# INTEL® MATH KERNEL LIBRARY FOR DEEP NEURAL NETWORKS (INTEL MKL-DNN)

ALCF SDL workshop, October 4th 2018

The Intel MKL-DNN team
Presenter: Mourad Gouicem

# Deep Learning Software Stack for Intel processors


**Deep learning and AI ecosystem** includes edge and datacenter applications.

- Open source frameworks (Tensorflow*, MXNet*, CNTK*, PaddlePaddle*)
- Intel deep learning products (Neon™ framework , BigDL, OpenVINO™ toolkit)
- In-house user applications

Intel MKL and Intel MKL-DNN optimize deep learning applications for Intel processors :

- through the collaboration with framework maintainers to upstream changes (Tensorflow*, MXNet*, PaddlePaddle*, CNTK*)
- through Intel optimized forks (Caffe*, Torch*, Theano*)
- by partnering to enable proprietary solutions

**Intel MKL-DNN** is an open source performance library for deep learning applications (available at https://github.com/intel/mkl-dnn)

- Fast open source implementations for wide range of DNN functions
- Early access to new and experimental functionality
- Open for community contributions

**Intel MKL** is a proprietary performance library for wide range of math and science applications

Distribution: Intel Registration Center, package repositories (apt, yum, conda, pip)

# Examples of speedups on Intel® Xeon® Scalable Processors

## INTEL-OPTIMIZED TENSORFLOW PERFORMANCE AT A GLANCE

**TRAINING THROUGHPUT**

14X

Intel-optimized TensorFlow ResNet50 training performance compared to default TensorFlow for CPU

**INFERENCE THROUGHPUT**

3.2X

Intel-optimized TensorFlow InceptionV3 inference throughput compared to Default TensorFlow for CPU

**System configuration**:
**CPU Thread(s) per core**: 2 **Core(s) per socket**: 28
**Socket(s)**: 2 **NUMA node(s)**: 2 **CPU family**: 6
**Model**: 85 **Model name**: Intel(R) Xeon(R) Platinum 8180 CPU @ 2.50GHz Stepping: 4
**HyperThreading**: ON **Turbo**: ON **Memory** 376GB (12 x 32GB) 24 slots, 12 occupied 2666 MHz Disks Intel RS3WC080 x 3 (800GB, 1.6TB, 6TB) **BIOS** SE5C620.86B.00.01.0004.071220170215 **OS** Centos

Inference and training throughput uses FP32 instructions

Unoptimized TensorFlow may not exploit the best performance from Intel CPUs.

| Model |
|---|
| VGG16 |
| InceptionV3 |
| ResNet50 |

*Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYS components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and purchases, including the performance of that product when combined with other products. For more complete information visit http://www.intel.com/performa*

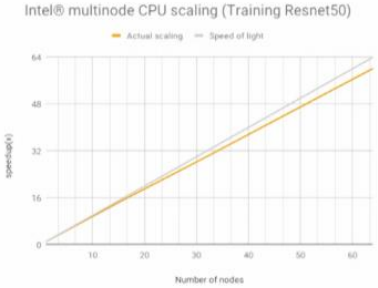
## PERFORMANCE GAINS REPORTED BY OTHERS

Intel TensorFlow Scalability Results Presented by Google @TF Summit March 30, '18

TensorFlow with Intel® MKL-DNN integration

**3x inference speedup** on Broadwell and Skylake

94% efficiency when training with 64 nodes cluster




"By making use of [Intel's] open source library [MKL-DNN], we were able to achieve a 3x performance benefit and great scaling efficiency on training. This is an example of how important it is to have strong collaborations with companies like Intel."

