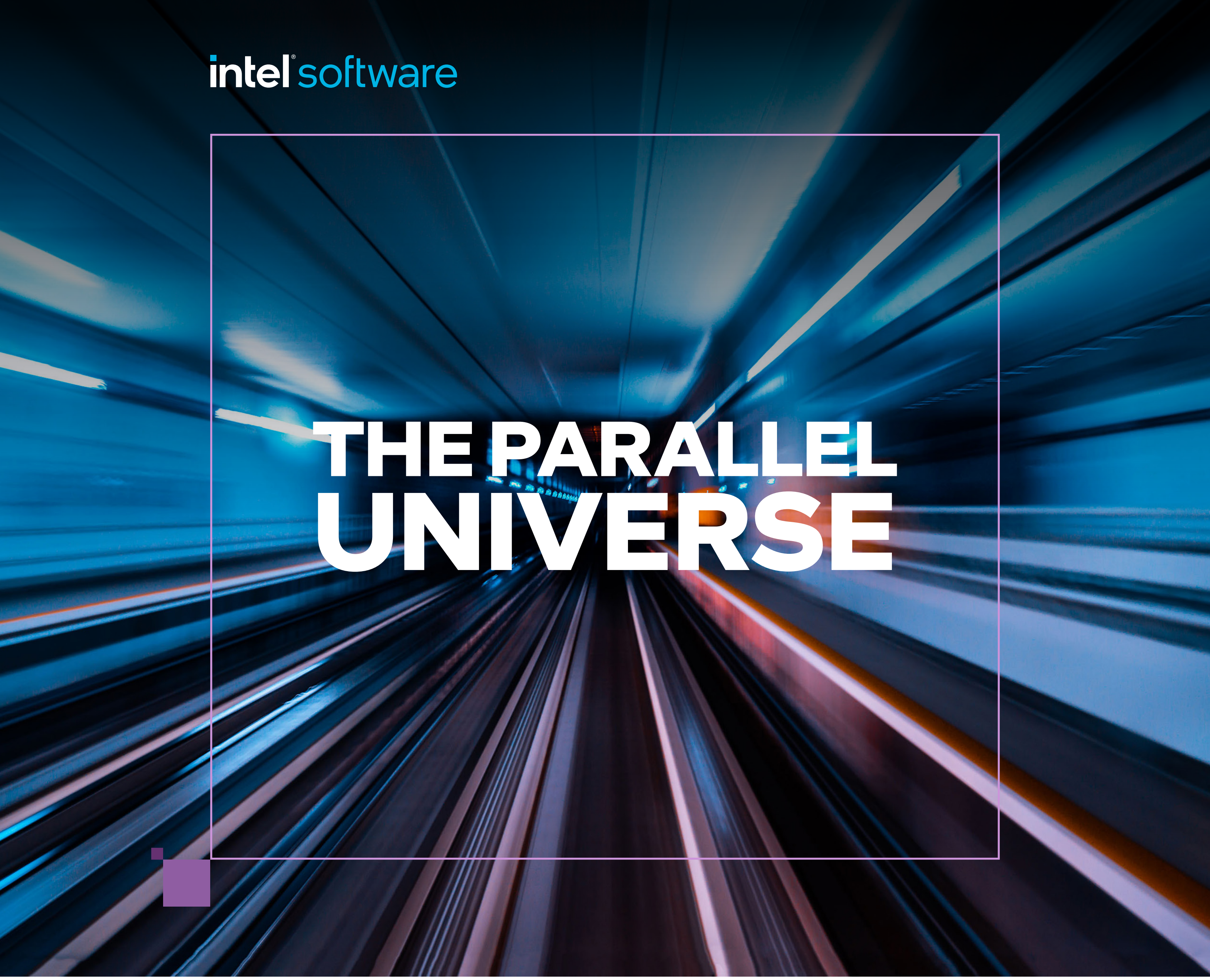
intel software



SYCLomatic: A New CUDA*-to-SYCL* Code Migration Tool

Free Your Software from Vendor Lock-in Using SYCL and oneAPI

The SigOpt Intelligent Experimentation Platform

Issue 49 2022

Contents

Sign up for future issues

Letter from the Editor

Henry A. Gabb, Senior Principal Engineer at Intel Corporation, is a longtime high-performance and parallel computing practitioner who has published numerous articles on parallel programming. He was editor/coauthor of "Developing Multithreaded Applications: A Platform Consistent Approach" and program manager of the Intel/Microsoft Universal Parallel Computing Research Centers.

Intel to Acquire Codeplay Software

Earlier this month, Intel signed an agreement to acquire Codeplay. Why does this matter? Codeplay is a leader in open-standard, cross-architecture development. This acquisition will help drive adoption of SYCL* and the oneAPI ecosystem. From Intel to Acquire Codeplay Software:

"Codeplay is globally recognized for its expertise and leadership in SYCL, the Khronos open standard programming models used in oneAPI, and its significant contributions to the industry ranging from open ecosystem activities like SYCL and OpenCL to RISC V* ... Codeplay has extensively delivered products supporting diverse hardware platforms globally, embracing the mission of bringing oneAPI to the masses."*

To celebrate, our first two articles, including one from Codeplay, cover SYCL. The feature article, **SYCLomatic: A New CUDA*-to-SYCL Code Migration Tool**, describes this new open-source project. SYCLomatic will give users an opportunity to contribute and provide feedback to improve the tool. It enables community collaboration to advance adoption of the SYCL standard, a key step in freeing developers from a single-vendor proprietary ecosystem. This is followed by **Free Your Software from Vendor Lock-in Using SYCL and oneAPI**, in which Rob Burns and Joe Todd from Codeplay demonstrate how SYCLomatic performs in practice.

The SigOpt Intelligent Experimentation Platform discusses the challenges of running modern recommender systems and shows how to democratize the end-to-end process of creating such systems. **Distributed Training on Intel® Xeon® Scalable Processors** presents a case study with our collaborators at Tencent on training an AI model for the Tencent AI Arena platform.

Some of you may know that my academic background is in life sciences, so it's gratifying to have two genetics articles. I coauthored **Delivering Cost-Effective Genomics for Precision Medicine** with Don Freed and Zhipan Li from Sentieon Inc. We used a genomics benchmark from the U.S. Food and Drug Administration to show that Intel Xeon processors give superior performance and TCO than the competition. In **Accelerating Single-Cell Genetics Analysis**, Intel Labs and a team of collaborators show how Intel Xeon processors deliver superior performance for an end-to-end genetics pipeline. These articles also describe how our competition sometimes hobbles Intel Xeon processor performance to inflate their speed-up results.

Sign up for future issues

We close this issue with an interesting demonstration of building oneAPI for Linux* and Windows* using the Windows subsystem for Linux 2* and Visual Studio Code*. There are plenty of interesting tidbits in **Leveraging Tools for Cross-Platform Software Development**, but seeing Linux and Windows running side-by-side in the same Visual Studio Code instance stands out for me. I hate switching between Linux and Windows on dual-boot systems. This article shows that it might finally be a thing of the past.

As always, don't forget to check out Tech.Decoded for more information on Intel® solutions for code modernization, visual computing, data center and cloud computing, data science, systems and IoT development, and heterogeneous parallel programming with oneAPI.

Henry A. Gabb
July 2022

*Other names and brands may be claimed as the property of others.
For more complete information about compiler optimizations, see our Optimization Notice.

intel. software

Code for the Future.

Grow beyond proprietary boundaries.

oneAPI

Expand your code's reach with a single, open programming model that supports multiple languages to deliver heterogeneous computing performance.

Rooted in open standards, oneAPI offers cross-architecture libraries, compilers and tools that open your code to more hardware choices—for unparalleled performance.

Discover oneAPI →

SYCLomatic: A New CUDA*-to-SYCL* Code Migration Tool

Contributions to this Open-Source Project Wanted!

Kent Moffat, Senior Product Manager, Intel Corporation

To achieve high performance and efficient developer productivity across CPUs, GPUs, FPGAs, and other architectures, developers need a unified programming model that enables them to select the optimal hardware for the task at hand. They need a high-level, open-standard, heterogeneous programming language that's both built on standards and extensible. It must boost developer productivity while providing consistent performance across architectures. SYCL*, a C++-based Khronos Group standard, addresses these challenges by extending C++ capabilities to support multiarchitecture and disjoint memory configurations.

To make it easier to adopt SYCL, developers may want to migrate their existing CUDA* GPU code so they don't have to start their SYCL development from a blank page. We've previously published articles on using the Intel® DPC++ Compatibility Tool to migrate CUDA to SYCL. This tool is included in the Intel® oneAPI Base Toolkit and supported by Intel technical consulting engineers.

SYCLomatic Open-Source Project

In response to developer requests, the compatibility tool has now been released as an open-source project under the name "SYCLomatic." Many organizations have successfully used the tool, and some also wanted to enhance and customize its capabilities to tune it to their needs. One of those organizations was Argonne National Laboratory.

"CRK-HACC is an N-body cosmological simulation code actively under development. To prepare for Aurora, the Intel DPC++ Compatibility Tool allowed us to quickly migrate over 20 [CUDA] kernels to SYCL. Since the current version of the code migration tool does not support migration to functors, we wrote a simple Clang tool to refactor the resulting SYCL source code to meet our needs. With the open-source SYCLomatic project, we plan to integrate our previous work for a more robust solution and contribute to making functors part of the available migration options," said Steve (Esteban) Rangel of HACC (Hardware/Hybrid Accelerated Cosmology Code), Cosmological Physics & Advanced Computing (anl.gov).

Utilizing the Apache* 2.0 license with LLVM exception, the SYCLomatic project hosted on GitHub offers a community for developers to contribute and provide feedback to further open heterogeneous development across CPUs, GPUs and FPGAs. The GitHub portal includes a "contributing.md" guide, describing the steps for technical contributions to the project. Developers are encouraged to use the tool and provide feedback and contributions to advance the tool's evolution. This open-source project enables community collaboration to advance adoption of the SYCL standard, a key step in freeing developers from a single-vendor proprietary ecosystem. Improvements made to SYCLomatic will also be incorporated in the Intel DPC++ Compatibility Tool product.

How the SYCLomatic Tool Works

SYCLomatic assists developers in porting CUDA code to SYCL, typically migrating 90–95% of CUDA code automatically to SYCL code.[1] To finish the process, developers complete the rest of the coding manually and then tune to the desired level of performance for the target architecture (**Figure 1**).

