WHITE PAPER

Internet Discovery
Artificial Intelligence (AI)

Achieving 2.5X[1] Higher Performance for the Taboola TensorFlow* Serving Application through Targeted Software Optimization

As one of the world's preeminent discovery platforms, Taboola, a member of Intel® AI Builders, delivers tailored recommendations to more than a billion unique Internet users every month to help them explore what's interesting and new across publisher sites, mobile apps, and other digital properties. Over the last decade, thousands of publishers and advertisers including CBS Interactive, Euronews, Pandora, and Samsung2 have partnered with Taboola to build audiences, increase engagement, and drive revenue. Taboola's proprietary deep learning algorithms, powered by one of the largest datasets of content consumption behavior across the open web, match people with content that they truly care about at the moments they are most receptive to new things.

Taboola delivers content recommendations to online users using an artificial intelligence (AI)-based solution that predicts the preferences of each visitor within the context of each visit. A variety of data are ingested in real time for each website visitor. The AI solution processes this data, taking into account both simple factors, such as time of day and recently viewed content, and more complex factors, such as context and trending topics. The accuracy of the recommendations, coupled with the simplicity and effectiveness of the solution, has driven global success for Taboola, and is helping some of the most innovative and highly visited digital properties increase user engagement, monetize traffic, and acquire quality audiences.

The Power of AI

The heart of the Taboola solution is a neural network based on the open source TensorFlow* framework that uses deep learning to infer visitor preferences. This AI-based strategy is fundamental to meeting speed and accuracy requirements while analyzing a variety of data for each website visitor. It also ensures that the Taboola AI algorithms can continue to learn from new data sources and from the way individual consumers respond to the recommendations. The self-learning power of AI drives ongoing improvements in recommendation accuracy, without the need for complex, hands-on programming.

Taking Performance to New Heights—on Existing Hardware

To deliver its recommendation service globally, Taboola runs seven data centers around the world. As the company continues to expand its online footprint and evolve the accuracy of its recommendation engine, it needs steady increases in the power and capacity of its computing infrastructure. A recent upgrade to servers based on the latest Intel® Xeon® Platinum 8168 processor provided a 1.49X boost in neural network performance (*for details, see the Intel Solution Brief: Taboola Optimizes Artificial Intelligence for Smarter Content Recommendations*). Given the rapid growth in workload demands, even more performance was needed.

To achieve higher performance without expanding their infrastructure footprint, Taboola engaged with Intel software engineers to optimize their code. The software optimization was completed in just a few weeks, resulting in a 2.5X[1] improvement in performance over the original, unoptimized code. Taboola is using those performance gains to deliver more and better recommendations at higher speeds. With upwards of ten thousand servers across multiple data centers, the benefits in cost savings, efficiency, and growth potential are substantial.

The Software Optimization Process

The Taboola AI solution uses the TensorFlow-Serving* (TFS) framework, which is an open source deployment service for running machine learning models in production environments. TFS is architected on top of TensorFlow and employs a client server workflow to deliver recommendations. Each TFS server hosts a pretrained model of the Taboola neural network. When the server receives a prediction request from a client (through gRPC), it runs the client data in a forward pass through the model and returns the result.

To improve performance, Intel engineers optimized TFS in three steps. Each step provided significant performance gains (Figure 1).

Step One: Use the Intel® Math Kernel Library for Deep Neural Networks (Intel® MKL-DNN)

Performance versus Baseline: 1.15X[1]

Tensor/matrix computations are used extensively in running client data through a trained AI model. TFS commonly relies on the open source Eigen* C++ template library to perform these operations. Although TFS itself has been highly optimized for Intel® architecture, Eigen has not. Intel® Math Kernel Library for Deep Neural Networks (Intel® MKL-DNN) provides primitives for neural network processing that are all highly optimized for performance on the latest Intel® microarchitecture. In the optimized test configuration, Intel MKL-DNN primitives were used by default. For operations that are currently not available in Intel MKL-DNN, the optimized application falls back to Eigen.