**Matt Wood** @mza

Follow

New optimized TensorFlow build for EC2 C5 instances (7.4x training performance improvement over stock TF 1.6) - now available on the #AWS Deep Learning AMI, Ubuntu, and Amazon Linux:


**Faster training with optimized TensorFlow 1.6 on Amazon EC2 C5 and P3 inst…**
The AWS Deep Learning AMIs come with latest pip packages of popular deep learning frameworks pre-installed in separate virtual environments so that develo…
aws.amazon.com

AIDC INTEL AI DEVCON 2018

Source: TENSORFLOW OPTIMIZED FOR INTEL® XEON™

# TensorFlow with Intel MKL/MKL-DNN

## Use Intel Distribution for Python*

- Uses Intel MKL for many NumPy operations thus supports MKL_VERBOSE=1
- Available via Conda, or YUM and APT package managers

## Use pre-built Tensorflow* wheels or build TensorFlow* with `bazel build --config=mkl`

- **Building from source required for integration with Intel Vtune™ Amplifier**
- Follow the CPU optimization advices including setting affinity and # of intra- and inter- ops threads
- More Intel MKL-DNN-related optimizations are slated for the next version: Use the latest TensorFlow* master if possible

# Intel distribution of Caffe

A fork of BVLC Caffe* maintained by Intel

The best-performing CPU framework for CNNs

Supports low-precision inference on Intel Xeon Scalable Processors (formerly known as Skylake)

# Intel MKL-DNN overview

**Features:**

- Training (float32) and inference (float32, int8)
- CNNs (1D, 2D and 3D), RNNs (plain, LSTM, GRU)
- Optimized for Intel processors

**Portability:**

- Compilers: Intel C++ compiler/Clang/GCC/MSVC*
- OSes: Linux*, Windows*, Mac*
- Threading: OpenMP*, TBB

**Frameworks that use Intel MKL-DNN:**

IntelCaffe, TensorFlow*, MxNet*, PaddlePaddle*

CNTK*, OpenVino, DeepBench*

| Primitives | Class |
|---|---|
| • (De-)Convolution<br>• Inner Product<br>• Vanilla RNN, LSTM, GRU | Compute intensive operations |
| • Pooling AVG/MAX<br>• Batch Normalization<br>• Local Response Normalization<br>• Activations (ReLU, Tanh, Softmax, ...)<br>• Sum | Memory bandwidth limited operations |
| • Reorder<br>• Concatenation | Data movement |

