INTEL AI WORKSHOP

Shailen Sobhee - Technical Consulting Engineer
Intel Architecture, Graphics and Software (IAGS)
Note: All slides in this slide deck were unhidden. During the three-hours of presentation, a select number of these slides that were relevant to the target audience were presented.

I am providing the entirety of the material for your own convenience.

Happy reading 😊
Legal Disclaimer & Optimization Notice

Performance results are based on testing as of September 2018 and may not reflect all publicly available security updates. See configuration disclosure for details. No product can be absolutely secure.

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.

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.

Copyright © 2019, Intel Corporation. All rights reserved. Intel, the Intel logo, Pentium, Xeon, Core, VTune, OpenVINO, Cilk, are trademarks of Intel Corporation or its subsidiaries 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
This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications and roadmaps.

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/performance.

Cost reduction scenarios described are intended as examples of how a given Intel-based product, in the specified circumstances and configurations, may affect future costs and provide cost savings. Circumstances will vary. Intel does not guarantee any costs or cost reduction.

Statements in this document that refer to Intel’s plans and expectations for the quarter, the year, and the future, are forward-looking statements that involve a number of risks and uncertainties. A detailed discussion of the factors that could affect Intel’s results and plans is included in Intel's SEC filings, including the annual report on Form 10-K.

The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel does not control or audit third-party benchmark data or the web sites referenced in this document. You should visit the referenced web site and confirm whether referenced data are accurate.

Intel, the Intel logo, Pentium, Celeron, Atom, Core, Xeon, Movidius 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.
Intel AI Workshop Agenda

Introduction to Intel Software Developer Tools

- We will go quickly through them
- Intel Distribution for Python
  - Hands-on exercises (NumPy, Numba, performance considerations)
  - Classical Machine learning (scikit-learn)
INTRODUCTION TO
INTEL SOFTWARE DEVELOPER TOOLS
Intel® Parallel Studio XE
What is it?
A comprehensive tool suite for building high-performance, scalable parallel code from enterprise to cloud, and HPC to AI applications.
- Includes C++, Fortran, & Python performance tools: industry-leading compilers, numerical libraries, performance profilers, & code analyzers
- Supports Windows*, Linux* & macOS*

Who needs this product?
- OEMs/ISVs
- C++, Fortran, & Python* developers
- Developers, domain specialists of enterprise, data center/cloud, HPC & AI applications

Why important?
- Accelerate performance on Intel® Xeon® & Core™ processors
- Deliver fast, scalable, reliable parallel code with less effort
- Modernize code efficiently—optimize for today's & future Intel® platforms
- Stay up-to-date with standards

Free 30-Day Trial—Download: software.intel.com/intel-parallel-studio-xe
Accelerate Parallel Code
Intel® Parallel Studio XE Capabilities

Build Fast, Scalable Parallel Applications from Enterprise to Cloud & HPC to AI

- Take advantage capabilities & performance on the latest Intel® platforms. Simplify modernizing code with proven techniques in vectorization, multi-threading, multi-node & memory optimization.

- Boost application performance, accelerate diverse workloads and machine learning with industry-leading compilers, libraries, and Intel® Distribution for Python*.

- Increase developer productivity—quickly spot high-payoff opportunities for faster code.
  - View memory, network, storage, MPI, CPU, and FPU usage with Application Performance Snapshots. Interactively build, validate algorithms with Flow Graph Analyzer. Find high-impact, under-performing loops with Roofline Analysis.
  - Use in popular development environments—profile enterprise applications inside Docker* and Mesos* containers, and running Java* services and daemons.

- Extend HPC solutions on the path to Exascale—gain scalability, reduce latency with Intel® MPI Library.

- Take advantage of Priority Support—get more from your code, overcome development challenges. Connect privately with Intel engineers for quick answers to technical questions.¹

¹ Applies to License purchases only. Free or discounted Intel Software Tools may be available for qualified Student & Academia.
What’s Inside Intel® Parallel Studio XE
Comprehensive Software Development Tool Suite

<table>
<thead>
<tr>
<th><strong>COMPOSER EDITION</strong></th>
<th><strong>PROFESSIONAL EDITION</strong></th>
<th><strong>CLUSTER EDITION</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BUILD</strong></td>
<td><strong>ANALYZE</strong></td>
<td><strong>SCALE</strong></td>
</tr>
<tr>
<td>Compilers &amp; Libraries</td>
<td><strong>Intel® VTune™ Amplifier</strong></td>
<td>**Inte...</td>
</tr>
<tr>
<td>Intel® Math Kernel Library</td>
<td>Performance Profiler</td>
<td><strong>Intel® MPI Library</strong></td>
</tr>
<tr>
<td>Intel® Data Analytics</td>
<td><strong>Intel® Inspector</strong></td>
<td>Message Passing Interface Library</td>
</tr>
<tr>
<td>Acceleration Library</td>
<td>Memory &amp; Thread Debugger</td>
<td>Intel® Trace Analyzer &amp; Collector</td>
</tr>
<tr>
<td>Intel Threading Building Blocks</td>
<td><strong>Intel® Advisor</strong></td>
<td>MPI Tuning &amp; Analysis</td>
</tr>
<tr>
<td>C++ Threading</td>
<td>Vectorization Optimization</td>
<td>Intel® Cluster Checker</td>
</tr>
<tr>
<td><strong>Intel® Integrated Performance Primitives</strong></td>
<td>Thread Prototyping &amp; Flow Graph Analysis</td>
<td>Cluster Diagnostic Expert System</td>
</tr>
<tr>
<td>Image, Signal &amp; Data Processing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Intel® Distribution for Python*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>High Performance Python</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Operating System: Windows*, Linux*, MacOS1*

Intel® Architecture Platforms

1Available only in the Composer Edition.

Copyright © 2019, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.
HPC & AI Software Optimization Success Stories

Intel® Parallel Studio XE

**SCIENCE & RESEARCH**

Up to 35X faster application performance

NERSC (National Energy Research Scientific Computing Center)

Read case study

**ARTIFICIAL INTELLIGENCE**

Performance speedup of up to 23X faster with Intel optimized scikit-learn vs. stock scikit-learn

Google Cloud Platform

**LIFE SCIENCE**

Simulations ran up to 7.6X faster with 9X energy efficiency

LAMMPS code - Sandia National Laboratories

Read technology brief

For more success stories, review Intel® Parallel Studio XE Case Studies

**Intel® Xeon Phi™ Processor Software Ecosystem Momentum Guide**

Performance results are based on the tests from 2016-2017 and may not reflect all publicly available security updates. See configuration disclosure for details. No product can be absolutely secure. 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 & functions. Any change to any of those factors may cause the results to vary. You should consult other information & performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more information go to www.intel.com/performance. See configurations in individual case study links. 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. 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. For more complete information about compiler optimizations, see our Optimization Notice.

Optimization Notice

Copyright © 2019, Intel Corporation. All rights reserved.

*Other names and brands may be claimed as the property of others.
Take Advantage of Intel Priority Support

Paid licenses of Intel® Software Development Tools include Priority Support for one year from your date of purchase, with options to extend support at a highly discounted rate.

Benefits

▪ **Performance & productivity**—get the most from your code on Intel hardware, and overcome performance bottlenecks or development challenges.

▪ **Direct, private** interaction with Intel engineers. Submit confidential inquiries & code samples for consultation.

▪ **Responsive help** with your technical questions & other product needs.

▪ **Free access** to all new product updates & access to older versions.

Additional Resources

▪ Learn from other experts via community product forums

▪ Access to a vast library of self-help documents that build off decades of experience with creating high performance code.
INTEL® PARALLEL STUDIO XE TOOLS DETAILS

BUILD
Intel® C++ Compiler
Intel® Fortran Compiler
Intel® Distribution for Python*
Intel® Math Kernel Library
Intel® Integrated Performance Primitives
Intel® Threading Building Blocks
Intel® Data Analytics Acceleration Library
Included in Composer Edition

ANALYZE
Intel® VTune™ Amplifier
Intel® Advisor
Intel® Inspector
Part of the Professional Edition

SCALE
Intel® MPI Library
Intel® Trace Analyzer & Collector
Intel® Cluster Checker
Part of the Cluster Edition
What’s New in Intel® Compilers 2019 (19.0)

Updates to All Versions

Advance Support for Intel® Architecture—use Intel® Compilers to generate optimized code for Intel Atom® processor through Intel® Xeon® Scalable processors.

Achieve Superior Parallel Performance—vectorize & thread your code (using OpenMP*) to take advantage of the latest SIMD-enabled hardware, including Intel® Advanced Vector Extensions 512 (Intel® AVX-512).

What’s New in C++

Additional C++17 Standard feature support
- Enjoy improvements to lambda & constant expression support
- Improved GNU C++ & Microsoft C++ compiler compatibility

Standards-driven parallelization for C++ developers
- Partial OpenMP* 5¹ support
- Modernize your code by using the latest parallelization specifications

What’s New in Fortran

Substantial Fortran 2018 support including
- Coarray features: EVENTS & COSHAPE
- IMPORT statement enhancements
- Default module accessibility

Complete OpenMP 4.5 support; user-defined reductions
- Check shape option for runtime array conformance checking

¹OpenMP 5 is currently a draft
## Industry-leading Application Performance on Linux* using Intel® C++ & Fortran Compilers (higher is better)

### Boost C++ Application Performance on Linux* using Intel® C++ Compiler

<table>
<thead>
<tr>
<th>Floating Point</th>
<th>Integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clang® 6.0</td>
<td>1.14</td>
</tr>
<tr>
<td>GCC® 8.1.0</td>
<td>1.2</td>
</tr>
<tr>
<td>Intel® C++ 19.0</td>
<td>1.34</td>
</tr>
</tbody>
</table>

**Estimated geometric mean of SPEC CPU2017 Floating Point rate base C/C++ benchmarks**

### Boost Fortran Application Performance on Linux* using Intel® Fortran Compiler

<table>
<thead>
<tr>
<th>Integer</th>
<th>1.00</th>
</tr>
</thead>
<tbody>
<tr>
<td>PGI® 18.5</td>
<td>1.14</td>
</tr>
<tr>
<td>gFortran® 8.1.0</td>
<td></td>
</tr>
</tbody>
</table>

**Estimated relative geometric performance, Polyhedron* benchmark– higher is better**

---

Performance results are based on testing as of Aug. 26, 2018 and may not reflect all publicly available security updates. See configuration disclosure for details. No product can be absolutely secure. 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**

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

Copyright © 2019, Intel Corporation. All rights reserved.
Accelerate Python* with Intel® Distribution for Python*
High Performance Python* for Scientific Computing, Data Analytics, Machine & Deep Learning

**FASTER PERFORMANCE**

<table>
<thead>
<tr>
<th>Performance Libraries, Parallelism, Multithreading, Language Extensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>▪ Accelerated NumPy/SciPy/scikit-learn with Intel® MKL&lt;sup&gt;1&lt;/sup&gt; &amp; Intel® DAAL&lt;sup&gt;2&lt;/sup&gt;</td>
</tr>
<tr>
<td>▪ Data analytics, machine learning &amp; deep learning with scikit-learn, pyDAAL, TensorFlow* &amp; Caffe*</td>
</tr>
<tr>
<td>▪ Scale with Numba* &amp; Cython*</td>
</tr>
<tr>
<td>▪ Includes optimized mpi4py, works with Dask* &amp; PySpark*</td>
</tr>
<tr>
<td>▪ Optimized for latest Intel® architecture</td>
</tr>
</tbody>
</table>

**GREATER PRODUCTIVITY**

<table>
<thead>
<tr>
<th>Prebuilt &amp; Accelerated Packages</th>
</tr>
</thead>
<tbody>
<tr>
<td>▪ Prebuilt &amp; optimized packages for numerical computing, machine/deep learning, HPC, &amp; data analytics</td>
</tr>
<tr>
<td>▪ Drop in replacement for existing Python-No code changes required</td>
</tr>
<tr>
<td>▪ Jupyter* notebooks, Matplotlib included</td>
</tr>
<tr>
<td>▪ Free download &amp; free for all uses including commercial deployment</td>
</tr>
</tbody>
</table>

**ECOSYSTEM COMPATIBILITY**

<table>
<thead>
<tr>
<th>Supports Python 2.7 &amp; 3.x, Conda &amp; PIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>▪ Supports Python 2.7 &amp; 3.x, optimizations integrated in Anaconda* Distribution</td>
</tr>
<tr>
<td>▪ Distribution &amp; optimized packages available via Conda, PIP, APT GET, YUM, &amp; DockerHub, numerical performance optimizations integrated in Anaconda Distribution</td>
</tr>
<tr>
<td>▪ Optimizations upstreamed to main Python trunk</td>
</tr>
<tr>
<td>▪ Priority Support with Intel® Parallel Studio XE</td>
</tr>
</tbody>
</table>

Operating System: Windows*, Linux*, MacOS<sup>1*</sup>

Intel® Architecture Platforms

---

<sup>1</sup>Intel® Math Kernel Library
<sup>2</sup>Intel® Data Analytics Acceleration Library

<sup>*Other names and brands may be claimed as the property of others.</sup>
Faster Python* with Intel® Distribution for Python*

Advance Performance Closer to Native Code
- Accelerated NumPy, SciPy, Scikit-learn for scientific computing, machine learning & data analytics
- Drop-in replacement for existing Python—no code changes required
- Highly optimized for the latest Intel® processors

What's New in the 2019 Release
- Faster machine learning with Scikit-learn: Support Vector Machine (SVM) & K-means prediction, accelerated with Intel® Data Analytics Acceleration Library
- Includes machine learning XGBoost library (Linux* only)
- Also available as easy command line standalone install

Performance results are based on testing as of July 9, 2018 and may not reflect all publicly available security updates. See configuration disclosure for details. No product can be absolutely secure. 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, see Performance Benchmark Test Disclosure. Testing by Intel as of July 9, 2018.

Configuration: Stock Python: python 3.6.6 hc3d631a_0 installed from conda, NumPy 1.15, numba 0.39.0, llvmlite 0.24.0, scipy 1.0.0, scikit-learn 0.19.2 installed from pip; Intel Python: Intel® Distribution for Python* 2019 Gold: python 3.6.5 intel_11, NumPy 1.14.3 intel_py36_5, mlk 2019.0 intel_101, mlk_fft 1.0.2 intel_np114py36_6, mkl_random 1.0.1 intel_np114py36_6, numba 0.39.0 intel_np114py36_6, llvmlite 0.24.0 intel_py36_6, scikit-learn 0.19.1 intel_np114py36_35; OS: CentOS Linux 7.3.1611, kernel 3.10.0-514.el7.x86_64; Hardware: Intel(R) Xeon(R) Gold 6140 CPU @ 2.30GHz (2 sockets, 18 cores/socket, HT:off), 256 GB of DDR4 RAM, 16 DIMMs of 16 GB@2666MHz

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.

For more complete information about compiler optimizations, see our Optimization Notice.
Fast, Scalable Code with Intel® Math Kernel Library (Intel® MKL)

- Speeds computations for scientific, engineering, financial and machine learning applications by providing highly optimized, threaded, and vectorized math functions.
- Provides key functionality for dense and sparse linear algebra (BLAS, LAPACK, PARDISO), FFTs, vector math, summary statistics, deep learning, splines and more.
- Dispatches optimized code for each processor automatically without the need to branch code.
- Optimized for single core vectorization and cache utilization.
- Automatic parallelism for multi-core and many-core.
- Scales from core to clusters.
- Available at no cost and royalty free.
- Great performance with minimal effort!

INTEL® MATH KERNEL LIBRARY OFFERS...

- Dense & Sparse Linear Algebra
- Fast Fourier Transforms
- Vector Math
- Vector RNGs
- Fast Poisson Solver
- & More!

