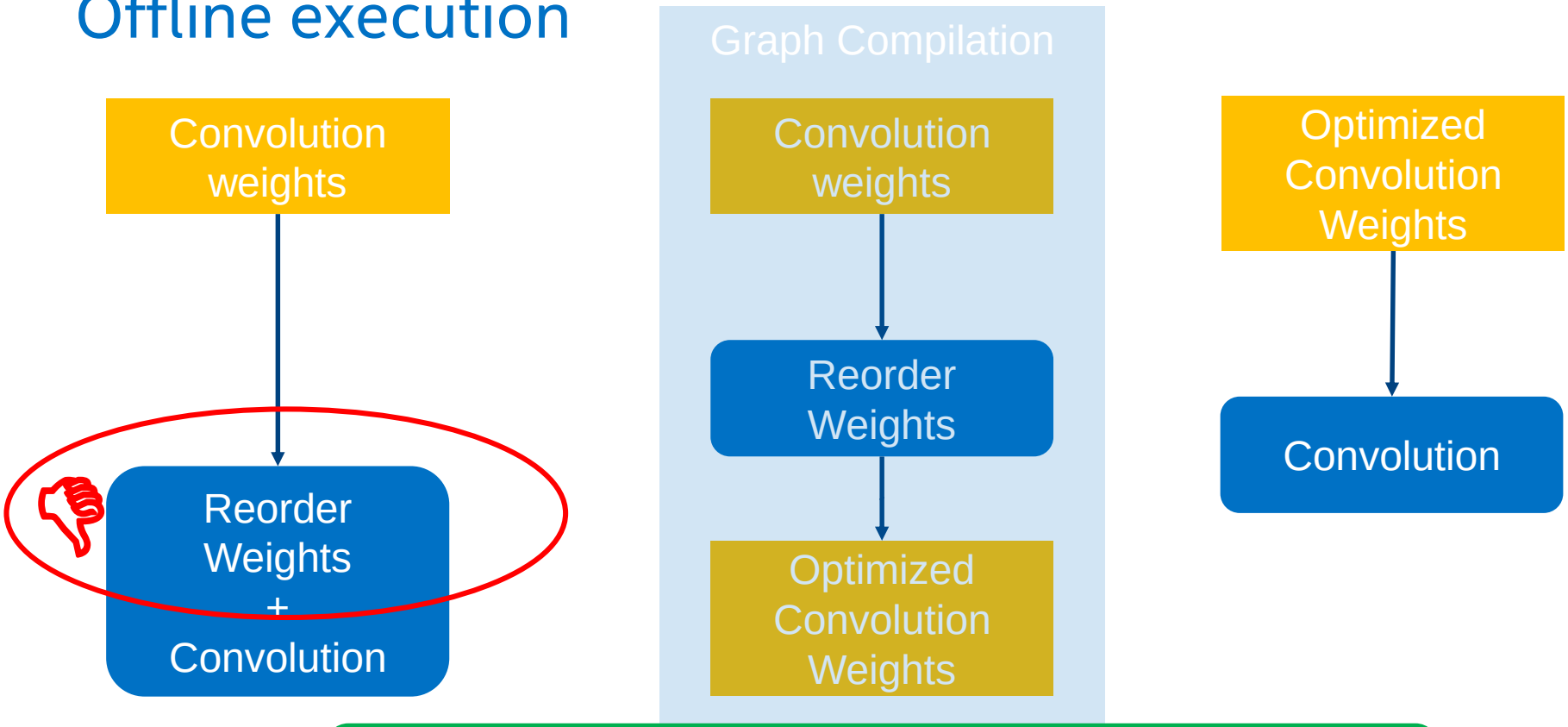
Convolution1'
Output in yxfb

Data(yxfb)

Convolution2'
Input in yxfb

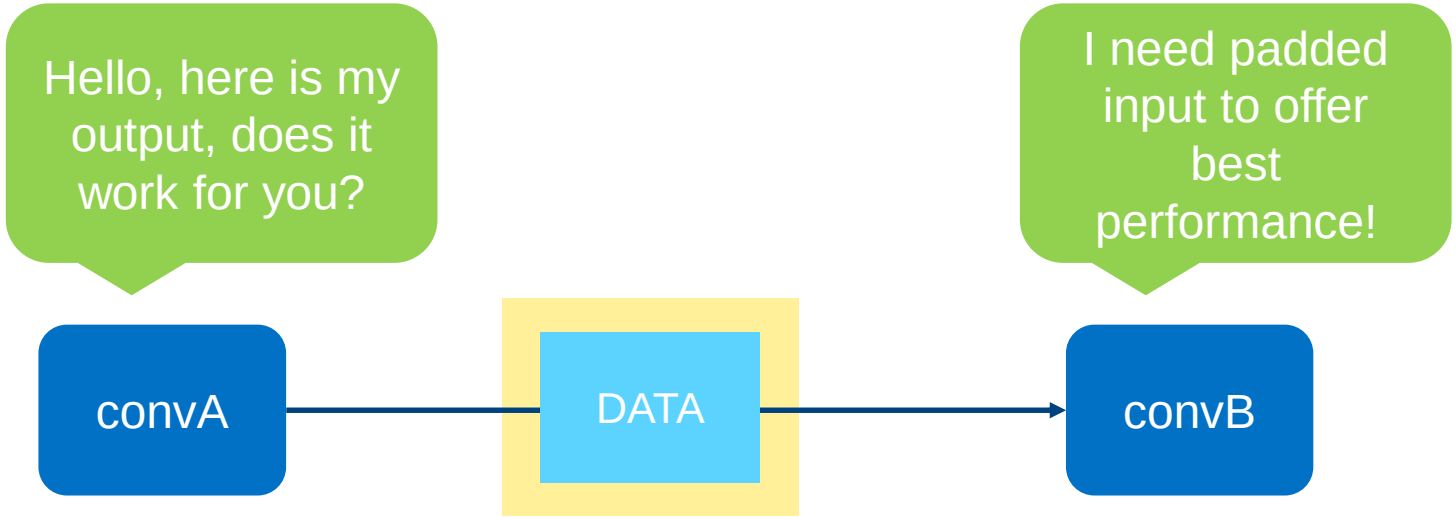


Offline execution



Expensive reorder done only once at graph compilation stage.
Optimized weights reused for subsequent runs.

