INTEL AND ML

Hans Pabst
Technical Computing Enabling, Switzerland
Developer Relations Division
Software and Services Group

March 20th 2017

Agenda

- Hardware and Software Innovation
    - JIT Code Specialization
    - ISA Extensions
- Open Source Software
    - Low-Level Primitives and Frameworks
    - Intel DAAL, MKL-DNN
    - LIBXSMM

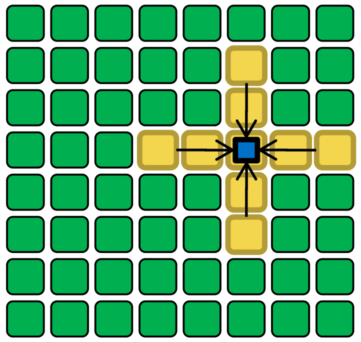
Hardware and Software Innovation

JIT Code Specialization

Do you remember Graphics processors turning into GPGPU? In the early days, people used to “concatenate strings” to build GPGPU programs (shaders).

- Implied embedding variables as constants, etc.
- Example: Advantages for Stencil Computation
  - ✓ Stencil (access-)pattern “baked” into code
  - ✓ Grid bounds are known constants
  - ✓ Code unrolling without remainder

  […]



Code specialization is an effective optimization!

- Small-size problems can be effectively hard-coded; no upfront code generation needed (may also avoid control-flow to select cases)
- Perfect match for unpredictable use cases (scripting framework, etc.)

Convolutional Neural Networks (CNNs) and JIT

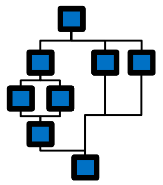
Direct convolutions are more efficient than a calculation in the frequency space

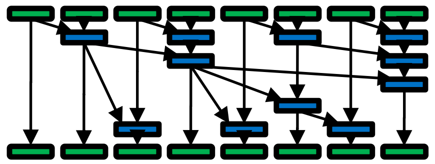
(There is a cross-over point when it comes to larger convolutions)

- Exploiting the memory bandwidth, caches, and register file is key
- ML frameworks are usually script-enabled (runtime dynamic)
- Network topology (graph structure) determines schedule, code fusion, etc.

JIT code generation: ideal for small convolutions and graph-based structure

- Can further exploit topology





→ Significant advantage with JIT code generation

Hardware Specialization

Innovation happens to large extent at ISA level (true for CPU and GPU)

- Domain specific ISA extensions (supported only at intrinsic level)
  Examples: CRC32 (SSE 4.2), AES-NI, SHA, F16 LD/ST (F16C)
- General compiler support (regularly generated from high-level code)
  Examples: AVX, AVX2, AVX-512

ML domain for Intel Architecture (IA) is enabled at ISA <u>and</u> platform level

- Complemented by segment-specific HW e.g., KNM (KNL derivative)
- Platform level e.g., Xeon sockets shared with FPGA
- ISA level e.g., FMA, QFMA, VNNI, QVNNI

Open Source Software

Open Source Software at Intel

Long-term commitment to Open Source Software (OSS)

- Linux OS (not just drivers): Intel contributes and ranks in top-10 since years
- Other contributions: compiler enabling (GCC), math library support (SVML), language standards (OpenMP: iOMP → Clang), in general: 01.org

… but Intel software product teams are also commited

- Free-of-charge Intel MKL (supported by user forum), which is also for commercial use (Premier support remains commercial)
- Open Source Software: Intel TBB, **Intel MKL-DNN**

→ Domain specific HW/SW strongly asks for "OSS backup"

Avoids "vendor lock", protects effort on customer's side, etc.