Available only in Intel® Parallel Studio Composer Edition.

Optimization Notice
Copyright © 2019, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.
What’s New in Intel® Math Kernel Library 2019?

Just-In-Time Fast Small Matrix Multiplication

- Improved speed of S/DGEMM for Intel® AVX2 and Intel® AVX-512 with JIT capabilities

Sparse QR Solvers

- Solve sparse linear systems, sparse linear least squares problems, eigenvalue problems, rank and null-space determination, and others

Generate Random Numbers for Multinomial Experiments

- Highly optimized multinomial random number generator for finance, geological and biological applications
Optimization Notice
Copyright © 2019, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Speed Imaging, Vision, Signal, Security & Storage Apps with Intel® Integrated Performance Primitives (Intel® IPP)

Accelerate Image, Signal, Data Processing & Cryptography Computation Tasks

- Multi-core, multi-OS and multi-platform ready, computationally intensive & highly optimized functions
- Use high performance, easy-to-use, production-ready APIs to quickly improve application performance
- Reduce cost & time-to-market on software development & maintenance

What’s New in 2019 Release

- Functions for ZFP floating-point data compression to help tackle large data storage challenges, great for oil/gas applications
- Optimization patch files for the bzip2 source 1.0.6
- Improved LZ4 compression & decompression performance on high entropy data
- New color conversion functions for convert RBG images to CIE Lab color models, & vice versa
- Extended optimization for Intel® AVX-512 & Intel® AVX2 instruction set
- Open source distribution of Intel® IPP Cryptography Library

Learn More: software.intel.com/intel-ipp
What’s Inside Intel® Integrated Performance Primitives
High Performance, Easy-to-Use & Production Ready APIs

- Image Processing
- Computer Vision
- Color Conversion
- Signal Processing
- Vector Math
- Data Compression
- Cryptography
- String Processing

Operating Systems: Windows*, Linux*, MacOS†*
Intel® Architecture Platforms

†Available only in Intel® Parallel Studio Composer Edition.
*Other names and brands may be claimed as the property of others.
Get the Benefits of Advanced Threading with Threading Building Blocks

Use Threading to Leverage Multicore Performance & Heterogeneous Computing

- Parallelize computationally intensive work across CPUs, GPUs & FPGAs,—deliver higher-level & simpler solutions using C++
- Most feature-rich & comprehensive solution for parallel programming
- Highly portable, composable, affordable, approachable, future-proof scalability

What's New in 2019 Release

- New capabilities in Flow Graph improve concurrency & heterogeneity through improved task analyzer & OpenCL* device selection
- New templates to optimize C++11 multidimensional arrays
- C++17 Parallel STL, OpenCL*, & Python* Conda language support
- Expanded Windows*, Linux*, Android*, MacOS* support

Learn More: software.intel.com/intel-tbb
What's Inside Threading Building Blocks

Parallel Execution Interfaces
- Flow Graph
- Generic Parallel Patterns
- Parallel STL

Low-Level Interfaces
- Tasks
- Task arenas
- Global Control

Interfaces Independent of Execution Model
- Concurrent Containers
  - Hash Tables
  - Queues
  - Vectors
- Memory Allocation
  - Scalable Allocator
  - Cache Aligned Allocator
- Primitives and Utilities
  - Synchronization Primitives
  - Thread Local Storage
Heterogeneous Support
Threading Building Blocks (TBB)

TBB flow graph as a coordination layer for heterogeneity—retains optimization opportunities & composes with existing models

- CPUs, integrated GPUs, etc.

Threading Building Blocks
OpenVX*
OpenCL*
COI/SCIF
....

TBB as a **composability layer** for library implementations
- One threading engine **underneath** all CPU-side work

TBB flow graph as a **coordination layer**
- Be the glue that connects heterogeneous hardware & software together
- Expose parallelism between blocks—simplify integration
Optimization Notice
Copyright © 2019, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Speedup Analytics & Machine Learning with Intel® Data Analytics Acceleration Library (Intel® DAAL)

- Highly tuned functions for classical machine learning & analytics performance from datacenter to edge running on Intel® processor-based devices
- Simultaneously ingests data & computes results for highest throughput performance
- Supports batch, streaming & distributed usage models to meet a range of application needs
- Includes Python*, C++, Java* APIs, & connectors to popular data sources including Spark* & Hadoop

What’s New in the 2019 Release

New Algorithms

- **Logistic Regression**, most widely-used classification algorithm
- **Extended Gradient Boosting Functionality** for inexact split calculations & user-defined callback canceling for greater flexibility
- **User-defined Data Modification Procedure** supports a wide range of feature extraction & transformation techniques

Learn More: software.intel.com/daal
Algorithms, Data Transformation & Analysis
Intel® Data Analytics Acceleration Library

Basic Statistics for Datasets
- Low Order Moments
- Quantiles
- Order Statistics

Correlation & Dependence
- Cosine Distance
- Correlation Distance
- Variance-Covariance Matrix

Matrix Factorizations
- SVD
- QR
- Cholesky

Dimensionality Reduction
- PCA
- Association Rule Mining (Apriori)
- Optimization Solvers (SGD, AdaGrad, lBFGS)

Outlier Detection
- Univariate
- Multivariate
- Math Functions (exp, log,...)

Optimization Notice
Copyright © 2019, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.
Algorithms & Machine Learning

Supervised Learning
- Regression
- Logistic Regression
- Ridge Regression
- Linear Regression
- Decision Forest
- Decision Tree
- Boosting (Ada, Brown, Logit)
- Naïve Bayes
- k-NN
- Support Vector Machine
- Collaborative Filtering
- Alternating Least Squares

Unsupervised Learning
- K-Means Clustering
- EM for GMM
- Alternating Least Squares

Neural Networks
- Neural Networks

Intel® Data Analytics Acceleration Library
Analyze & Tune Application Performance & Scalability with Intel® VTune™ Amplifier—Performance Profiler

Save Time Optimizing Code

- Accurately profile C, C++, Fortran*, Python*, Go*, Java*, or any mix
- Optimize CPU, threading, memory, cache, storage & more
- Save time: rich analysis leads to insight

What’s New in 2019 Release (partial list)

- Enhanced Application Performance Snapshot: Focus on useful data with new data selection & pause/resume options (Linux*)
- Analyze CPU utilization of physical cores
- Improved JIT profiling for server-side/cloud applications
- A more accessible user interface provides a simplified profiling workflow

Learn More: software.intel.com/intel-vtune-amplifier-xe
Rich Set of Profiling Capabilities for Multiple Markets

Intel® VTune Amplifier

- **Single Thread**: Optimize single-threaded performance.
- **Multithreaded**: Effectively use all available cores.
- **System**: See a system-level view of application performance.
- **Media & OpenCL™ Applications**: Deliver high-performance image and video processing pipelines.
- **HPC & Cloud**: Access specialized, in-depth analyses for HPC and cloud computing.
- **Memory & Storage Management**: Diagnose memory, storage, and data plane bottlenecks.
- **Analyze & Filter Data**: Mine data for answers.
- **Environment**: Fits your environment and workflow.
What’s New for 2019?

Intel® VTune Amplifier

New, Simplified Setup, More Intelligible Results

New Platform Profiler – Longer Data Collection
- Find hardware configuration issues
- Identify poorly tuned applications

Smarter, Faster Application Performance Snapshot
- Smarter: CPU utilization analysis of physical cores
- Faster: Lower overhead, data selection, pause/resume

Added Cloud, Container & Linux .NET Support
- JIT profiling on LLVM* or HHVM PHP servers
- Java* analysis on OpenJDK 9 and Oracle* JDK 9
- .NET* support on Linux* plus Hyper-V* support

SPDK & DPDK I/O Analysis - Measure “Empty” Polling Cycles

Balance CPU/FPGA Loading

Additional Embedded OSs & Environments
Better, Faster Application Performance Snapshot

Intel® VTune™ Amplifier

Better Answers
- CPU utilization analysis of physical cores

Less Overhead
- Lower MPI trace overhead & faster result processing
- New data selection & pause/resume let you focus on useful data

Easier to Use
- Visualize rank-to-rank & node-to-node MPI communications
- Easily configure profiling for Intel® Trace Analyzer & Collector
Tune Workloads & System Configuration

Intel® VTune Amplifier

Finds
- Configuration issues
- Poorly tuned software

Target Users
- Infrastructure Architects
- Software Architects & QA

Performance Metrics
- Extended capture (minutes to hours)
- Low overhead – coarse grain metrics
- Sampling OS & hardware performance counters
- RESTful API for easy analysis by scripts

Timelines & Histograms

Core to Core Comparisons

Server Topology Overview
Modernize Your Code with Intel® Advisor
Optimize Vectorization, Prototype Threading, Create & Analyze Flow Graphs

Modern Performant Code
- Vectorized (uses Intel® AVX-512/AVX2)
- Efficient memory access
- Threaded

Capabilities
- Adds & optimizes vectorization
- Analyzes memory patterns
- Quickly prototypes threading

New for 2019 Release (partial list)
- Enhanced hierarchical roofline analysis
- Shareable HTML roofline
- Flow graph analysis

Performance Increases Scale with Each New Hardware Generation

‘Automatic’ Vectorization is Not Enough
Explicit pragmas and optimization are often required

2010
Intel® Xeon® Processor X5680
codenamed Westmere

130x

2012
Intel Xeon Processor E5-2600
codenamed Sandy Bridge

2013
Intel Xeon Processor E5-2600 v2
codenamed Ivy Bridge

2014
Intel Xeon Processor E5-2600 v3
codenamed Haswell

2016
Intel Xeon Processor E5-2600 v4
codenamed Broadwell

2017
Intel® Xeon® Platinum Processor 81xx
codenamed Skylake Server

Optimization Notice
Copyright © 2019, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.


Benchmark: Binomial Options Pricing Model
Performance results are based on testing as of August 2017 and may not reflect all publicly available security updates. See configuration disclosure for details. No product can be absolutely secure. For more complete information about performance and benchmark results, visit www.intel.com/benchmarks. See Vectorize & Thread or Performance Dies Configurations for 2010-2017 Benchmarks in Backup. Testing by Intel as of August 2017.

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, & SSSE3 instruction sets & 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
‘Automatic’ Vectorization is Often Not Enough
A good compiler can still benefit greatly from vectorization optimization—Intel® Advisor

Compiler will not always vectorize
- With Intel® Advisor, check for Loop Carried Dependencies
- All clear? Force vectorization. C++ use: pragma simd, Fortran use: SIMD directive

Not all vectorization is efficient vectorization
- Stride of 1 is more cache efficient than stride of 2 & greater - use Advisor to Analyze
- Consider data layout changes Intel® SIMD Data Layout Templates can help

Benchmarks (prior slide) did not all ‘auto vectorize.’ Compiler directives were used to force vectorization & get more performance.

Arrays of structures are great for intuitively organizing data, but less efficient than structures of arrays. Use SIMD Data Layout Templates to map data into a more efficient layout for vectorization.
Get Breakthrough Vectorization Performance
Intel® Advisor—Vectorization Advisor

Faster Vectorization Optimization
▪ Vectorize where it will pay off most
▪ Quickly ID what is blocking vectorization
▪ Tips for effective vectorization
▪ Safely force compiler vectorization
▪ Optimize memory stride

Data & Guidance You Need
▪ Compiler diagnostics + Performance Data + SIMD efficiency
▪ Detect problems & recommend fixes
▪ Loop-Carried Dependency Analysis
▪ Memory Access Patterns Analysis

Optimize for Intel® Advanced Vector Extensions 512 (Intel® AVX-512) with or without access to Intel AVX-512 hardware
Find Effective Optimization Strategies
Intel® Advisor—Cache-aware Roofline Analysis

Roofline Performance Insights

- Highlights poor performing loops
- Shows performance ‘headroom’ for each loop
  - Which can be improved
  - Which are worth improving
- Shows likely causes of bottlenecks
- Suggests next optimization steps

“I am enthusiastic about the new “integrated roofline” in Intel® Advisor. It is now possible to proceed with a step-by-step approach with the difficult question of memory transfers optimization & vectorization which is of major importance.”

Nicolas Alferez, Software Architect
Onera – The French Aerospace Lab
Visualize Parallelism—Interactively Build, Validate & Analyze Algorithms

Intel® Advisor—Flow Graph Analyzer (FGA)

- Visually generate code stubs
- Generate parallel C++ programs
- Click & zoom through your algorithm’s nodes & edges to understand parallel data & program flow
- Analyze load balancing, concurrency, & other parallel attributes to fine tune your program

Use Threading Building Blocks or OpenMP* 5 (draft) OMPT APIs
Debug Memory & Threading with Intel® Inspector
Find & Debug Memory Leaks, Corruption, Data Races, Deadlocks

Correctness Tools Increase ROI by 12%-21%¹
- Errors found earlier are less expensive to fix
- Races & deadlocks not easily reproduced
- Memory errors are hard to find without a tool

Debugger Integration Speeds Diagnosis
- Breakpoint set just before the problem
- Examine variables and threads with the debugger

What’s New in 2019 Release
Find Persistent Memory Errors
- Missing / redundant cache flushes
- Missing store fences
- Out-of-order persistent memory stores
- PMDK transaction redo logging errors

Learn More: intel.ly/inspector-xe

INTEL® PARALLEL STUDIO XE COMPONENT TOOLS

BUILD
Intel® C++ Compiler
Intel® Fortran Compiler
Intel® Distribution for Python*
Intel® Math Kernel Library
Intel® Integrated Performance Primitives
Intel® Threading Building Blocks
Intel® Data Analytics Acceleration Library
Included in Composer Edition

ANALYZE
Intel® VTune™ Amplifier
Intel® Advisor
Intel® Inspector
Part of the Professional Edition

SCALE
Intel® MPI Library
Intel® Trace Analyzer & Collector
Intel® Cluster Checker
Part of the Cluster Edition
Standards Based Optimized MPI Library for Distributed Computing

- Built on open source MPICH Implementation
- Tuned for low latency, high bandwidth & scalability
- Multi-fabric support for flexibility in deployment

What’s New in 2019 Release

- New MPI code base- MPI-CH4 (on the path to Exascale & beyond)
- Greater scalability & shortened CPU paths
- Superior MPI Multi-threaded performance
- Supports the latest Intel® Xeon® Scalable processor

Learn More: software.intel.com/intel-mpi-library
Intel® MPI Library Features

Optimized MPI Application Performance
- Application-specific tuning
- Automatic tuning
- Support for Intel® Omni-Path Architecture Fabric

Multi-vendor Interoperability & Lower Latency
- Industry leading latency
- Performance optimized support for the fabric capabilities through OpenFabrics* (OFI)

Faster MPI Communication
- Optimized collectives

Sustainable Scalability
- Native InfiniBand* interface support allows for lower latencies, higher bandwidth, and reduced memory requirements

More Robust MPI Applications
- Seamless interoperability with Intel® Trace Analyzer & Collector

Applications
- CFD
- Crash
- Climate
- OCD
- BIO
- Other...