Padding

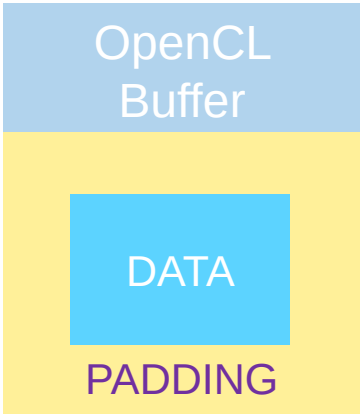

Hello, here is my output, does it work for you?
I need padded input to offer best performance!
convA
DATA
convB

Padding types - physical

How:
- Buffer created with larger size
- Data “outside” of buffer filled with zeroes

Good:
- Best performance for compute bound kernels

Bad:
- Requires management of special pool with allocations
- Increases memory footprint
- Reduces memory bandwidth



Padding types - logical

How:
- Kernel logic contains code preventing out of bounds access (returning zeroes for those accesses)

Good:
- Input allocations without any changes may be used

Bad:
- Worst performance for compute bound kernels (code contains branches which is not good for SIMD architecture)

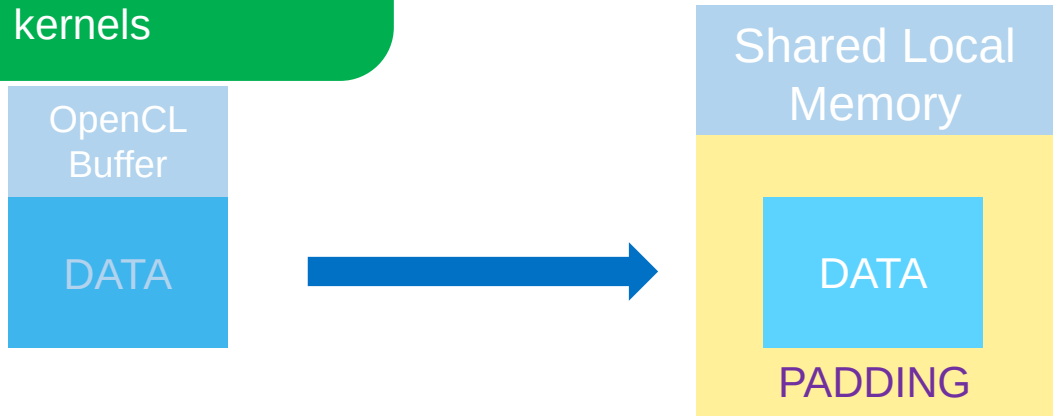
```
if ((y_offset + patch_row < 0) ||
    ((y_offset + patch_row) >= INPUT_SIZE_Y))
{
    blockA00 = { 0 };
}
else
{
    blockA00 = src0[src0_read_offset - partial_left];
```

Padding types - virtual

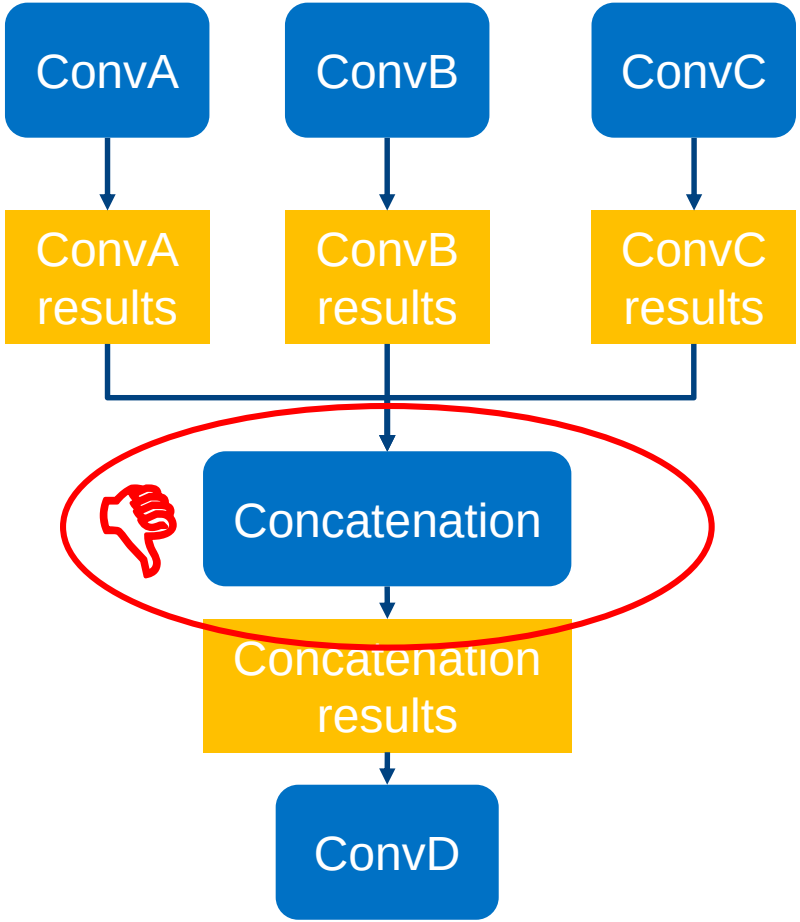
How:
- Data is downloaded to Shared Local Memory
- Data in shared local memory is surrounded with padding

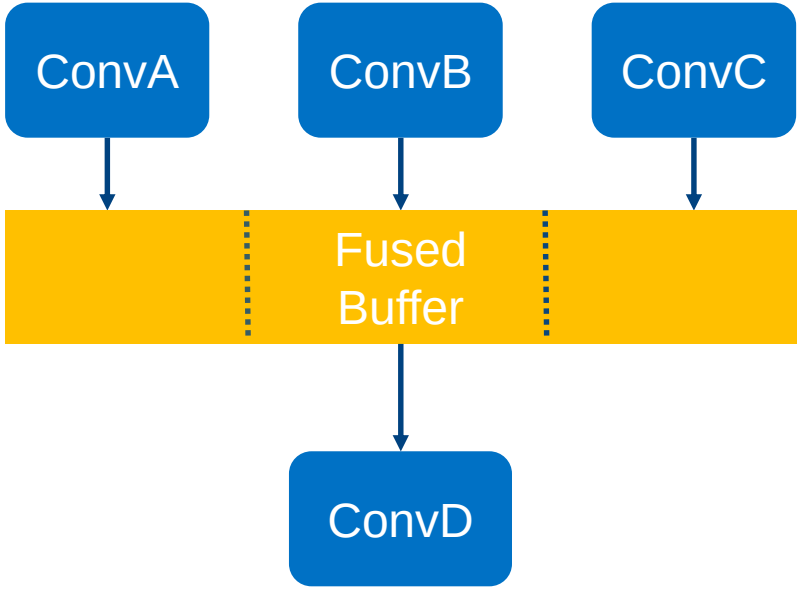
Good:
- Input allocation without any changes may be used
- Best performance for compute & memory bound kernels

