White Paper

Visual Technologies
Multi-device Enabling

Unlocking the Power of Intel® Deep Link

Part Two: Increasing Workload Throughput Using Intel® GPUs

This paper is part two in a series of white papers designed to provide details regarding openly available development tools that can be used to take full advantage of Intel® Deep Link Technology.

Together, these papers will introduce and demonstrate some of the tools and processes that can be used to leverage Deep Link and allow developers to build better performing and more efficient applications. Included will be use cases that showcase Deep Link's current and future potential.

This second paper focuses on applying Deep Link concepts for improving workload throughput using various backend options, presents Leela Chess Zero as a use case, and introduces oneDNN: a high performance library for deep learning network applications.

Authors

Nick Yang
Senior Software Application Engineer

Max Domeika
Principal Engineer

Table of Contents

Introduction

With the release of the Intel® 11th Generation mobile processor and the Intel® Iris® Xe and Intel® Iris® Xe MAX graphics architecture, Deep Link was introduced to the world and a new era of innovation was born.

Developers now have the ability to strategically apply computing power that was previously unavailable, and to assign tasks to parts of the machine which would otherwise just lie dormant. Imagine having the ability to significantly boost the performance of your application using not much more than a strategic approach and some lines of code.

That is the power of Deep Link.

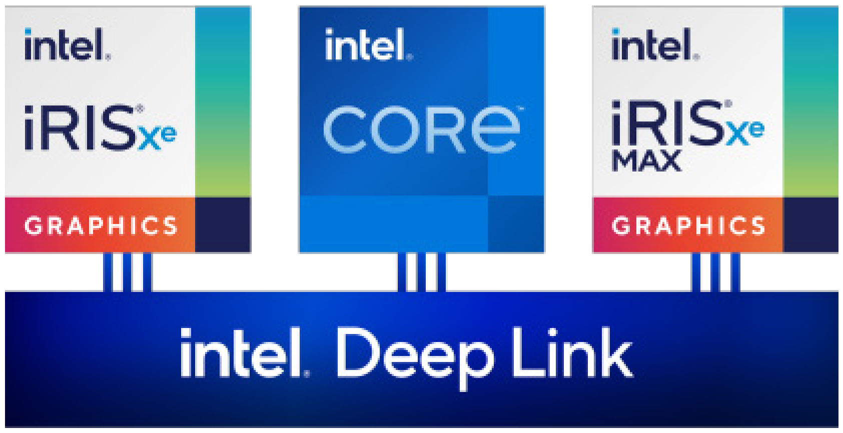


Figure 1. Intel® Deep Link combines an 11th Generation processor with an Iris® Xe integrated GPU and an Iris® Xe MAX discrete GPU, and manages the use of multiple GPUs simultaneously.

Deep Link Technology

At its core, Intel® Deep Link Technology is a reimagining and reshaping of the way that a machine's Central Processing Unit (CPU) and Graphics Processing Units (GPUs) interact. Already available on a number of devices, Deep Link offers the ability to combine the computing power of a discrete GPU with that of a powerful integrated GPU.

Using Deep Link, complex workloads and pipelines can be constructed which use multiple computing elements with a high degree of code re-use, thereby simplifying code development and reducing overall effort. This approach offers significant gains in both performance and efficiency, boosting the functional capabilities of a given application by offering expanded computing capabilities and processing options.

Deep Link Puts Multiple GPUs to Work

By partitioning computational elements into logical segments, Deep Link GPUs can equitably share the workload - each one working independently from and concurrently with the other. The Intel® Iris® Xe (also referred to as the integrated, or iGPU) and Intel® Iris® Xe MAX (also referred to as the discrete, or dGPU) can be used together to equitably split tasks, often allowing a workload to be completed in roughly half the time.

Deep Link Gets the Most out of Intel® Hardware

Deep Link enables Intel® computing components to work together at a level of speed and efficiency that was not available before by combining the computing power of multiple GPUs with similar characteristics. Because the two graphics processors use the same kernel code and have similar computing power and performance characteristics, there is little additional overhead introduced when partitioning tasks between the two.

Deep Neural Networks

For as long as computers have been around, there has always been a desire to find ways to help computers think and act (and even look) more human. For many years, any realization of this desire was limited to the clean rooms of tight-lipped technology companies or the imaginations of authors and movie directors. In the last fifteen years, however, there has been an explosion of consistent advancement in not only the ability but also the availability of Artificial Intelligence (AI) systems, and those advances are beginning to find their way into our everyday lives.