[1]Intel estimates as of September 2021. Based on measurements on a set of 70 HPC benchmarks and samples, with examples like Rodinia, SHOC, PENNANT. Results may vary.

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

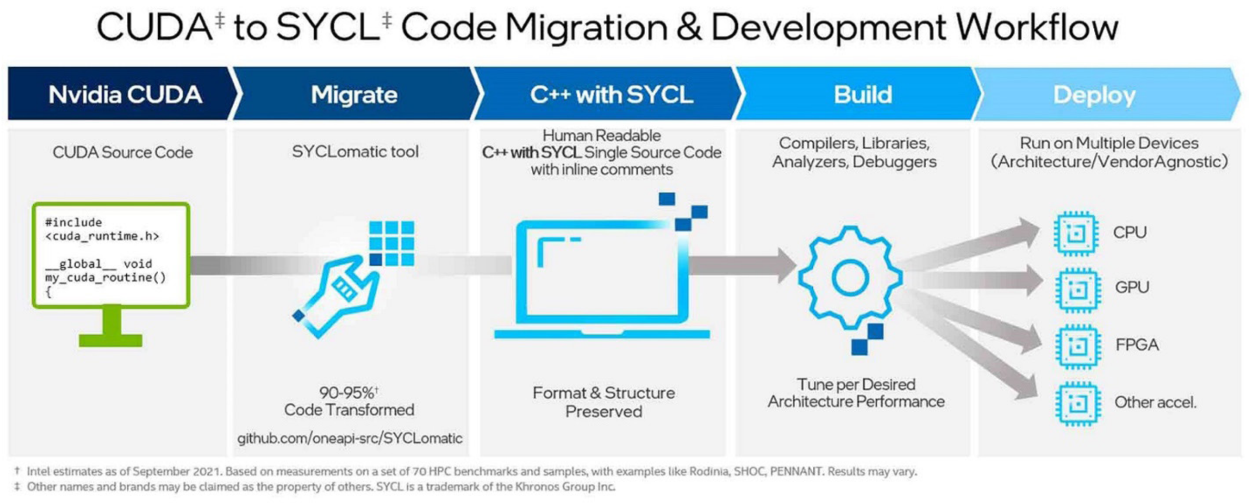

CUDA‡ to SYCL‡ Code Migration & Development Workflow
Nvidia CUDA
Migrate
C++ with SYCL
Build
Deploy
CUDA Source Code
#include <cuda_runtime.h>
__global__ void my_cuda_routine() {
SYCLomatic tool
Human Readable C++ with SYCL Single Source Code with inline comments
Compilers, Libraries, Analyzers, Debuggers
Run on Multiple Devices (Architecture/VendorAgnostic)
CPU
GPU
FPGA
Other accel.
90-95%† Code Transformed
github.com/oneapi-src/SYCLomatic
Format & Structure Preserved
Tune per Desired Architecture Performance
† Intel estimates as of September 2021. Based on measurements on a set of 70 HPC benchmarks and samples, with examples like Rodinia, SHOC, PENNANT. Results may vary.
‡ Other names and brands may be claimed as the property of others. SYCL is a trademark of the Khronos Group Inc.

Figure 1. The SYCLomatic workflow.

Successful Code Migrations

Research organizations and Intel customers have successfully used the Intel DPC++ Compatibility Tool, which has the same technologies as SYCLomatic, to migrate CUDA code to SYCL (or Data Parallel C++, oneAPI's implementation of SYCL) on multiple vendors' architectures. Examples include the University of Stockholm with GROMACS 2022, Zuse Institute Berlin (ZIB) with easyWave, Samsung Medison, and Bittware. (Go to oneAPI DevSummit content for more examples.) Multiple customers are also testing code on current and upcoming Intel® Iris® X^e architecture-based GPUs, including the Argonne National Laboratory Aurora supercomputer, Leibniz Supercomputing Centre (LRZ), GE Healthcare*, and others.

Example: Migrating CUDA Vector Addition to SYCL

To provide a practical overview of the migration process, this article uses a simple implementation of vector addition in CUDA*. We take a closer look at the code that SYCLomatic generates. Mainly, we focus on the code sections where CUDA and SYCL differ the most.

We'll be using SYCLomatic and the Intel® oneAPI DPC++/C++ Compiler from the Intel oneAPI Base Toolkit for the task at hand. To install the toolkit, follow the Intel® oneAPI installation guide. Use the following workflow to migrate your existing CUDA* application to SYCL*:

1. Use the *intercept-build* utility to intercept commands issued by the *Makefile* and save them in a JSON-format compilation database file. This step is optional for single-source projects.
2. Migrate your CUDA code to SYCL using SYCLomatic.
3. Verify the generated code for correctness and complete the migration manually if warning messages indicate this explicitly. Check the Intel DPC++ Compatibility Tool Developer Guide and Reference to fix the warnings.
4. Compile the code using the Intel oneAPI DPC++/C++ Compiler, run the program, and then check the output.

You can then use Intel's oneAPI analysis and debug tools, including Intel® VTune™ Profiler, to optimize your code further.

Let's take vector addition as an example. Vector addition involves adding the elements from vectors A and B into vector C. A CUDA* kernel computes this as follows:

```
__global__ void vector_sum(const float *A,
                           const float *B,
                           float *C,
                           const int num_elements)
{
    int idx = blockDim.x * blockIdx.x + threadIdx.x;
    if (idx < num_elements) C[idx] = A[idx] + B[idx];
}
```

In CUDA, a group of threads is a thread block equivalent to a workgroup in SYCL; however, we compute thread indexing differently. In CUDA, we use built-in variables to identify a thread. (See how we calculated the *idx* variable in the code above.) Once migrated to SYCL, the same kernel looks like this:

```
void vector_sum(const float *A,
                const float *B,
                float *C,
                const int num_elements,
                sycl::nd_item<3> item_ct1)
{
    int idx = item_ct1.get_local_range().get(2) *
              item_ct1.get_group(2) +
              item_ct1.get_local_id(2);
    if (idx < num_elements) C[idx] = A[idx] + B[idx];
}
```

Like a CUDA thread, a work item in SYCL has a global identifier in a global space or a local identifier within a workgroup. We can get these identifiers from the *nd_item* variable. So, we no longer need to compute the global identifier explicitly. However, this demonstration shows how we do it in SYCL, so we see the similarities to CUDA's built-in variables. Notice that *nd_item* is three-dimensional because of the *dim3* type in CUDA. In this context, we can make *nd_item* one-dimensional. This action maps a work item to each element in the vector. To run a CUDA kernel, we must set the block size and how many blocks we need. In SYCL, we must define the execution range. As the code below shows, we do this with an *nd_range* variable that combines the global range and local range. The global range represents the total number of work items, while the local range is the size of a workgroup.

```
const int num_elements = 512;
dpct::device_info prop;
dpct::dev_mgr::instance().get_device(0).get_device_info(prop);
const size_t max_block_size = prop.get_max_work_group_size();
const size_t block_size = std::min<size_t>(max_block_size, num_elements);

range<1> global_rng(num_elements);
range<1> local_rng(block_size);
nd_range<1> kernel_rng(global_rng, local_rng);
```

To invoke our SYCL kernel, we use a *parallel_for* and the execution range to submit the kernel to a queue. Each work item invokes the kernel once. We have the same number of work items for each vector element in this context. Let's see how this looks:

```
dpct::get_default_queue().parallel_for(kernel_rng, [=](nd_item<1> item_ct1)
{
    vector_sum(d_A, d_B, d_C, num_elements, item_ct1);
});
```

So far, we've explored how to implement and run a kernel. However, before running the kernel, we need to think about memory allocation and copy the data to the device.

1. First, we allocate memory for the operand vectors in the host and initialize them.
2. Then, we do the same on the device. CUDA uses the *cudaMalloc* routine. By default, the DPCT migrates this routine to *malloc_device*, which uses unified shared memory (USM).
3. Now, we use the *memcpy* command to copy the vectors from the host memory to the device.

After these steps, we run our kernel. Once the execution completes, we copy the result back to the host. We then check the result for correctness. Finally, we free the memory in the host and device by calling *free* and *sycl::free*, respectively.

Conclusion

The Khronos SYCL C++ standard is the open path for developing heterogeneous code that runs across multiple architectures. SYCLomatic, the new open-source project, provides the same CUDA to SYCL code migration benefits as the Intel DPC++ Compatibility Tool that came before it. And now, anyone can contribute to help improve and/or tune the tool to their needs. Give it a try today.

Resources for Developers

- Comparing programming models: CUDA and SYCL
- Learn about SYCL
- oneAPI specification
- Intel oneAPI toolkits
- SYCLomatic project on GitHub | Contributing.md guide
- Get started developing: Book: Mastering Programming of Heterogeneous Systems using C++ & SYCL | Training: Essentials of SYCL
- CodeProject: Using oneAPI to convert CUDA code to SYCL
- Intel® DevCloud: A free environment to access Intel oneAPI tools and develop and test code across a variety of Intel® architectures (CPU, GPU, and FPGA).

Free Your Software from Vendor Lock-in Using SYCL* and oneAPI

Migrating from CUDA* to SYCL Just Got Easier

Rod Burns and Joe Todd, Codeplay Software

The use of accelerators is increasing every year, with software developers taking advantage of GPUs in particular to run a variety of HPC and AI algorithms on highly parallel systems. The data center accelerator market is projected to grow from $13.7 billion in 2021 to $65.3 billion by 2026 according to research from MarketsandMarkets[1].

During the past decade or so, software developers have largely been bound to CUDA* to write highly parallel software that can make use of GPUs that, whilst originally designed for graphics processing, are now being used in a wide range of disciplines that include AI and machine learning. The challenge with this approach for software developers is that CUDA is a proprietary programming interface and can only be used to run on processors from NVIDIA. This ties organizations into a single vendor and limits the ability to innovate with the latest processor architectures.

[1] https://www.marketsandmarkets.com/Market-Reports/data-center-accelerator-market-48984803.html

The market is changing, and there is increasing choice from a wide range of processor vendors, including Intel (especially with the upcoming Intel® Iris® Xe GPU), and new specialized processors, such as those harnessing the RISC-V instruction set architecture. Software developers need to be able to not only take maximum advantage of the existing processor targets but also adapt and capitalize on this explosion of new architectures and innovations — but how is this possible without writing new code for each of these new processors from different vendors?