Bad:
- Requires to use Shared Local Memory



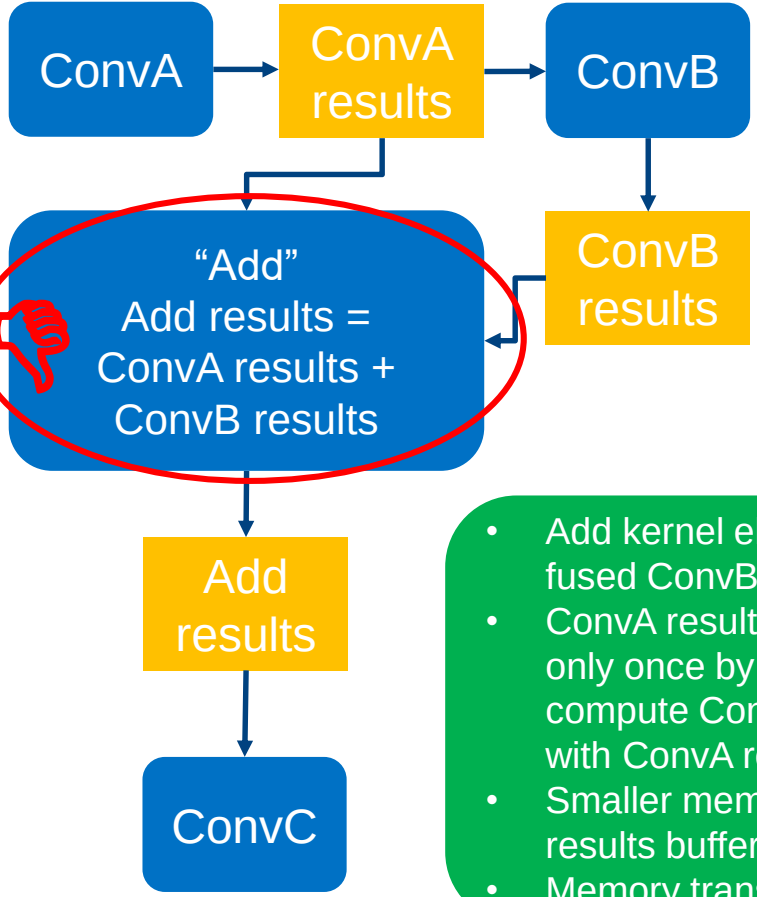
Memory Fusing



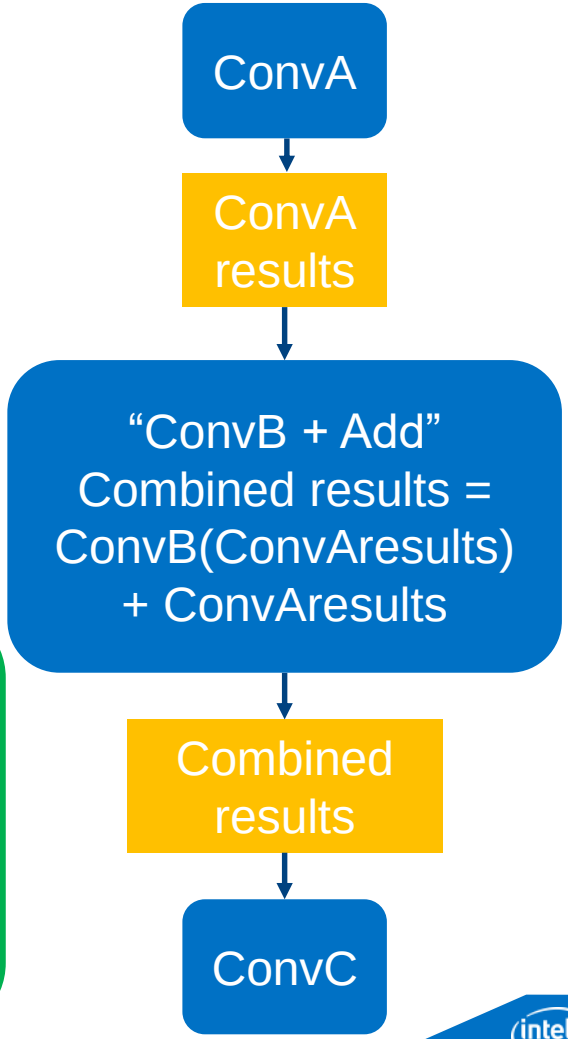


- Concatenation kernel eliminated
- Memory transfers reduced
- Smaller memory footprint (half memory needed)