Develop applications for one fabric

Intel® MPI Library
Select interconnect fabric at runtime

TCP/IP
Omni-Path
InfiniBand
iWarp
Shared Memory
...Other Networks
Fabrics

Achieve optimized MPI performance

Cluster

Intel® MPI Library = 1 library to develop, maintain & test for multiple fabrics
Profile & Analyze High Performance MPI Applications

Intel® Trace Analyzer & Collector

Powerful Profiler, Analysis & Visualization Tool for MPI Applications

- Low overhead for accurate profiling, analysis & correctness checking
- Easily visualize process interactions, hotspots & load balancing for tuning & optimization
- Workflow flexibility: Compile, Link or Run

What's New in 2019 Release

- Minor updates & enhancements
- Supports the latest Intel® Xeon® Scalable processors

Learn More: software.intel.com/intel-trace-analyzer
Efficiently Profile MPI Applications
Intel® Trace Analyzer & Collector

Helps Developers
- Visualize & understand parallel application behavior
- Evaluate profiling statistics & load balancing
- Identify communication hotspots

Features
- Event-based approach
- Low overhead
- Excellent scalability
- Powerful aggregation & filtering functions
- Idealizer
- Scalable
Use an Extensive Diagnostic Toolset for High Performance Compute Clusters—Intel® Cluster Checker (for Linux*)

Ensure Cluster Systems Health

- Expert system approach providing cluster systems expertise - verifies system health: find issues, offers suggested actions
- Provides extensible framework, API for integrated support
- Check 100+ characteristics that may affect operation & performance – improve uptime & productivity

New in 2019 Release: Output & Features Improve Usability & Capabilities

- Simplified execution with a **single command**
- **New output** format with overall summary
  - Simplified issue assessment for ‘CRITICAL’, ‘WARNING’, or ‘INFORMATION’
  - Extended output to logfile with details on issue, diagnoses, observations
- Added **auto-node discovery** when using Slurm*
- Cluster State **2 snapshot comparison** identifies changes
- And more…

For application developers, cluster architects & users, & system administrators
Functionality, Uniformity, & Performance Tests
Intel® Cluster Checker

Comprehensive pre-packed cluster systems expertise out-of-the-box

✔ Suitable for HPC experts & those new to HPC
✔ Tests can be executed in selected groups on any subset of nodes

<table>
<thead>
<tr>
<th>Intel® Cluster Checker</th>
<th>Functionality Tests</th>
<th>Uniformity Tests</th>
<th>Performance Tests</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>System-level</strong></td>
<td>▪ Node</td>
<td>▪ CPUs</td>
<td><strong>Benchmarks</strong></td>
</tr>
<tr>
<td></td>
<td>▪ Connectivity</td>
<td>▪ Memory</td>
<td>▪ DGEMM</td>
</tr>
<tr>
<td></td>
<td>▪ Cluster</td>
<td>▪ Interconnect</td>
<td>▪ HPCG</td>
</tr>
<tr>
<td><strong>Validation</strong></td>
<td>▪ Application platform compliance</td>
<td>▪ Disks</td>
<td>▪ HPL</td>
</tr>
<tr>
<td></td>
<td>▪ Solution compliance</td>
<td></td>
<td>▪ Intel® MPI Benchmarks</td>
</tr>
<tr>
<td><strong>Hardware</strong></td>
<td>▪ Installed packages &amp; versions</td>
<td></td>
<td>▪ IOzone</td>
</tr>
<tr>
<td><strong>Software</strong></td>
<td>▪ Numerous kernel &amp; BIOS settings</td>
<td></td>
<td>▪ STREAM</td>
</tr>
</tbody>
</table>

API Available for Integration

✔ Get compact reports, find problems, validate status
Speaker – the speaker notes are important for this presentation. Be sure to read them.

WHICH TOOL SHOULD I USE?
Optimizing Performance on Parallel Hardware
Intel® Parallel Studio XE—It’s an iterative process...

- **Cluster Scalable?**
  - Y: Tune MPI
  - N: Effective threading?
    - Y: Vectorize
    - N: Thread

- **Memory Bandwidth Sensitive?**
  - Y: Optimize Bandwidth
  - N: Ignore if you are not targeting clusters.

- **Possible System Configuration Issues?**
  - Y: Intel® Cluster Checker
  - N: Effective threading?
Performance Analysis Tools for Diagnosis
Intel® Parallel Studio

Cluster Scalable?

Tune MPI

Intel® Trace Analyzer & Collector
Intel® MPI Tuner

Effective threading?

Vectorize

Memory Bandwidth Sensitive?

Optimize Bandwidth

Thread

Intel® VTune™ Amplifier

Inte® VTune™ Amplifier

Intel® Advisor

Application Performance Snapshot

Optimization Notice
Copyright © 2019, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.
Tools for High Performance Implementation
Intel® Parallel Studio XE

Cluster Scalable?
Y → Vectorize
N → Tune MPI

Effective threading?
Y → Vectorize
N → Thread

Memory Bandwidth Sensitive?
Y → Optimize Bandwidth
N → Tune MPI

Possible System Configuration Issues?
Y → Intel® Cluster Checker
N → Intel® Compiler

Intel® MPI Library
Intel® Math Kernel Library
Intel® Integrated Performance Primitives – Media & Data Library
Intel® Data Analytics Acceleration Library
Intel® OpenMP*

Threading Building Blocks – Threading Library

Intel® Compiler
Intel® MPI Library
Intel® Math Kernel Library
Intel® Integrated Performance Primitives – Media & Data Library
Intel® Data Analytics Acceleration Library
Intel® OpenMP*

Threading Building Blocks – Threading Library

Intel® Cluster Checker
INTRODUCTION TO
MACHINE LEARNING AND DEEP LEARNING
**Artificial Intelligence**

is the ability of machines to learn from experience, without explicit programming, in order to perform cognitive functions associated with the human mind.
**MACHINE VS. DEEP LEARNING**

**MACHINE LEARNING**
How do you engineer the best features?

\[ N \times N \]

- Roundness of face
- Dist between eyes
- Nose width
- Eye socket depth
- Cheek bone structure
- Jaw line length
- ...etc.

**DEEP LEARNING**
How do you guide the model to find the best features?

\[ N \times N \]

**CLASSIFIER ALGORITHM**
- SVM
- Random Forest
- Naïve Bayes
- Decision Trees
- Logistic Regression
- Ensemble methods

**NEURAL NETWORK**

**Arjun**
DEEP LEARNING BREAKTHROUGHS

Machines able to meet or exceed human image & speech recognition

**IMAGE RECOGNITION**

- 2010:
  - Error: 30%
  - Human: 97%
  - Using Deep Learning: 8%

- Present:
  - Error: 0%
  - Using Deep Learning: 8%

**SPEECH RECOGNITION**

- 2000:
  - Error: 30%
  - Human: 97%
  - Using Deep Learning: 8%

- Present:
  - Error: 0%
  - Using Deep Learning: 8%

**Examples**

- TUMOR DETECTION
- DOCUMENT SORTING
- OIL & GAS SEARCH
- VOICE ASSISTANT
- DEFECT DETECTION
- GENOME SEQUENCING

DEEP LEARNING BASICS

TRAINING

- Human
- Bicycle
- Strawberry

Lots of labeled data!

Model weights

Forward "Strawberry"

Backward

Error

IFERENCE

"Bicycle"?

Forward "Bicycle"

"Bicycle"?

DID YOU KNOW?

Training with a large data set AND deep (many layered) neural network often leads to the highest accuracy inference.

Optimization Notice
Copyright © 2019, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.
Deliver significant AI performance with hardware and software optimizations on Intel® Xeon® Scalable Family

**INFORMATION THROUGHPUT**

- **Up to 277×**
  - Intel® Xeon® Platinum 8180 Processor
  - higher optimized Caffe GoogleNet v1 with Intel® MKL
  - inference throughput compared to Intel® Xeon® Processor E5-2699 v3 with BVLC-Caffe

**TRAINING THROUGHPUT**

- **Up to 241×**
  - Intel® Xeon® Platinum 8180 Processor
  - higher Intel Optimized Caffe AlexNet with Intel® MKL
  - training throughput compared to Intel® Xeon® Processor E5-2699 v3 with BVLC-Caffe

Inference and training throughput uses FP32 instructions

---

1. The benchmark results may need to be revised as additional testing is conducted. The results depend on the specific platform configurations and workloads utilized in the testing, and may not be applicable to any particular user’s components, computer system or workloads. The results are not necessarily representative of other benchmarks and other benchmark results may show greater or lesser impact from mitigations. 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: [http://www.intel.com/performance](http://www.intel.com/performance) 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: [http://www.intel.com/performance](http://www.intel.com/performance) Source: Intel measured as of June 2018. Configurations: See slide 4.
WHAT IS MACHINE LEARNING?

Applying Algorithms to observed data and make predictions based on data.
Supervised Learning

We train the model. We feed the model with correct answers. Model Learns and finally predicts.

We feed the model with “ground truth”.
Unsupervised Learning

Data is given to the model. Right answers are not provided to the model. The model makes sense of the data given to it.

Can teach you something you were probably not aware of in the given dataset.
Types of Supervised and Unsupervised learning

**SUPERVISED**
- Classification
- Regression

**UNSUPERVISED**
- Clustering
- Recommendation
Regression
Predict a real numeric value for an entity with a given set of features.

Property Attributes
- Price
- Address
- Type
- Age
- Parking
- School
- Transit
- Total sqft
- Lot Size
- Bathrooms
- Bedrooms
- Yard
- Pool
- Fireplace

Linear Regression Model

$ \text{sqft}$
CLUSTERING
Group entities with similar features

MARKET SEGMENTATION

Play time in hours

Serious Gamers

Causal Gamers

No Gamers

Age

10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90

CLUSTERING
Group entities with similar features
What is the Issue with Linear Classifiers We Have Learnt So Far?

Linear functions can solve the AND problem.

<table>
<thead>
<tr>
<th>X1</th>
<th>X2</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
What is the Issue with Linear Classifiers We Have Learnt So Far?

Linear functions can solve the OR problem.

<table>
<thead>
<tr>
<th>X1</th>
<th>X2</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Why Deep Learning – What is wrong with Linear Classifiers?

XOR
The counter example to all models
We need non-linear functions

<table>
<thead>
<tr>
<th>X1</th>
<th>X2</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

We Need Layers Usually Lots with Non-linear Transformations

**XOR = (X1 and not X2) OR (Not X1 and X2)**

<table>
<thead>
<tr>
<th>X1</th>
<th>X2</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Threshold to 0 or 1

\[
(1 \times 1) + (0 \times 1) < 1.5 = 0
\]

\[
(1 \times 1) + (0 \times -2) + (0 \times 1) = 1 > 0.5 = 1
\]
We Need Layers Usually Lots with Non-linear Transformations

XOR = (X1 and not X2) OR (Not X1 and X2)

Input

1

1 x 1

+1

1 x 1

1 x 1

2 > 1.5

1.5

-2

1 x -2

1 x 1

+1

1 x 1

(1 x 1) + (1 x 1) = 2 > 1.5

(1x1) + (1x -2) + (1x1) = 0 < .5 =0

Output

Threshold to 0 or 1

X1 | X2 | y
---|---|---
0  | 0  | 0
0  | 1  | 1
1  | 0  | 1
1  | 1  | 0

Optimization Notice
Copyright © 2019, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.
This is a brewing domain called Deep Learning

In the machine learning world, we use neural networks. The idea comes from biology. Each layer learns something.

“Deep learning is a set of algorithms in machine learning that attempt to model high-level abstractions in data by using architectures composed of multiple non-linear transformations.”

- Wikipedia*
Motivation for Neural Nets

- Use biology as inspiration for mathematical model
- Get signals from previous neurons
- Generate signals (or not) according to inputs
- Pass signals on to next neurons
- By layering many neurons, can create complex model
Each Layer Learns Something

Layer 1

Faces

Cars

Elephants

Chairs

Layer 2

Layer N

Fully Connected layer

Prediction

Elephant
THE BASICS OF BUILDING A NEURAL NETWORK
Basic Neuron Visualization

$$z = x_1w_1 + x_2w_2 + x_3w_3 + b$$

Activation Function

$$f(z)$$
Types of Activation Functions

- **Sigmoid function**
  - Smooth transition in output between (0,1)

- **Tanh function**
  - Smooth transition in output between (-1,1)

- **ReLU function**
  - $f(x) = \max(x,0)$

- **Step function**
  - $f(x) = (0,1)$
Why Neural Nets?

- Why not just use a single neuron? Why do we need a larger network?
- A single neuron (like logistic regression) only permits a linear decision boundary.
- Most real-world problems are considerably more complicated!
Feedforward Neural Network

\[ x_1 \rightarrow \sigma \rightarrow \sigma \rightarrow \hat{y}_1 \]
\[ x_2 \rightarrow \sigma \rightarrow \sigma \rightarrow \hat{y}_2 \]
\[ x_3 \rightarrow \sigma \rightarrow \sigma \rightarrow \hat{y}_3 \]
Weights
Weights (Represented by Matrices)
Input Layer
Hidden Layers

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]
Net Input (Sum of Weighted Inputs, Before Activation Function)
Activations (Output of Neurons to Next Layer)
Output Layer

\[
\begin{align*}
\sigma & \quad \sigma \\
\sigma & \quad \sigma \\
\sigma & \quad \sigma \\
\end{align*}
\]

\[
\begin{align*}
\mathbf{x}_1 & \quad \mathbf{y}_1 \\
\mathbf{x}_2 & \quad \mathbf{y}_2 \\
\mathbf{x}_3 & \quad \mathbf{y}_3 \\
\end{align*}
\]
How to Train a Neural Net?

- Put in Training inputs, get the output
- Compare output to correct answers: Look at loss function $J$
- Adjust and repeat!
- Backpropagation tells us how to make a single adjustment using calculus.
Convolutional Neural Nets

Primary Ideas behind Convolutional Neural Networks:

- Let the Neural Network learn which kernels are most useful
- Use same set of kernels across entire image (translation invariance)
- Reduces number of parameters and “variance” (from bias-variance point of view)
- Can Think of Kernels as “Local Feature Detectors”

<table>
<thead>
<tr>
<th>Vertical Line Detector</th>
<th>Horizontal Line Detector</th>
<th>Corner Detector</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1 1 -1</td>
<td>-1 -1 -1</td>
<td>-1 -1 -1</td>
</tr>
<tr>
<td>-1 1 -1</td>
<td>1 1 1</td>
<td>-1 1 1</td>
</tr>
<tr>
<td>-1 1 -1</td>
<td>-1 -1 -1</td>
<td>-1 1 1</td>
</tr>
</tbody>
</table>
CNN for Digit Recognition

Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.
Convolutional Neural Networks (CNN) for Image Recognition

Convolution

- Each element in the output is the result of a dot product between two vectors

Source: http://cs231n.github.io/
Pooling: Max-pool

- For each distinct patch, represent it by the maximum
- 2x2 Max-Pool shown below

```
2  1  0  -1
-3  8  2  5
1 -1  3  4
0  1  1 -2
```

```
8  5
1  4
```
Differences Between CNN and Fully Connected Networks

**Convolutional Neural Network**
- Each neuron connected to a small set of nearby neurons in the previous layer
- Uses same set of weights for each neuron
- Ideal for spatial feature recognition, Ex: Image recognition
- Cheaper on resources due to fewer connections

**Fully Connected Neural Networks**
- Each neuron is connected to every neuron in the previous layer
- Every connection has a separate weight
- Not optimal for detecting features
- Computationally intensive – heavy memory usage
CLASSIC ML TOOLS
INTEL PERFORMANCE LIBRARIES

INTEL® MATH KERNEL LIBRARY (MKL)
INTEL® DATA ANALYTICS ACCELERATION LIBRARY (DAAL)
INTEL® MATH KERNEL LIBRARY
Intel® MKL
Faster, Scalable Code with Intel® Math Kernel Library

- Speeds computations for scientific, engineering, financial and machine learning applications by providing highly optimized, threaded, and vectorized math functions
- Provides key functionality for dense and sparse linear algebra (BLAS, LAPACK, PARDISO), FFTs, vector math, summary statistics, deep learning, splines and more
- Dispatches optimized code for each processor automatically without the need to branch code
- Optimized for single core vectorization and cache utilization
- Automatic parallelism for multi-core and many-core
- Scales from core to clusters
- Available at no cost and royalty free
- Great performance with minimal effort!