Deep Neural Networks (DNNs) are one of the gigantic leaps forward, giving computers the ability to see and understand a situation and to act upon the information that it receives. Built in part to resemble the way that the human mind is believed to process information, a DNN is similarly made up of neurons (the cells which receive, transform and relay inputs) and synapses (the ties that bind the neurons together with other neurons). In a computational neural network, the neurons are assembled in a hierarchical pattern and separated into individual layers. Each DNN has an input layer, an output layer, and one or more internal 'hidden' layers, and each neuron can be connected to any or all of the neurons in the adjacent layers, as shown in Figure 2 below. The direction of the flow of information is generally from input to output, but it can change depending on both the overall function and the current state or status of the DNN.

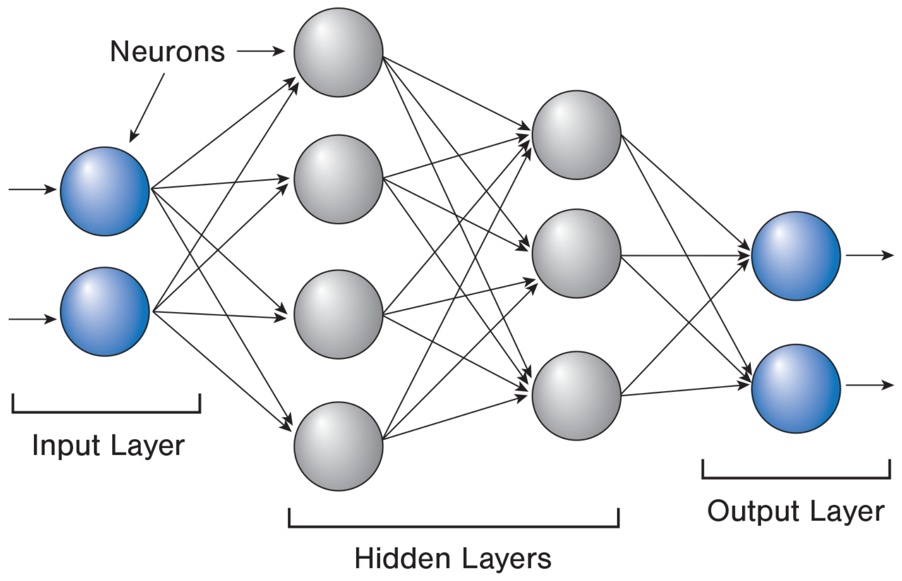


Figure 2. A simple depiction of a Deep Neural Network showing its basic components, interconnectivity and typical information flow.

Weights and Biases

Weight and bias values are assigned to each neuron according to its function and its effect on the next layer as data moves along on its way to the output layer. Weights and biases are multipliers that are applied to each neuron within the network, and are typically indicated and stored as a number between 0.0 and 1.0. The weight of a neuron indicates its relative importance as compared to other neutrons in its layer, and the bias increases or decreases the threshold required to satisfy or activate the neuron function.

The weights and biases are initially created using a random value, and are then modified as necessary during the training phase to ensure that their effect on the data being processed is in-line with the desired (correct) output.

Deep Neural Network Training

Finding the right balance of weights and biases throughout the network typically requires some level of human intervention. During training, when a DNN output is wrong, it is instructed that the output is incorrect and the weights and biases along the path that led to the incorrect output must be adjusted. These adjustments are made to the neurons along the errant path to prevent the same error from occurring in future calculations. This method of DNN training is known as supervised learning, and the adjustment of the weights and biases is known as backpropagation.

But manual training on that level is not always necessary. Over the last few years, developers have begun work on applications where a DNN-based AI engine is provided with a basic environment and basic goals and then equipped and encouraged to explore all possible avenues and solutions to find the best one. This approach has led to achievements in some areas which some thought would never be possible, including the playing of games that experts believed could only be played by humans who could apply imagination, creativity and ingenuity. One of the areas where the advancements of self-trained DNNs is most evident is in the playing of strategy games. No longer do AI engines require vast amounts of input from game masters to program responses to all potential moves by its opponents. Through repeated self-play, at a rate of hundreds or even thousands of games per day, DNN engines can learn for themselves how to competently play complicated strategy games, like chess.