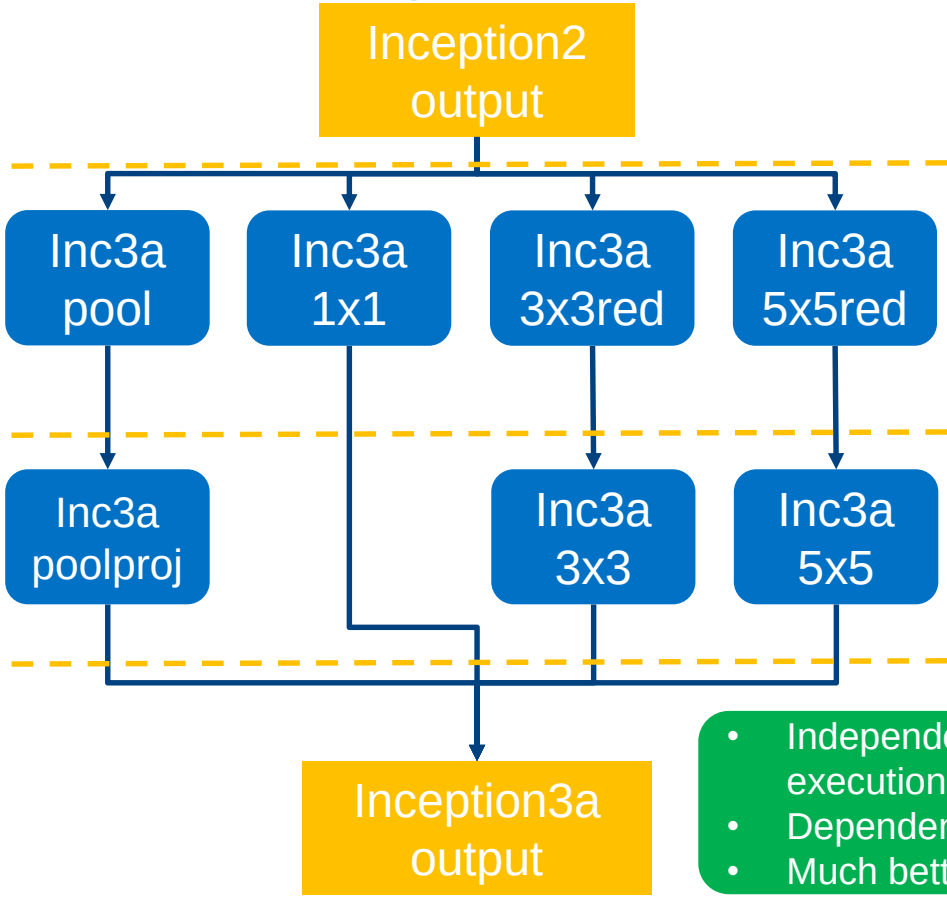
Primitive Fusing



- Add kernel eliminated (now part of fused ConvB kernel)
- ConvA results read from memory only once by fused kernel, used to compute ConvB results and sum with ConvA results
- Smaller memory footprint (no ConvB results buffer)
- Memory transfers reduced