This is where SYCL* and oneAPI offer a flexible and non-proprietary alternative. SYCL is an industry defined, royalty free, open standard interface for writing software that runs on highly parallel processor targets. SYCL is already widely adopted: it is being used to enable performance portability on some of the fastest supercomputers in the world. Argonne, Lawrence Berkeley and Oak Ridge National Laboratories in the United States are using SYCL to enable their researchers to write software that can be run on Aurora (using Intel® GPUs); Perlmutter, Polaris, and Summit (using NVIDIA GPUs); and Frontier (using AMD GPUs) supercomputers whilst achieving the best performance. In Europe, the Lumi supercomputer team have chosen SYCL as a primary programming model to enable performance portability with existing and future machines across the continent. Other government and commercial organizations are using SYCL to deploy complex software to their multi-vendor systems.

SYCL sits at the heart of oneAPI, an open and standards-based programming environment for accelerator architectures. oneAPI defines and implements a set of commonly used libraries and frameworks that enable highly parallel software to run with performance and portability across architectures. oneAPI includes libraries for common math and neural network algorithms alongside the building blocks required to write highly optimized applications. Together, this provides software developers with everything they need to write HPC and AI applications.

You may be thinking, SYCL sounds great but how difficult is it going to be to move all my code from CUDA to SYCL? To understand how straightforward it is, let's walk through an N-body simulation project we have been working on and demonstrate how to migrate this from CUDA to SYCL.

An N-body simulation is used to show gravitational interaction in a fictional galaxy using a defined set of equations (**Figure 1**). This project is based on an existing open-source N-body simulation written by Sarah Le Luron using C++. This code was adapted to implement a kernel to run some of the simulation calculations in parallel on an NVIDIA GPU, helping to achieve a faster execution time compared to running on a CPU.

Figure 1. N-body simulation of a fictional galaxy.

How do we port this kernel code to SYCL? Whilst this project has a single kernel, other applications may have hundreds of CUDA kernels in a code base, and this next step will remove a lot of the heavy lifting and save significant development time.

The first step involves using the recently open-sourced SYCLomatic tool, which provides a semi-automated way to port CUDA to SYCL. Given one or more CUDA source files, it converts the source code to SYCL. Simply tell the tool what source files need to be converted, and it will produce a C++ source file containing the ported code. **Figure 2** shows how simple the steps are at this stage using the *intercept-build* script. This tracks and saves the commands and flags in a JSON file, which is useful for projects with multiple source files. By pointing the tool at the source files that need to be ported, it produces a set of SYCL code files alongside some helper classes and functions that are used to simplify the porting at this stage.

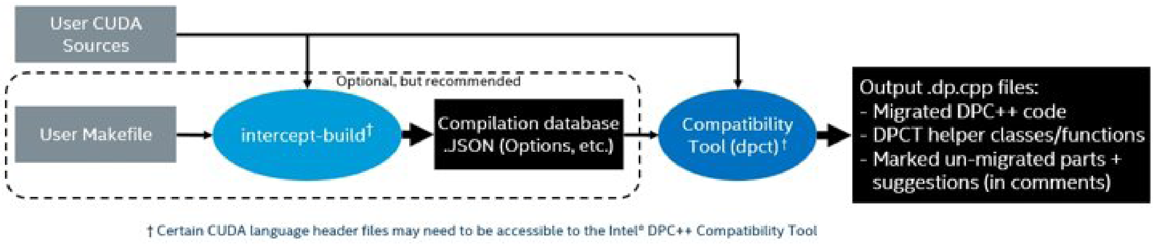
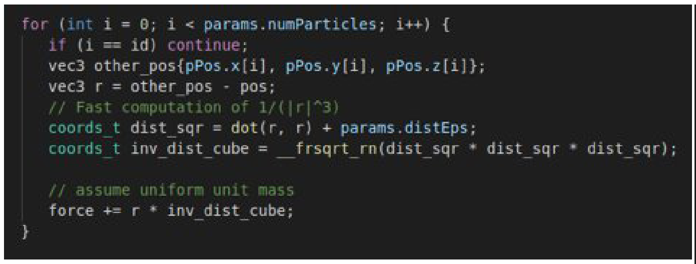
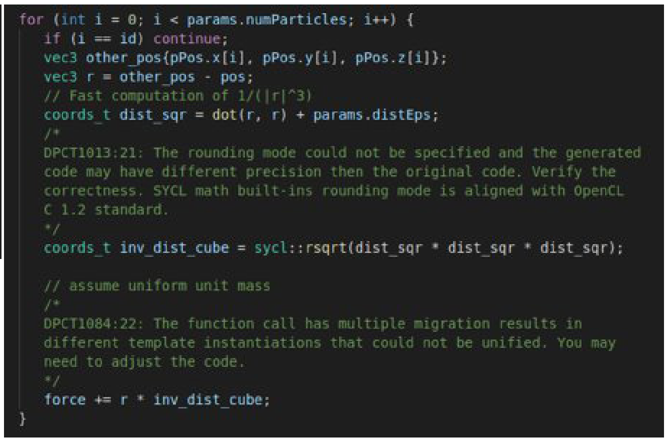

User CUDA Sources
Optional, but recommended
User Makefile
intercept-build†
Compilation database .JSON (Options, etc.)
Compatibility Tool (dpct)†
Output .dp.cpp files:
- Migrated DPC++ code
- DPCT helper classes/functions
- Marked un-migrated parts + suggestions (in comments)
† Certain CUDA language header files may need to be accessible to the Intel® DPC++ Compatibility Tool

Figure 2. SYCLomatic workflow.

Let's look at some of the code that SYCLomatic generates (**Figure 3**). This comparison shows a portion of the code that was converted by the tool. One of the changes made is to the built-in square root method. This is a simple translation from the CUDA version to the SYCL equivalent. You can also see that the tool generates comments that help to give the developers guidance and hints on further changes that they might need to make to optimize or ensure accuracy in their application. There are other changes to the kernel code that can be examined in the project repository. It's worth noting that, for this N-body project, no further changes were required to the SYCLomatic-generated source code. This demonstrates the effectiveness of this tool, but it's worth understanding that usually developers will need to do some level of manual code changes for their project.



```
for (int i = 0; i < params.numParticles; i++) {
    if (i == id) continue;
    vec3 other_pos{pPos.x[i], pPos.y[i], pPos.z[i]};
    vec3 r = other_pos - pos;
    // Fast computation of 1/(|r|^3)
    coords_t dist_sqr = dot(r, r) + params.distEps;
    coords_t inv_dist_cube = __frsqrt_rn(dist_sqr * dist_sqr * dist_sqr);

    // assume uniform unit mass
    force += r * inv_dist_cube;
}
```



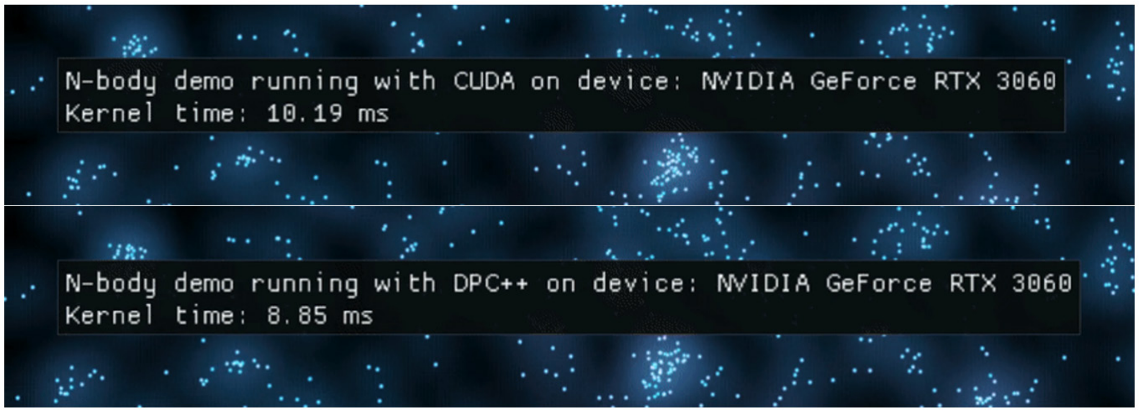
```
for (int i = 0; i < params.numParticles; i++) {
    if (i == id) continue;
    vec3 other_pos{pPos.x[i], pPos.y[i], pPos.z[i]};
    vec3 r = other_pos - pos;
    // Fast computation of 1/(|r|^3)
    coords_t dist_sqr = dot(r, r) + params.distEps;
    /*
    DPCT1013:21: The rounding mode could not be specified and the generated
    code may have different precision then the original code. Verify the
    correctness. SYCL math built-ins rounding mode is aligned with OpenCL
    C 1.2 standard.
    */
    coords_t inv_dist_cube = sycl::rsqrt(dist_sqr * dist_sqr * dist_sqr);

    // assume uniform unit mass
    /*
    DPCT1084:22: The function call has multiple migration results in
    different template instantiations that could not be unified. You may
    need to adjust the code.
    */
    force += r * inv_dist_cube;
}
```

Figure 3. Converting CUDA* (left) to SYCL* (right) using SYCLomatic.

Now that the porting is done, it's possible to run this SYCL code on a range of processors. Let's focus on using an NVIDIA GPU target with the DPC++ compiler. DPC++ is an open-source SYCL compiler that is part of the oneAPI initiative. It includes support for Intel® processors, NVIDIA GPUs and AMD GPUs. There are instructions on how to set up your environment to use the NVIDIA and AMD targets on the Codeplay website and the open source DPC++ repository.

Running the N-body simulation using CUDA and SYCL on the same NVIDIA hardware shows comparable performance (**Figure 4**). Running the CUDA and SYCL versions of this N-body simulation on the same NVIDIA GeForce* GPU demonstrates it is possible to achieve comparable performance using either native CUDA or SYCL. The times in the images show the kernel time for the N-body gravity simulation is close, and in this instance the SYCL version is slightly faster compared to the native CUDA code. It's worth noting that your mileage will vary, but there are examples across both research and commercial organizations that show performance results using SYCL that closely match CUDA. See the Zuse Institute Berlin's video presentation showing how their Tsunami simulation code was able to achieve this.