Contribute and participate without replicating status quo

Common Architecture for Machine & Deep Learning

INTEL® XEON® PROCESSOR + LAKE CREST

Best in class neural network training performance

INTEL® XEON® PROCESSORS

Most widely deployed machine learning platform

INTEL® XEON PHI™ PROCESSORS

Higher performance, general purpose machine learning

INTEL® XEON® PROCESSOR + FPGA

Higher perf/watt inference, programmable

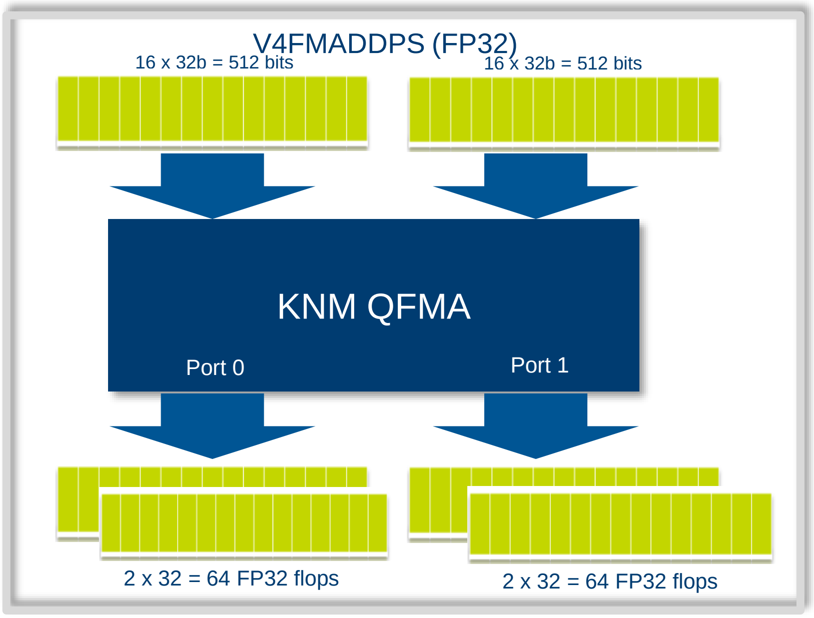
TARGETED ACCELERATION

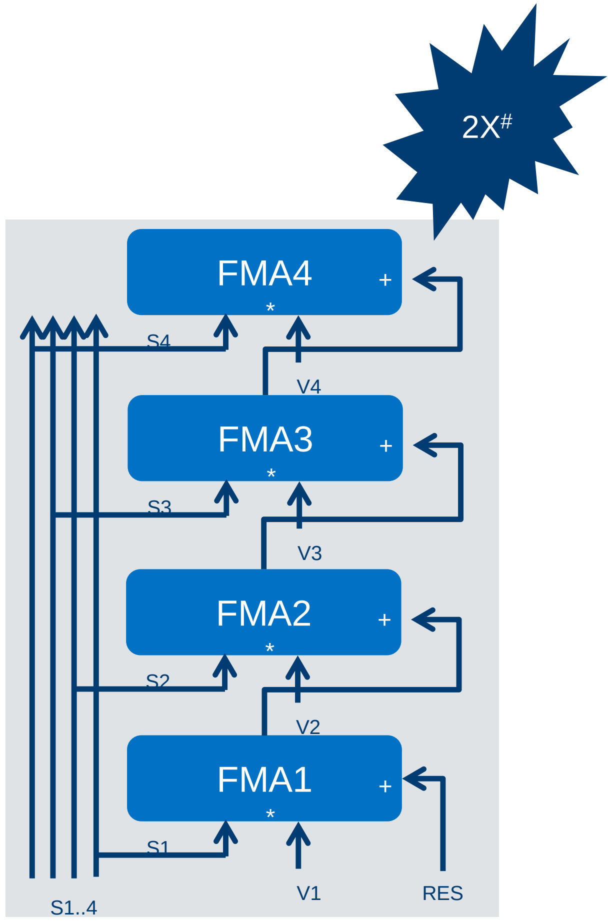
* Intel AI Days (series of worldwide events) shared more vision and roadmap information

Knights Mill (KNM): QFMA Instruction*

Enhanced ISA QFMA instructions in Knights Mill delivers:

- ✓ Higher Peak Flops for CNN, RNN, DNN, LSTM
- ✓ Higher Efficiency (One Quad FMA executed in two cycles)
- ✓ 2X FP operations per cycle





QMADD packs 4 IEEE FMA ops in a single instruction

# 2X faster per clock than KNL SP

* See Intel Architecture Reference Manual

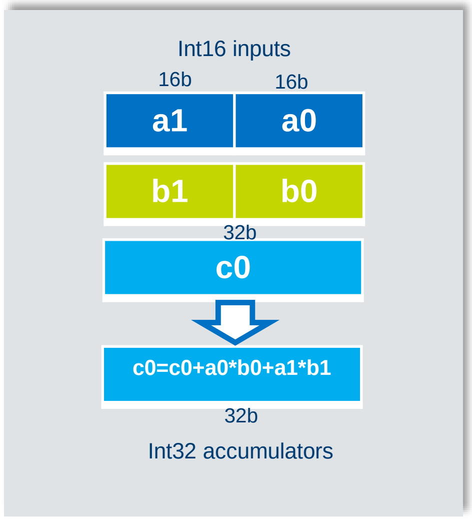
Knights Mill (KNM) Variable Precision Instructions*

Enabling higher throughput for ML training tasks

- 2x the flops by using INT16 inputs
- Similar accuracy as SP by using INT32 accumulated output