Optimizing Execution Order

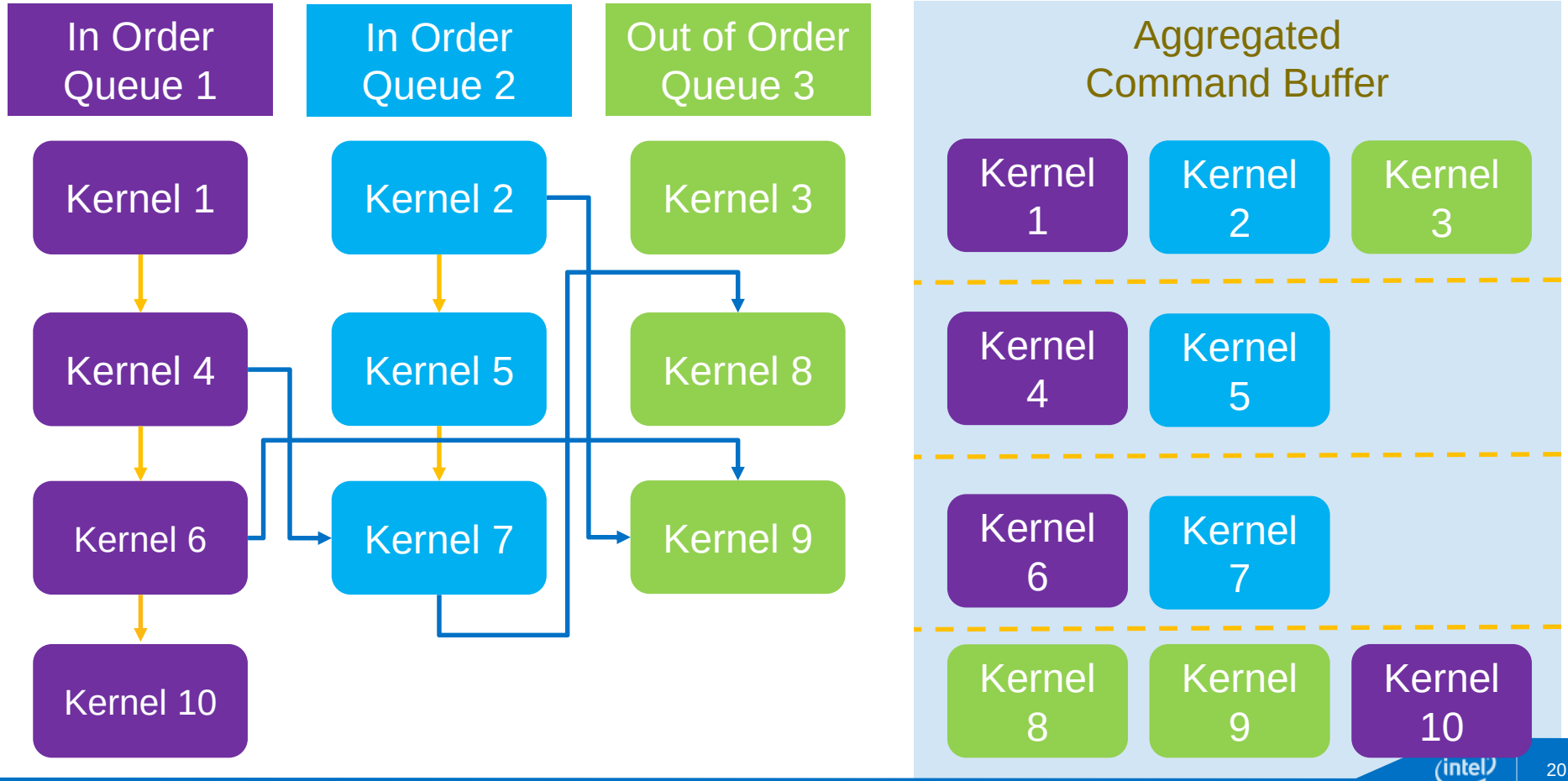


Out Of Order Queue

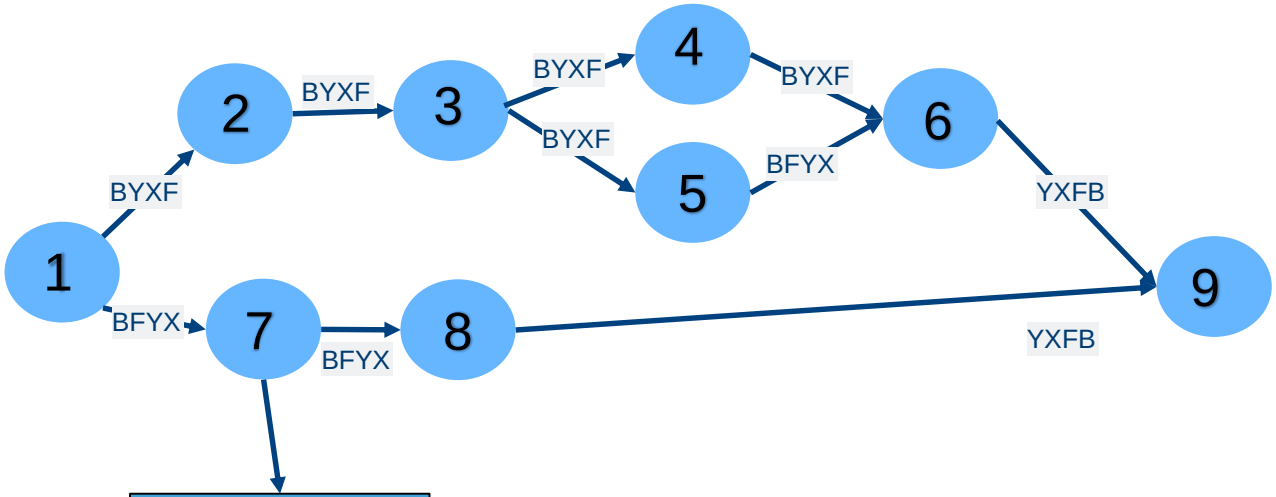
```
//no events are used in all calls below
clEnqueueBarrierWithWaitList( queue )

clEnqueueNDRangeKernel(queue, inception3a_pool )
clEnqueueNDRangeKernel(queue, inception3a_1x1 )
clEnqueueNDRangeKernel(queue, inception3a_3x3_reduce )
clEnqueueNDRangeKernel(queue, inception3a_5x5_reduce )

clEnqueueBarrierWithWaitList( queue )

clEnqueueNDRangeKernel( queue, inception3a_pool_proj )
clEnqueueNDRangeKernel( queue, inception3a_3x3 )
clEnqueueNDRangeKernel( queue, inception3a_5x5 )

clEnqueueBarrierWithWaitList( queue )
```

- Independent kernels grouped together to enable concurrent execution
- Dependencies resolved via Barrier calls
- Much better GPU utilization

Unleashing concurrency – queues with events



Kernel Selector & Auto-Tuner



raw.githubusercontent.com/intel/clDNN/master/kernel_selector/core/cache/cache.json


```
{
    "24": {
        "17988378644620438881": ["fully_connected_gpu_bf_io_input_spatial", 1],
        "1447947330145817080": ["convolution_gpu_bfyx_gemm_like", 2],
        "7822260665195993699": ["convolution_gpu_bfyx_depthwise_weights_lwg",0],
        "8834376889372261135": ["convolution_gpu_bfyx_gemm_like",2],
        "13198642774931141302": ["convolution_gpu_bfyx_gemm_like",1],
        "14147966687151087307": ["convolution_gpu_bfyx_depthwise_weights_lwg",1],
        "12416108838449201073": ["convolution_gpu_bfyx_gemm_like",2],
        "2981613830919028333": ["convolution_gpu_bfyx_gemm_like",2],
        "8083720773671701257": ["convolution_gpu_bfyx_depthwise_weights_lwg",2],
        "2088791910163600059": ["convolution_gpu_bfyx_gemm_like",2],
        "1050184225892328595 2": ["convolution_gpu_bfyx_gemm_like",2],
        "1837715130996775469 8": ["convolution_gpu_bfyx_depthwise_weights_lwg",0],
        "1146342377444615826 4": ["convolution_gpu_bfyx_os_iyx_osv16",325],
        "1907052741356343855": ["convolution_gpu_bfyx_gemm_like",2],
        "1710783679575025000 5": ["convolution_gpu_bfyx_depthwise_weights_lwg",2],
        "1239224302266630483 0": ["convolution_gpu_bfyx_os_iyx_osv16",323],
        "7210665245866922495": ["convolution_gpu_bfyx_gemm_like",1],
        "1537769288062085067 4": ["convolution_gpu_bfyx_os_iyx_osv16",1078],
        "2235284465019694961": ["convolution_gpu_bfyx_os_iyx_osv16",1072],
        "1789119171827764135 6": ["convolution_gpu_bfyx_os_iyx_osv16",1072],
        "1150656768910357913 6": ["convolution_gpu_bfyx_os_iyx_osv16",1075],
        "1356645259189040992 1": ["convolution_gpu_bfyx_os_iyx_osv16",697],
        "1098416792786227998 2": ["convolution_gpu_bfyx_gemm_like",2],
```

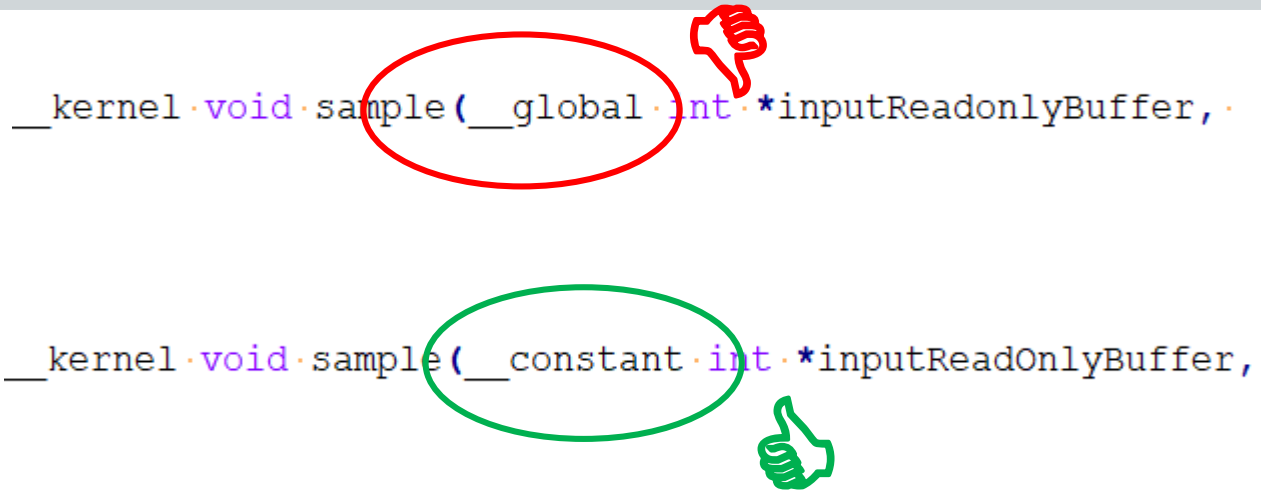

conv
Format = BFYX
Pading = 1,1
Kernel = 3x3
Input = 8,128,13,13
Output = 8,256,15,15
Split = 1;
Dilatation = 1,1,1,1
Bias = yes