*Other names and brands may be claimed as the property of others.
For more complete information about compiler optimizations, see our Optimization Notice.

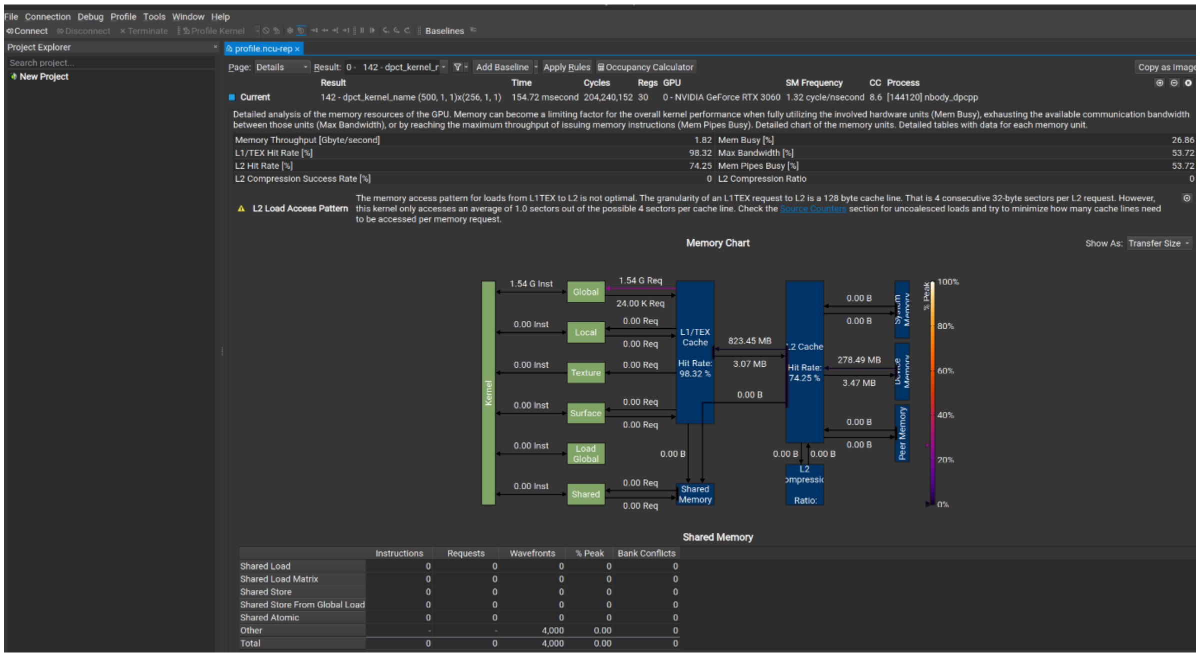


```
N-body demo running with CUDA on device: NVIDIA GeForce RTX 3060
Kernel time: 10.19 ms
```

```
N-body demo running with DPC++ on device: NVIDIA GeForce RTX 3060
Kernel time: 8.85 ms
```

Figure 4. The original CUDA* N-body code (top) and the converted SYCL* code (bottom) give comparable performance.

Whilst performance portability is achievable with SYCL on NVIDIA GPUs using DPC++, there will be times when it's necessary to figure out where your code needs to be fine-tuned to bring out the best from the hardware. Alongside the ability to run SYCL code on NVIDIA hardware, it's also possible to enjoy the benefits of the NVIDIA profiling tooling including NVIDIA Nsight* (**Figure 5**). Without any code or configuration changes, these same familiar tools can be used to help you fine tune and optimize your application to get the absolute best performance.



Result	Time	Cycles	Regs	GPU	SM Frequency	CC	Process
Current 142 - dpct_kernel_name (500, 1, 1)x(256, 1, 1)	154.72 msecond	204,240,152	30	0 - NVIDIA GeForce RTX 3060	1.32 cycle/nsecond	8.6	[144120] nbody_dpcpp

Detailed analysis of the memory resources of the GPU. Memory can become a limiting factor for the overall kernel performance when fully utilizing the involved hardware units (Mem Busy), exhausting the available communication bandwidth between those units (Max Bandwidth), or by reaching the maximum throughput of issuing memory instructions (Mem Pipes Busy). Detailed chart of the memory units. Detailed tables with data for each memory unit.

Memory Throughput [Gbyte/second]	1.82	Mem Busy [%]	26.86
L1/TEX Hit Rate [%]	98.32	Max Bandwidth [%]	53.72
L2 Hit Rate [%]	74.25	Mem Pipes Busy [%]	53.72
L2 Compression Success Rate [%]	0	L2 Compression Ratio	0

⚠ **L2 Load Access Pattern** The memory access pattern for loads from L1TEX to L2 is not optimal. The granularity of an L1TEX request to L2 is a 128 byte cache line. That is 4 consecutive 32-byte sectors per L2 request. However, this kernel only accesses an average of 1.0 sectors out of the possible 4 sectors per cache line. Check the Source Counters section for uncoalesced loads and try to minimize how many cache lines need to be accessed per memory request.

	Instructions	Requests	Wavefronts	% Peak	Bank Conflicts
Shared Load	0	0	0	0	0
Shared Load Matrix	0	0	0	0	0
Shared Store	0	0	0	0	0
Shared Store From Global Load	0	0	0	0	0
Shared Atomic	0	0	0	0	0
Other	-	-	4,000	0.00	0
Total	0	0	4,000	0.00	0

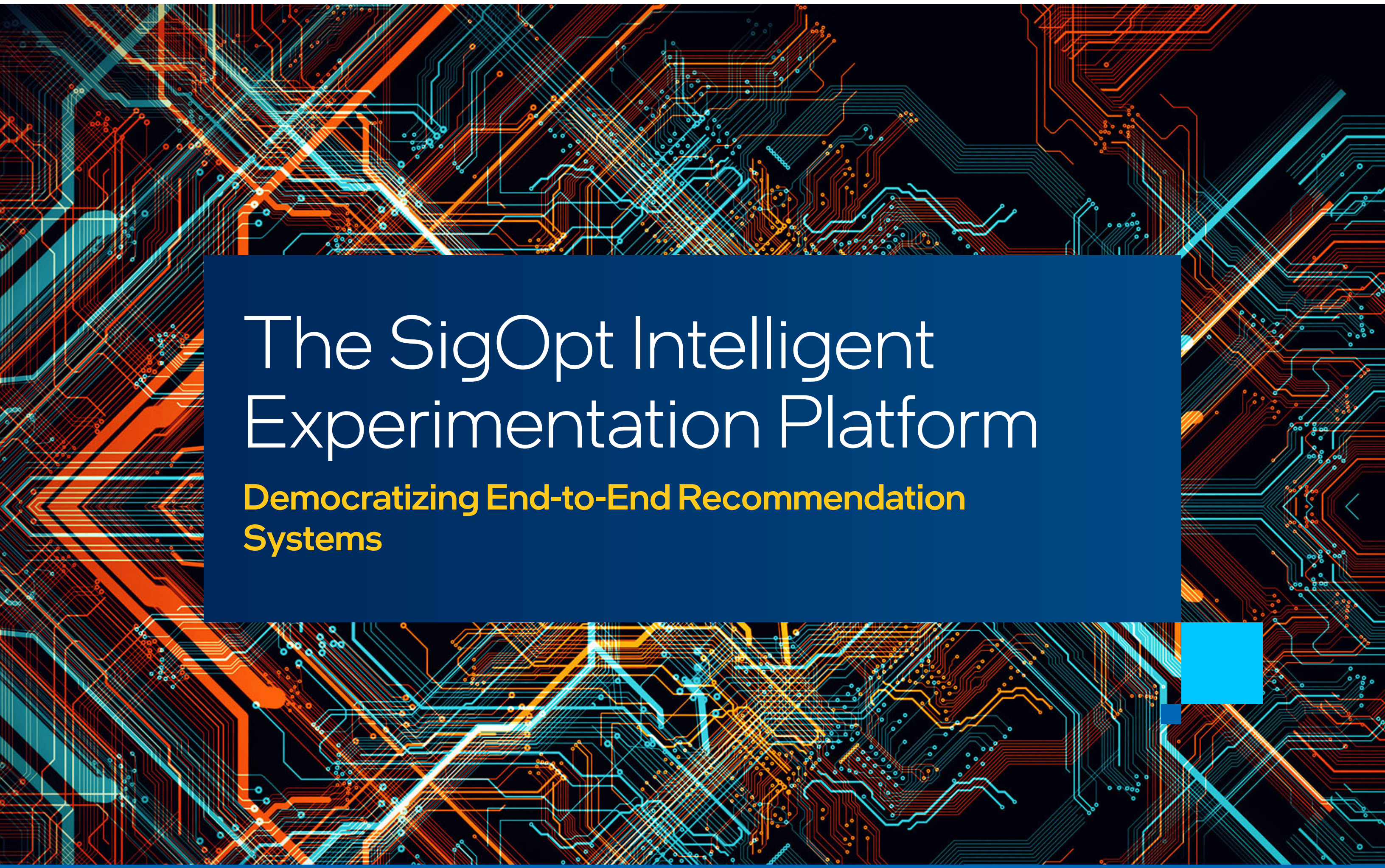
Figure 5. NVIDIA Nsight* can be used to profile SYCL* applications when using the CUDA* backend.

*Other names and brands may be claimed as the property of others.
For more complete information about compiler optimizations, see our Optimization Notice.

Sign up for future issues

By examining the information from these tools, it is possible to track down kernel and memory optimizations, making it easier to pinpoint the area of your code to focus on. If you are wanting to get into the very fine details, it's also possible to examine the generated low-level PTX instructions using the compiler output. What this means is that developers can use the tools most suitable for their target architecture using SYCL, whether that's NVIDIA Nsight for NVIDIA GPUs or Intel® VTune™ for Intel® platforms. The path from CUDA to SYCL is not as daunting as it seems, and the benefits are clear to see. Code written with SYCL enables a future-proofed software development environment, allowing software development teams to target existing accelerator processors, such as those from Intel, NVIDIA, and AMD, and be ready to adopt the latest architectures, such as novel RISC-V based accelerators.

Take some time to examine the open-source N-body project code we have developed, use SYCLomatic to port your CUDA code to SYCL and future-proof your software for the next generation of processors.