After integrating Intel MKL-DNN, the unoptimized and optimized versions of TFS were run on the same two-socket server configured with Intel® Xeon® Platinum 8180 processors. The addition of the Intel MKL-DNN library delivered 1.15X[1] the performance of the unoptimized version of TFS. The performance gains resulted primarily from faster matrix-matrix multiplication (SGEMM) operations.

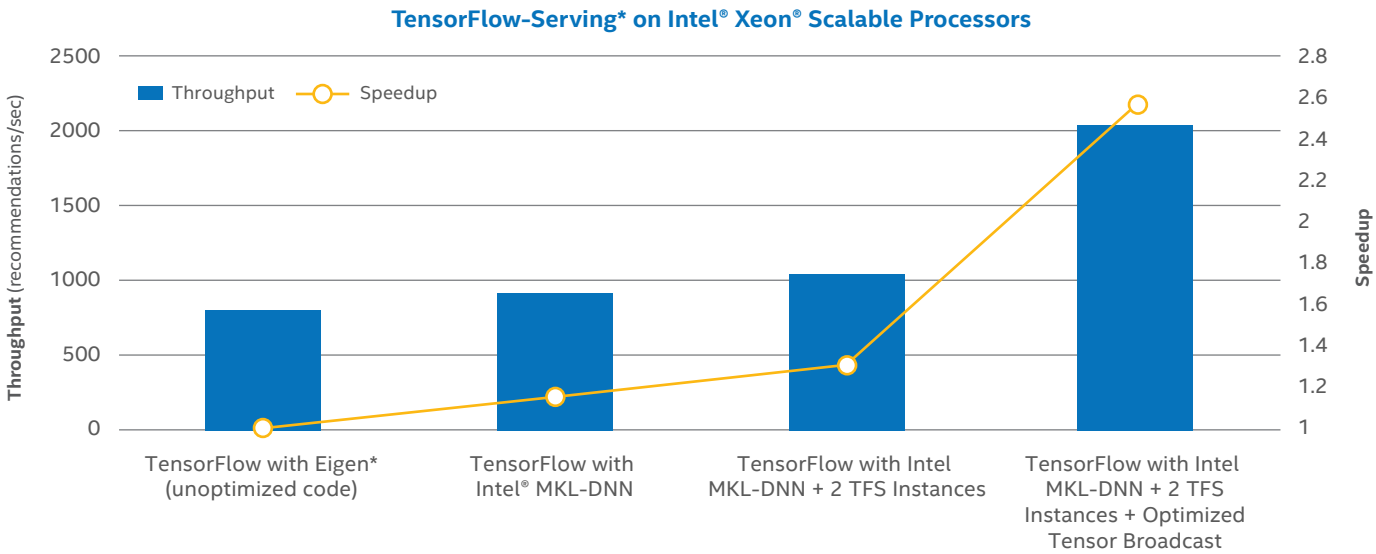

Figure 1. TFS Performance Gains: Performance comparisons for the optimized versions of TFS versus the unoptimized baseline version.

Step Two: Pin Application Threads and Memory Requests

Performance versus Baseline: 1.3X[1]

A two-socket server based on the Intel Xeon Platinum 8180 processor provides 56 cores. Intel engineers have found that hosting two instances of TFS per server and allocating the processor and memory resources efficiently to each instance improves performance. To accomplish this, they pinned the application threads from each instance of TFS to a corresponding processor socket. They also pinned memory requests originating from each TFS instance to the associated non-uniform memory access (NUMA) memory domain.

With this additional optimization, performance for the optimized TFS version rose to 1.3X[1] the performance of the original, unoptimized version.