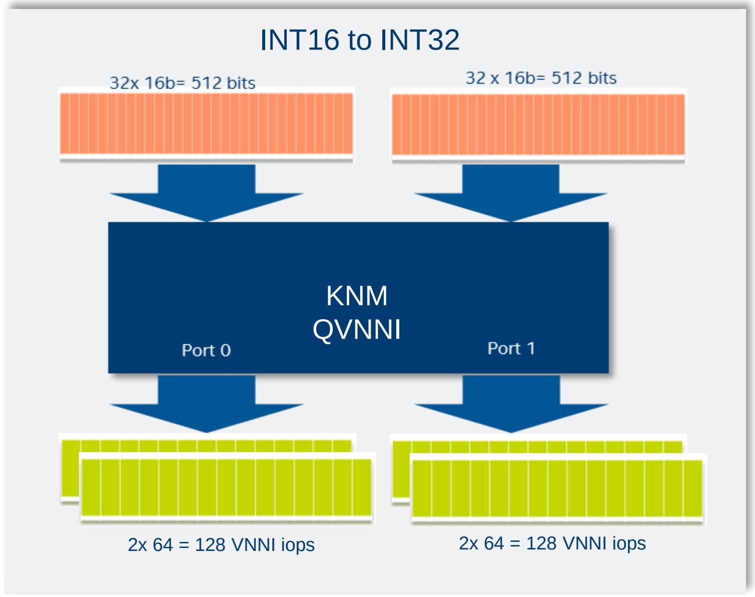
4X#

VNNI



QVNNI

- 2x VNNI operations per port
- 4x* ML performance than regular AVX512-SP



# 4x faster per clock than KNL SP

* See Intel Architecture Reference Manual

Libraries, Frameworks and Tools*

	Intel® Math Kernel Library							
	Intel® MKL	MKL-DNN	Intel® MLSL	Intel® Data Analytics Acceleration Library (DAAL)	Intel® Distribution	OpenSource Frameworks	Intel Deep Learning SDK	Intel® Computer Vision SDK
High Level Overview	High performance math primitives granting low level of control	Free open source DNN functions for high-velocity integration with deep learning frameworks	Primitive communication building blocks to scale deep learning framework performance over a cluster	Broad data analytics acceleration object oriented library supporting distributed ML at the algorithm level	Most popular and fastest growing language for machine learning	Toolkits driven by academia and industry for training machine learning algorithms	Accelerate deep learning model design, training and deployment	Toolkit to develop & deploying vision-oriented solutions that harness the full performance of Intel CPUs and SOC accelerators
Primary Audience	Consumed by developers of higher level libraries and Applications	Consumed by developers of the next generation of deep learning frameworks	Deep learning framework developers and optimizers	Wider Data Analytics and ML audience, Algorithm level development for all stages of data analytics	Application Developers and Data Scientists	Machine Learning App Developers, Researchers and Data Scientists.	Application Developers and Data Scientists	Developers who create vision-oriented solutions
Example Usage	Framework developers call matrix multiplication, convolution functions	New framework with functions developers call for max CPU performance	Framework developer calls functions to distribute Caffe training compute across an Intel® Xeon Phi™ cluster	Call distributed alternating least squares algorithm for a recommendation system	Call scikit-learn k-means function for credit card fraud detection	Script and train a convolution neural network for image recognition	Deep Learning training and model creation, with optimization for deployment on constrained end device	Use deep learning to do pedestrian detection

* http:// software.intel.com/ai

Libraries, Frameworks and Tools*

	Intel® Math Kernel Library							
	Intel® MKL	MKL-DNN	Intel® MLSL	Intel® Data Analytics Acceleration Library (DAAL)	Intel® Distribution	OpenSource Frameworks	Intel Deep Learning SDK	Intel® Computer Vision SDK
High Level Overview	High performance math primitives granting low level of control	Free open source DNN functions for high-velocity integra[illegible]on w[illegible] deep[illegible]	Primitive communication building blocks to scale deep learning framework performance over a cluster	Broad data analytics acceleration object oriented library supportin[illegible] dist[illegible]ted [illegible] at th[illegible]	Most popular and fastest growing language for machine learning	Toolkits driven by academia and industry for training machine learning algorithms	Accelerate deep learning model design, training and deployment	Toolkit to develop & deploying vision-oriented solutions that harness the full performance of Intel CPUs and SOC accelerators
Primary Audience	Consumed by developers of higher level libraries and Applications	[illegible] g[illegible]neration of deep learning frameworks	Deep learning framework developers and optimizers	[illegible]e, [illegible] level [illegible]velopment for all stages of data analytics	Application Developers and Data Scientists	Machine Learning App Developers, Researchers and Data Scientists.	Application Developers and Data Scientists	Developers who create vision-oriented solutions
Example Usage	Framework developers call matrix multiplication, convolution functions	New framework with functions developers call for max CPU performance	Framework developer calls functions to distribute Caffe training compute across an Intel® Xeon Phi™ cluster	Call distributed alternating least squares algorithm for a recommendation system	Call scikit-learn k-means function for credit card fraud detection	Script and train a convolution neural network for image recognition	Deep Learning training and model creation, with optimization for deployment on constrained end device	Use deep learning to do pedestrian detection

Open Source

Open Source

* http:// software.intel.com/ai

Deep learning with Intel MKL-DNN (C/C++ API)