# KEY PERFORMANCE CONSIDERATIONS ON INTEL PROCESSORS

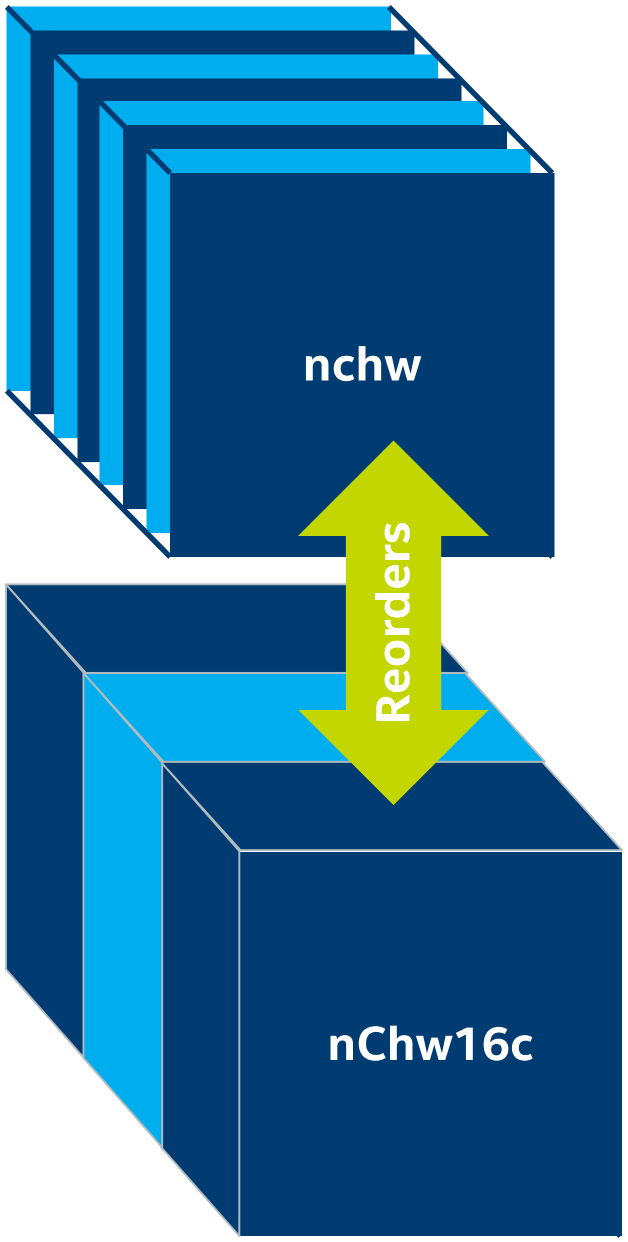
# Memory layouts

## Most popular memory layouts for image recognition are **nhwc** and **nchw**

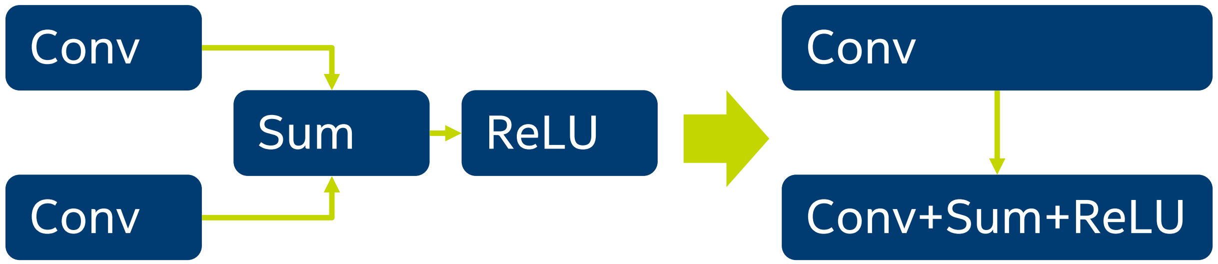
- Challenging for Intel processors either for vectorization or for memory accesses (cache thrashing)

## Intel MKL-DNN convolutions use blocked layouts

- Example: **nhwc** with channels blocked by 16 – **nChw16c**
- Convolutions define which layouts are to be used by other primitives
- Optimized frameworks track memory layouts and perform reorders **only** when necessary