Kernel Selector

impl1	impl2
Support:	Support:
Batching, padding, split, bias, all formats	Batching, padding, split, bias, yxfb format only
Require:	Require:
Batch % 8 == 0 Ofm % 32 ==0	No limitations

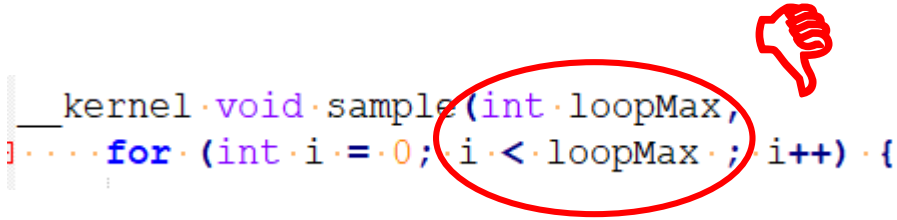
Kernel level optimizations – use __constant

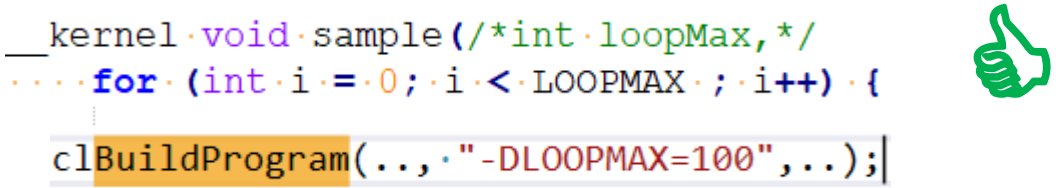
How:
If buffer is read only during kernel execution, instead of declaring it __global put it in __constant address space. This will hint the compiler to optimize reading schemes as value may not change during kernel execution so subsequent reads of the same value are not necessary.



Kernel level optimizations – pass scalars to compiler

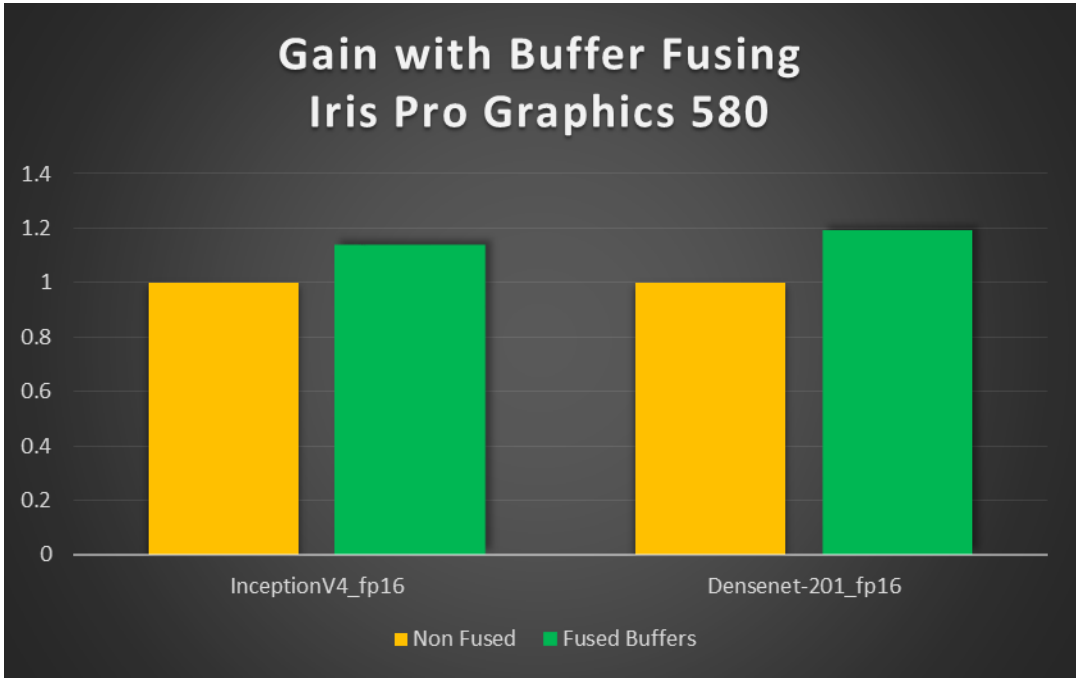
How:
Instead of passing arguments as scalars, create dedicated version of kernel that has this argument value present in compile time. This way compiler can easier apply many optimizations (i.e. loop unrolling)