Step Three: Optimize Tensor Operations

Performance versus Baseline: 2.5X[1]

To take performance to the next level, Intel engineers used Intel® VTune™ Amplifier to identify performance bottlenecks by profiling the application during runtime. With Intel VTune Amplifier, engineers can visualize the contribution of each software module to the overall runtime of the application. They can also look more closely to identify the precise lines of source code within those modules that are impairing performance and are good candidates for optimization. Not surprisingly for an application called TensorFlow, the most time-consuming operation turned out to be a tensor operation known as broadcasting.

A tensor is an n-dimensional array of numbers. A broadcast operation involves replicating the input tensor by a specified factor on any given dimension (Figure 2). Performance analysis of the Taboola TFS solution showed that a request from a single client results in roughly 25,000 tensor broadcast operations, which consume a large portion of the total processing time.

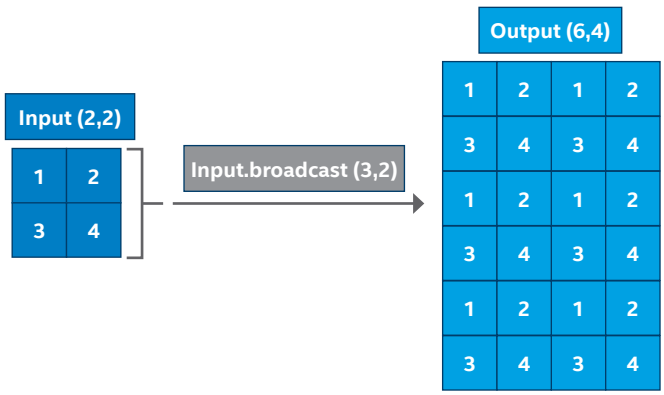

Figure 2. Example of a tensor broadcast: A 2x2 tensor is broadcast to a 6x4 tensor by replicating the first and second dimensions by a factor of 3 and 2, respectively.

Operations such as tensor broadcasting involve executing an instruction many times across a large number of data points. This makes them an ideal fit for the single instruction multiple data (SIMD) capabilities that are built into Intel® Xeon® processors through a technology called Intel® Advanced Vector Extensions (Intel® AVX). The latest Intel® Xeon® Scalable processors support Intel® Advanced Vector Extensions 512 (Intel® AVX-512), which allows a single instruction to be executed simultaneously on multiple data elements stored in a 512-bit vector register. Optimizing software for this strategy is known as vectorization, and can dramatically increase performance for operations that can be parallelized in this way.

As revealed by the Intel VTune Amplifier analysis, the Eigen implementation of tensor broadcasting relies heavily on scalar instructions that do not take advantage of vector processing capabilities available in Intel Xeon Scalable processors. The scalar instructions are used in calculating the target index in the input tensor, which specifies how the elements are copied to the output tensor. The engineering team also found that the required number of index calculations for a broadcast are excessive unless the dimensions of the tensors are a multiple of the width of the vector registers in the processor (vector register width is 16 for an FP32 data type on Intel Xeon Scalable processors).

The software optimization team vectorized the tensor broadcast functions in Eigen using Intel AVX-512 instructions. As part of the optimization effort, the engineering team created two new tensor broadcasting member functions based on the types of input tensors identified in Taboola's application: (packetNByOne) and (packetOneByN). For both types of tensors, the engineering team was able to significantly reduce the number of operations in a typical broadcast operation.

The following examples show how the operations are accelerated in representative cases.

- **Broadcasting a 5x1 input tensor into a 5x16 output tensor** (Figure 3): Using unoptimized Eigen, this operation requires 80 separate scalar calculations, one calculation for each tensor member in the output tensor. Using Intel AVX-512 instructions in the optimized version of Eigen, that same operation can be performed using just five calculations, one calculation for each element of the input tensor. In other words, the optimized code reduces the number of required calculations by a factor of 16.
- **Broadcasting a 1x20 input tensor into 5x20 output tensor** (Figure 4): This operation is more complicated because the 20 elements of the output tensor do not fit evenly within the 512-bit vector registers of Intel Xeon Scalable processors (20 elements times 32 bits per element equals 640 bits). In this case, the baseline Eigen version takes advantage of some SIMD functionality, but still relies on 52 scalar operations. The optimized code performs the same operation without scalar operations, which significantly reduces the total number of required calculations.