# Fusing computations




## On Intel processors a high % of time is typically spent in BW-limited ops

- ~40% of ResNet-50, even higher for inference

## The solution is to fuse BW-limited ops with convolutions or one with another to reduce the # of memory accesses

- Conv+ReLU+Sum, BatchNorm+ReLU, etc
- Done for inference, WIP for training

## The FWKs are expected to be able to detect fusion opportunities

- IntelCaffe already supports this

## Major impact on implementation

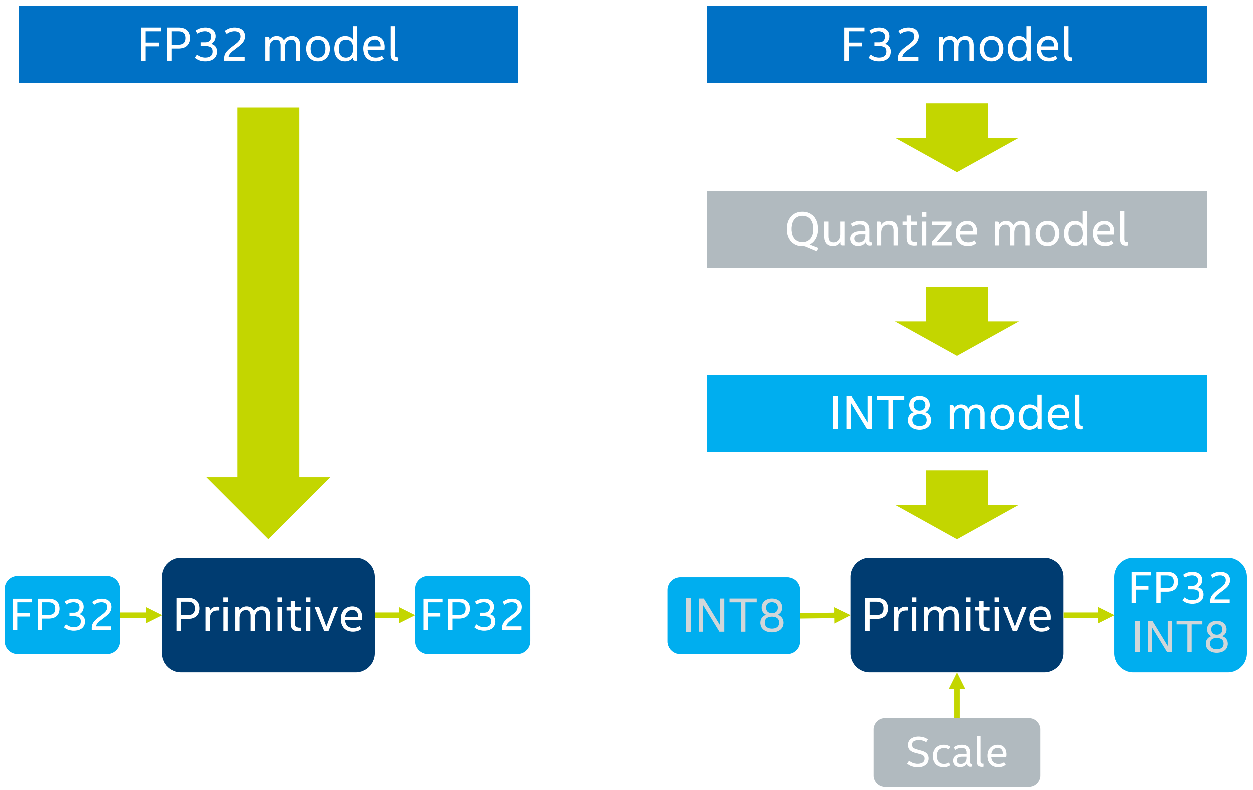
- All the impls. must be made aware of the fusion to get max performance
- Intel MKL-DNN team is looking for scalable solutions to this problem