Available as standalone or as a part of Intel® Parallel Studio XE and Intel® System Studio

INTEL® MKL OFFERS...

- DENSE AND SPARSE LINEAR ALGEBRA
- FAST FOURIER TRANSFORMS
- VECTOR MATH
- VECTOR RNGS
- FAST POISSON SOLVER
- AND MORE!

Intel® Architecture Platforms

Operating System: Windows*, Linux*, MacOS*¹

¹ Available only in Intel® Parallel Studio Composer Edition.

Other names and brands may be claimed as the property of others.
### Automatic Dispatching to Tuned ISA-specific Code Paths

More cores → More Threads → Wider vectors

<table>
<thead>
<tr>
<th>Product Family</th>
<th>Up to Core(s)</th>
<th>Up to Threads</th>
<th>SIMD Width</th>
<th>Vector ISA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel® Xeon® Processor 64-bit</td>
<td>1</td>
<td>2</td>
<td>128</td>
<td>Intel® SSE3</td>
</tr>
<tr>
<td>Intel® Xeon® Processor 5100 series</td>
<td>2</td>
<td>2</td>
<td>128</td>
<td>Intel® SSE3</td>
</tr>
<tr>
<td>Intel® Xeon® Processor 5500 series</td>
<td>4</td>
<td>8</td>
<td>128</td>
<td>Intel® SSE-4.1</td>
</tr>
<tr>
<td>Intel® Xeon® Processor 5600 series</td>
<td>6</td>
<td>12</td>
<td>128</td>
<td>Intel® SSE 4.2</td>
</tr>
<tr>
<td>Intel® Xeon® Processor E5-2600 v2 series</td>
<td>12</td>
<td>24</td>
<td>256</td>
<td>Intel® AVX</td>
</tr>
<tr>
<td>Intel® Xeon® Processor E5-2600 v3 series</td>
<td>18-22</td>
<td>36-44</td>
<td>256</td>
<td>Intel® AVX2</td>
</tr>
<tr>
<td>Intel® Xeon® Processor E5-2600 v4 series</td>
<td>28</td>
<td>56</td>
<td>512</td>
<td>Intel® AVX-512</td>
</tr>
</tbody>
</table>

Intel® Xeon Phi™ x200 Processor (KNL)

- More cores
- More Threads
- Wider vectors

1. Product specification for launched and shipped products available on ark.intel.com.
What’s New for Intel® MKL 2019?

Just-In-Time Fast Small Matrix Multiplication

• Improved speed of S/DGEMM for Intel® AVX2 and Intel® AVX-512 with JIT capabilities

Sparse QR Solvers

• Solve sparse linear systems, sparse linear least squares problems, eigenvalue problems, rank and null-space determination, and others

Generate Random Numbers for Multinomial Experiments

• Highly optimized multinomial random number generator for finance, geological and biological applications
Performance Benefits for the latest Intel Architectures

DGEMM, SGEMM Optimized by Intel® Math Kernel Library
2019 Gold for Intel® Xeon® Platinum Processor

The benchmark results reported above may need to be revised as additional testing is conducted. The results depend on the specific platform configurations and workloads utilized in the testing, and may not be applicable to any particular user's components, computer system or workloads. The results are not necessarily representative of other benchmarks and other benchmark results may show greater or lesser impact from mitigations.

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.

Configuration: Intel® Xeon® Platinum 8180 H0 205W 2x28@2.5GHz 192GB DDR4-2666
Benchmark Source: Intel® Corporation.

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.
## Intel® MKL 11.0 - 2018 Noteworthy Enhancements

- Conditional Numerical Reproducibility (CNR)
- Intel® Threading Building Blocks (TBB) Composability
- Intel® Optimized High Performance Conjugate Gradient (HPCD) Benchmark
- Small GEMM Enhancements (Direct Call) and Batch
- Compact GEMM and LAPACK Support
- Sparse BLAS Inspector-Executor API
- Extended Cluster Support (MPI wrappers and macOS*)
- Parallel Direct Sparse Solver for Clusters
- Extended Eigensolvers
What's Inside Intel® MKL

**Linear Algebra**
- BLAS
- LAPACK
- ScaLAPACK
- Sparse BLAS
- Iterative sparse solvers
- PARDISO
- Cluster Sparse Solver

**FFT**
- Multidimensional
- FFTW interfaces
- Cluster FFT

**Vector RNGs**
- Congruential
- Wichmann-Hill
- Mersenne Twister
- Sobol
- Neiderreiter
- Non-deterministic

**Summary Statistics**
- Kurtosis
- Variation coefficient
- Order statistics
- Min/max
- Variance-covariance

**Vector Math**
- Trigonometric
- Hyperbolic
- Exponential
- Log
- Power
- Root
- Fast Poisson Solver
- Splines
- Interpolation
- Trust Region
- Fast Poisson Solver

*Other names and brands may be claimed as the property of others.*
# Intel® MKL BLAS (Basic Linear Algebra Subprograms)

<table>
<thead>
<tr>
<th>De-facto Standard APIs since the 1980s</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>100s of Basic Linear Algebra Functions</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

| Precisions Available                        | Real – Single and Double |
|                                           | Complex - Single and Double |

| BLAS-like Extensions                       | Direct Call, Batched, Packed and Compact |

# Intel® MKL LAPACK (Linear Algebra PACKage)

## De-facto Standard APIs since the 1990s

<table>
<thead>
<tr>
<th>1000s of Linear Algebra Functions</th>
<th>Matrix factorizations - LU, Cholesky, QR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Solving systems of linear equations</td>
</tr>
<tr>
<td></td>
<td>Condition number estimates</td>
</tr>
<tr>
<td></td>
<td>Symmetric and non-symmetric eigenvalue problems</td>
</tr>
<tr>
<td></td>
<td>Singular value decomposition</td>
</tr>
<tr>
<td></td>
<td>and many more ...</td>
</tr>
</tbody>
</table>

## Precisions Available

- Real – Single and Double,
- Complex – Single and Double

## Reference Implementation

- [http://netlib.org/lapack/](http://netlib.org/lapack/)
# Intel® MKL Fast Fourier Transforms (FFTs)

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FFTW Interfaces support</strong></td>
<td>C, C++ and FORTRAN source code wrappers provided for FFTW2 and FFTW3. FFTW3 wrappers are already built into the library.</td>
</tr>
<tr>
<td><strong>Cluster FFT</strong></td>
<td>Perform Fast Fourier Transforms on a cluster</td>
</tr>
<tr>
<td></td>
<td>Interface similar to DFTI</td>
</tr>
<tr>
<td></td>
<td>Multiple MPIS supported</td>
</tr>
<tr>
<td><strong>Parallelization</strong></td>
<td>Thread safe with automatic thread selection</td>
</tr>
<tr>
<td><strong>Storage Formats</strong></td>
<td>Multiple storage formats such as CCS, PACK and Perm</td>
</tr>
<tr>
<td><strong>Batch support</strong></td>
<td>Perform multiple transforms in a single call</td>
</tr>
<tr>
<td><strong>Additional Features</strong></td>
<td>Perform FFTs on partial images</td>
</tr>
<tr>
<td></td>
<td>Padding added for better performance</td>
</tr>
<tr>
<td></td>
<td>Transform combined with transposition</td>
</tr>
<tr>
<td></td>
<td>Mixed-language usage supported</td>
</tr>
</tbody>
</table>
## Intel® MKL Vector Math

### Example:

\[ y(i) = e^{x(i)} \text{ for } i = 1 \text{ to } n \]

### Broad Function Support

- Basic Operations – add, sub, mult, div, sqrt
- Trigonometric – sin, cos, tan, asin, acos, atan
- Exponential – exp, pow, log, log10, log2
- Hyperbolic – sinh, cosh, tanh
- Rounding – ceil, floor, round
- And many more

### Precisions Available

- Real – Single and Double
- Complex - Single and Double

### Accuracy Modes

- High - almost correctly rounded
- Low - last 2 bits in error
- Enhanced Performance - 1/2 the bits correct
<table>
<thead>
<tr>
<th>Random Number Generators (RNGs)</th>
<th>Pseudorandom, quasi-random and non-deterministic random number generators with continuous and discrete distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary Statistics</td>
<td>Parallelized algorithms to compute basic statistical estimates for single and double precision multi-dimensional datasets</td>
</tr>
<tr>
<td>Convolution and Correlation</td>
<td>Linear convolution and correlation transforms for single and double precision real and complex data</td>
</tr>
<tr>
<td><strong>Intel® MKL Sparse Solvers</strong></td>
<td></td>
</tr>
<tr>
<td>-----------------------------</td>
<td></td>
</tr>
<tr>
<td><strong>PARDISO - Parallel Direct Sparse Solver</strong></td>
<td></td>
</tr>
<tr>
<td>Factor and solve $Ax = b$ using a parallel shared memory $LU$, $LDL$, or $LL^T$ factorization</td>
<td></td>
</tr>
<tr>
<td>Supports a wide variety of matrix types including real, complex, symmetric, indefinite, ...</td>
<td></td>
</tr>
<tr>
<td>Includes out-of-core support for very large matrix sizes</td>
<td></td>
</tr>
<tr>
<td><strong>Parallel Direct Sparse Solver for Clusters</strong></td>
<td></td>
</tr>
<tr>
<td>Factor and solve $Ax = b$ using a parallel distributed memory $LU$, $LDL$, or $LL^T$ factorization</td>
<td></td>
</tr>
<tr>
<td>Supports a wide variety of matrix types (real, complex, symmetric, indefinite, ... )</td>
<td></td>
</tr>
<tr>
<td>Supports $A$ stored in 3-array CSR3 or BCSR3 formats</td>
<td></td>
</tr>
<tr>
<td><strong>DSS – Simplified PARDISO Interface</strong></td>
<td></td>
</tr>
<tr>
<td>An alternative, simplified interface to PARDISO</td>
<td></td>
</tr>
<tr>
<td><strong>ISS – Iterative Sparse Solvers</strong></td>
<td></td>
</tr>
<tr>
<td>Conjugate Gradient (CG) solver for symmetric positive definite systems</td>
<td></td>
</tr>
<tr>
<td>Generalized Minimal Residual (GMRes) for non-symmetric indefinite systems</td>
<td></td>
</tr>
<tr>
<td>Rely on Reverse Communication Interface (RCI) for matrix vector multiply</td>
<td></td>
</tr>
</tbody>
</table>
### Intel® MKL General Components

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sparse BLAS</td>
<td>NIST-like and inspector execute interfaces</td>
</tr>
<tr>
<td>Data Fitting</td>
<td>1D linear, quadratic, cubic, step-wise and user-defined splines, spline-based interpolation and extrapolation</td>
</tr>
<tr>
<td>Partial Differential Equations</td>
<td>Helmhotz, Poisson, and Laplace equations</td>
</tr>
<tr>
<td>Optimization</td>
<td>Trust-region solvers for nonlinear least square problems with and without constraints</td>
</tr>
<tr>
<td>Service Functions</td>
<td>Threading controls, Memory management, Numerical reproducibility</td>
</tr>
</tbody>
</table>
## Intel® MKL Summary

<table>
<thead>
<tr>
<th>Boosts application performance with minimal effort</th>
<th>feature set is robust and growing provides scaling from the core, to multicore, to manycore, and to clusters automatic dispatching matches the executed code to the underlying processor future processor optimizations included well before processors ship</th>
</tr>
</thead>
<tbody>
<tr>
<td>Showcases the world’s fastest supercomputers¹</td>
<td>Intel® Distribution for LINPACK* Benchmark Intel® Optimized High Performance Conjugate Gradient Benchmark</td>
</tr>
</tbody>
</table>

¹http://www.top500.org
## Intel® MKL Resources

<table>
<thead>
<tr>
<th>Resource</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel® MKL Website</td>
<td><a href="https://software.intel.com/en-us/intel-mkl">https://software.intel.com/en-us/intel-mkl</a></td>
</tr>
</tbody>
</table>
INTEL® DATA ANALYTICS ACCELERATION LIBRARY
INTEL® DAAL
Speed-up Machine Learning and Analytics with Intel® Data Analytics Acceleration Library (Intel® DAAL)

Boost Machine Learning & Data Analytics Performance
- Helps applications deliver better predictions faster
- Optimizes data ingestion & algorithmic compute together for highest performance
- Supports offline, streaming & distributed usage models to meet a range of application needs
- Split analytics workloads between edge devices and cloud to optimize overall application throughput

What’s New in the 2019 Release
New Algorithms
- **High performance Logistic Regression**, most widely-used classification algorithm
- **Extended Gradient Boosting Functionality** provides inexact split calculations & algorithm-level computation canceling by user-defined callback for greater flexibility
- **User-defined Data Modification Procedure in CSV & IDBC data sources to implement** a wide range of feature extraction & transformation techniques

Learn More: software.intel.com/daal

Pre-processing | Transformation | Analysis | Modeling | Validation | Decision Making
---|---|---|---|---|---
Decompression, Filtering, Normalization | Aggregation, Dimension Reduction | Summary Statistics Clustering, etc. | Machine Learning (Training) Parameter Estimation Simulation | Hypothesis Testing Model Errors | Forecasting Decision Trees, etc.
Regression

Problems

- A company wants to define the impact of the pricing changes on the number of product sales
- A biologist wants to define the relationships between body size, shape, anatomy and behavior of the organism

Solution: Linear Regression

- A linear model for relationship between features and the response

\[ \hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 + \ldots + \hat{\beta}_N x_N \]

Source: Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani. (2014). An Introduction to Statistical Learning. Springer
Classification

Problems

– An emailing service provider wants to build a spam filter for the customers
– A postal service wants to implement handwritten address interpretation

Solution: Support Vector Machine (SVM)

– Works well for non-linear decision boundary

– Two kernel functions are provided:
  – Linear kernel
  – Gaussian kernel (RBF)

– Multi-class classifier
  – One-vs-One

Source: Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani. (2014). *An Introduction to Statistical Learning*. Springer
Cluster Analysis

Problems

- A news provider wants to group the news with similar headlines in the same section
- Humans with similar genetic pattern are grouped together to identify correlation with a specific disease

Solution: K-Means

- Pick $k$ centroids
- Repeat until converge:
  - Assign data points to the closest centroid
  - Re-calculate centroids as the mean of all points in the current cluster
  - Re-assign data points to the closest centroid

Source: Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani. (2014). An Introduction to Statistical Learning. Springer
Dimensionality Reduction

Problems

- Data scientist wants to visualize a multi-dimensional data set
- A classifier built on the whole data set tends to overfit

Solution: Principal Component Analysis

- Compute eigen decomposition on the correlation matrix
- Apply the largest eigenvectors to compute the largest principal components that can explain most of variance in original data

Source: Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani. (2014). *An Introduction to Statistical Learning*. Springer
Performance Scaling with Intel® Data Analytics Acceleration Library (Intel® DAAL)

Within a CPU Core
- SIMD vectorization: optimized for the latest instruction sets, Intel® AVX2, AVX512...
- Internally relies on sequential Math Kernel Library

Scale to Multicores or Many Cores
- Threading Building Blocks threading

Scale to Cluster
- Distributed processing done by user application (MPI, MapReduce, etc.)
- Intel® DAAL provides
  - Data structures for partial and intermediate results
  - Functions to combine partial or intermediate results into global result
Processing Modes

**Batch Processing**

\[ R = F(D_1, \ldots, D_k) \]

**Online Processing**

\[ S_{i+1} = T(S_i, D_i) \]
\[ R_{i+1} = F(S_{i+1}) \]

**Distributed Processing**

\[ R = F(R_1, \ldots, R_k) \]
Data Transformation & Analysis Algorithms

Intel® Data Analytics Acceleration Library

Basic Statistics for Datasets
- Low Order Moments
- Quantiles
- Order Statistics

Correlation & Dependence
- Cosine Distance
- Correlation Distance
- Variance-Covariance Matrix

Matrix Factorizations
- SVD
- QR
- Cholesky

Dimensionality Reduction
- PCA
- Association Rule Mining (Apriori)
- Optimization Solvers (SGD, AdaGrad, lBFGS)

Outlier Detection
- Univariate
- Multivariate
- Math Functions (exp, log,...)