The SigOpt Intelligent Experimentation Platform

Democratizing End-to-End Recommendation Systems

Jian Zhang, AI Software Engineering Manager, Intel Corporation; and Tobias Andreasen, Machine Learning Specialist, SigOpt, an Intel company

As everything from streaming services to e-commerce platforms and social networks grows, so does the available data and the need for efficient algorithms. To power these platforms and engage the end users, an increasing number of companies are investing in recommendation systems. Modern recommendation systems require a complex pipeline to handle both data processing and feature engineering at tremendous scale, while promising high service-level agreements for complex deep learning (DL) models. Additionally, these complex DL models also need to be constantly updated and retrained to guarantee the best possible performance, while keeping the training price point down.

In this article, we introduce some of the challenges of running modern recommender systems. To tackle these challenges, we propose an end-to-end (E2E) democratization solution that includes optimized parallel data processing based on Apache Spark*, and user-guided automatic model democratization via SigOpt AutoML. We present how the solution improves the E2E pipeline efficiency on commodity clusters for common recommender system workloads.

Motivations

Challenges of Recommendation Systems

The goal of a recommendation system is to automatically predict a user's preference based on context about the user and context about all possible preferences. The use of recommendation systems has been growing steadily in retail and e-commerce, healthcare, transportation, and more. For some platforms, recommendations from automated recommendation systems account for as much as 30% of revenue. It is said that 35% of what consumers buy on Amazon comes from automated recommendations, and that 75% of what users watch on Netflix comes from automated recommendations (source: How Retailers Can Keep Up with Consumers).[1]

Figure 1 shows the architecture of a modern recommendation system consisting of two major workstreams: data processing and modeling. Data processing handles data collection and processing, and uses feature engineering to generate features for model training. Modeling can be divided into two subcategories: model training and model serving.

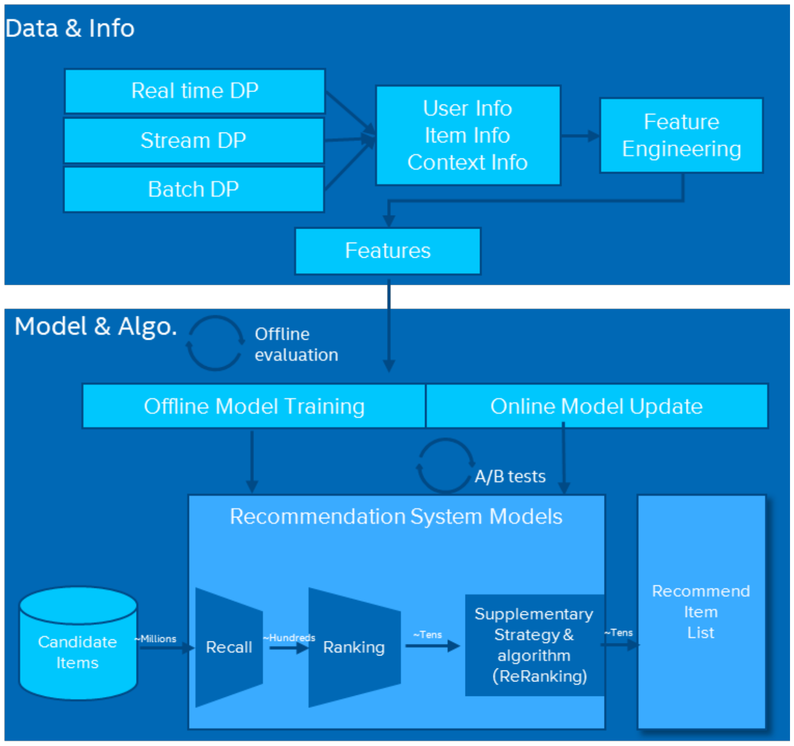


Figure 1. Recommendation system architecture.

[1]https://www.mckinsey.com/industries/retail/our-insights/how-retailers-can-keep-up-with-consumers

Accuracy is critical when building an E2E recommendation system, but challenges arise with excessively large models and datasets:

- **Huge datasets:** Recommendation systems are often trained on large datasets (terabytes and sometime even petabytes), which require large clusters to store and process the data. Slow data movement between the data processing and training/inference clusters can be challenging.
- **Data preprocessing:** Datasets need to be loaded, cleaned, preprocessed, and transformed into a suitable format for DL models and frameworks. This requires complex data processing technologies, like batch/streaming data processing based on the type of data being processed.
- **Feature engineering:** Numerous sets of features need to be created, engineered, and tested, which is error-prone and time-consuming.
- **Models and algorithms:** Companies develop complex models and algorithms to generate the best business predictions. Those models require deep expertise and unique capabilities.
- **Repeated experiments:** Building and maintaining the best performing models is an iterative process that requires multiple experiments.
- **Huge embedding tables:** Categorical features require embedding and a large amount of memory, something that tends to be very bandwidth intensive.
- **Distributed training:** The heavy models usually require extreme computing power, so distributed training is a must, which makes both hardware and software scalability a critical challenge to be resolved.

Artificial Intelligence (AI) Democratization

The purpose of AI democratization is to make it accessible and affordable to every organization and end-user. Currently, AI is restricted to data scientists and data analysts who are specifically trained in this field. Therefore, making AI accessible to a wider user base is one of the goals of AI democratization. Moreover, AI requires hardware that can be expensive, so another goal of AI democratization is to make AI scale on commodity hardware that exists in most data centers.

There are multiple areas to be democratized:

- **Data:** Data democratization is a priority because AI cannot generate insights without data. Besides the amount of data, improving data quality and making data access easier and faster is also critical.
- **Infrastructure:** The effectiveness of AI relies on the infrastructure. Architecting the software and hardware platform, efficiently managing resource allocation, and auto-scaling are all important factors for infrastructure democratization.
- **Hardware:** The compute-intensive nature of AI requires specialized, and sometimes expensive, accelerators. Democratization aims to migrate AI frameworks from expensive accelerators to commodity hardware to reduce cost.
- **Algorithms:** To get an accurate prediction or inference, the models and algorithms can be very complex. Democratization tries to simplify the use, development, and sharing of AI algorithms to reduce the entry barriers.

SigOpt

The SigOpt intelligent experimentation platform helps users produce the best models. SigOpt is completely agnostic to the modeling framework. It automatically tracks and logs everything that has been done throughout the modeling process. This allows the user to visualize everything in a hosted dashboard. SigOpt also offers a set of proprietary optimization algorithms that help users optimize their models for the best performance. It gives modelers the ability to optimize multiple metrics and add both thresholds and constraints to guardrail these metrics and their impact on the final model.

Smart Democratization Advisor

After seeing customers struggle to bring deep learning recommendation models (DLRM) into production, we decided to democratize them to remove the common pain points like slow training times, large memory consumption, slow convergence, and communication overhead. We developed Smart Democratization Advisor (SDA), a human intelligence enhanced toolkit to generate recipes for SigOpt AutoML with parametrized configuration files (**Figure 2**). SDA uses SigOpt to automatically make the training faster through intelligent optimization, reduce the size of the model, find the optimal gradient descend method, and reduce the need for looking through massive look-up tables.

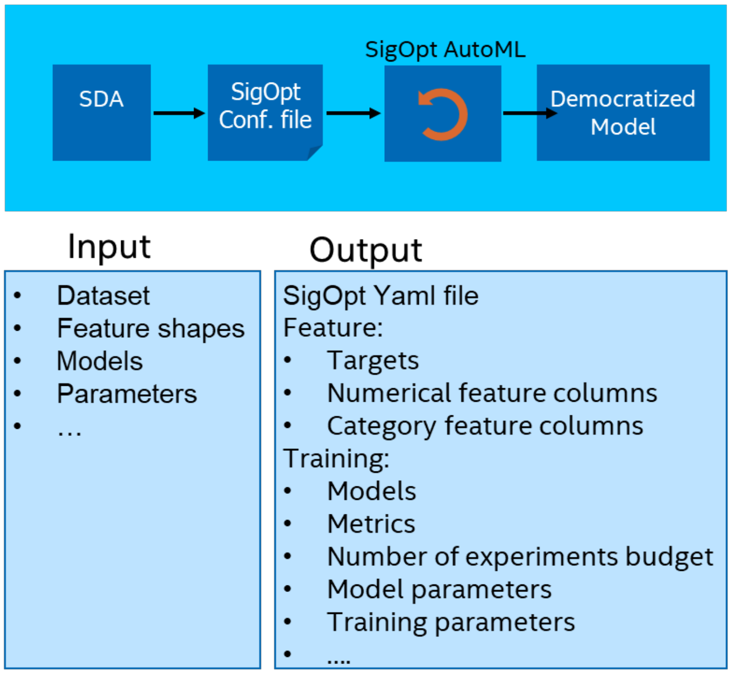


Figure 2. SDA overview.

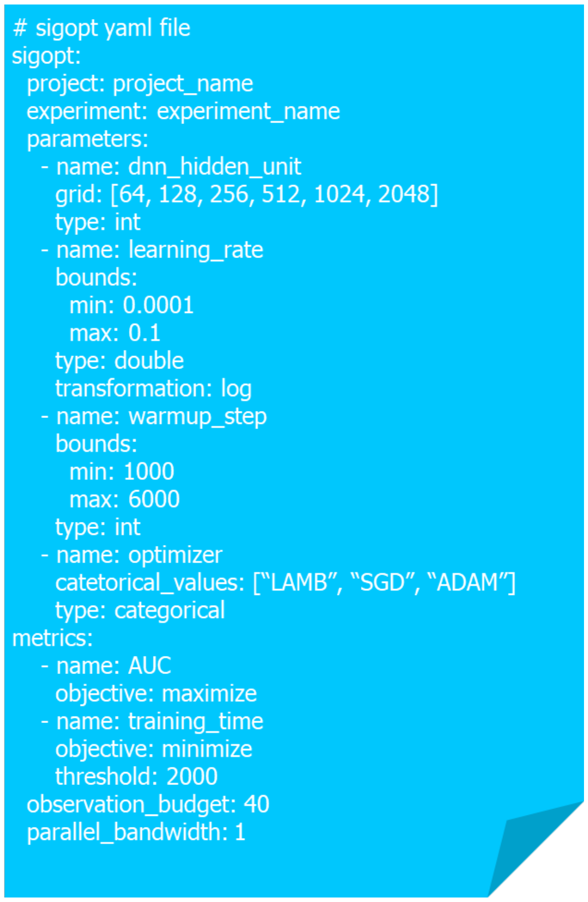
SDA facilitates the generation of a SigOpt recipe based on expert-level knowledge of how a particular model, like DLRM, can be parametrized. SDA takes input choices around data, feature shapes, and models, and then uses this information to automatically generate all the information needed to start a

SigOpt experiment. Through built-in expert-level knowledge — such as the best suited number of layers, optimizers, and hyperparameters — SDA automatically generates a parameterized configuration to facilitate SigOpt AutoML.