# Low-precision inference

Proven only for certain CNNs by IntelCaffe at the moment

A trained float32 model quantized to int8

Some operations still run in float32 to preserve accuracy

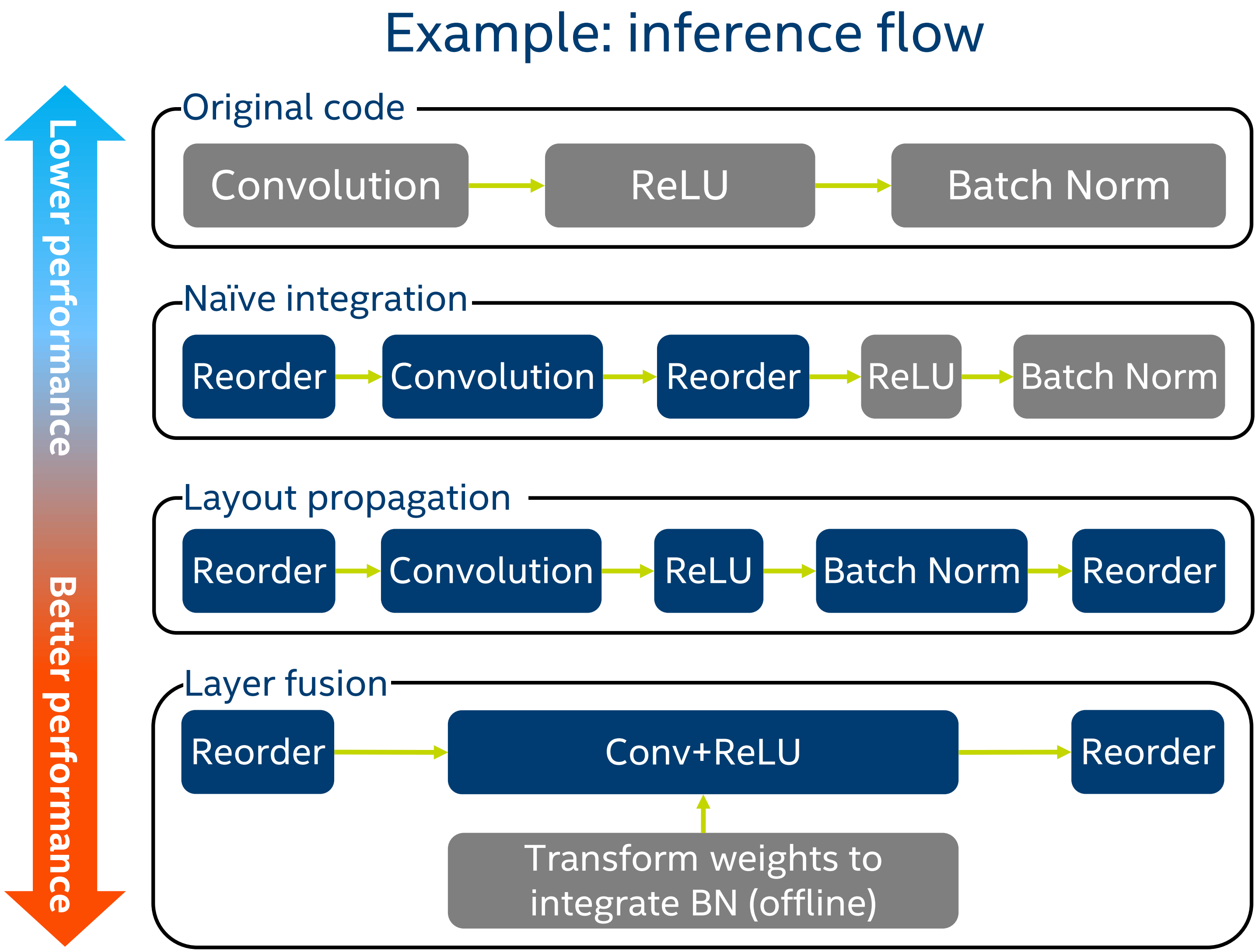


# Intel MKL-DNN integration levels

Intel MKL-DNN is designed for best performance.

However, topology level performance will depend on Intel MKL-DNN integration.

- Naïve integration will have reorder overheads.
- Better integration will propagate layouts to reduce reorders.
- Best integration will fuse memory bound layers with compute intensive ones or with each other.



# INTEL MKL-DNN LIBRARY PHILOSOPHY

# Intel MKL-DNN concepts

**Descriptor:** a structure describing memory and computation properties

**Primitive**: a handle to a particular compute operation

- Examples: Convolution, ReLU, Batch Normalization, etc.
- Three key operations on primitives: **create**, **execute** and **destroy**
- Separate **create** and **destroy** steps help amortize setup costs (memory allocation, code generation, etc.) across multiple calls to **execute**

**Memory:** a handle to data

**Stream:** a handle to an execution context

**Engine:** a handle to an execution device

# Layout propagation: the steps to create a primitive

1. Create memory descriptors
   - These describe the shapes and memory layouts of the tensors the primitive will compute on
   - Use the **layout ‘any’** as much as possible for every input/output/weights if supported (e.g. convolution or RNN). Otherwise, use the **same layout as the previous layer output**.
2. Create primitive descriptor and primitive
3. Create needed input reorders
   - Query the primitive for the input/output/weight layout it expects
   - Create the needed memory buffers and reorder primitives to accordingly reorder the data to the appropriate layout
4. Enqueue primitives and reorders in the stream queue for execution