And in the world of AI chess, Leela Chess Zero has built itself into a champion.

AI Application: Leela Chess Zero

Although computer chess has been around in one form or another for decades, Leela Chess Zero is unique - not only in its application but also in its approach to gameplay. Also known to chess enthusiasts as Lc0, Leela Chess Zero is unlike other chess engines that preceded it. Lc0 is built around a neural network architecture which is completely open-source and relies on a network of enthusiasts, programmers and designers to provide the computing horsepower needed to train and deploy its neural network-based chess engine.

Similar to AlphaZero, the DeepMind-developed, Google-owned proprietary chess engine that it was based on, Lc0 was initially provided only with the rules of chess, including the conditions that result in the game ending (checkmate, stalemate, draw). No information was provided with respect to any openings, tactics, strategy, or even the relative usefulness or value of the various pieces. Lc0 was left to figure all of those things out on its own after simply being programmed to go and play. The programmers provided Lc0 with a basic DNN structure to assist in exploring options and making decisions, but all of the lessons were initially learned by the AI through self-play and applied by adjusting the weights and biases that represent the variable components of its neural network.

Lc0 Network Topology

While Lc0 utilizes what in many ways is a typical neural network structure, there are some key specializations that are designed specifically for playing and managing chess matches. The input block of Lc0's neural network is an 8x8x112 image stack which stores positional data representing the current game state, or node (i.e., the current position of all game pieces on the board for both sides). The stack also stores the position following each of the previous seven moves, as well as some data regarding other specific situational game states (such as castling and en passant).

The input layer feeds a number of residual blocks. The number of residual blocks can change depending on the specific implementation. Lc0 networks are categorized based on the specific number of blocks and filters that make up each network (often notated as just the number of blocks due to the fact that the number of filters is generally proportional to the number of blocks). The more common configurations use either 10, 15, 20, 24 or 30 blocks, and each block has a number of individual layers. Filters represent the number of convolutions in a convolution layer. The Lc0 deep neural networks can range from 10 blocks x 128 filters to 30 blocks x 384 filters. Greater numbers of blocks and filters generally result in a higher level of play, but require more hardware computing power (typically one or more GPUs), more time to process and evaluate positional data, and more time to train. For this reason, a 10b Lc0 network might choose 2-3 actions to explore, while a more powerful 24b network might choose 6-8 actions or more.

Within each residential block is a series of layers and filters, each with its own exploration/evaluation function. Using a series of residual blocks in this way ensures that as each node is evaluated throughout a match and the training of the neural network progresses, the changes made to each neuron are small steps instead of giant leaps (this results in steady course correction instead of large corrective measures which are more likely to lead to overcorrection and errors).

The stack of residual blocks form a residual tower (also known as a Residual Network, or ResNet), with the input block at one end and separate Policy and Value heads acting as the output at the other end. The Policy head is used to calculate the probability that an action will be chosen, while the Value head calculates the likelihood that the game will end in victory if recommended action is chosen.

AI Implementation

From any given position during a chess game (referred to in AI chess as the Root Node), there is a finite number of legal moves that can be made. As the chess engine begins to consider the possible moves that branch out from those positions two, three, four or more moves deeper into the game (as well as the opponent's correlating moves) this creates a seemingly infinite[1] number of possible board positions - each with its own advantages and disadvantages as viewed from one side of the board or the other.

Managing the size and scope of all of the various move options and their consequences during each turn is a tall task, even for a machine capable of performing billions of calculations per second. There is also a limit on the time allotted for each turn. For these reasons, Lc0 utilizes a number of strategies to narrow its focus during gameplay.

One of these strategies is the use of Predictor + Upper Confidence Tree search (PUCT) to use its current position to evaluate the possible actions and decide upon its next move. For example, at the very beginning of a chess game, there are twenty legal moves that can be made (as shown in Figure 3 below).