The generated configuration includes target features, models, model metrics, and some training-specific parameters, like epochs, steps, etc. SDA converts the time-consuming, manual model-tuning and optimization process to automated hyperparameter optimization (HPO), which simplifies the process of AI democratization, increases efficiency, and empowers data scientists to deliver more value.

SDA Examples

SDA automatically generates a parameter configuration for all built-in models in YAML format that contains model parameters, training parameters, SigOpt parameters, etc. **Figure 3** shows an example for a wide and deep model that includes model parameters (e.g., number of layers, hidden units per layer, size of embeddings, and dropout rate), training parameters (e.g., learning rate, learning rate decay, warmup steps, batch size, and optimizer), and SigOpt parameters (e.g., observation budget and parallel bandwidth). Users also have the freedom to add or remove parameters in any of the categories.



```
# sigopt yaml file
sigopt:
  project: project_name
  experiment: experiment_name
  parameters:
    - name: dnn_hidden_unit
      grid: [64, 128, 256, 512, 1024, 2048]
      type: int
    - name: learning_rate
      bounds:
        min: 0.0001
        max: 0.1
      type: double
      transformation: log
    - name: warmup_step
      bounds:
        min: 1000
        max: 6000
      type: int
    - name: optimizer
      catetorical_values: ["LAMB", "SGD", "ADAM"]
      type: categorical
metrics:
    - name: AUC
      objective: maximize
    - name: training_time
      objective: minimize
      threshold: 2000
  observation_budget: 40
  parallel_bandwidth: 1
```

Figure 3. SDA-generated YAML file for a wide and deep model.

Once the YAML file has been generated, the user can start the SigOpt optimization loop. With SDA-generated model parameters, SigOpt experiments budget can be greatly reduced. Meanwhile, SDA can leverage SigOpt multimetric optimizations to further improve AutoML efficiency. The joint solution reduces the time needed to get the desired models into production.

Performance Evaluation

To evaluate how SDA performs, we designed several experiments to show whether SDA can democratize typical recommendation systems automatically, and how it performs compared with original models.

DLRM is a personalization model open-sourced by Meta (**Figure 4**). It was conceived by the union of the two perspectives in recommendation systems: both collaborative filtering and predictive analytics-based approaches. DLRM is able to work efficiently with production-scale data while delivering state-of-the-art results. It leverages embeddings to process sparse features and adopts multilayer perceptron (MLP) to process dense (numerical) features, and then it interacts with features explicitly using statistical techniques. Finally, it predicts the event probability by postprocessing the interactions with another MLP.

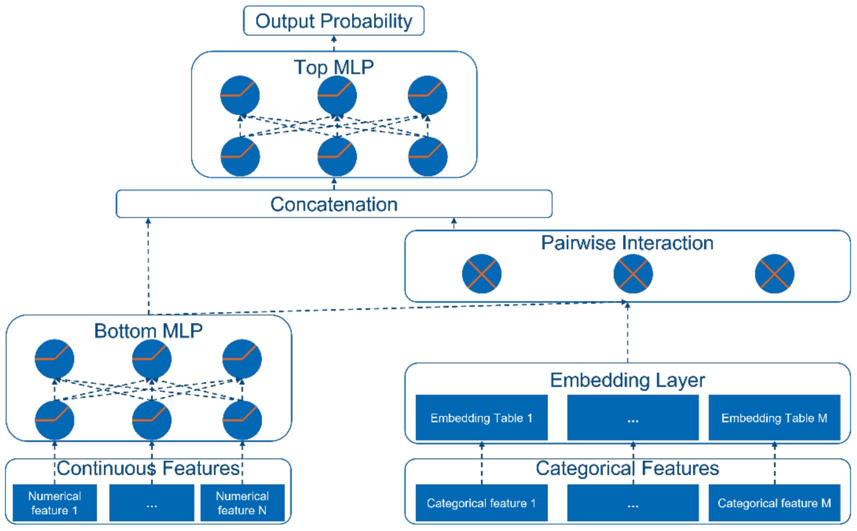
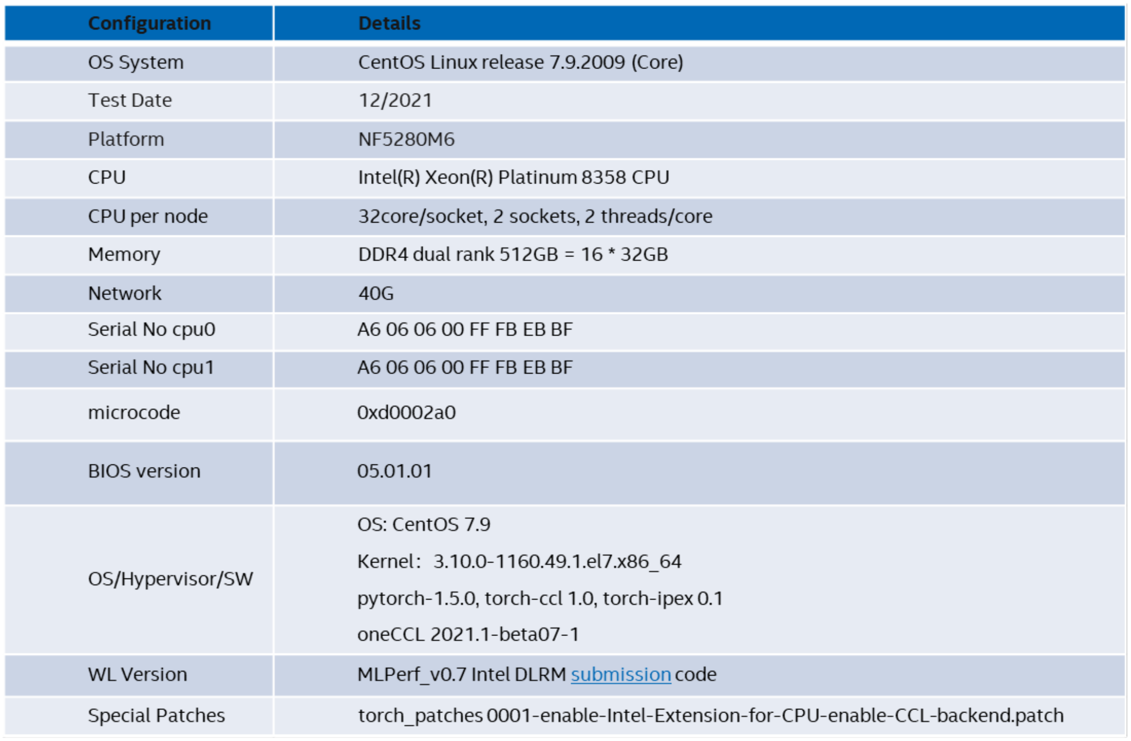


Figure 4. DLRM model architecture.

DLRM on commodity CPUs can be challenging. The initial NumPy data processing and model training can take several days. For data processing, the single-threaded NumPy data processing becomes a bottleneck. To resolve this issue, Intel developed and open-sourced RecDP, a parallel data processing toolkit for recommendation systems based on PySpark. RecDP can fully utilize multithreading and multi-node benefits when doing data processing, and can deliver up to a 100x speed-up over the original data processing solution.

The tests were conducted on a four-node cluster, each node equipped with two Intel® Xeon® Platinum 8358 processors, 512GB memory, and connected with 40GB Ethernet. A 1TB Intel® SSD DC P4500 NVMe was used as a data drive. **Table 1** shows a detailed configuration.



Configuration	Details
OS System	CentOS Linux release 7.9.2009 (Core)
Test Date	12/2021
Platform	NF5280M6
CPU	Intel(R) Xeon(R) Platinum 8358 CPU
CPU per node	32core/socket, 2 sockets, 2 threads/core
Memory	DDR4 dual rank 512GB = 16 * 32GB
Network	40G
Serial No cpu0	A6 06 06 00 FF FB EB BF
Serial No cpu1	A6 06 06 00 FF FB EB BF
microcode	0xd0002a0
BIOS version	05.01.01
OS/Hypervisor/SW	OS: CentOS 7.9 Kernel: 3.10.0-1160.49.1.el7.x86_64 pytorch-1.5.0, torch-ccl 1.0, torch-ipex 0.1 oneCCL 2021.1-beta07-1
WL Version	MLPerf_v0.7 Intel DLRM submission code
Special Patches	torch_patches 0001-enable-Intel-Extension-for-CPU-enable-CCL-backend.patch

Table 1. System configuration.

To evaluate the improvement from SDA, we designed four test cases (**Table 2**):

- **Original DLRM:** The original DLRM model, using E2E model training time as metrics
- **SDA-assisted DLRM:** The SDA-optimized model, including all model tunings and optimizations
- **SigOpt AutoML:** The total execution time of five SigOpt experiments
- **SDA-assisted SigOpt AutoML:** The total execution time of SigOpt experiments, using SDA-generated configurations

	Original model	SDA	SigOpt AutoML	SDA+SigOpt
Workload	DLRM			
Data Set	Criteo Terabyte Dataset			
Run Method	mpirun			
Iterations and result (median, average, min, max)	1 epoch max			
Dataset size	training: 4,096,896k testing: 87,104k			
Number of runs	1	1	5	5

Special configuration	train_batch_size=32k
	test_batch_size=64k
	learning_rate=18.0
	lamblr=None
	arch-mlp-top="1024-1024-512-256-1"
	arch-sparse-feature-size=128
	lr-num-warmup-steps=4000
	lr-decay-start-step=5760
	lr-num-decay-steps=27000
		train_batch_size=256k
		test_batch_size=128k
		learning_rate=50.0
		lamblr=22
		arch-mlp-bot="13-128-64"
		arch-mlp-top="256-128-1"
		arch-sparse-feature-size=64
		lr-num-warmup-steps=8000
		lr-decay-start-step=70000
		lr-num-decay-steps=30000
			batch_size=32k
			test_batch_size=64k
			learning_rate=5~50
			lamblr=None
			arch-mlp-bot="13-512-256-128"
			arch-mlp-top="1024-1024-512-256-1"
			arch-sparse-feature-size=128
			lr-num-warmup-steps=2000~8000
			lr-decay-start-step=4500~9000
			lr-num-decay-steps=5000~15000
				batch_size=256k
				test_batch_size=128k
				learning_rate=5~50
				lamblr=5~50
				arch-mlp-bot="13-128-64"
				arch-mlp-top="256-128-1"
				arch-sparse-feature-size=64
				lr-num-warmup-steps=2000~8000
				lr-decay-start-step=4500~9000
				lr-num-decay-steps=5000~15000

Table 2. Test configuration.