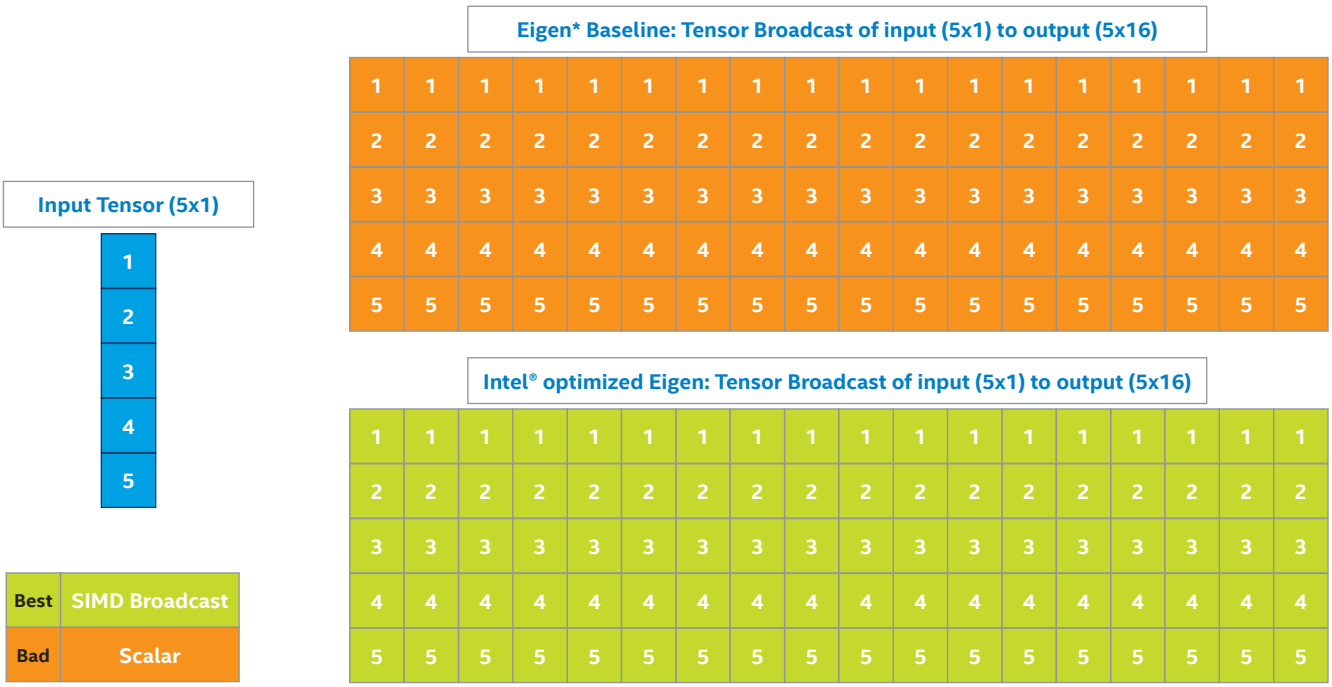

Figure 3. Comparison of a tensor broadcast of a 5x1 tensor (packetNByOne class) into a 5x16 tensor using unoptimized Eigen* (upper graphic) and Intel® optimized Eigen (lower graphic). The unoptimized code requires 80 separate scalar calculations for the broadcast operation versus just 5 operations for the optimized code.