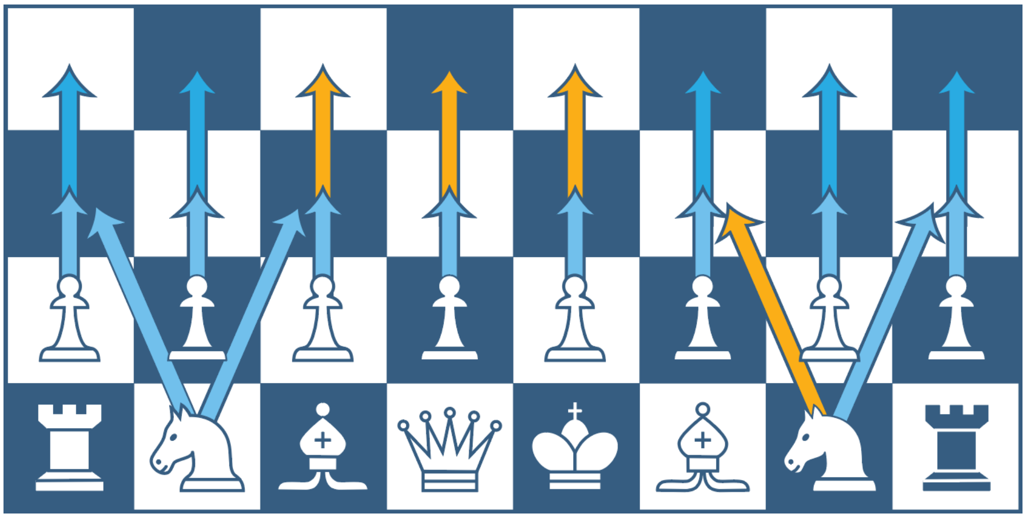


Figure 3. The twenty legal opening moves that can be made at the beginning of a chess match, with the four most popular shown in orange.

1 - It is estimated that there are more potential chess board position variations than there are atoms in the known universe, and according to those estimates it's not even close (Chess: 10^{123} board positions; Atoms in the known universe: 10^{78}).

For each possible move (known in AI chess as an action) from the current state, there are 4 numbers that are stored and used to determine which potential moves should be explored further. For each possible move, an Lc0 network has stored in memory:

(n)	The number of times that this action has been taken from this specific state
(w)	The total value of the next state if this action is taken
(q)	The mean value of the next state if this action is taken
(p)	The probability of selecting this action

A fifth value (u) is determined by evaluating and comparing the (p) and (n) values to identify actions which have been used less often compared to the other actions, or to identify actions which have a significantly higher or lower probability of success than the other actions.

When deciding which potential actions to explore, Lc0 compares the (q) and (u) values for each action and chooses the actions with the maximum benefit. This does not always represent the actions with the highest (q)+(u) total, because the value of either (q) or (u) may be weighted higher or lower depending on the maturity of the network and the current training stage.

Figure 4. The Leela Chess Zero logo.

Once the actions with the greatest potential have been determined, Lc0 runs simulations for each action and the most likely potential follow-up actions from both sides. Each option is followed through the most likely scenarios all the way to the end of the match (called a playout).

As Lc0 works its way through the PUCT decision tree for each action, the workload is stored in a batch buffer and sent to the compute device(s). If there are multiple GPUs available for use, there are two ways to handle the distribution of the workload:

1. A split workload approach, where the data stored in the batch buffer is split into multiple sub-batches and each sub-batch is distributed to the individual GPUs to be processed in parallel.

2. A round-robin approach, where each GPU is assigned a specific portion of the workload to process and then comes back to be assigned another portion once it is finished. The process is repeated over and over until the entire workload is complete.

With either approach, the processed data is accumulated in common output storage for aggregation. After running through thousands or tens of thousands of simulations, each action is assigned a value which represents the number of times that it was tried and the number of times it led to a win (ratio of wins to total playouts). The decision as to which move is best is made based on an evaluation of the entirety of the workload, and the action with the greatest potential for starting - or continuing - the path to a winning match is chosen[2].

For this reason, the network may have to evaluate hundreds of thousands of nodes as it progresses through all of the positional variables and their pros and cons. This requires a massive amount of computational capability and speed. The most widely used metric for measuring the speed at which the network is able to carry on these simulations is Nodes per Second (NPS). The NPS rate that a network is capable of is largely based upon two characteristics: the topology of the network and the characteristics of the hardware platform.

2 - This assumes that the match is being played to win, which is not always the case. Especially early in its training, a network may be configured to choose moves stochastically (in a measured but much more random fashion) to ensure that all potential moves from a given node are chosen roughly the same number of times. As the network moves along in training and begins playing competitive matches - against humans or other AI networks - it is much more likely that the action chosen using the PUCT method is the one that is selected.