The DLRM performance comparison of the four test cases is shown in **Figure 5**. It shows that SDA can deliver a 5.4x performance improvement over the original DLRM model and a 5.2x speed-up for SigOpt AutoML of five experiments while maintaining the same AUC (area under the ROC curve) as the original model.

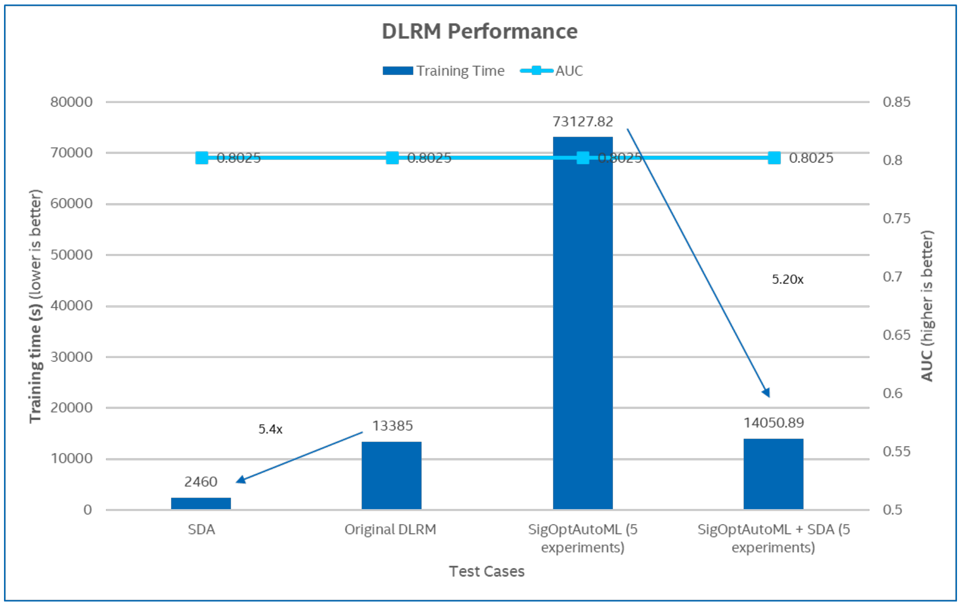


Figure 5. DLRM performance comparison.

SDA cannot deliver such results without the help of SigOpt multi-metrics and metrics thresholds. To evaluate the effect of multi-metrics (accuracy and training time) in the SigOpt tuning process, we compare the performance of SigOpt AutoML plus SDA with or without multiple metrics. **Figure 6** shows that multi-metrics and metrics thresholds were able to reduce training time, delivering up to a 1.26x speed-up for the best metrics.

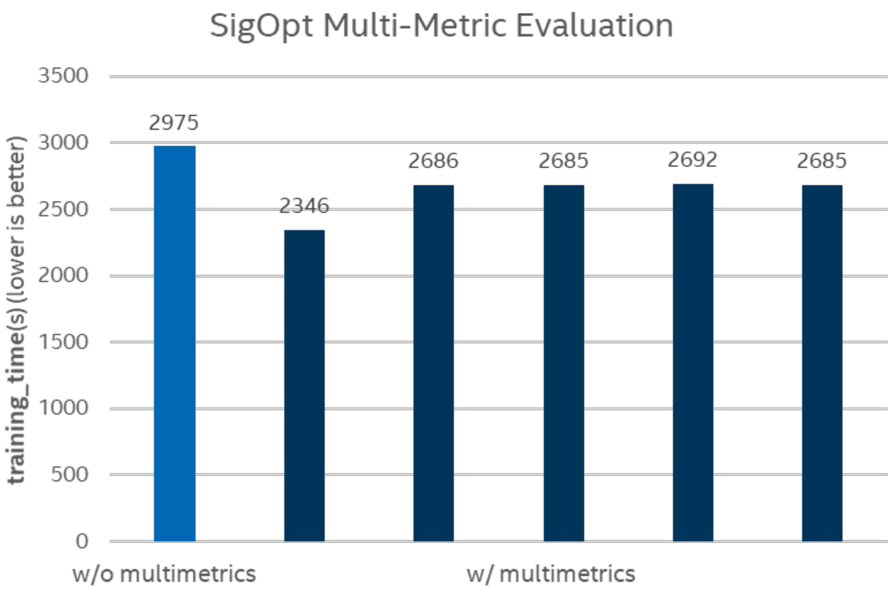


Figure 6. Multiple metrics performance.

To analyze the effect of multiple metrics, we explored both the time and AUC metrics for with and without multi-metrics (**Figure 7**). With multi-metric and metric thresholds, the number of experiments that delivered the same AUC but with lower training time increased.

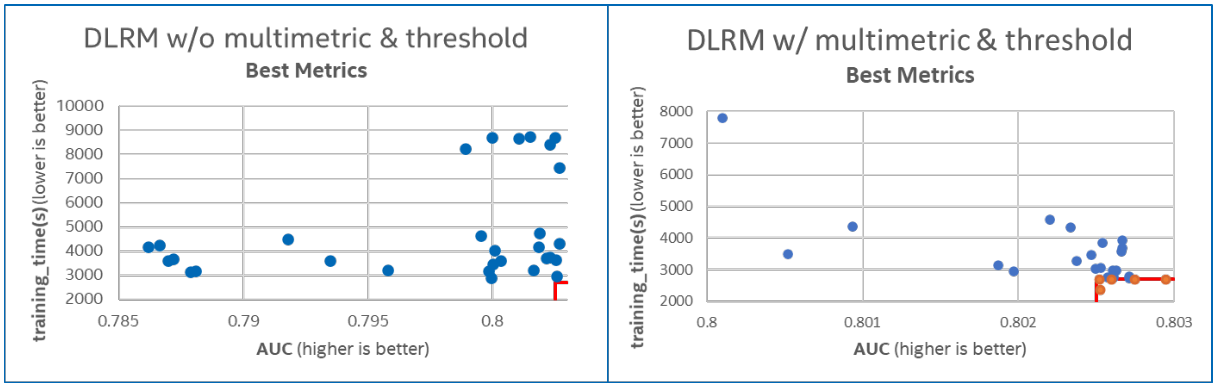


Figure 7. Best metrics comparison with and without multi-metrics and metrics thresholds.

Summary

Recommendation systems are becoming increasingly popular because they bring genuine business value, but they also pose a lot of challenges due to their unique nature. E2E AI democratization is targeted to address those challenges, and to make AI accessible and affordable to every organization and every user. This article introduced SDA, one component of the Intel AI democratization kit, to simplify and automate AI democratization. SDA is a toolkit to generate SigOpt recipes for AutoML with parametrized configuration files generated by built-in expertise. Performance evaluation showed SDA-assisted SigOpt AutoML delivers very promising results — up to a 5x performance speed-up for a DLRM model while remaining at the same AUC. Meanwhile, SigOpt multi-metrics and metrics thresholds further improve efficiency.

SDA is our first attempt toward AI democratization. If you want to use it, please check out the RecDP repo and stay tuned for future articles. If you want to use SigOpt for your own projects, visit sigopt.com/signup and sign up for free.

Distributed Training on Intel® Xeon® Scalable Processors

A Case Study of Training the AI Model on Tencent AI Arena Platform

Ashiq Imran, Wenyue Hu, Ashraf Bhuiyan, AG Ramesh, and Geetanjali Krishna, Intel Corporation; and Wenxi Zhu and Minwen Deng, Tencent

It is a general misconception that GPUs are necessary to train deep learning (DL) models. There are many complex DL models that train more efficiently on CPUs using distributed training configurations. Intel® Xeon® Scalable processors have built-in AI acceleration with Intel® Deep Learning Boost instructions. This article describes how to set up distributed training on a cluster using TensorFlow* and Horovod*.

TensorFlow is a widely used DL framework that is optimized for Intel® processors and other architectures using Intel® oneAPI Deep Neural Network (oneDNN), an open-source, cross-platform library for DL applications. The TensorFlow optimizations enabled by oneDNN accelerate key performance-intensive operations, such as convolution, matrix multiplication, batch normalization, and many more. Horovod is an open-source package that facilitates distributed DL with TensorFlow and other popular frameworks, such as PyTorch*. It is widely used to train models across multiple GPUs and CPUs.

Sign up for future issues

AI Arena is an open platform presented by Tencent for the research of multi-agent intelligence and complex decision-making. The platform includes a wide range of services that can help AI researchers to build their experiment environment conveniently. Tencent has shown a real-world case of building an AI agent in the smash-hit mobile game *Honor of Kings* with the AI Arena platform, which involves multi-agent competition and cooperation, imperfect information, complex action control, and massive state-action space. This AI agent could make players have more fun to enhance game playability. We present a case study of using Tencent* production-level model training on a 16-node cluster comprising Intel Xeon Scalable processors, achieving up to a 15.2x speed-up over a single node.

Running Distributed Deep Learning (DL) Training

You can either use a prebuilt Docker* container with TensorFlow with oneDNN enabled, which has everything you need:

```
docker pull intel/intel-optimized-tensorflow:2.6.0-ubuntu-18.04-mpi-horovod
```

Or you can manually install TensorFlow (v2.6 or later) with oneDNN enabled, Horovod (v0.22.1 or later), and Open MPI (v4.0 or later). Be sure to check your GCC version. If you are using Ubuntu* 16.04 or older, install GCC v8.4.1 or later.