Optimization Notice
Copyright © 2019, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.
Classification

Problems
An email service provider wants to build a spam filter for the customers
A postal service wants to implement handwritten address interpretation

Solution: Support Vector Machine (SVM)
Works well for non-linear decision boundary

Two kernel functions are provided:
- Linear kernel
- Gaussian kernel (RBF)

Multi-class classifier
- One-vs-One

Source: Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani. (2014). *An Introduction to Statistical Learning*. Springer
Performance Example: Read And Compute
SVM Classification with RBF kernel

Training dataset: CSV file (PCA-preprocessed MNIST, 40 principal components) \( n=42000, p=40 \)

Testing dataset: CSV file (PCA-preprocessed MNIST, 40 principal components) \( n=28000, p=40 \)

![Graph showing performance comparison between Scikit-Learn, Pandas, and pyDAAL for read and compute tasks.](image)

- **Training (sec):**
  - Scikit-Learn, Pandas: 25 seconds
  - pyDAAL: 10 seconds
  - **60% faster CSV read**

- **Prediction (sec):**
  - Scikit-Learn, Pandas: 15 seconds
  - pyDAAL: 2.2 seconds
  - **66x faster**

**Optimization Notice**

Performance improvements are indicative of optimized data handling and computation, showcasing the benefits of using pyDAAL for machine learning tasks.
Projection Methods for Outlier Detection

**Principal Component Analysis (PCA)**

- Computes principal components: the directions of the largest variance, the directions where the data is mostly spread out

**PCA for outlier detection**

- Project new observation on the space of the first $k$ principal components
- Calculate score distance for the projection using first $k$ singular values
- Compare the distance against threshold

![Image](http://i.stack.imgur.com/uYaTv.png)
More Resources
Intel® Data Analytics Acceleration Library (Intel® DAAL)

Download Now

- Free version with Intel® Performance Libraries
- Bundled in Intel® Parallel Studio XE or Intel® System Studio, includes Intel Priority Support

Product Information

- software.intel.com/intel-daal

Getting Started Guides

- software.intel.com/intel-daal-support/training
- Webinars, how-to videos & articles on Intel® Tech.Decoded

View Video: Speed up your machine learning application code, turn data into insight and actionable results with Intel® DAAL and Intel® Distribution for Python*
INTEL® DISTRIBUTION FOR PYTHON 2019
The most popular languages for Data Science

"Python wins the heart of developers across all ages, according to our Love-Hate index. Python is also the most popular language that developers want to learn overall, and a significant share already knows it"

2018 Developer Skills Report

- Python, Java, R are top 3 languages in job postings for data science and machine learning jobs
The most popular ML packages for Python
The most popular ML package for Python

scikit-learn
Machine Learning in Python

- Simple and efficient tools for data mining and data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

Classification
Identifying to which category an object belongs to.

**Applications:** Spam detection, Image recognition.

**Algorithms:** SVM, nearest neighbors, random forest, ...

Regression
Predicting a continuous-valued attribute associated with an object.

**Applications:** Drug response, Stock prices.

**Algorithms:** SVR, ridge regression, Lasso, ...

Clustering
Automatic grouping of similar objects into sets.

**Applications:** Customer segmentation, Grouping experiment outcomes

**Algorithms:** k-Means, spectral clustering, mean-shift, ...

Performance gap between C and Python

Black–Scholes Formula

\[
V_{\text{call}} = S_0 \cdot \text{CDF}(d_1) - e^{-rT} \cdot X \cdot \text{CDF}(d_2)
\]

\[
V_{\text{put}} = e^{-rT} \cdot X \cdot \text{CDF}(-d_2) - S_0 \cdot \text{CDF}(-d_1)
\]

\[
d_1 = \frac{\ln\left(\frac{S_0}{X}\right) + \left(r + \frac{\sigma^2}{2}\right)T}{\sigma \sqrt{T}}
\]

\[
d_2 = \frac{\ln\left(\frac{S_0}{X}\right) + \left(r - \frac{\sigma^2}{2}\right)T}{\sigma \sqrt{T}}
\]
Performance gap between C and Python

Hardware and software efficiency crucial in production (Perf/Watt, etc.)

Efficiency = Parallelism
- Instruction Level Parallelism with effective memory access patterns
- SIMD
- Multi-threading

* Roofline Performance Model  [https://crd.lbl.gov/departments/computer-science/PAR/research/roofline/]
Performance matters at every stage

Prototyping Development cost

1. Pre-Processing
2. Analysis
3. Modeling
4. Model Validation
5. Visualization

High migration costs

1x

Production

1. Pre-processing
2. Model Calibration
3. Decision Making
4. Model Validation
5. Reporting

HPC/Big Data Cluster

Development cost

3-10x and more

*Other names and brands may be claimed as the property of others.
# What's Inside Intel® Distribution for Python

High Performance Python* for Scientific Computing, Data Analytics, Machine Learning

<table>
<thead>
<tr>
<th><strong>FASTER PERFORMANCE</strong></th>
<th><strong>GREATER PRODUCTIVITY</strong></th>
<th><strong>ECOSYSTEM COMPATIBILITY</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance Libraries, Parallelism, Multithreading, Language Extensions</td>
<td>Prebuilt &amp; Accelerated Packages</td>
<td>Supports Python 2.7 &amp; 3.x, conda, pip</td>
</tr>
<tr>
<td>Accelerated NumPy/SciPy/scikit-learn with Intel® MKL¹ &amp; Intel® DAAL²</td>
<td>Prebuilt &amp; optimized packages for numerical computing, machine/deep learning, HPC, &amp; data analytics</td>
<td>Compatible &amp; powered by Anaconda*, supports conda &amp; pip</td>
</tr>
<tr>
<td>Data analytics, machine learning &amp; deep learning with scikit-learn, pyDAAL</td>
<td>Drop in replacement for existing Python - No code changes required</td>
<td>Distribution &amp; individual optimized packages also available via conda, pip YUM/APT, Docker image on DockerHub</td>
</tr>
<tr>
<td>Scale with Numba* &amp; Cython*</td>
<td>Jupyter* notebooks, Matplotlib included</td>
<td>Optimizations upstreamed to main Python trunk</td>
</tr>
<tr>
<td>Includes optimized mpi4py, works with Dask* &amp; PySpark*</td>
<td>Conda build recipes included in packages</td>
<td>Commercial support through Intel® Parallel Studio XE</td>
</tr>
<tr>
<td>Optimized for latest Intel® architecture</td>
<td>Free download &amp; free for all uses including commercial deployment</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Intel® Architecture Platforms</strong></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating System: Windows*, Linux*, MacOS¹*</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

¹Intel® Math Kernel Library
²Intel® Data Analytics Acceleration Library

---

¹ Available only in Intel® Parallel Studio Composer Edition.

*Other names and brands may be claimed as the property of others.
What’s New for 2019?
Intel® Distribution for Python*

Faster Machine learning with Scikit-learn functions
- Support Vector Machine (SVM) and K-means prediction, accelerated with Intel® DAAL

Built-in access to XGBoost library for Machine Learning
- Access to Distributed Gradient Boosting algorithms

Ease of access installation
- Now integrated into Intel® Parallel Studio XE installer.

Access Intel-optimized Python packages through

YUM/APT repositories

Standalone Python Distribution

Intel optimized packages via conda

Docker Hub

python Package Index

Optimization Notice
Copyright © 2019, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.
Optimizing scikit-learn with Intel® DAAL

- The most popular package for machine learning
- Hundreds of algorithms with different parameters
- Has a very flexible and easy-to-use interface

Intel DAAL own Python API (middleware)

- High performance of analytical and machine learning algorithms on Intel architecture

- High performance basic mathematical routines (BLAS, vector math, RNG, ...)

scikit-learn

DAAL4Py

Intel® DAAL

Optimized kernels from Intel® MKL
## Installing Intel® Distribution for Python* 2018

|----------------------|--------------------------------------------------------------------------------------------------|
| Anaconda.org         | > conda config --add channels intel  
> conda install intelpython3_full  
> conda install intelpython3_core |
| PyPI                 | > pip install intel-numpy  
> pip install intel-scipy  
> pip install mkl_fft  
> pip install mkl_random + Intel library Runtime packages  
+ Intel development packages |
| Docker Hub           | docker pull intelpython/intelpython3_full |

### 2.7 & 3.6 (3.7 coming soon)

- **Linux***
- **Windows***
- **OS X***
Scikit-learn functions now faster with Intel® DAAL

Optimization Notice

Performance results are based on testing as of July 9, 2018 and may not reflect all publicly available security updates. See configuration disclosure for details. No product can be absolutely secure.

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, see Performance Benchmark Test Disclosure.

Testing by Intel as of July 9, 2018. Configuration: Stock Python; python 3.6.6 h23d621a_0 installed from conda, numpy 1.15, numba 0.39.0, llvmlite 0.24.0, scipy 1.1.0, scikit-learn 0.19.2 installed from pipp; Intel® Python; Intel® Distribution for Python® 2019 Gold; python 3.6.6 intel_11, numpy 1.14.3 intel_py36_5, scikit-learn 0.19.2, numpy 1.16.2, numba 0.39.0, intel_rtlib 0.114.0, llvmlite 0.24.0, scipy 1.1.0, intel_python 0.19.2, scikit-learn 0.19.2.

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 #20110504.

For more complete information about compiler optimizations, see our Optimization Notice.
But Wait.....There's More!

Outside of optimized Python*, how efficient is your Python/C/C++ application code?

Are there any non-obvious sources of performance loss?

Performance analysis gives the answer!
Tune Python* + Native Code for Better Performance
Analyze Performance with Intel® VTune™ Amplifier (available in Intel® Parallel Studio XE)

Challenge
▪ Single tool that profiles Python + native mixed code applications
▪ Detection of inefficient runtime execution

Solution
▪ Auto-detect mixed Python/C/C++ code & extensions
▪ Accurately identify performance hotspots at line-level
▪ Low overhead, attach/detach to running application
▪ Focus your tuning efforts for most impact on performance

Auto detection & performance analysis of Python & native functions

Available in Intel® VTune™ Amplifier & Intel® Parallel Studio XE
Diagnose Problem code quickly & accurately

Details Python* calling into native functions

Identifies exact line of code that is a bottleneck
Deeper Analysis with Call stack listing & Time analysis

Call Stack Listing for Python* & Native Code

Detailed Time Analysis
A 2-prong approach for Faster Python* Performance
High Performance Python Distribution + Performance Profiling

Step 1: Use Intel® Distribution for Python
- Leverage optimized native libraries for performance
- Drop-in replacement for your current Python - no code changes required
- Optimized for multi-core and latest Intel processors

Step 2: Use Intel® VTune™ Amplifier for profiling
- Get detailed summary of entire application execution profile
- Auto-detects & profiles Python/C/C++ mixed code & extensions with low overhead
- Accurately detect hotspots - line level analysis helps you make smart optimization decisions fast!
- Available in Intel® Parallel Studio XE Professional & Cluster Edition
HANDS-ON PREPARATION
Hands-On Sessions are for You!

Take your time to understand the Python code samples – don’t just execute Jupyter cells 1by1

Also... there are solution files available, while it is in your own interest trying to find a solution yourself ...
Prerequisites for the hands-on part

1) Internet connection
2) SSH client (e.g. Putty)
3) Browser (Jupyter, NoVNC)

Who want’s to join the hands-on?
START INSTANCES

C5.xlarge
Audience Community Effort

1) We have N attendees of the workshop
2) While Shailen is preparing N nodes ...
3) Audience task
   a) Collectively solve the following problem
   b) Each workshop participant gets a unique index 0 < I <= N
4) Write down the IP address related to your index from Michael’s sheet
Login Credentials

Username: workshop
Password: Intel!1234
VNC Password: Intel!1234

We need two different SSH tunnels:

- 12345:localhost:12345
- 12346:localhost:12346
Putty Setup

PuTTY Configuration

Category: Session
- Logging
- Terminal
- Keyboard
- Bell
- Features

Category: Window
- Appearance
- Behaviour
- Translation
- Selection
- Colours

Category: Connection
- Data
- Proxy
- Telnet
- Rlogin
- SSH
- Kex
- Cipher
- Auth
- TTY

Category: Options controlling SSH port forwarding
- Port forwarding
  - Local ports accept connections from other hosts
  - Remote ports do the same (SSH-2 only)
- Forwarded ports:
  - L12345, localhost:12345
  - L12346, localhost:12346

Add new forwarded port:
- Source port 12346
- Destination localhost:12346
- Type Local
- Protocol IPv4
Native Shell

```bash
$ ssh -L 12345:localhost:12345 -L 12346:localhost:12346 \ 
workshop@$IP
```
Workshop Setup

$ cd labs/
$ ll

total 0

drwx------. 4 workshop workshop 147 Nov 14 13:43 idp_ml
drwxrwxr-x. 4 workshop workshop 127 Nov 15 12:35 tf_basics
drwxrwxr-x. 2 workshop workshop 6 Nov 15 10:20 tf_distributed
IDP HANDS-ON CLASSIC ML
Workshop Setup

```
$ cd ~/labs/idp_ml/
$ ll

 total 16
-rwx-------. 1 workshop workshop 230 Nov 14 13:32 01_start_vnc_server.sh
-rw-------. 1 workshop workshop 136 Nov 14 13:42 02_source_environments.sh
-rwx-------. 1 workshop workshop  74 Nov 14 13:43 03_start_notebook.sh
-rwx-------. 1 workshop workshop  48 Nov 14 13:28 04_kill_vnc.sh
drwx-------. 4 workshop workshop 122 Nov 14 16:34 numpy
drwx-------. 3 workshop workshop 124 Nov 14 16:35 sklearn
```
Start VNC Server and Jupyter Notebook

$ ./01_start_vnc_server.sh

New 'ip-172-31-38-147.eu-central-1.compute.internal:1 (workshop)' desktop is ip-172-31-38-147.eu-central-1.compute.internal:1

Starting applications specified in /home/workshop/.vnc/xstartup
Log file is /home/workshop/.vnc/ip-172-31-38-147.eu-central-1.compute.internal:1.log

Now open in your local browser: http://localhost:12345/vnc.html?host=localhost&port=12345

$ source ./02_source_environments.sh

Copyright (C) 2009-2018 Intel Corporation. All rights reserved.
Intel(R) VTune(TM) Amplifier 2018 (build 574913)

$ ./03_start_notebook.sh

[I 13:46:33.936 NotebookApp] 0 active kernels
[I 13:46:33.936 NotebookApp] The Jupyter Notebook is running at:
[I 13:46:33.936 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 13:46:33.936 NotebookApp]

Copy/paste this URL into your browser when you connect for the first time, to login with a token:

http://127.0.0.1:12346/?token=646642d51856d5385aa7cbe38228717da201c166003e4fbf
Open VNC Session
Open Jupyter Notebook
numpy/numpy_demo.ipynb – 15 Minutes

1) Why is the performance using the NumPy functions is lower as expected?
2) Implement the black_scholes function in a NumPy like fashion
3) Measure the speedup and explain where exactly it is coming from
4) Do benchmarking with Vtune
   a) Result will open in VNC session
   b) Proof your arguments from 3) using the VTune result
   c) Look at the call-stack in order to see native vs managed code
NumPy Demo Summary

- Use NumPy for compute intensive operations (MKL enabled)
- Make sure to apply operations to as many elements as possible at a time
- Check with VTune if there are performance hotspots outside of optimized code
- Speedup = #Cores * Vector Width * Other optimizations (e.g. cache blocking)
sklearn/kmeans.ipynb – 15 Minutes

1) What is the K-Means Algorithm?