How an AI Chess Engine Improves

But why is computing power important to a chess engine? After playing so many games, isn't gameplay for a computer a simple matter of remembering what it has done in the past and repeating it? Well, sort of, but there is a lot more involved for a top-rated AI chess engine like Lc0, and those differences require horsepower.

Just like a human chess player, every game that the AI can play (either against itself or an opponent) is an opportunity to fine-tune its decision making and become a better player. With its ability to play thousands of games per day, Lc0's learning curve is very steep, but along with that ability to learn quickly comes the need for large amounts of computing capability.

Lc0 relies on a group of individuals and companies from around the world who devote compute space on their machines to be used by Lc0 in its training. Community-lent CPUs and GPUs are organized, combined and coordinated to form networks that are used to harness the power of many computers at once. This unique crowd-computing approach gives Lc0 the computing power necessary to play millions of games every day.

Can a Computer Possess the Desire to Win?

Although a computer cannot be made to desire enjoyment, fulfillment or gratification through gameplay the way a human does, it can easily be programmed to seek out what it has been told is a more desirable outcome. Leela Chess Zero has been programmed and trained this way using a process known as reinforcement learning.

In reinforcement learning, computers are given the opportunity to freely explore the available options within their environment, to carry those options through to their natural conclusion, and then to evaluate whether or not the options chosen resulted in a successful outcome. If the outcome is successful, the weights and biases of the neurons that led to the successful result (checkmate) can be adjusted to make them a more favorable option in the future. The opposite is true for neurons that led to an unsuccessful outcome (stalemate or a lost match). As more and more playouts are performed and as options and paths are explored (often at random, especially when a network is newer and has run fewer games), this fine-tuning of weights and biases throughout the DNN leads to a trained weight file which can recognize positions and choose options that are most likely to end in a win.

Leela Chess Zero Tournament Championships

The Top Chess Engine Championship (TCEC) has been sponsoring championship tournaments for computer chess engines since 2010, and in January of 2019 Leela Chess Zero became only the fourth chess engine to win a TCEC tournament. The TCEC Cup 2 tournament featured 31 other chess engines, including all three former champions. This tournament win included head-to-head wins over two of the three former champions.

Since that win Lc0 has gone on to win five Chess.com Computer Chess Championships, including four in a row (Tournament 11 in 2019, and Tournaments 12, 13 and 14), has won the TCEC main tournament two times, and continually performs well in other tournaments, proving itself as one of the top chess engines in the world.

Lc0 Software

Regardless of whether you intend to use Lc0 as a player, as a community resource lender or as a developer, Lc0 requires the download and installation of two separate software components in order to run: the Lc0 engine executable (lc0.exe), and a network file.

The Lc0 executable and support files are available to download for free at:

https://lczero.org/play/download/

and support Windows, Linux, MacOS and Android installations. For Windows installations, it also supports a wide variety of backend options which can be used to suit different hardware configurations.

There are many different existing networks available for download on the lczero.org website, depending on the number of blocks and filters that you wish to employ, as well as the purpose (gameplay, training, CPU only vs. GPU inclusion, etc.). A list of some of the more popular networks is available by following this link:

https://lczero.org/play/networks/bestnets/

or the full list of available networks is available here:

https://training.lczero.org/networks/?show_all=0

(this link opens a truncated list of the available networks as the full list is extremely long - a link that can be used to display the full list is provided at the bottom of the target page).

In order to interact with the software, Lc0 also requires the use of a GUI. There are several options available due to the fact that Lc0 utilizes the standard Universal Chess Interface (UCI) open communication protocol. lczero.org lists BanksiaGUI and Nibbler as chess GUIs which are free and have Lc0 specific features.

Accelerating Lc0 with oneDNN

As described above, the Lc0 backend employs a very deep neural network that is used to evaluate multiple possible moves from a given position. This approach to network training and gameplay requires large amounts of processing power. Using Intel Deep Link technology, these node-based evaluation functions can be spread across multiple devices (CPU, GPUs) to vastly extend system capabilities without adding hardware.

Intel® oneDNN Overview

The Intel® oneAPI Deep Neural Network Library (oneDNN) helps developers improve productivity and enhance the performance of their deep learning frameworks. oneDNN consists of a set of building blocks, known as primitives, that enable developers to assemble a deep neural network quickly by using pre-configured oneDNN APIs.