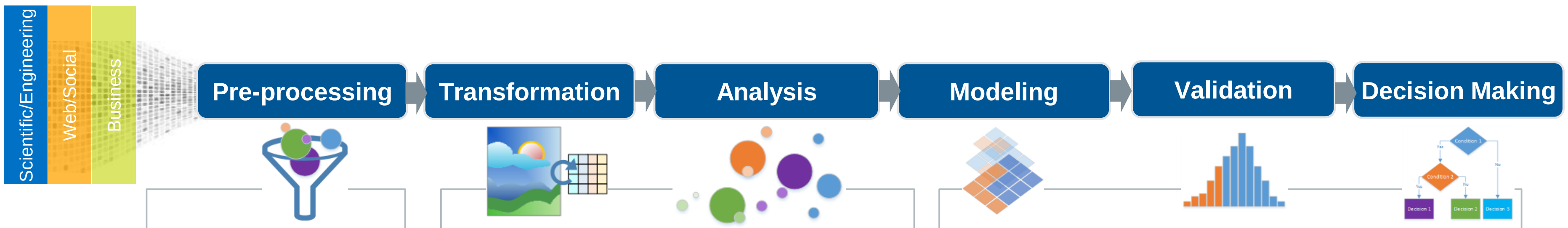
Intel® MKL-DNN Programming Model

- **Primitive** – any operation (convolution, data format re-order, memory)
    - **Operation/memory descriptor** - convolution parameters, memory dimensions
    - **Descriptor** - complete description of a primitive
    - **Primitive** – a specific instance of a primitive relying on descriptor
- **Engine** – execution device (e.g., CPU)
- **Stream** – execution context

```
/* Initialize CPU engine */
auto cpu_engine = mkldnn::engine(mkldnn::engine::cpu, 0);
/* Create a vector of primitives */
std::vector<mkldnn::primitive> net;
/* Allocate input data and create a tensor structure that describes it */
std::vector<float> src(2 * 3 * 227 * 227);
mkldnn::tensor::dims conv_src_dims = {2, 3, 227, 227};
/* Create memory descriptors, one for data and another for convolution input */
auto user_src_md = mkldnn::memory::desc({conv_src_dims},
mkldnn::memory::precision::f32, mkldnn::memory::format::nchw);
auto conv_src_md = mkldnn::memory::desc({conv_src_dims},
mkldnn::memory::precision::f32, mkldnn::memory::format::any);
/* Create convolution descriptor */
auto conv_desc = mkldnn::convolution::desc(
mkldnn::prop_kind::forward, mkldnn::convolution::direct,
conv_src_md, conv_weights_md, conv_bias_md, conv_dst_md,
{1, 1}, {0, 0}, mkldnn::padding_kind::zero);

/* Create a convolution primitive descriptor */
auto conv_pd = mkldnn::convolution::primitive_desc(conv_desc, cpu_engine);

/* Create a memory descriptor and primitive */
auto user_src_memory_descriptor
= mkldnn::memory::primitive_desc(user_src_md, engine);
auto user_src_memory = mkldnn::memory(user_src_memory_descriptor, src);

/* Create a convolution primitive and add it to the net */
auto conv = mkldnn::convolution(conv_pd, conv_input, conv_weights_memory,
conv_user_bias_memory, conv_dst_memory);
net.push_back(conv);

/* Create a stream, submit all primitives and wait for completion */

mkldnn::stream().submit(net).wait();
```

Intel DAAL Overview

Industry leading performance, open source C++/Java/Python library for machine learning and deep learning optimized for Intel®



(De-)Compression

PCA
Statistical moments
Variance matrix
QR, SVD, Cholesky
Apriori

Linear regression
Naïve Bayes
SVM
Classifier boosting

Kmeans
EM GMM

Collaborative filtering

Neural Networks

What's the difference? Intel MKL vs. Intel DAAL

	Intel MKL	Intel DAAL
DNN primitives	Performance critical	Performance critical and convenience
DNN layers	No	All building blocks for NN topology
Optimization solvers	No	Yes
Performance	Top in the class, full control from user side	Build on top of MKL, more convenience, on-par with MKL
Distributed memory	No (not yet)	APIs and samples for Spark, Hadoop, MPI
Language support	Low level: C, C++ - coming	High level: C++, Java, Python
Target audience	Code ninjas, and users who want to speedup existing frameworks	Wider ML audience, and end users. For example, users who want to prototype or build from scratch

LIBXSMM LIBRARY TARGETING INTEL ARCHITECTURE (X86)

FOR SMALL, DENSE OR SPARSE MATRIX MULTIPLICATIONS, AND SMALL CONVOLUTIONS.

Hans Pabst
Intel High Performance and Throughput Computing
Switzerland

Alexander Heinecke
Parallel Computing Lab
Intel Labs

Greg Henry
SSG Pathfinding
USA

https://github.com/hfp/libxsmm

LIBXSMM, for small, dense or sparse matrix multiplications, and small convolutions on Intel Architecture

General-purpose code cannot be optimal:

- If all cases are supported, the library is too large, too much branching, etc.
- Lack of specialization hurts when matrices are 1-10 SIMD units on a side.

Runtime specialization captures best of both worlds:

- “Perfect” code for only the cases needed; unused code is never generated.
- Just-in-Time (JiT) compilation for general code is hard.
- Specific domain (SMM, DNN, etc.) allows for JiT code generation without a compiler.

LIBXSMM Function Domains

Main function domains in LIBXSMM

SMM	Small Matrix Multiplication Kernels (original library)
DNN	Deep Neural Network Kernels for CNNs (v1.5)
SPMDM	Sparse Matrix Dense Matrix Multiplication for CNNs (v1.6)
AUX	Mem. alloc., synchronization, debugging, profiling

There is more functionality…

- Tiled GEMM routines based on SMM kernels (also parallelized)
- Stand-alone out-of-place matrix transpose routines (non-JIT, soon JIT)
- Matrix-copy kernels (JIT)
- Other “sparse routines”