# Primitive attributes

## Fusing layers through post-ops

1. Create a post_ops structure
2. Append the layers to the post-ops structure (currently supports sum and elementwise operations)
3. Pass the post-op structure to the primitive descriptor creation through attributes

## Quantized models support through attributes (more details)

1. Set the scaling factors and rounding mode in an attribute structure
2. Pass the attribute structure to the primitive descriptor creation

# KEY TAKEAWAYS

# Key Takeaways

1. Application developers already benefit of Intel MKL-DNN through integration in popular frameworks
2. Framework developers can get better performance on Intel processors by integrating Intel MKL-DNN
3. There are different levels of integration, and depending on the level you will get different performance
4. Profiling can help you identify performance gaps due to
   - Integration not fully enabling Intel MKL-DNN potential (more on that in the hands-on session).
   - Performance sensitive function not enabled with Intel MKL-DNN (make requests on Github*)
   - Performance issue in Intel MKL-DNN (raise the issue on Github*)

intel®

# Legal Disclaimer & Optimization Notice

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. For more complete information visit www.intel.com/benchmarks.



**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

# BACKUP

# Primitives and their implementations

| Operation | Implementations | | | | | | |
|---|---|---|---|---|---|---|---|
| **Convolutions fp32** | JIT Winograd (AVX512 SKX/KNL only) | ***Separate implementations for SSE4.2, AVX2 and AVX512F+*** | | | | GEMM (Intel MKL: all ISA, JIT: AVX512F+ only) | Reference |
| | | 1x1 JIT | non-1x1 JIT FWD | non-1x1 JIT BWD_D | non-1x1 JIT BWD_W | | |
| **Convolutions int8** | JIT (AVX512BW) | Intel MKL GEMM (WIP) | Reference | | | | |
| **InnerProduct fp32** | JIT (AVX512F+ only) | Intel MKL GEMM | Reference | | | | |
| **BatchNorm fp32** | JIT (any ISA) | Reference | | | | | |
| **LRN fp32** | JIT (any ISA) | Reference | | | | | |
| **Pooling fp32 / int8** | JIT (any ISA) | JIT (nchw, any ISA) | Reference | | | | |
| **Elementwise** | JIT (any ISA) | | | | | | |
| **Reorders** | JIT (AVX2 | Reference | | | | | |

## Multiple conv impls. to support diff. features and have diff. perf.

- Conv 1x1 – special vectorization and blocking
- Conv non-1x1 – better support for 3x3, 5x5, etc
- GEMM – support for dilation (hard to implement in direct JIT)
- Winograd is only for 3x3; only the (special) GEMM part is JIT-ed