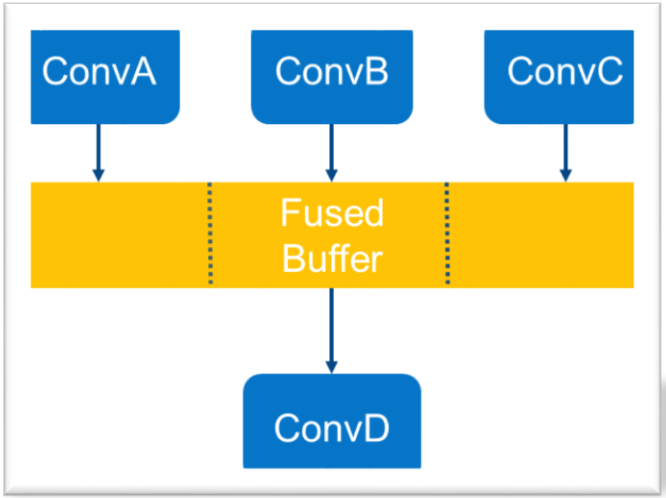




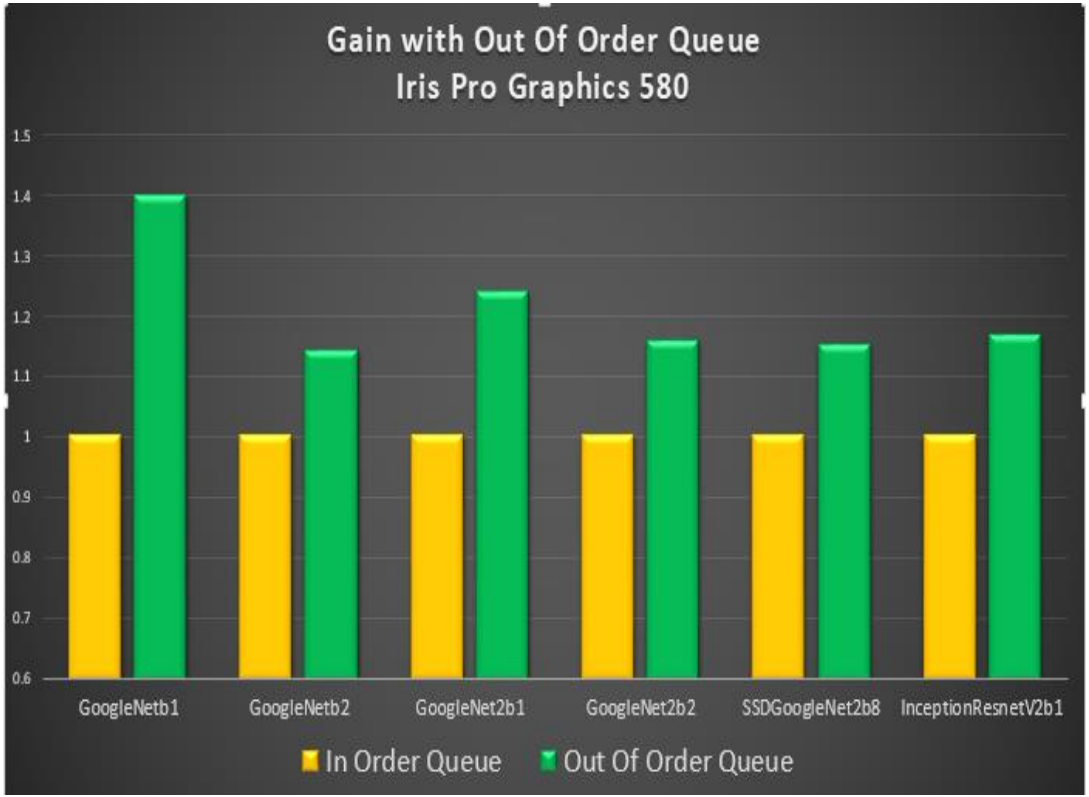
Performance Results

Gain with buffer fusing





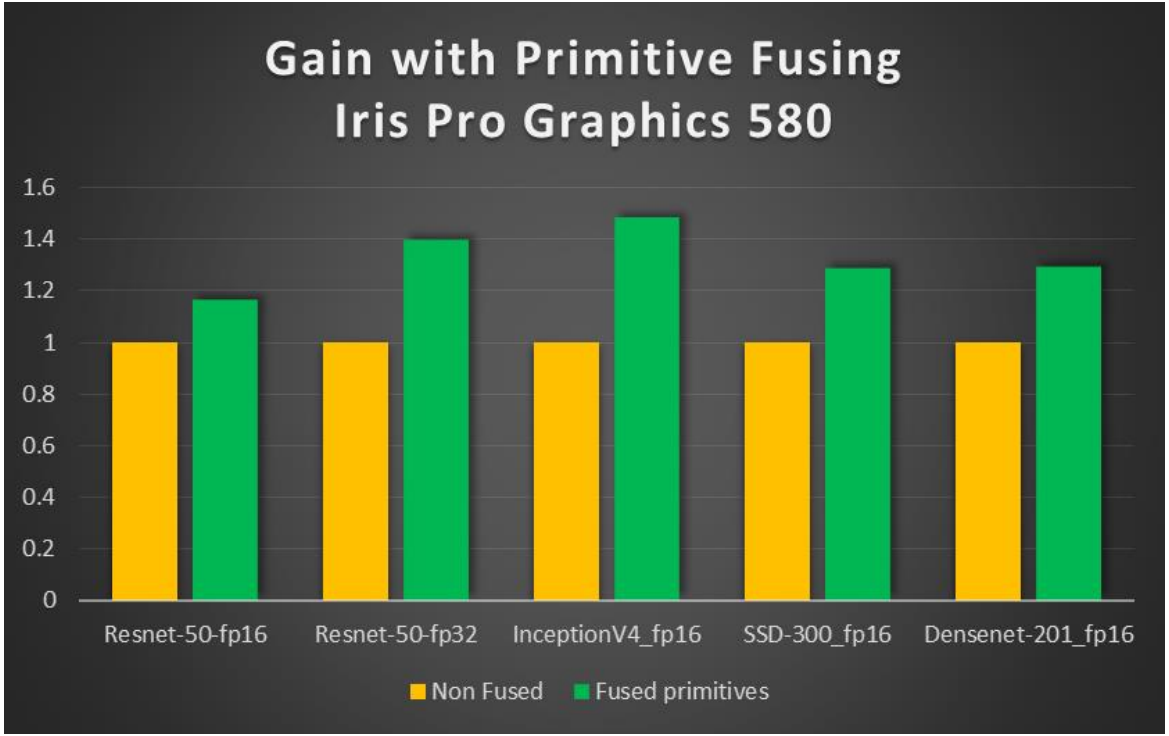
Gain with optimizing execution order

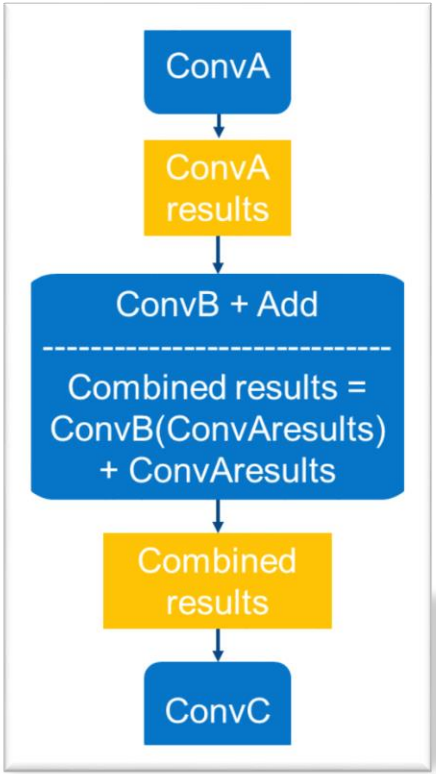


Out Of Order Queue

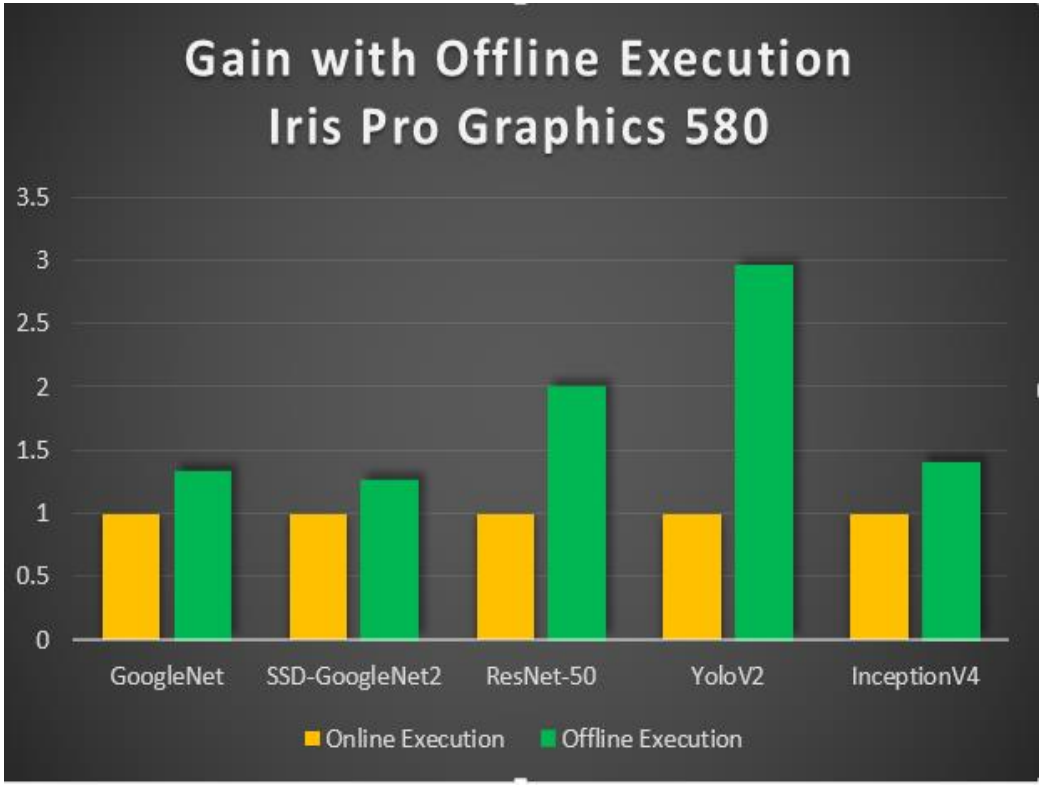
```
//no events are used in all calls below
clEnqueueBarrierWithWaitList( queue )

clEnqueueNDRangeKernel(queue, inception3a_pool )
clEnqueueNDRangeKernel(queue, inception3a_1x1 )
clEnqueueNDRangeKernel(queue, inception3a_3x3_reduce )
clEnqueueNDRangeKernel(queue, inception3a_5x5_reduce )

clEnqueueBarrierWithWaitList( queue )

clEnqueueNDRangeKernel( queue, inception3a_pool_proj )
clEnqueueNDRangeKernel( queue, inception3a_3x3 )
clEnqueueNDRangeKernel( queue, inception3a_5x5 )

clEnqueueBarrierWithWaitList( queue )
```