LIBXSMM: Overview

Highly efficient Frontend

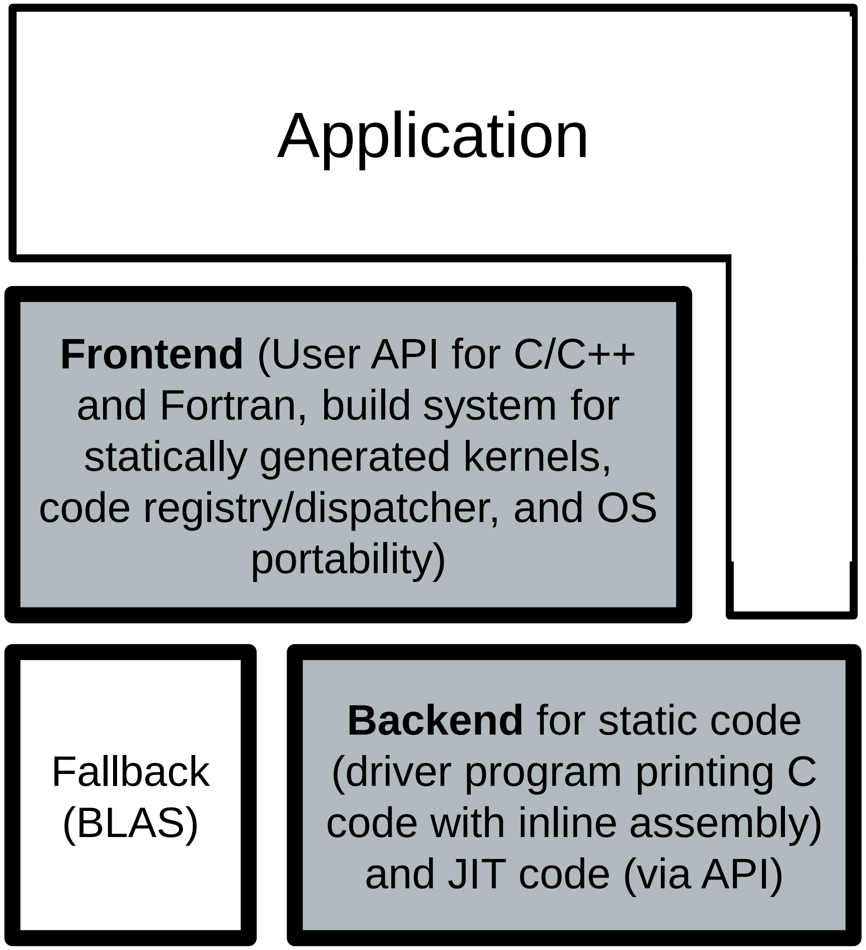
- BLAS compatible (DGEMM, SGEMM) including LD_PRELOAD
- Support for F77, C89/C99, F2003, C++
- Zero-overhead calls into assembly
- Two-level code cache

Code Generator

- Supports all Intel Architectures since 2005, focus on AVX-512
- Prefetching across small GEMMs
- Can generate assembly (*.s), inline assembly (*.h/*.c), and in-memory code

Just-In-Time (JIT) Encoder

- Encodes instructions based on basic blocks
- Very fast code generation (no compilation)


Application
Frontend (User API for C/C++ and Fortran, build system for statically generated kernels, code registry/dispatcher, and OS portability)
Fallback (BLAS)
Backend for static code (driver program printing C code with inline assembly) and JIT code (via API)

JIT Technologies (2015)

Evaluation of suitable JIT code generators

- Numerous projects evaluated: jitasm, libgccjit, etc.
- Selection/rejection criterions
  - Support for recent Intel Architectures,
  - Active development
- Interesting candidates (with comments)
  - LLVM Full-blown (with IR, phases, etc.), “slow” JIT, complex
  - Xbyak compiler and JIT-assembler, incomplete AVX-512 (2015!)
  - XED Closed source (2016: https://github.com/intelxed/xed)

Final decision in 2015: own development needed “JIT Assembler”

- Only “a few” instructions needed for a certain domain (still true)
- No legacy support needed (AVX/2 and beyond is fine)

LIBXSMM Backend: Runtime Code Generation (Very High Level Idea)

Idea: leveraged GNU Compiler extension "Computed GOTO"

```
LABEL1:
   c = a + b;
LABEL2:

memcpy(code, &&LABEL1, &&LABEL2 - &&LABEL1);
```

Reality: LIBXSMM manually encodes all instructions needed

- Basic form is encoded with placeholder(s) for varying parts (immediates)
- Emitting an instruction: call a function (arguments may cover instruction variants and/or immediates), to write a whole kernel is like using a DSL ("assembly programming domain")

LIBXSMM Backend: Code Generation (cont.)

Quick facts about in-memory JIT code generation (JIT assembler)

- No intermediate representation
- No automatic register allocation
- No (compiler-)optimizations