2) How does the K-Means Algorithm work?

3) Select different sizes for n_colors (K) and compare the training runtime

4) Implement the inference function “labels = ”

5) What is the random codebook and how does it compare to K-Means?

6) Compare the outcome images with different cluster sizes (K)

7) Implement the function to disable our DAAL optimizations underneath Scikit-Learn and do some tests without it (plain vanilla Scikit-Learn)
K-Means Demo Summary

- K-Means is a powerful clustering algorithm
- SciKit-Learn K-Means is accelerated with DAAL inside IDP
- The optimized K-Means runs faster and consumes less memory
- We found a way to compress images!!!
1) What is a Support Vector Machine (SVM)?

2) How does the SVM work?

3) What is the MNIST dataset? Can classic ML algorithms classify images?

4) How can a binary classifier categorize 10 different classes?

5) How is the data is partitioned? And why?

6) What is a confusion matrix?

7) Implement the missing code to show mispredicted images

8) Do you recognize patterns from the mispredicted images?
SVM Demo Summary

• SVM is a powerful classifier
• Complex classification is not an exclusively deep learning field
• Classic machine learning, wherever applicable can safe time and resources
• The confusion matrix is actually not so confusing
• NumPy is powerful, can transform and operate on whole arrays
Save your accomplishments

$ ./05_pack_work.sh
...
$ ll ~/Downloads/
total 28
-rw-rw-r--. 1 workshop workshop 24692 Nov 21 15:14 idp_ml.tar.bz2

From your system:

cp -r workshop@$IP:~/Downloads/* .
TERMINATE INSTANCES
LUNCH BREAK

... finally ...
DEEP LEARNING TOOLS
INTEL PERFORMANCE LIBRARIES

INTEL® MATH KERNEL LIBRARY FOR DEEP NEURAL NETWORKS (MKL-DNN)
INTEL® MACHINE LEARNING SCALING LIBRARY (MLSL)
INTEL® MATH KERNEL LIBRARY FOR DEEP NEURAL NETWORKS

INTEL® MKL-DNN
Intel’s Open-Source *Math Kernel Library* for *Deep Neural Networks*

For developers of deep learning frameworks featuring optimized performance on Intel hardware

**Distribution Details**

- Open Source
- Apache 2.0 License
- Common DNN APIs across all Intel hardware.
- Rapid release cycles, iterated with the DL community, to best support industry framework integration.
- Highly vectorized & threaded for maximal performance, based on the popular Intel® MKL library.

**Examples:**

- Direct 2D Convolution
- Local response normalization (LRN)
- Rectified linear unit neuron activation (ReLU)
- Maximum pooling
- Inner product

[github.com/01org/mkl-dnn](https://github.com/01org/mkl-dnn)

*All products, computer systems, dates, and figures are preliminary based on current expectations, and are subject to change without notice.*
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)

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

INFORMATION THROUGHPUT

3.2X

Intel-optimized TensorFlow Inceptionv3 inference throughput compared to Default TensorFlow for CPU

Inference and training throughput uses FP32 instructions

PERFORMANCE GAINS REPORTED BY OTHERS

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

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

*Other names and brands may be claimed as the property of others.
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](https://software.intel.com/en-us/articles/intel-vtune-amplifier-xe) 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**

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

Example: inference flow
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
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

\[
\text{# building Intel MKL-DNN using cmake}
\]
\[
\text{cmake -DVTUNEROOT=/opt/intel/vtune_amplifier_2018 .. \\&
\text{make install}
\]
\[
\text{# an alternative: building Intel MKL-DNN using sources directly, e.g. in TensorFlow}
\]
\[
\text{CFLAGS="-I$VTUNEROOT/include -DJIT_PROFILING_VTUNE" LDFLAGS="-L$VTUNEROOT/lib64 -ljitprofiling" bazel build}
\]
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](#))

```bash
$ # 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_0hwi8o,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
```

Optimization Notice
Copyright © 2019, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.
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](https://github.com/intel/mkl-dnn/issues).
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® MACHINE LEARNING SCALING LIBRARY
INTEL® MLSL
Deep Learning Training

Complex Networks with billions of parameters can take days to train on a modern processor*

Hence, the need to reduce time-to-train using a cluster of processing nodes

Deep Learning Training

- Forward propagation: calculate loss function based on the input batch and current weights;

- Backward propagation: calculate error gradients w.r.t. weights for all layers (using chain rule);

- Weights update: use gradients to update weights; there are different algorithms exist: vanilla SGD, Momentum, Adam, etc.

\[ W_n^* = W_n - \alpha \cdot \frac{\partial E}{\partial W_n} \text{ or variants} \]
Why Machine Learning Scaling Library (MLSL)?

Scale Out Deep Learning: Requirements

- Choosing optimal work partitioning strategy
- Enabling scalability for small/large batch size
- Reducing communication volume
- Choosing optimal communication algorithm
- Prioritizing latency-bound communication
- Portable / efficient implementation
- Workload coverage across CNNs, RNNs, LSTMs, ...
- Integration with Deep Learning Frameworks

Communication dependent on work partitioning strategy
Data parallelism = Allreduce (or) Reduce_Scatter + Allgather
Model parallelism = AlltoAll

Data Parallelism

Model Parallelism

Hybrid Parallelism

Numerous DL Frameworks

Multiple NW Fabrics

Ethernet
OmniPath® Infiniband®
MLSL : Key features & ideas

Abstraction:
- MLSL abstracts communication patterns and backend and supports data/model/hybrid parallelism

Flexibility:
- C, C++, Python languages are supported out of box

Usability
- MLSL API is being designed to be applicable to variety of popular FWs

Optimizations:
- MLSL uses not only the existing MPI functionality, but also extensions
- Domain awareness to drive MPI in a performant way
- Best performance across interconnects– transparent to frameworks
MLSL: Parallelism options

Fully connected layer

\( I \in \mathbb{R}^{N \times K} \)

Input

\( W \in \mathbb{R}^{K \times M} \)

Weights or model

\( O \in \mathbb{R}^{N \times M} \)

Output or activations

Several options for parallelization
MLSL : Parallelism options

Data parallelism:

- Replicate the model across nodes;
- Feed each node with its own batch of input data;
- Communication for gradients is required to get their average across nodes;
- Can be either
  - AllReduce pattern
  - ReduceScatter + AllGather patterns

\[
\begin{align*}
I = \text{Input data} & \quad \text{W = Weights or model} \\
\times & \quad = \\
O = \text{Output or activations}
\end{align*}
\]
MLSL: Parallelism options

Data Parallelism
MLSL: Parallelism options

Model parallelism (#1):

- Model is split across nodes;
- Feed each node with slice of input data;
- Communication for partial activations is required to proceed to the next layer;
MLSL : Parallelism options

Model parallelism (#2):

- Model is split across nodes;
- Feed each node with the same batch of input data;
- Communication for partial activations is required to gather the result and proceed further;
MLSL : Parallelism options

Hybrid parallelism:

• Split nodes into groups;
• Model parallelism inside the groups;
• Data parallelism between the groups;
• Communicate both gradients and activations;
MLSL: Parallelism at Scale

General rule of thumb

- Use data parallelism when activations > weights
- Use model parallelism when weights > activations

Side effects of data and model parallelism

- Data parallelism at scale makes activations << weights
- Model parallelism at scale makes weights << activations
- Communication time dominates at scale
MLSL : Message prioritization

Node 1:

Result is required ASAP

Node k:

Result is required later

Optimization Notice
Copyright © 2019, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.
MLSL calls hide communication patterns used underneath:

- StartComm may involve reduce_scatter or all2all depending on the distributions or may not require any communication at all
- StartGradientComm/WaitGradientComm hides the details of distributed solver implementation
- API hides the details of communication backend
- Ideal for Caffe likes
MLSL : Collective API

Goal:

- Ease of enabling graph-based frameworks (allreduce op)

Collective Ops supported (non-blocking):

- Reduce/Allreduce
- Alltoall(v)
- Gather/Allgather(v)
- Scatter, Reduce_Scatter
- Bcast

Features:

- High performance (EP-based)
- Efficient asynchronous progress
- Prioritization (WIP)

/*Create MLSL environment*/
Environment env = Environment::GetEnv();
env.Init(&argc, &argv);

/* Create distribution
 * Arguments define how compute resources are split
 * between GROUP_DATA and GROUP_MODEL
 * Example below: all nodes belong to GROUP_DATA*/
Distribution* distribution = env.CreateDistribution(nodeCount, 1);

/*Handle for non-blocking comm operation*/
CommReq cr;

/*Start non-blocking op*/
distribution->AllReduce(sendbuffer, recvbuffer, size, DT_FLOAT, RT_SUM, GROUP_ALL, &cr);

/*Blocking wait call*/
env.Wait(&cr);
MLSL: Features

Current features:

✓ Non-blocking DL Layer and Collective interface
✓ Python/C++/C bindings
✓ Asynchronous communication progression
✓ Optimized algorithms
✓ Support for data, model, hybrid parallelism
✓ Initial support for quantization – available in IntelCaffe/MLSL
✓ Built-in inversed prioritization (through env. variable) – available in IntelCaffe/MLSL

• Upcoming features (in development or research):

✓ Explicit prioritization API
✓ Sparse data allreduce
✓ Gradient quantization and compression
✓ Cloud native features

https://github.com/intel/MLSL
## Scale-out in Cloud environment

### DAWNbench:

<table>
<thead>
<tr>
<th>Date</th>
<th>Model</th>
<th>Time</th>
<th>Cores</th>
<th>Nodes</th>
<th>Memory</th>
<th>Processor</th>
<th>Efficiency Scaling</th>
<th>Optimized Caffe</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apr 2018</td>
<td>ResNet50</td>
<td>3:25:55</td>
<td>N/A</td>
<td>128</td>
<td>144 GB</td>
<td>Xeon Platinum 8124M</td>
<td>93.02%</td>
<td>Intel(R) Optimized Caffe</td>
</tr>
<tr>
<td>Apr 2018</td>
<td>ResNet56</td>
<td>3:31:47</td>
<td>N/A</td>
<td>128</td>
<td>144 GB</td>
<td>Xeon Platinum 8124M</td>
<td>93.11%</td>
<td>Intel(R) Optimized Caffe</td>
</tr>
<tr>
<td>Apr 2018</td>
<td>ResNet50</td>
<td>6:09:50</td>
<td>N/A</td>
<td>64</td>
<td>144 GB</td>
<td>Xeon Platinum 8124M</td>
<td>93.05%</td>
<td>Intel(R) Optimized Caffe</td>
</tr>
</tbody>
</table>

*RN50:*

- 81 epochs for 64 nodes
- 85 epochs for 128 nodes
- 94% efficiency scaling from 64 to 128 nodes
Scale-out in HPC environment

- **IntelCaffe**: MLSL–based multinode solution; **Horovod, nGraph**: WIP

- MLSL is enabled in Baidu’s DeepBench

- SURFSara: used IntelCaffe/MLSL to achieve ResNet50 time-to-train record (~40 minutes, 768 SKX) *

- UC-Berkeley, TACC, and UC-Davis: 14 minutes TTT for ResNet50 with IntelCaffe/MLSL (2048 KNL) **

---
Deep Learning at 15PF *

- Joint work between NERSC, Stanford University and Intel
- Novel approach to distributed SGD: synchronous within the group, asynchronous across the groups
- Record scaling: in terms of number of nodes collaboratively training the same model (9600 KNL)
- Record peak performance: ~15PF
- Communication approach: MLSL for intragroup communication, MPI for intergroup
- The mechanism is available in IntelCaffe/MLSL

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

https://arxiv.org/abs/1708.05256

Deep Learning at 15PF *
INFERENC IN PRODUCTION?
Intel® OpenVINO™ toolkit
(Open Visual Inference & Neural Network Optimization)
What’s Inside the OpenVINO™ toolkit

**Intel® Deep Learning Deployment Toolkit**

- **Model Optimizer**
  - Convert & Optimize
  - 20+ Pre-trained Models

- **Inference Engine**
  - Optimized Inference
  - Computer Vision Algorithms

- **IR**

**Optimized Libraries**

- OpenCV*
- OpenVX*
- Photography Vision
- Code Samples

*For Intel® CPU & CPU with integrated graphics

**Traditional Computer Vision Tools & Libraries**

- **Intel® Media SDK**
  - Open Source version
- **OpenCL™ Drivers & Runtimes**
  - For CPU with integrated graphics

**Optimize Intel® FPGA**

- **FPGA RunTime Environment**
  - (from Intel® FPGA SDK for OpenCL™)
- **Bitstreams**
  - FPGA – Linux* only

**OS Support**

- CentOS* 7.4 (64 bit)
- Ubuntu* 16.04.3 LTS (64 bit)
- Microsoft Windows* 10 (64 bit)
- Yocto Project* version Poky Jethro v2.0.3 (64 bit)

**Intel® Architecture-Based Platforms Support**

Optimization Notice

Copyright © 2019, Intel Corporation. All rights reserved.

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

OpenVX and the OpenVX logo are trademarks of the Khronos Group Inc.

OpenCL and the OpenCL logo are trademarks of Apple Inc. used by permission by Khronos
Intel® Deep Learning Deployment Toolkit
Take Full Advantage of the Power of Intel® Architecture

**Model Optimizer**
- **What it is:** Preparation step -> imports trained models
- **Why important:** Optimizes for performance/space with conservative topology transformations; biggest boost is from conversion to data types matching hardware.

**Inference Engine**
- **What it is:** High-level inference API
- **Why important:** Interface is implemented as dynamically loaded plugins for each hardware type. Delivers best performance for each type without requiring users to implement and maintain multiple code pathways.

**Trained Model**
- Caffe*
- TensorFlow*
- MxNet*
- ONNX*
- Kaldi*

**Model Optimizer**
- Convert & optimize to fit all targets

**Inference Engine**
- Common API (C++ / Python)
- Optimized cross-platform inference

**Inference Engine**
- **CPU Plugin**
- **GPU Plugin**
- **FPGA Plugin**
- **Myriad Plugin**
- **GNA Plugin**

**IR**
- Intermediate Representation format

**Load, infer**

**Extendibility**
- C++
- OpenCL™
- OpenCL/TBD
- TBD

**Optimization Notice**
Copyright © 2019, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

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

GPU = Intel CPU with integrated graphics processing unit/Intel® Processor Graphics
Improve Performance with Model Optimizer

- Easy to use, Python*-based workflow does not require rebuilding frameworks
- Import Models from various supported frameworks - Caffe*, TensorFlow*, MXNet*, ONNX*, Kaldi*.
- More than 100 models for Caffe, MXNet and TensorFlow validated. All public models on ONNX* model zoo supported.
- With support for Kaldi, the model optimizer extends inferencing for non-vision networks.
- IR files for models using standard layers or user-provided custom layers do not require Caffe.
- Fallback to original framework is possible in cases of unsupported layers, but requires original framework
Optimal Model Performance Using the Inference Engine

- Simple & Unified API for Inference across all Intel® architecture
- Optimized inference on large IA hardware targets (CPU/GEN/FPGA)
- Heterogeneity support allows execution of layers across hardware types
- Asynchronous execution improves performance
- FutureproofSCALE YOUR development for future Intel® processors

Transform Models & Data into Results & Intelligence
Increase Deep Learning Workload Performance on Public Models using OpenVINO™ toolkit & Intel® Architecture

Comparison of Frames per Second (FPS)

Relative Performance Improvement

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Std. Caffe</td>
<td>Blue</td>
<td>Green</td>
<td>Orange</td>
<td>Red</td>
<td>Blue</td>
<td>Green</td>
<td>Orange</td>
<td>Red</td>
<td>Blue</td>
</tr>
<tr>
<td>OpenCV on CPU</td>
<td>Yellow</td>
<td>Yellow</td>
<td>Yellow</td>
<td>Yellow</td>
<td>Yellow</td>
<td>Yellow</td>
<td>Yellow</td>
<td>Yellow</td>
<td>Yellow</td>
</tr>
<tr>
<td>OpenVINO on CPU+Intel® Processor Graphics (GPU) / (FP16)</td>
<td>Purple</td>
<td>Purple</td>
<td>Purple</td>
<td>Purple</td>
<td>Purple</td>
<td>Purple</td>
<td>Purple</td>
<td>Purple</td>
<td>Purple</td>
</tr>
</tbody>
</table>

Fast Results on Intel Hardware, even before using Accelerators

1Depending on workload, quality/resolution for FP16 may be marginally impacted. A performance/quality tradeoff from FP32 to FP16 can affect accuracy; customers are encouraged to experiment to find what works best for their situation. The benchmark results reported in this deck may need to be revised as additional testing is conducted. Performance results are based on testing as of April 10, 2018 and may not reflect all publicly available security updates. See configuration disclosure for details. No product can be absolutely secure. For more complete information about performance and benchmark results, visit www.intel.com/benchmarks.