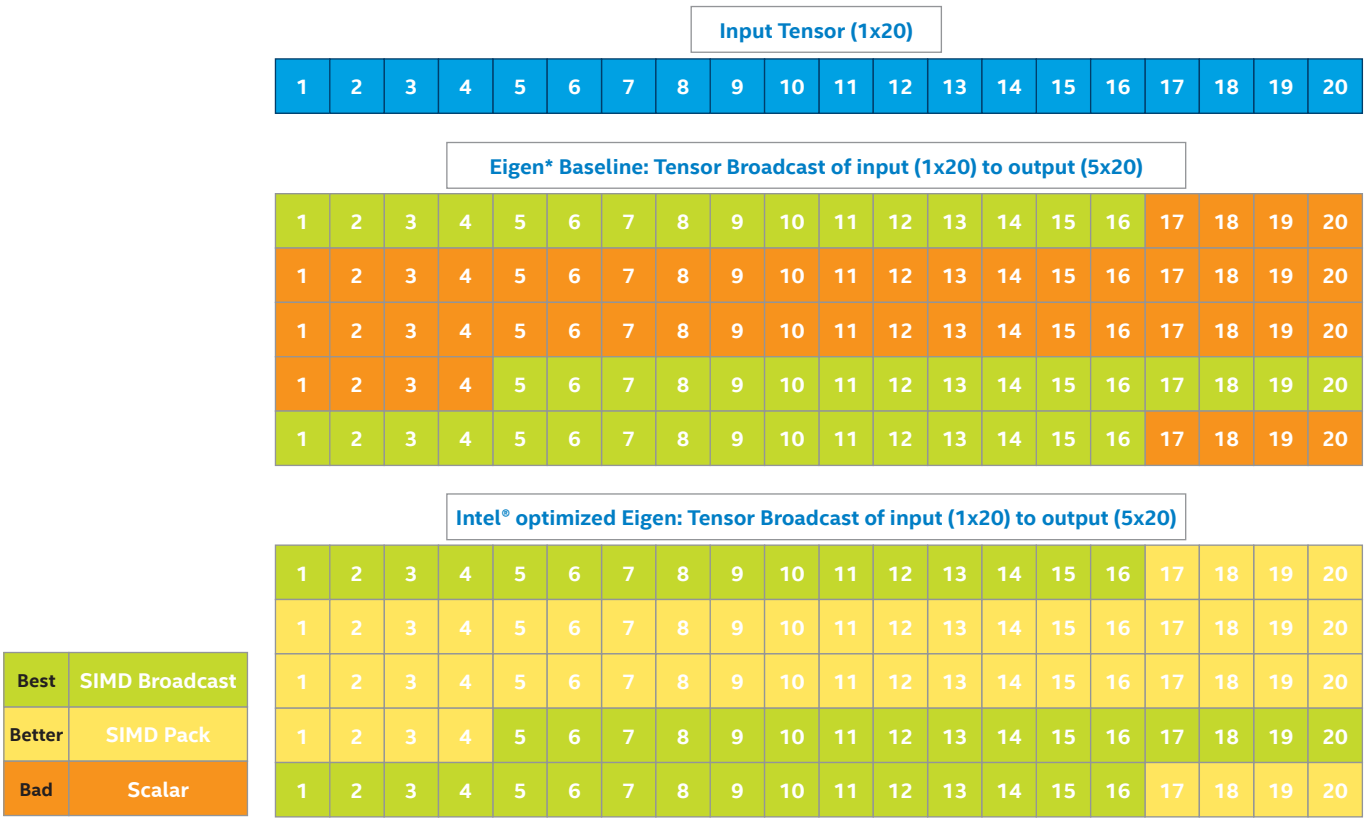

Figure 4. Comparison of a tensor broadcast of a 1x20 input tensor (packetOneByN class) into a 5x20 output tensor using unoptimized Eigen* (upper graphic) and using Intel® optimized Eigen (lower graphic). Although not as streamlined as the example in Figure 3, the optimized version replaces 52 scalar operations with much more efficient SIMD operations.