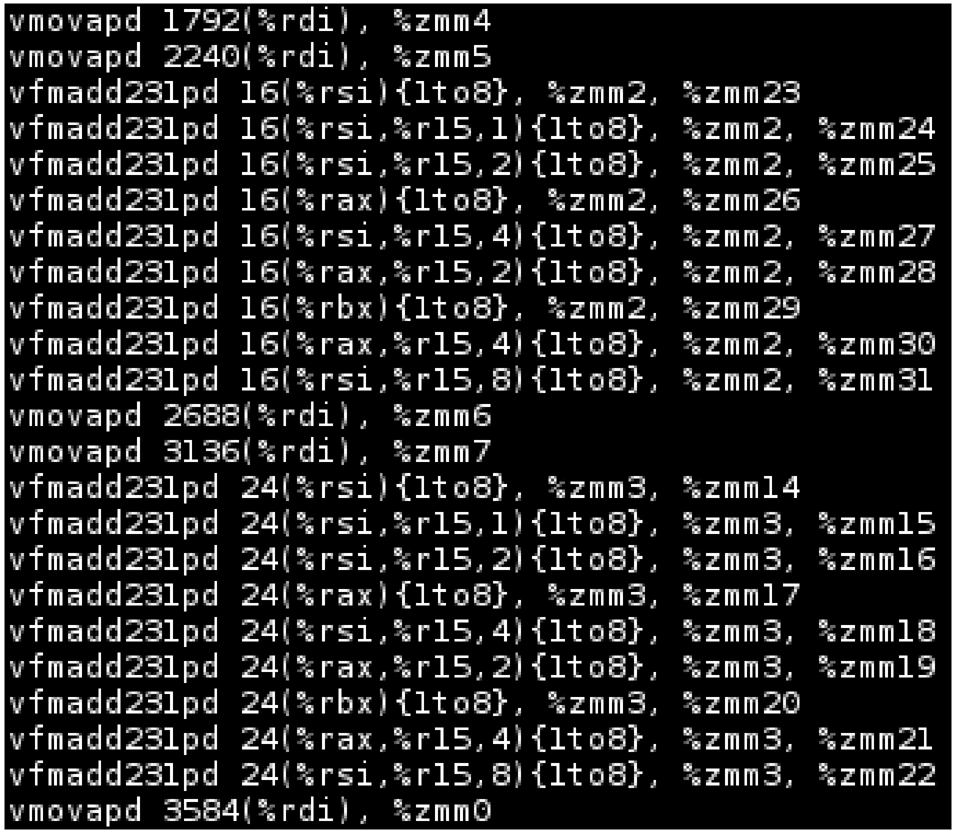
What is the advantage of JIT code?

- It is able to leverage instruction variants/immediates to hardcode runtime knowledge (hard to statically compile equivalent code!)

  Example: hard-coded stride for load instruction address (broadcast ld.)

- Why is there a particular focus on AVX-512? There is a lot of potential in the instruction set e.g., EVEX may also encode certain values into instruction

LIBXSMM AVX512 code for N=9



```
vmovapd 1792(%rdi), %zmm4
vmovapd 2240(%rdi), %zmm5
vfmadd231pd 16(%rsi){1to8}, %zmm2, %zmm23
vfmadd231pd 16(%rsi,%r15,1){1to8}, %zmm2, %zmm24
vfmadd231pd 16(%rsi,%r15,2){1to8}, %zmm2, %zmm25
vfmadd231pd 16(%rax){1to8}, %zmm2, %zmm26
vfmadd231pd 16(%rsi,%r15,4){1to8}, %zmm2, %zmm27
vfmadd231pd 16(%rax,%r15,2){1to8}, %zmm2, %zmm28
vfmadd231pd 16(%rbx){1to8}, %zmm2, %zmm29
vfmadd231pd 16(%rax,%r15,4){1to8}, %zmm2, %zmm30
vfmadd231pd 16(%rsi,%r15,8){1to8}, %zmm2, %zmm31
vmovapd 2688(%rdi), %zmm6
vmovapd 3136(%rdi), %zmm7
vfmadd231pd 24(%rsi){1to8}, %zmm3, %zmm14
vfmadd231pd 24(%rsi,%r15,1){1to8}, %zmm3, %zmm15
vfmadd231pd 24(%rsi,%r15,2){1to8}, %zmm3, %zmm16
vfmadd231pd 24(%rax){1to8}, %zmm3, %zmm17
vfmadd231pd 24(%rsi,%r15,4){1to8}, %zmm3, %zmm18
vfmadd231pd 24(%rax,%r15,2){1to8}, %zmm3, %zmm19
vfmadd231pd 24(%rbx){1to8}, %zmm3, %zmm20
vfmadd231pd 24(%rax,%r15,4){1to8}, %zmm3, %zmm21
vfmadd231pd 24(%rsi,%r15,8){1to8}, %zmm3, %zmm22
vmovapd 3584(%rdi), %zmm0
```

→ **Max. theoretical efficiency: 90%!**

- Column-major storage; working on all 9 columns and 8 rows simultaneously
- Loads to A (vmovapd) are spaced out to cover L1$ misses; K-loop is fully unrolled
- B-elements are broadcasted within the FMA instruction to save execution slots (SIB)
- SIB addressing mode to keep instruction size <= 8 byte for 2 decodes per cycle (16 byte I-fetch per cycle)
- Multiple accumulators (zmm31-xmm23 and zmm22-zmm14) for hiding FMA latencies

