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.

Intel, the Intel logo and others are trademarks of Intel Corporation in the U.S. and/or other countries. *Other names and brands may be claimed as the property of others.

© 2019 Intel Corporation.

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.

Copyright © 2019, Intel Corporation. All rights reserved. Intel, Pentium, Xeon, Xeon Phi, Core, VTune, Cilk, and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

### 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!
ConvolutionA
Output in byxf

Data(byxf)

Change Layout byxf->yxfb

Data2(yxfb)

ConvolutionB
Input in yxfb
Layout selection

Convolution1
Output in byxf

Convolution1’
Output in yxfb

Convolution2
Input in byxf

Convolution2’
Input in yxfb

Data(byxf)

Data(yxfb)

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

• Layout changing kernel eliminated
• Memory footprint reduced

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

• Layout changing kernel eliminated
• Memory footprint reduced
Offline execution

Graph Compilation

Convolution weights

Reorder Weights

Optimized Convolution Weights

Convolution

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!
Padding types - physical

Good:
• Best performance for compute bound kernels

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

Bad:
• Requires management of special pool with allocations
• Increases memory footprint
• Reduces memory bandwidth
Padding types - logical

**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)

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

```c
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

ConvA → ConvA results → ConvB

“Add”
Add results = ConvA results + ConvB results

Add results → ConvC

- 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

“ConvB + Add”
Combined results = ConvB(ConvA results) + ConvA results

ConvA results → ConvB results → “ConvB + Add”
Combined results → ConvC
Inception2 output

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

Out Of Order Queue

```c
// no events are used in all calls below
c1EnqueueBarrierWithWaitList( queue )
c1EnqueueNDRangeKernel( queue, inception3a_pool  )
c1EnqueueNDRangeKernel( queue, inception3a_1x1  )
c1EnqueueNDRangeKernel( queue, inception3a_3x3_reduce )
c1EnqueueNDRangeKernel( queue, inception3a_5x5_reduce )
c1EnqueueBarrierWithWaitList( queue )
c1EnqueueNDRangeKernel( queue, inception3a_pool_proj  )
c1EnqueueNDRangeKernel( queue, inception3a_3x3  )
c1EnqueueNDRangeKernel( queue, inception3a_5x5  )
c1EnqueueBarrierWithWaitList( queue )
```
Unleashing concurrency – queues with events

In Order Queue 1
Kernel 1
Kernel 4
Kernel 6
Kernel 10

In Order Queue 2
Kernel 2
Kernel 5
Kernel 7

Out of Order Queue 3
Kernel 3
Kernel 8
Kernel 9

Aggregated Command Buffer
Kernel 1
Kernel 2
Kernel 3
Kernel 4
Kernel 5
Kernel 6
Kernel 7
Kernel 8
Kernel 9
Kernel 10
Kernel Selector & Auto-Tuner

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

**impl1**
- Support:
  - Batching, padding, split, bias, all formats
- Require:
  - Batch % 8 == 0
  - Ofm % 32 == 0

**impl2**
- Support:
  - Batching, padding, split, bias, yxfb format only
- Require:
  - 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.

```c
__kernel void sample(__global int *inputReadonlyBuffer,
```

```c
__kernel void sample(__constant int *inputReadOnlyBuffer,
```
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)

```c
__kernel void sample(int loopMax,
};

for (int i = 0; i < loopMax; i++) {

__kernel void sample(/*int loopMax,* /
};

for (int i = 0; i < LOOPMAX; i++) {

clBuildProgram(..., "-DLOOPMAX=100", ...);
```
Performance Results
Gain with buffer fusing
Gain with optimizing execution order

Gain with Out Of Order Queue
Iris Pro Graphics 580

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)
Gain with Primitive Fusing
Iris Pro Graphics 580

- Resnet-50-fp16
- Resnet-50-fp32
- InceptionV4_fp16
- SSD-300_fp16
- Densenet-201_fp16

- Non Fused
- Fused primitives
Offline Execution

Gain with Offline Execution
Iris Pro Graphics 580

Graph Compilation
- Convolution weights
- Reorder Weights
- Optimized Convolution Weights
  - Convolution

Online Execution
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

▪ 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.