Before testing the impact of improved vectorization on TFS performance, the team benchmarked Eigen tensor broadcast independently of TFS by running Eigen on a single core of the Intel Xeon Platinum 8180 processor. For the Nx1 type of input tensors (packetNByOne class), the speedup was 58-65X[1]; for the 1xN type of input tensors (packetOneByN class), the speedup was 3-4X[1] (Figure 5).

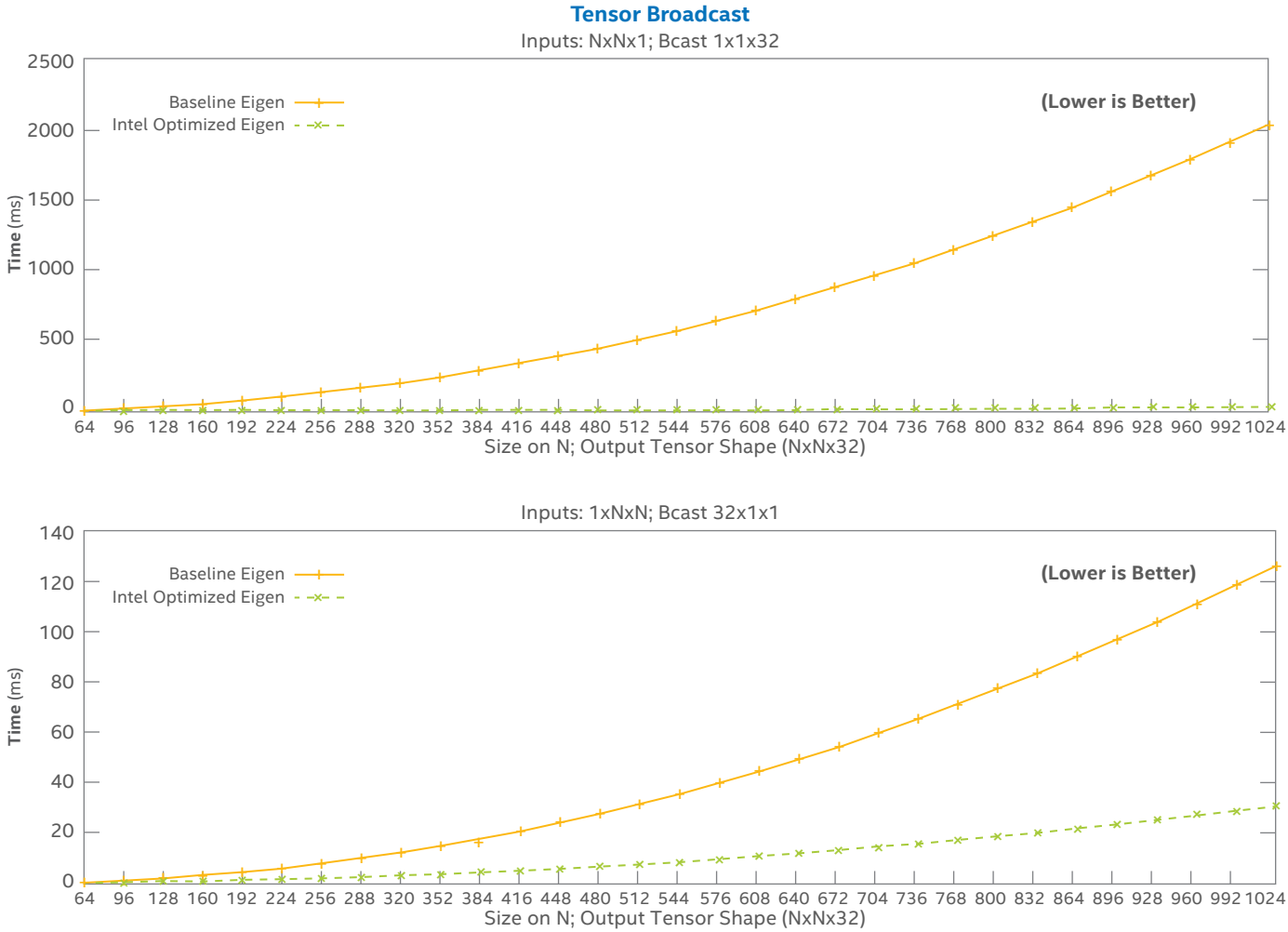

Figure 5. Eigen* Performance Gains: Performance comparison of tensor broadcast operations with and without Intel® optimizations (lower is better). The 58-65X[1] performance gains shown in the upper graph are applicable to packetNByOne input tensors; the 3-4X[1] gains shown in the lower graph are applicable to packetOneByN input tensors.