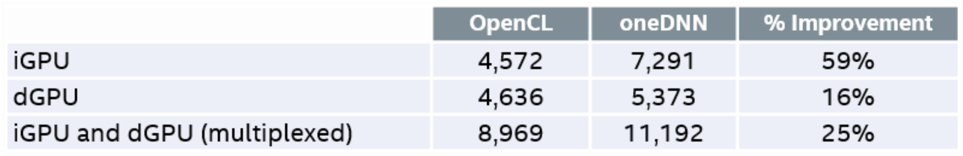
Information regarding all supported oneDNN primitives can be found online at:

https://oneapi-src.github.io/oneDNN/group_dnnl_api_primitives.html

along with a developer guide, code samples and more. Developers can use the same APIs to develop networks running on CPUs, GPUs or both. When used in applications, the best performance option is selected or compiled at runtime based on the detected hardware features. All oneDNN primitives are optimized based on the specific hardware capabilities of the targeted devices.

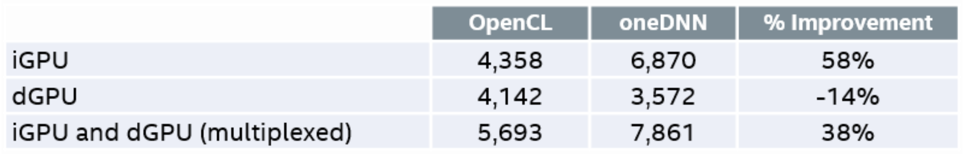
Lc0 Performance Using oneDNN

Lc0 has a built-in benchmarking feature which is used to measure performance (measured in NPS). This benchmark was used to compare the performance of Lc0 using two different backend options: OpenCL and oneDNN. The following tables clearly show that there are significant gains to be made when combining the capabilities of multiple GPUs using Deep Link Technology.



	OpenCL	oneDNN	% Improvement
iGPU	4,572	7,291	59%
dGPU	4,636	5,373	16%
iGPU and dGPU (multiplexed)	8,969	11,192	25%

Table 1. OpenCL vs. oneDNN throughput comparison on the Intel Software Development Platform.



	OpenCL	oneDNN	% Improvement
iGPU	4,358	6,870	58%
dGPU	4,142	3,572	-14%
iGPU and dGPU (multiplexed)	5,693	7,861	38%

Table 2. OpenCL vs. oneDNN throughput comparison on the ASUS VivoBook.

NOTE: The data shown in the table above was collected running Lc0 v0.28-RC1 with oneDNN v2.3, and the network used was 591226. All oneDNN vs. OpenCL benchmark comparison data was collected using identical parameter options when possible. One DNN is running in fp16 and OpenCL is running in fp32 (Lc0 does not support fp16 with OpenCL).

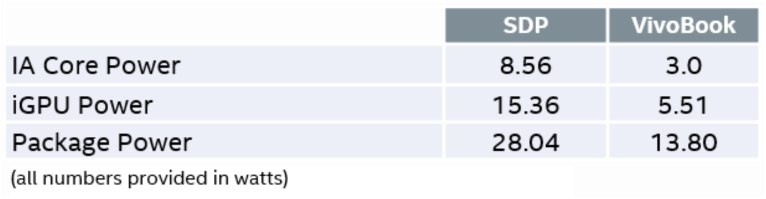
Lc0 using oneDNN is available for download from the github Leela Chess Zero page here:

https://github.com/LeelaChessZero/lc0/releases/tag/v0.28.0

Power and Thermal Considerations

Power sharing and management between the CPU and GPU devices in Deep Link applications is performed using one or more standard power policies. Using this approach, power can be distributed on an as-needed basis while remaining within the overall power budget.

As is always the case in computing, excess heat generated by the system increases as the processors are taxed. This invariably leads to will cause performance degradation. In this case, the increase in heat associated with engaging both GPUs has a significant impact on performance. To combat this, the VivoBook power was throttled down to prevent overheat, which is why the data in Table 2 shows significantly slower NPS speeds than in Table 1, despite the fact that the software configuration was virtually identical. Table 3 (below) shows the actual power consumption differences between the two hardware platforms.