LIBXSMM: Other Features

CPUID-dispatched (critical) code paths

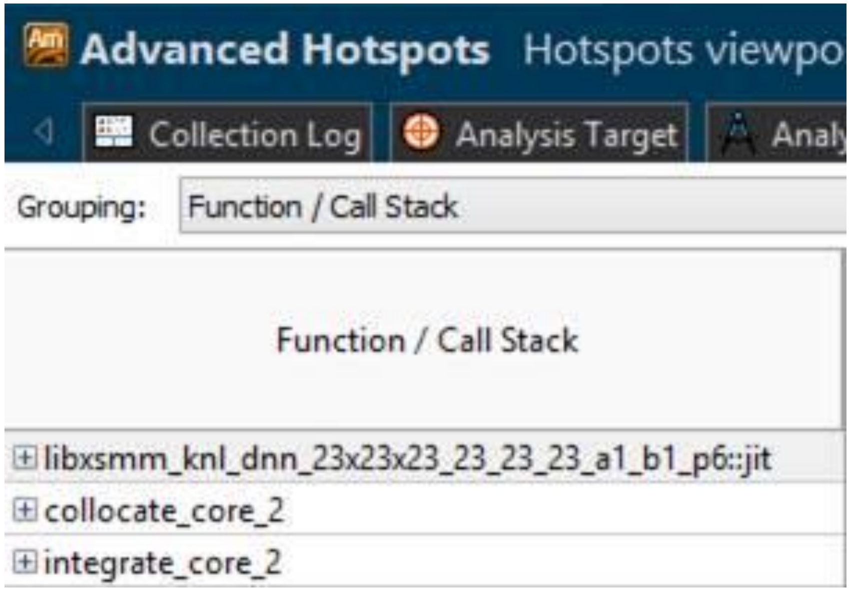
Makes LIBXSMM suitable for Linux distributions where the code path (target system) is unpredictable (1 package)

Link-time and Runtime Wrapper

Intercepts existing xGEMM calls at runtime (LD_PRELOAD) or at link-time (LD's --wrap)

JIT Profiling

Support for ***Intel VTune Amplifier*** and ***Linux Perf*** (contributed by Google)


Advanced Hotspots Hotspots viewpo
Collection Log
Analysis Target
Analy
Grouping: Function / Call Stack
Function / Call Stack

```
⊞ libxsmm_knl_dnn_23x23x23_23_23_23_a1_b1_p6::jit
⊞ collocate_core_2
⊞ integrate_core_2
```

libxsmm_hsw_dnn_23x23x23_23_23_23_a1_b1_p0::jit

- Encodes an Intel AVX-512 ("knl") double-precision kernel ("d") which is multiplying matrices without transposing them ("nn"),
- Rest of the name encodes M=N=K=LDA=LDB=LDC=23, Alpha=Beta=1.0 (all similar to GEMM),
- No prefetch strategy ("p0").

LIBXSMM: Developments

Support for “medium-sized” and “big” matrix multiplication

- Tiled matrix multiplication routines to go beyond SMM
- OpenMP for multicore support
    - Thread-based multicore with internal parallel region
    - Task based when called from parallel region
- Plan: more LIBXSMM for Eigen-library,
  and non-OpenMP based multithreading

Initial support for stand-alone matrix transposes

- Tiled transpose optionally with task-based OpenMP

LIBXSMM DNN

* https://github.com/hfp/libxsmm/#interface-for-convolutions

LIBXSMM: Interface for Convolutions (DNN API)

DNN API for Convolutional Neural Networks (CNNs)

- Introduced in LIBXSMM 1.5, refined in v1.6 and v1.7
- Features:
  - Fallback code, JIT-code: AVX2, and AVX-512 (Common, Core, KNM)
  - Forward convolution, backward convolution, and weight transformation
  - Data formats: NHWC, RSCK, and custom formats
  - Compute types: f32, and i16 (+ i8 as in-/output)
- Major additions in v1.8: logical padding, Winograd, KNM

Quick summary

- Handle-based API to generate/perform the requested transformation

LIBXSMM: Getting Started with DNN API

LIBXSMM DNN API

- Sample code (samples/dnn) to also act as benchmark for convolutions
- Results

  DeepBench: https://software.intel.com/en-us/articles/intel-xeon-phi-delivers-competitive-performance-for-deep-learning-and-getting-better-fast

TensorFlow

https://github.com/hfp/libxsmm/blob/master/documentation/tensorflow.md

- Initial integration only in master revision of TensorFlow
- Scheduled for TF 1.1

LIBXSMM: Applications HPC

[1] https://cp2k.org/: Open Source Molecular Dynamics with its DBCSR component processing batches of small matrix multiplications ("matrix stacks") out of a problem-specific distributed block-sparse matrix. Starting with CP2K 3.0, LIBXSMM can be used to substitute CP2K's 'libsmm' library. Prior to CP2K 3.0, only the Intel-branch of CP2K integrated LIBXSMM (see https://github.com/hfp/libxsmm/raw/master/documentation/cp2k.pdf).

[2] https://github.com/SeisSol/SeisSol/: SeisSol is one of the leading codes for earthquake scenarios, for simulating dynamic rupture processes. LIBXSMM provides highly optimized assembly kernels which form the computational back-bone of SeisSol (see https://github.com/TUM-I5/seissol_kernels/).

[3] https://github.com/NekBox/NekBox: NekBox is a highly scalable and portable spectral element code, which is inspired by the Nek5000 code. NekBox is specialized for box geometries, and intended for prototyping new methods as well as leveraging FORTRAN beyond the FORTRAN 77 standard. LIBXSMM can be used to substitute the MXM_STD code. Please also note LIBXSMM's NekBox reproducer.