Configuration: Testing by Intel as of April 10, 2018. Intel® Core™ i7-6700K CPU @ 2.90GHz fixed, GPU GT2 @ 1.00GHz fixed Internal ONLY testing, Test v312.30 – Ubuntu® 16.04, OpenVINO™ 2018 RC4. Tests were based on various parameters such as model used (these are public), batch size, and other factors. Different models can be accelerated with different Intel hardware solutions, yet use the same Intel software tools.
Increase Deep Learning Workload Performance on Public Models using OpenVINO™ toolkit & Intel® Architecture

Comparison of Frames per Second (FPS)

Get an even Bigger Performance Boost with Intel® FPGA

19.9x

*Depending on workload, quality/resolution for FP16 may be marginally impacted. A performance/quality tradeoff from FP32 to FP16 can affect accuracy; customers are encouraged to experiment to find what works best for their situation. Performance results are based on testing as of June 13, 2018 and may not reflect all publicly available security updates. See configuration disclosure for details. No product can be absolutely secure. For more complete information about performance and benchmark results, visit www.intel.com/benchmarks. Configuration: Testing by Intel as of June 13, 2018. Intel® Core™ i7-6700K CPU @ 2.90GHz fixed, GPU GT2 @ 1.00GHz fixed Internal ONLY testing, Test v3.15.21 – Ubuntu® 16.04, OpenVINO 2018 RC4, Intel® Arria® 10 FPGA 1150GX. Tests were based on various parameters such as model used (these are public), batch size, and other factors. Different models can be accelerated with different Intel hardware solutions, yet use the same Intel software tools. 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
SECURITY BARRIER RECOGNITION MODEL USING INTEL® DEEP LEARNING DEPLOYMENT TOOLKIT
Load Input Image(s)

Run Inference 1:
Model vehicle-license-plate-detection-barrier-0007
Detects Vehicles

Run Inference 2:
Model vehicle-attributes-recognition-barrier-0010
Classifies vehicle attributes

Run Inference 3:
Model license-plate-recognition-barrier-0001
Detects License Plates

Display Results
INTRODUCTION TO TENSORFLOW WITH INTEL® OPTIMIZATIONS
Agenda

• Introduction to TensorFlow
• Neural Networks with TensorFlow
• Convolutional Neural Network with TensorFlow to perform image classification
• Build and Install Intel® optimized TensorFlow
• Optimizations and performance comparisons
INTEL AI FRAMEWORKS

Popular DL Frameworks are now optimized for CPU!

Choose your favorite framework:

TensorFlow, Caffe, mxnet, BigDL for Spark, and others to be enabled via Intel® nGraph™ Library

See installation guides at ai.intel.com/framework-optimizations/

More under optimization: Caffe2, PyTorch, Microsoft CNTK, and PaddlePaddle

See ALSO: Machine Learning Libraries for Python (Scikit-learn, Pandas, NumPy), R (Caret, randomForest, e1071), Distributed (MLlib on Spark, Mahout)

*Limited availability today
Other names and brands may be claimed as the property of others.
Getting intel-optimized tensorflow: using pip

# Python 2.7

```bash
pip install https://anaconda.org/intel/tensorflow/1.6.0/download/tensorflow-1.6.0-cp27-cp27mu-linux_x86_64.whl
```

# Python 3.5

```bash
pip install https://anaconda.org/intel/tensorflow/1.6.0/download/tensorflow-1.6.0-cp35-cp35m-linux_x86_64.whl
```

# Python 3.6

```bash
pip install https://anaconda.org/intel/tensorflow/1.6.0/download/tensorflow-1.6.0-cp36-cp36m-linux_x86_64.whl
```
Build TensorFlow MKL-DNN


```bash
$ git clone https://github.com/hfp/tensorflow-xsmm.git

▪ (or rely on https://github.com/tensorflow/tensorflow/releases/latest)

$ cd tensorflow-xsmm; ./configure

$ bazel build -c opt --copt=-O2 \
   --cxxopt=-D_GLIBCXX_USE_CXX11_ABI=0 \
   --copt=-mfma --copt=-mavx2 \
   //tensorflow/tools/pip_package:build_pip_package

$ bazel-bin/tensorflow/tools/pip_package/build_pip_package \
   /tmp/tensorflow_pkg

* AVX-512: --copt=-mfma --copt=-mavx512f --copt=-mavx512cd --copt=-mavx512bw --copt=-mavx512vl --copt=-mavx512dq
Build TensorFlow (cont.)

Package the TensorFlow Wheel file

$ bazel-bin/tensorflow/tools/pip_package/build_pip_package \
/tmp/tensorflow_pkg

- Optional (save Wheel file for future installation):

$ cp /tmp/tensorflow_pkg/tensorflow-1.2.1-cp27-cp27mu-linux_x86_64.whl \
/path/to/mysafeplace

Install the TensorFlow Wheel

- [user] $ pip install --user --upgrade -I \
/tmp/tensorflow_pkg/tensorflow-1.2.1-cp27-cp27mu-linux_x86_64.whl

- [root] $ sudo -H pip install --upgrade -I \
/tmp/tensorflow_pkg/tensorflow-1.2.1-cp27-cp27mu-linux_x86_64.whl
TensorFlow History

- **2nd gen. open source ML framework from Google**
  - Widely used by Google’s: search, Gmail, photos, translate, etc.
  - Open source implementation released in November 2015

- **Core in C++, frontend wrapper is in Python**
  - Core: key computational kernel, extensible per user-ops
  - Python script to specify/drive computation

- **Runtime**
  - Multi-node originally per GRPC protocol, MPI added later
  - Own threading runtime (not OpenMP, TBB, etc.)

**Milestones**

- **02’16**: TensorFlow Serving
- **02’16**: TensorFlow Serving
- **01’17**: Accelerated Linear Algebra (XLA)
- **02’17**: TensorFlow Fold

*Other names and brands may be claimed as the property of others.
Why do we need Optimizations for CPU?

- TensorFlow* on CPU has been very slow

- **With optimization**, up to 14x Speedup in Training and 3.2x Speedup in Inference! Up-streamed and Ready to Use!
Main TensorFlow API Classes

Graph
- Container for operations and tensors

Operation
- Nodes in the graph
- Represent computations

Tensor
- Edges in the graph
- Represent data
Computation Graph

Nodes represent computations

- input
- mul
- add
- add
- input
Computation Graph

Edges represent numerical data flowing through the graph
Data Flow
tf.constant() creates an Operation that returns a fixed value

tf.placeholder() defines explicit input that vary run-to-run

```python
>>> a = tf.placeholder(tf.float32, name="input1")
>>> c = tf.add(a, b, name="my_add_op")
```
We use a Session object to execute graphs. Each Session is dedicated to a single graph.

```python
>>> sess = tf.Session()
```

Session

Graph: `default`

Variable values:

![Diagram of a computational graph with nodes for input1, input2, my add op, and my mul op, and edges connecting them to form the graph structure.](image)
ConfigProto is used to set configurations of the Session object.

```python
>>> config = tf.ConfigProto(inter_op_parallelism_threads=2,
                            intra_op_parallelism_threads=44)
```

```python
>>> tf.Session(config=config)
```

Session `sess`

Graph: `default`

Variable values:
placeholders require data to fill them in when the graph is run.

We do this by creating a dictionary mapping Tensor keys to numeric values:

```python
>>> feed_dict = {a: 3.0, b: 2.0}
```

Let's consider a simple graph with two inputs, `input1` and `input2`, and two operations, `my_add_op` and `my_mul_op`. The graph is as follows:

```
input1 ---- a ----> my_add_op
     |        ^        |
     v        v        v
input2 ---- b ----> my_mul_op
```

The variable values are stored in the dictionary `feed_dict` which maps the Tensor keys to numeric values:

```
feed_dict: {a: 3.0, b: 2.0}
```
We execute the graph with `sess.run(fetches, feed_dict)`

`sess.run` returns the fetched values as a NumPy array

>>> out = sess.run(d, feed_dict=feed_dict)
Two-Step Programming Pattern

1. Define a computation graph

2. Run the graph
NEURAL NETWORKS WITH TENSORFLOW
Neural Networks

Use biology as inspiration for math model

Neurons:
- Get signals from previous neurons
- Generate signal (or not) according to inputs
- Pass that signal on to future neurons

By layering many neurons, can create complex model
Reads roughly the same as a TensorFlow graph

- Data flows into neuron from previous layers
- Some form of computation transforms the inputs
- The neuron outputs the transformed data

activation function
Inside a single neuron (TensorFlow graph)

Represents the function $z = W^t X + b$
Inside a single neuron (TensorFlow graph)

The activation function applies a non-linear transformation and passes it along to the next layer.
To keep visual noise down, we’ll use this notation for now

\[ x_1 \]

\[ x_2 \]

\[ +1 \]

\[ \alpha \]
A single neural layer

But having different weights means neurons respond to inputs differently

Each neuron has the same value for $x_1, x_2$ plugged in

$x_1$  
$x_2$  
$x_3$  
$\sigma$  
$\sigma$  
$\sigma$  
$\sigma$
CONVOLUTIONAL NEURAL NETWORK WITH TENSORFLOW
Convolutional Neural Nets

**Convolution Parameters:**
- Number of outputs/feature-maps: < 4 >
- Filter size: < 3 x 3 >
- Stride: < 2 >
- Pad_size (for corner case): < 1 >
Convolution In TensorFlow

tf.nn.conv2d(input, filter, strides, padding)

**input**: 4d tensor [batch_size, height, width, channels]

**filter**: 4d: [height, width, channels_in, channels_out]

- Generally a Variable

**strides**: 4d: [1, vert_stride, horiz_strid, 1]

- First and last dimensions must be 1 (helps with under-the-hood math)

**padding**: string: ‘SAME’ or ‘VALID’
TRAINING AND INference

**Step 1: Training**  
(Over Hours/Days/Weeks)

- Input data
- Create Deep network
- Output Classification

**Step 2: Inference**  
(Real Time)

- New input from camera and sensors
- Trained neural network model
- Output Classification

Input data → Person → 90% person, 8% traffic light → Trained Model

Trained Model -> New input from camera and sensors -> Output Classification

97% person
INTEL® TENSORFLOW OPTIMIZATIONS
intel-tensorflow optimizations

1. Operator optimizations
2. Graph optimizations
3. System optimizations
Operator optimizations

In TensorFlow, computation graph is a data-flow graph.
Operator optimizations

Replace default (Eigen) kernels by highly-optimized kernels (using Intel® MKL-DNN)

Intel® MKL-DNN has optimized a set of TensorFlow operations.


<table>
<thead>
<tr>
<th>Forward</th>
<th>Backward</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conv2D</td>
<td>Conv2DGrad</td>
</tr>
<tr>
<td>Relu, TanH, ELU</td>
<td>ReLUGrad, TanHGrad, ELUGrad</td>
</tr>
<tr>
<td>MaxPooling</td>
<td>MaxPoolingGrad</td>
</tr>
<tr>
<td>AvgPooling</td>
<td>AvgPoolingGrad</td>
</tr>
<tr>
<td>BatchNorm</td>
<td>BatchNormGrad</td>
</tr>
<tr>
<td>LRN</td>
<td>LRNGrad</td>
</tr>
<tr>
<td>MatMul, Concat</td>
<td></td>
</tr>
</tbody>
</table>
OPERATOR OPTIMIZATIONS IN RESNET50

Intel-optimized TensorFlow timeline

Default TensorFlow timeline
Graph optimizations: fusion

Before Merge

After Merge
Graph optimizations: fusion

Before Merge

After Merge
Graph optimizations: layout propagation

What is layout?

- How do we represent N-D tensor as a 1-D array.

{N:2, R:5, C:5}
Graph optimizations: layout propagation

Converting to/from optimized layout can be less expensive than operating on un-optimized layout.

All MKL-DNN operators use highly-optimized layouts for TensorFlow tensors.
Graph optimizations: layout propagation

Did you notice anything wrong with previous graph?

Problem: redundant conversions
System optimizations: load balancing

TensorFlow graphs offer opportunities for parallel execution.

Threading model

1. `inter_op_parallelism_threads` = max number of operators that can be executed in parallel

2. `intra_op_parallelism_threads` = max number of threads to use for executing an operator

3. `OMP_NUM_THREADS` = MKL-DNN equivalent of `intra_op_parallelism_threads`
tf.ConfigProto is used to set the inter_op_parallelism_threads and intra_op_parallelism_threads configurations of the Session object.

```python
>>> config = tf.ConfigProto()
>>> config.intra_op_parallelism_threads = 56
>>> config.inter_op_parallelism_threads = 2
>>> tf.Session(config=config)
```

https://www.tensorflow.org/performance/performance_guide#tensorflow_with_intel_mkl_dnn
System optimizations: load balancing

Incorrect setting of threading model parameters can lead to over- or under-subscription, leading to poor performance.

Solution:

▪ Set these parameters for your model manually.

▪ Guidelines on TensorFlow webpage

OMP: Error #34: System unable to allocate necessary resources for OMP thread:

OMP: System error #11: Resource temporarily unavailable

OMP: Hint: Try decreasing the value of OMP_NUM_THREADS.
Setting the threading model correctly


Example setting MKL variables with python `os.environ`:

```python
os.environ['KMP_BLOCKTIME'] = "1"
os.environ['KMP_AFFINITY'] = "granularity=fine,compact,1,0"
os.environ['KMP_SETTINGS'] = "0"
os.environ['OMP_NUM_THREADS'] = "56"
```

Tuning MKL for the best performance

This section details the different configurations and environment variables that can be used to tune the MKL to get optimal performance. Before tweaking various environment variables make sure the model is using the NCHW (channels_first) data format. The MKL is optimized for NCHW and Intel is working to get near performance parity when using NHWC.

MKL uses the following environment variables to tune performance:

- `KMP_BLOCKTIME` - Sets the time, in milliseconds, that a thread should wait, after completing the execution of a parallel region, before sleeping.
- `KMP_AFFINITY` - Enables the run-time library to bind threads to physical processing units.
- `KMP_SETTINGS` - Enables (true) or disables (false) the printing of OpenMP* run-time library environment variables during program execution.
- `OMP_NUM_THREADS` - Specifies the number of threads to use.

Optimizing for CPU

CPUs, which include Intel® Xeon®Phi™, achieve optimal performance when TensorFlow is built from source with all of the instructions supported by the target CPU.

Beyond using the latest instruction sets, Intel® has added support for the Intel® Math Kernel Library for Deep Neural Networks (Intel® MKL-DNN) to TensorFlow. While the name is not completely accurate, these optimizations are often simply referred to as "MKL" or TensorFlow with MKL. TensorFlow with Intel® MKL-DNN contains details on the MKL optimizations.

The two configurations listed below are used to optimize CPU performance by adjusting the thread pools.

- **intra_op_parallelism_threads**: Nodes that can use multiple threads to parallelize their execution will schedule the individual pieces into this pool.
- **inter_op_parallelism_threads**: All ready nodes are scheduled in this pool.

These configurations are set via the `tf.ConfigProto` and passed to `tf.Session` in the `config` attribute as shown in the snippet below. For both configuration options, if they are unset or set to 0, will default to the number of logical CPU cores. Testing has shown that the default is effective for systems ranging from one CPU with 4 cores to multiple CPUs with 70+ combined logical cores. A common alternative optimization is to set the number of threads in both pools equal to the number of physical cores rather than logical cores.