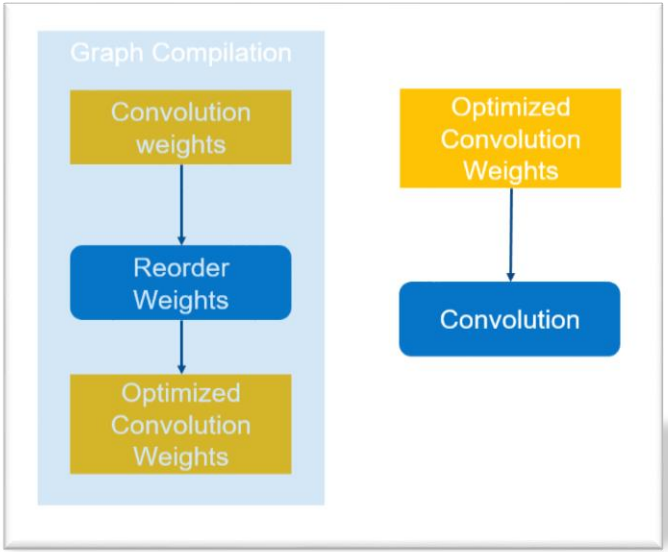
Primitive Fusing





Offline Execution





Summary and Call to Action

OpenCL is great to build neural network libraries !

To the Khronos OpenCL Working Group:

- All those optimizations doesn't require any vendor extensions!

Try our compute libraries and give us feedback!

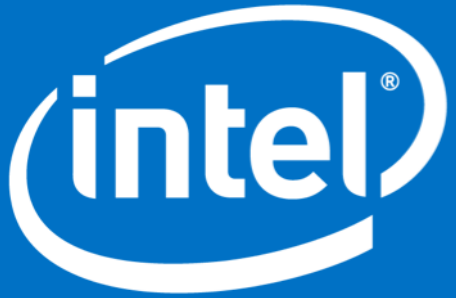
- Check out how we implemented those optimizations in clDNN library
  https://github.com/opencv/dldt/tree/2019/inference-engine/thirdparty/clDNN
- Check out how our OpenCL driver supports those optimizations
  https://github.com/intel/compute-runtime
- Send Issues and Pull Requests

To OpenCL Developers:

- Try those techniques and optimize your kernels!

Thank You!

Much thanks to Tomasz Poniecki and Ben Ashbaugh for help with material preparation, guidance and detailed review.


intel