[4] https://github.com/Nek5000/Nek5000: Nek5000 is the open-source, highly-scalable, always-portable spectral element code from https://nek5000.mcs.anl.gov/. The development branch of the Nek5000 code incorporates LIBXSMM.

[5] http://pyfr.org/: PyFR is an open-source Python based framework for solving advection-diffusion type problems on streaming architectures using the flux reconstruction approach. PyFR 1.6.0 optionally incorporates LIBXSMM as a matrix multiplication provider for the OpenMP backend. Please also note LIBXSMM's PyFR-related code sample.

LIBXSMM: Applications ML

[6] https://github.com/baidu-research/DeepBench: The primary purpose of DeepBench is to benchmark operations that are important to deep learning on different hardware platforms. LIBXSMM's DNN primitives have been incorporated into DeepBench to demonstrate an increased performance of deep learning on Intel hardware. In addition, LIBXSMM's DNN sample folder contains scripts to run convolutions extracted from popular benchmarks in a stand-alone fashion.

[7] https://www.tensorflow.org/: TensorFlow™ is an open source software library for numerical computation using data flow graphs. TensorFlow was originally developed by researchers and engineers working on the Google Brain Team for the purposes of conducting machine learning and deep neural networks research. LIBXSMM can be used to increase the performance of TensorFlow on Intel hardware.

LIBXSMM: References

[1] http://sc16.supercomputing.org/presentation/?id=pap364&sess=sess153: LIBXSMM: Accelerating Small Matrix Multiplications by Runtime Code Generation (paper). SC'16: The International Conference for High Performance Computing, Networking, Storage and Analysis, Salt Lake City (Utah).

[2] http://sc15.supercomputing.org/sites/all/themes/SC15images/tech_poster/tech_poster_pages/post137.html: LIBXSMM: A High Performance Library for Small Matrix Multiplications (poster and abstract). SC'15: The International Conference for High Performance Computing, Networking, Storage and Analysis, Austin (Texas).

[3] https://software.intel.com/en-us/articles/intel-xeon-phi-delivers-competitive-performance-for-deep-learning-and-getting-better-fast: Intel Xeon Phi Delivers Competitive Performance For Deep Learning - And Getting Better Fast. Article mentioning LIBXSMM's performance of convolution kernels with DeepBench. Intel Corporation, 2016.

Questions?

Seeded Questions

1. **From the industry angle, how do you see the field of machine learning (ML) evolving over the next decade?**

   Aside from “insights” on how technology emerges: “pre-hype”, “hype”, “late adopters”, etc.   I am seeing ML still in an explorative phase (wrt known use cases such as imaging, voice recognition, etc.), scientific and non-imaging cases still need adequate primitives (e.g., need to currently “color code” scientific data), a few cases are also entirely “rebadged” applications. I expect surprisingly powerful applications to surpass the experience of services (“press 1 to…”), ever ongoing consolidation of for instance services currently supplied by humans, improved foundations and tighter bounds, and better estimates for machine learning.

Seeded Questions

2. **What are the limitations of ML today and the road ahead?**

   Scientific foundations did not accommodate ML so far (which does not mean there are no foundations), and better/adapted theoretical coverage is much needed. We also need “recipes” (not so scientific) on “how to achieve” or “how to avoid”. On the computation side, I do not see big hurdles to ever proceed. However, if something shows (proofs?) to be incorrect or superseded – “ML bits” may leave a significant legacy.

3. **Is realtime (~< microsecond) application of sophisticated ML possible?**

   Inferring from a trained existing model is subject to realtime decision making (e.g., industrial processes, autonomous driving). Thinking of today (“2…4 GHz clocked CPU”) – micro (10^{-6}) is in the thousands (10^3) of cycles/instructions of giga (10^9) and enables inference. Learning however, may be possible in that range as well but requires “scale” or sufficient sources (“data center”). This makes microsecond-timeframes unreliable for learning given today’s options.

Seeded Questions (cont.)

4. **Physicists use ML in many areas: classification of particles and events, measurement of particle properties from increasingly lower-level data, some unsupervised learning. In your opinion, what are we not doing yet? e.g. what exciting new ideas of ML have not yet made it to particle physics?**

   I cannot really provide advise on physics. Unfortunately (fortunately?), the most interesting things are not classified, not learned already, and do not adhere to an expectation. Perhaps it’s possible to learn “nothing” (e.g., noise rather than filtering it) in order to find something i.e., the “disjunction of nothing” (at least to check whether it leaves room for an event or not).

Seeded Questions (cont.)

5. **Many physicists choose to go on to industry careers. What are the skills that they need to master in ML to be more competitive.**

   This has yet to be seen. I have seen fluctuations in industry in both directions from/to companies, and that includes “known to be good” people as well as the opposite with no big perceived difference in booking the heads. Supplying skills will differentiate very soon if not differentiated already. For physicists (just as for any scientist), I can see a focus on “non-traditional applications” (if this already exists for ML) might be valuable: working with time series data, measurements, or sensors and data acquisition in general, uncertainty, and quantifying natural effects, and being able to embed applications (HW/SW combo) might be advantageous.

Legal Disclaimer & Optimization Notice

intel®

experience what's inside™