# PROFILING

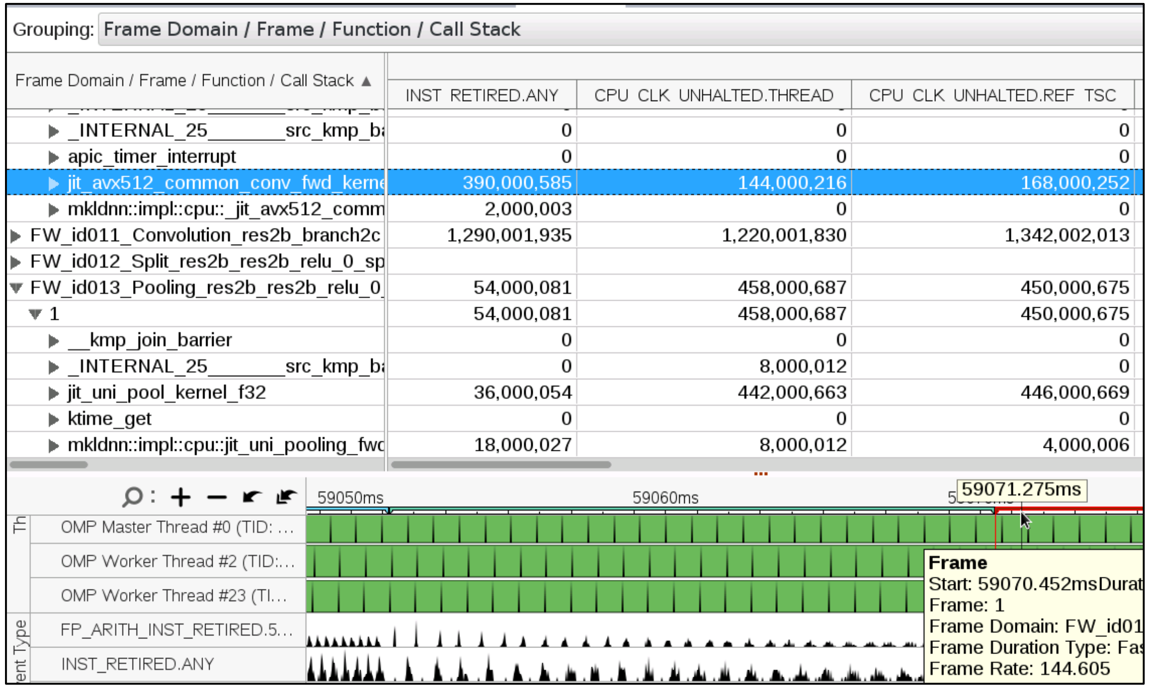
# Integration with Intel VTune Amplifier

## Full application analysis

## Report types:

- CPU utilization
- Parallelization efficiency
- Memory traffic

## Profiling of run-time generated code must be enabled at compile time




```
$ # building Intel MKL-DNN using cmake
$ cmake -DVTUNEROOT=/opt/intel/vtune_amplifier_2018 .. && make install

$ # an alternative: building Intel MKL-DNN using sources directly, e.g. in TensorFlow
$ CFLAGS="-I$VTUNEROOT/include -DJIT_PROFILING_VTUNE" LDFLAGS="-L$VTUNEROOT/lib64 -ljitprofiling" bazel build
```

# Intel MKL-DNN verbose mode overview

## Simple yet powerful analysis tool:

- Similar to [Intel MKL verbose]()
- Enabled via environment variable or function call
- Output is in CSV format

## Output includes:

- The marker, state and primitive kind
- Implementation details (e.g. jit:avx2)
- Primitive parameters
- Creation or execution time (in ms)

Example below (details [here]())

```
$ # MKLDNN_VERBOSE is unset
$ ./examples/simple-net-c
passed

$ export MKLDNN_VERBOSE=1 # report only execution parameters and runtime
$ ./examples/simple-net-c # | grep "mkldnn_verbose"
mkldnn_verbose,exec,reorder,jit:uni,undef,in:f32_oihw out:f32_Ohwi8o,num:1,96x3x11x11,12.2249
mkldnn_verbose,exec,eltwise,jit:avx2,forward_training,fdata:nChw8c,alg:eltwise_relu,mb8ic96ih55iw55,0.437988
mkldnn_verbose,exec,lrn,jit:avx2,forward_training,fdata:nChw8c,alg:lrn_across_channels,mb8ic96ih55iw55,1.70093
mkldnn_verbose,exec,reorder,jit:uni,undef,in:f32_nChw8c out:f32_nchw,num:1,8x96x27x27,0.924805
passed
```

# Performance gaps causes

**Functional gaps:** your hotspot is a commonly/widely used primitive and is not enabled in Intel MKL-DNN

**Integration gaps:** your hotspot uses Intel MKL-DNN but runs much faster in a standalone benchmark (more details in the hands-on session)

**Intel MKL-DNN performance issue:** your hotspot uses Intel MKL-DNN but is very slow given its parameters

In any of these cases, feel free to contact the Intel MKL-DNN team through the Github* page issues section.