```python
config = tf.ConfigProto()
config.intra_op_parallelism_threads = 44
config.inter_op_parallelism_threads = 44
tf.Session(config=config)
```

The *Comparing compiler optimizations* section contains the results of tests that used different compiler optimizations.

TensorFlow with Intel® MKL-DNN

Intel® has added optimizations to TensorFlow for Intel® Xeon® and Intel® Xeon Phi™ though the use of Intel® Math Kernel Library for Deep Neural Networks (Intel® MKL-DNN). These optimizations also provide speedups for the consumer line of processors, e.g. G5 and G7 Intel processors. The Intel published paper *TensorFlow* Optimization on Modern Intel® Architecture contains additional details on the implementation.

https://www.tensorflow.org/performance/performance_guide#tensorflow_with_intel_mkl_dnn
Intel-Optimized tensorflow Performance at a glance

**TRAINING THROUGHPUT**

14X

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

**INERENCE THROUGHPUT**

3.2X

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

Inference and training throughput uses FP32 instructions

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

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 Linux 7.4.1708 (Core) Kernel 3.10.0-693.11.6.el7.x86_64

**TensorFlow Source**:
https://github.com/tensorflow/tensorflow

**TensorFlow Commit ID**:
926fc13f7378d14fa7980963c4fe774e5922e336.

**TensorFlow benchmarks**:
https://github.com/tensorflow/benchmarks

<table>
<thead>
<tr>
<th>Model</th>
<th>Data_format</th>
<th>Intrac_op</th>
<th>Inter_op</th>
<th>OMP_NUM_THREADS</th>
<th>KMP_BLO CTKTIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>VGG16</td>
<td>NCHW</td>
<td>56</td>
<td>1</td>
<td>56</td>
<td>1</td>
</tr>
<tr>
<td>InceptionV3</td>
<td>NCHW</td>
<td>56</td>
<td>2</td>
<td>56</td>
<td>1</td>
</tr>
<tr>
<td>ResNet50</td>
<td>NCHW</td>
<td>56</td>
<td>2</td>
<td>56</td>
<td>1</td>
</tr>
</tbody>
</table>

Software and system 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, applications, and functions. Results may vary for any of those factors. 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. This product is not designed or intended to be the sole or primary means of data protection. Additional features or software from Intel or other parties may be required. Any other products. For more complete information visit [http://www.intel.com/performance](http://www.intel.com/performance). Copyright © 2018, Intel Corporation.
**Intel-Optimized TensorFlow Training Performance**

Training Improvement with Intel-optimized TensorFlow over Default (Eigen) CPU Backend

- Improvement with Intel-optimized TensorFlow (NHWC)
- Improvement with Intel-optimized TensorFlow (NCHW)

**System configuration:**
- CPU Thread(s) per core: 2
- 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 Linux 7.4.1708 (Core) Kernel 3.10.0-693.11.6.el7.x86_64

**TensorFlowSource:**
[https://github.com/tensorflow/tensorflow](https://github.com/tensorflow/tensorflow)

**TensorFlow Commit ID:**
926fc13f7378d14fa79980963c4fe774e5922e336.

**TensorFlow benchmarks:**
[https://github.com/tensorflow/benchmarks](https://github.com/tensorflow/benchmarks)

<table>
<thead>
<tr>
<th>Model</th>
<th>Data_format</th>
<th>Intra_op</th>
<th>Inter_op</th>
<th>OMP_NUM_THREADS</th>
<th>KMP_BOC_KTIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>VGG16</td>
<td>NCHW</td>
<td>56</td>
<td>1</td>
<td>56</td>
<td>1</td>
</tr>
<tr>
<td>InceptionV3</td>
<td>NCHW</td>
<td>56</td>
<td>2</td>
<td>56</td>
<td>1</td>
</tr>
<tr>
<td>ResNet50</td>
<td>NCHW</td>
<td>56</td>
<td>2</td>
<td>56</td>
<td>1</td>
</tr>
</tbody>
</table>
INTEL-OPTIMIZED TENSORFLOW INFEERENCE PERFORMANCE

Inference Improvement with Intel-optimized TensorFlow over Default (Eigen) CPU Backend

- Improvement with Intel-optimized TensorFlow (NHWC)
- Improvement with Intel-optimized TensorFlow (NCHW)

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 Linux 7.4.1708 (Core) Kernel 3.10.0-693.11.6.el7.x86_64

TensorFlowSource: https://github.com/tensorflow/tensorflow
TensorFlow Commit ID: 926fc13f7378d14fa7980963c4fe774e5922e336.

TensorFlow benchmarks: https://github.com/tensorflow/benchmarks

<table>
<thead>
<tr>
<th>Model</th>
<th>Data_format</th>
<th>Intra_op</th>
<th>Inter_op</th>
<th>OMP_NUM_THREADS</th>
<th>KMP_BCASTTIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>VGG16</td>
<td>NCHW</td>
<td>56</td>
<td>1</td>
<td>56</td>
<td>1</td>
</tr>
<tr>
<td>InceptionV3</td>
<td>NCHW</td>
<td>56</td>
<td>2</td>
<td>56</td>
<td>1</td>
</tr>
<tr>
<td>ResNet50</td>
<td>NCHW</td>
<td>56</td>
<td>2</td>
<td>56</td>
<td>1</td>
</tr>
</tbody>
</table>
Distributed TensorFlow™ Compare

The parameter server model for distributed training jobs can be configured with different ratios of parameter servers to workers, each with different performance profiles.

The ring all-reduce algorithm allows worker nodes to average gradients and disperse them to all nodes without the need for a parameter server.

Source: https://eng.uber.com/horovod/
Run as Distributed Training Across Multiple Nodes & Multiple Sockets

- No Parameter Server required
- Each **socket** on each worker node running 2 or more Framework Streams
- Internode communication with horovod MPI library
HOROVOD for multinode:

from Parameter server (PS):

NP=4
PER_PROC=10
HOSTLIST=192.168.10.110
MODEL=inception3
BS=64
BATCHES=100
INTRA=10
INTER=2

/usr/lib64/openmpi/bin/mpirun --allow-run-as-root -np $NP -cpus-per-proc $PER_PROC -map-by-socket -H $HOSTLIST --report-bindings --oversubscribe -x LD_LIBRARY_PATH python ./tf_cnn_benchmarks.py --model $MODEL --batch_size $BS --data_format NCHW --num_batches $BATCHES --distortions=True --mkl=True --local_parameter_device cpu --num_warmup_batches 10 --optimizer rmsprop --display_every 10 --kmp_blocktime 1 --variable_update horovod --horovod_device cpu --num_intra_threads $INTRA --num_inter_threads $INTER --data_dir /home/tf_imagenet --data_name imagenet
Scaling TensorFlow

There is way more to consider when striking for peak performance on distributed deep learning training.:

Summary

Convolutional Neural Network with TensorFlow

Getting Intel-optimized TensorFlow is easy.

TensorFlow performance guide is the best source on performance tips.

Intel-optimized TensorFlow improves TensorFlow CPU performance by up to 14X.

Stay tuned for updates - https://ai.intel.com/tensorflow
START INSTANCES

C5.2xlarge
Audience Community Effort

1) We have N attendees of the workshop
2) While Michael is preparing N nodes ...
3) Audience task
   a) Collectively solve the following problem
   b) Each workshop participant gets a unique index $0 < I \leq N$
4) Write down the IP address related to your index from Michael’s sheet
TENSORFLOW HANDS-ON IMAGE CLASSIFICATION

Basics
Workshop Setup

$ cd ~/labs/tf_basics/
$ ll

```
total 8
-rw-------. 1 workshop workshop 160 Nov 15 20:49 01_source_environments.sh
-rwx-------. 1 workshop workshop 394 Nov 15 20:49 02_start_notebook.sh
drwxrwxr-x. 5 workshop workshop 199 Nov 15 22:01 mnist
drwxrwxr-x. 2 workshop workshop  30 Nov 15 10:33 test
```
Start Jupyter Notebook

$ source ./01_source_environments.sh

$ ./02_start_notebook.sh


[I 17:27:37.744 NotebookApp] The Jupyter Notebook is running at:

[I 17:27:37.744 NotebookApp] http://127.0.0.1:12346/?token=7e7b503b855e94721b6041daf4abe1e470f5c42f31539957

[I 17:27:37.744 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).

[C 17:27:37.44 NotebookApp]

Copy/paste this URL into your browser when you connect for the first time, to login with a token:

http://127.0.0.1:12346/?token=7e7b503b855e94721b6041daf4abe1e470f5c42f31539957
Open Jupyter Notebook
mnist/01_mnist_softmax.ipynb – 15 Minutes

1) What is a Bias?

2) How does the matrix multiplication look like in TensorFlow?

3) What is the cross entropy?

4) What optimizer is being used?

5) How can you extract the correct prediction?

6) What is the accuracy of the trained model?

7) Is the evaluation accuracy using different data?
MNIST Softmax Demo Summary

- The bias represents some activation-independent offset for each neuron.
- Cross entropy is used to compute the difference (loss) between vectors.
- The accuracy is determined using a different evaluation dataset.
mnist/02_mnist_deep.ipynb – 20 Minutes

1) How is h_conv2 connected to the topology?

2) What is keep_prob representing?

3) What optimizer is being used?

4) How is the evaluation of the accuracy being done during training?

5) Can you compare the performance of different Jupyter kernels?

6) Is MKL-DNN used by each kernel?

7) What are the MKL-DNN primitives consuming most of the time?
MNIST CNN Demo Summary

- Conv2 is activated by the pooling layer after Conv1
- Keep_prob represents the dropout
- The “vanilla_tf” kernel does not use MKL-DNN, while “idp_tf” does
- The convolutions take the majority of CPU time – almost 20 seconds
- Switch off MKLDNN_VERBOSE for maximum performance
TENSORFLOW HANDS-ON IMAGE CLASSIFICATION
Distributed
Workshop Setup

$ cd ~/labs/tf_distributed
$ ll

total 8
-rw-------. 1 workshop workshop 146 Nov 20 17:53 01_source_environments.sh
-rwx-------. 1 workshop workshop 145 Nov 20 16:04 02_start_notebook.sh
drwxrwxr-x. 5 workshop workshop 152 Nov 21 13:22 images
drwxrwxr-x. 5 workshop workshop 245 Nov 20 17:59 mnist
Start Jupyter Notebook

$ source ./01_source_environments.sh

Intel(R) Parallel Studio XE 2019 Update 1 for Linux*
Copyright (C) 2009-2018 Intel Corporation. All rights reserved.

$ ./02_start_notebook.sh
[I 15:50:49.123 NotebookApp] 0 active kernels
[I 15:50:49.123 NotebookApp] The Jupyter Notebook is running at:
[I 15:50:49.123 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 15:50:49.124 NotebookApp]

Copy/paste this URL into your browser when you connect for the first time, to login with a token:
http://127.0.0.1:12346/?token=041bb0345290e3354f45f8d7474341044e3ace3862764551
mnist/03_mnist_deep_monitored.ipynb – 15 Min

1) What is the global step?

2) How does the MonitoredTrainingSession help you?

3) What happens if the training gets disrupted and continued later on?

4) How can a checkpoint be re-opened?

5) How can a checkpoint be re-stored?
MNIST CNN Monitored Training Session Demo Summary

- The global step helps when checkpointing and restarting the training
- The MonitoredTrainingSession
  - Does automatic checkpoints
  - Re-opens checkpoints automatically
  - Does automatic logging
  - Allows distributed runs
1) How to initialize Horovod and why is it necessary?

2) Why is it necessary to adapt the learning rate with larger batches?

3) How can you dynamically adapt the learning rate?

4) How to identify rank #1 (0)?

5) Why is it necessary to adapt the number of training steps according to the number of workers / larger batches?

6) How can you dynamically adapt the number of training steps?

7) How is the single process performance vs 2 ranks vs 4 ranks?
Horovod initializes the MPI communication underneath and therefore defines rank() and size()

In order to reduce the Time To Train with multiple workers, therefore increasing the batch size, the learning rate needs to scale

Same for the # of steps for training

4 ranks can be faster since less threading efficiency is required in small convolutions
images/05_custom_images.ipynb – 20 Min

1) What additional configuration variables are defined?
2) Why does read_images initialize the random seed with 42?
3) How does the next_batch_index function work?
4) What changes are needed for the original MNIST CNN topology?
5) How is the data being split into training and evaluation?
6) Why is the initial accuracy during training always around 0.2?
7) How can you extract the misclassified images?
8) How is the single process performance vs 2 ranks vs 4 ranks?
MNIST CNN Horovod Demo Summary

- Configuration variables like image size, batch size, training / eval split
- The training batches are partitioned in a way that each worker gets a different sub-batch – this requires aligned data. Also when re-starting a checkpoint, the train / eval split would be messed up otherwise
- Approx. init. accuracy = 1 / #classes
- Identify misclassified by leveraging the prediction_class
TENSORFLOW HANDS-ON CNN BENCHMARKING

Distributed
Workshop Setup

$ cd ~/labs/tf_benchmark
$ ll

```
total 12
-rw-------. 1 workshop workshop 111 Nov 21 12:29 01_source_environments.sh
-rwxrwxr-x. 1 workshop workshop 510 Nov 21 13:16 02_run_half_node.sh
-rwxrwxr-x. 1 workshop workshop 510 Nov 21 13:16 03_run_full_node.sh
drwxrwxr-x. 4 workshop workshop  65 Nov 21 12:08 benchmarks
```
Benchmark CNN – ResNet50 Example – 15 Min

$ source ./01_source_environments.sh
Intel(R) Parallel Studio XE 2019 Update 1 for Linux*
Copyright (C) 2009-2018 Intel Corporation. All rights reserved.

$ ./02_run_half_node.sh
...

$ ./03_run_full_node.sh
...

Play with these scripts and parameters – mind the limited memory

1) What is the KMP_BLOCKTIME?
2) What is NCHW?
3) How much difference does –mkl=True make?
4) How much difference does the pinning make (KMP_AFFINITY)?
5) Can you find a better Intra- Threads vs Inter- Threads combination?
6) What effect does the batch size have?
Save your accomplishments

$ ./04_pack_work.sh
...
$ ll ~/Downloads/
total 20
-rw-rw-r-- 1 workshop workshop 18262 Nov 21 17:25 tf_labs.tar.bz2

From your system:

scp -r workshop@$IP:~/Downloads/* .
TERMINATE INSTANCES