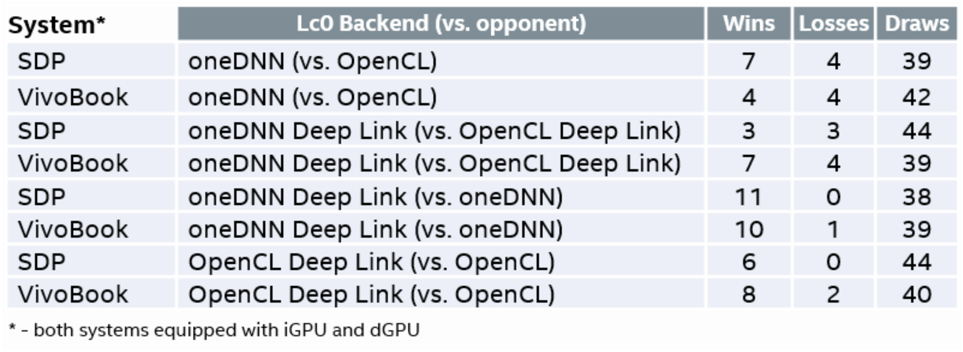
	SDP	VivoBook
IA Core Power	8.56	3.0
iGPU Power	15.36	5.51
Package Power	28.04	13.80

(all numbers provided in watts)

Table 3. Comparison of power allotment in the software development platform vs. the ASUS VivoBook.

Backend Head-to-Head Tournament

Pitted against each other in a series of eight 50-match, head-to-head tournaments, the oneDNN backend showed that it is more than capable of holding its own when compared directly to the OpenCL backend. In addition, the application of Deep Link Technology showed that the ability to efficiently use both GPUs simultaneously offered benefits to both computing systems, regardless of which backend was being used.



System*	Lc0 Backend (vs. opponent)	Wins	Losses	Draws
SDP	oneDNN (vs. OpenCL)	7	4	39
VivoBook	oneDNN (vs. OpenCL)	4	4	42
SDP	oneDNN Deep Link (vs. OpenCL Deep Link)	3	3	44
VivoBook	oneDNN Deep Link (vs. OpenCL Deep Link)	7	4	39
SDP	oneDNN Deep Link (vs. oneDNN)	11	0	38
VivoBook	oneDNN Deep Link (vs. oneDNN)	10	1	39
SDP	OpenCL Deep Link (vs. OpenCL)	6	0	44
VivoBook	OpenCL Deep Link (vs. OpenCL)	8	2	40

* - both systems equipped with iGPU and dGPU

Table 4. Comparison of backend performance in a head-to-head tournament.

Test Configuration

All throughput totals presented in this paper were collected using one of two machines: (1) a Deep Link-enabled ASUS® VivoBook™, and (2) an Intel internal Software Development Platform (SDP). Information on both systems is shown below. All tests were performed in August, 2021.

VivoBook system information:

Laptop Model: TP470EZ
Processor: 11th Gen Intel® Core i7-1165G7
OS: Windows® 10 Home v.2004, build 19041,1165
BIOS Version and Date: American Megatrends International, LLC. TP470EZ.300, 11/4/2020
Memory: 16GB DDR4
Graphics: Intel® Iris® X^e (integrated); Intel® Iris® X^e MAX (discrete)
Graphics Driver Version: 30.0.100.9684 for both GPUs

SDP system information:

Processor: 11th Gen Intel® Core i7-1185G7
OS: Windows® 10 Pro v.20H2, build 19042,1165
BIOS Version and Date: Intel Corporation TGLSFWI1.R00.3373.A00.2009091720, 9/9/2020
Memory: 16GB DDR4
Graphics: Intel® Iris® X^e (integrated); Intel® Iris® X^e MAX (discrete)
Graphics Driver Version: Intel® engineering build 8876 for both GPUs

Conclusion

Intel® Deep Link Technology offers new avenues and paths for data processing that are just begging to be explored, and tools like oneDNN™ make increase throughput to exceptional levels. By combining Deep Link with tools like oneDNN, you can speed up virtually any application by spreading compute functions across multiple devices. Developers can also create applications more quickly using the primitives available in the library, and they can further benefit from the fact that they are already optimized to work alongside Intel hardware devices.

It is our sincere hope that you have found the information in this white paper helpful and informative. Other papers in this series promise to bring to light other tools that can be paired with Deep Link to provide exceptional processing speed and performance for your new and existing applications. From dynamic power sharing to accelerated video rendering and much more, the opportunities for performance improvement are nearly endless.