Finally, the team compared performance for the optimized and unoptimized versions of TFS. All three optimizations were used: 1) Intel MKL-DNN; 2) two instances of TFS with CPU and NUMA pinning, and 3) vectorized tensor broadcast operations. With all three enhancements, the performance of the optimized code was 2.5X[1] that of the original, unoptimized code (Figure 1).

Built-in High Performance for Future AI Solutions

As in many Intel software optimization engagements, the work done with Taboola offers potential value for a much broader community. To better support emerging AI users, Intel generalized the tensor broadcast optimizations to support tensors of all dimensions, and then up-streamed the code improvements to the public distribution of Eigen. The optimized code will be included in TensorFlow-Serving release 1.10. As a result, future Eigen and TFS users will benefit from significantly faster tensor broadcasting when running their applications on Intel Xeon Scalable processors, or on the many other Intel® processors that support Intel AVX, Intel® Advanced Vector Extensions 2 (Intel® AVX2), and Intel AVX-512.

A Scalable Path Forward for AI Developers

Adding SIMD capabilities to software is fundamental to optimizing performance on modern processors. The techniques used in optimizing TFS and Eigen can be applied to many other software codes and can potentially deliver major performance gains for a wide range of applications running on Intel processor-based platforms. For commonly used neural network primitives, AI developers can rely on Intel MKL-DNN to get the optimal performance on Intel processors. Identifying and optimizing the most time-consuming code segments is an iterative process that offers a path toward unleashing even higher performance on both current and future hardware platforms.

Conclusion

Taboola has achieved rapid, worldwide growth by matching individuals with brand and editorial content that's interesting and relevant to them across the open web. Both speed and accuracy are fundamental to the success of Taboola's discovery platform, and a highly optimized AI framework on Intel architecture makes it easier to achieve these goals without overspending on hardware infrastructure.

Intel continues to deliver new hardware optimizations with each new processor generation and collaborates with both open source communities and commercial organizations to help unleash the full performance benefits in real-world deployments. In recent processor generations, many of these advances have targeted the heavy processing demands of AI workloads. Taking advantage of these advances can help organizations build better AI solutions today using their existing infrastructure. It can also help them scale their solutions more easily and cost effectively on future Intel Xeon Scalable processors.

[1] Performance results are based on Taboola and Intel testing as of 6 August, 2018 and may not reflect all publicly available security updates. System Configuration: Two-socket server configured with 2 x Intel® Xeon® Platinum processor 8180 (2.50 GHz, 28 cores), 192 GB DDR4@2666MHz memory (12 x 16 GB DIMMS), 1.5 TB Intel® SSD (SC2BX01), CentOS Linux* release 7.5.1804 (Core) (3.10.0-862.9.1.el7.x86_64); Baseline software application: TensorFlow-Serving r1.9 (https://github.com/tensorflow/serving); Intel Optimized software application: TensorFlow-Serving r1.9 + Intel MKL-DNN (https://mirror.bazel.build/github.com/intel/mkl-dnn/archive/0c1cf54b63732e5a723c5670f66f6dfb19b64d20.tar.gz) + optimizations (availability of optimizations expected in TensorFlow-Serving release 1.10).

[2] For these and other Taboola case studies, visit the Taboola website at https://www.taboola.com/resources/case-studies

 1218/RA/MESH/PDF 338507-001US