Use the following commands to run distributed TensorFlow using the Horovod framework on a Linux* cluster. First, set the following environment variables:

```
export LD_LIBRARY_PATH=<path to OpenMP lib>:$LD_LIBRARY_PATH
export PATH=<path to Open MPI bin>:$PATH
export OMP_NUM_THREADS=#of_cores of the machine [e.g., lscpu | grep "Core"]
export KMP_AFFNITY=granularity=fine,compact,1,0
export KMP_BLOCKTIME=1
```

Before starting the training, we can find out how many sockets are in the system by using the following command:

```
lscpu | grep "Socket"
```

Use the following command to run the training on one server with two sockets. Here, the total number of workers is two:

```
horovodrun -np 2 -H localhost:2 --autotune python train.py
```

To run the training across four servers, each with two sockets, use this command. Here, the total number of workers is eight (one worker on each socket):

```
horovodrun -np 8 -H server1:2,server2:2,server3:2,server4:2 --autotune python train.py
```

To run on eight servers, each with one socket, use this command. Here, the total number of workers is also eight:

```
horovodrun -np 8 \
-H server1:1,server2:1,server3:1,server4:1,server5:1,server6:1,server7:1,server8:1 \
--autotune python train.py
```

As you can see, you can scale the number of servers up or down depending on the time-to-train that you want to achieve for your model. Time-to-train is expected to scale almost linearly with the number of servers used. Hyperparameter optimization is also done, as it would be for training on multiple GPUs. The learning rate and effective batch size can often be scaled by the number of workers. An increase in the learning rate can often compensate for the increased batch size.

Case Study: Training Wukong AI*

Wukong AI*, an artificial intelligence program, playing *Honor of Kings*, a popular MOBA game published by Tencent, exceeds the performance of top-professional players. Wukong AI uses reinforcement learning (RL). The goal of RL is to select and optimize appropriate policies to strengthen the AI agent. Selection of these policies can be optimized in deep RL training. Distributed training is used to scale the training process to multiple RL learners on the cluster of 2nd Generation Intel Xeon Scalable processors.

We started off on a single node and with a single worker to measure the baseline performance, and then scaled up to more workers until we reached the desired time-to-train. We achieved the required performance with 16 nodes. In this case study, distributed training gave up to a 15.2x speed-up over the baseline performance (**Figure 1**). This is nearly linear speedup with distributed training on our Intel Xeon processor-based cluster.

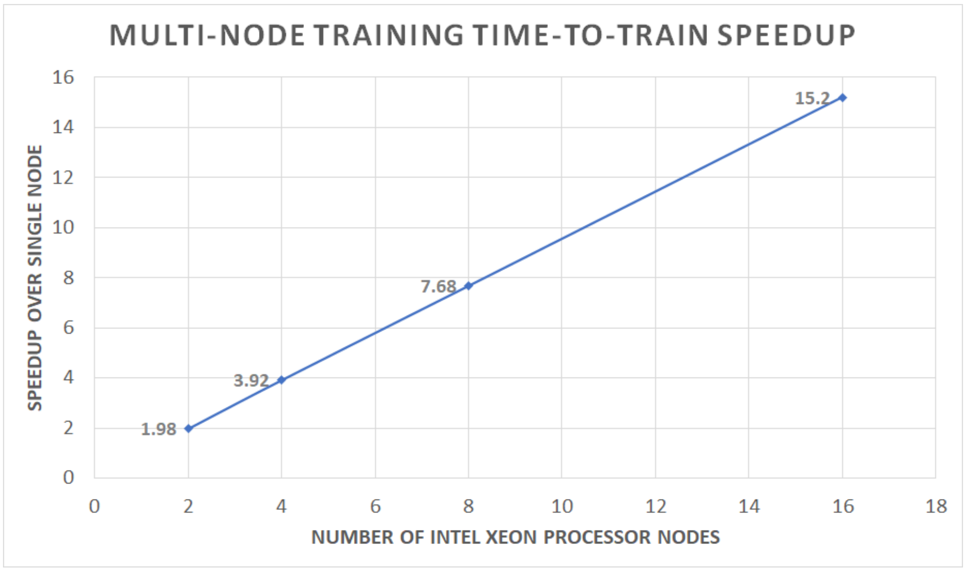


Figure 1. Distributed training gives significant speed-up over baseline performance.

Intel Xeon Scalable processors provide the performance required to train a variety of production workloads. Using Horovod* for distributed training reduces the time-to-train. In this blog, we shared a recent case study used by Tencent AI Lab.

Stay tuned for more blogs and articles as Intel adds more hardware acceleration in the next generation of Intel Xeon Scalable processors and software acceleration to continue to meet users' needs.

Resources and Support

You can get more information from the following sites:

- TensorFlow
- Intel TensorFlow optimizations on PyPI*
- Intel oneAPI Deep Neural Network Library

For help with technical questions, visit the following communities and forums to find answers and get support:

- TensorFlow issues (GitHub)
- Intel Optimized AI Frameworks
- Horovod

System Configuration

TensorFlow source code	https://github.com/tensorflow/tensorflow
TensorFlow version	2.6.0
CPU	76
Threads per core	2
Cores per socket	19
Sockets	2
NUMA nodes	2
Vendor ID	GenuineIntel
BIOS vendor ID	Smdbmds
CPU family	6
Model	85
Model name	Intel Xeon Platinum 8255C Processor @ 2.50 GHz
BIOS model name	3.0
Stepping	5
Hyper-threading	ON
Turbo	ON
Memory	256 GB
OS	Red Hat Enterprise CentOS Linux* version 8.2 (Core) kernel 4.18.0-305.3.1.el8.x86_64 x86_64

Delivering Cost-Effective Genomics for Precision Medicine

Comparing Sentieon DNASeq* Performance to NVIDIA Clara Parabricks*

Henry A Gabb, Intel Corporation; and Don Freed and Zhipan Li, Sentieon Inc.

Next-generation sequencing (NGS) technologies have significantly reduced the cost and time required to sequence whole genomes and exomes. NGS and efficient secondary analysis have brought precision medicine to the clinical setting and even to the point of care. Sentieon optimized their genome analytics software for 3rd Gen Intel® Xeon® Scalable processors and the 4th Gen Intel Xeon Scalable processor (formerly code-named Sapphire Rapids). It is designed to scale on multicore systems to achieve best-in-class performance — whether the clinical requirement is fast turnaround (e.g., in the emergency department to predict adverse drug reactions from a single patient genome) or high throughput (e.g., in an oncology lab to analyze multiple samples from the same tumor or from different patients).

The Sentieon software is vectorized for modern processors, particularly Intel Xeon processors, to achieve high performance without proprietary programming languages or specialized hardware, which eliminates vendor lock-in and reduces software development, deployment, and maintenance costs. We wanted to compare Sentieon performance and accuracy to alternatives, like NVIDIA Clara Parabricks*, to see if specialized hardware is cost-effective or even necessary.

Recent performance data is available for comparison: Benchmarking the NVIDIA Clara Parabricks Germline Pipeline on AWS. This article reports performance and cost data for the following HG001 tests:

- Whole exome sequencing (WES) @ 50x, 75x, and 100x coverage
- Whole genome sequencing (WGS) @ 30x and 50x coverage

We will focus on the HG001 WGS 30x test from the PrecisionFDA Truth Challenge. A Parabricks vs. Genome Analysis Toolkit (GATK) performance comparison is provided for this test (**Figure 1**). GATK is the standard by which variant calling accuracy is judged, but it is written in Java*, so it is not the gold standard of performance. The University of Illinois and the Mayo Clinic have already established that Sentieon significantly outperforms GATK with no loss of accuracy: Sentieon DNASeq* Variant Calling Workflow Demonstrates Strong Computational Performance and Accuracy. Therefore, we will not bother with a GATK comparison. Our goal is to compare the Sentieon software (written in C++ and optimized for modern vector CPUs) to Parabricks (written in CUDA* and optimized for NVIDIA GPUs).

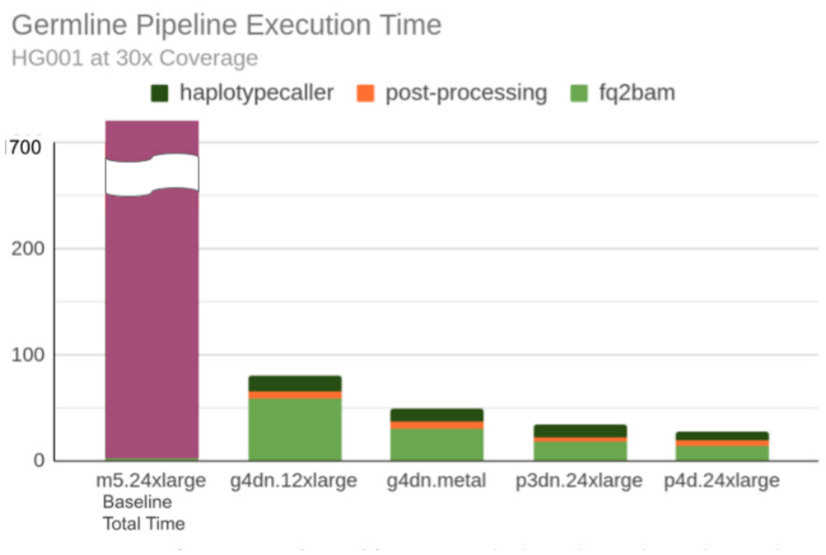


Figure 1. Comparison of NVIDIA Clara Parabricks* execution time (in minutes) against a GATK baseline on various AWS EC2* instance types (source: Benchmarking the NVIDIA Clara Parabricks Germline Pipeline on AWS, Figure 4).

We used the benchmark description and performance data from **Figure 1** to get as close as possible to an apples-to-apples performance comparison of Sentieon and Parabricks. We mapped the haplotypecaller, post-processing, and fq2bam steps from **Figure 1** to the typical stages of the variant calling pipeline (**Table 1**). Our mapping is based on the following description from the Parabricks benchmarks:

"The fq2bam *step includes bwa-mem and parts of coordinate sorting,* post-processing *includes* parts of *coordinate sorting, marking duplicates followed by bqsr.* haplotypecaller *the applybqsr step applied on the input bam, which is then fed to the variant calling step."*

Pipeline Stage	Parabricks*	Sentieon
Alignment	fq2bam	BWA MEM + Sort utility
Sorting		
Deduplication	post-processing	LocusCollector + Dedup, QualCal
BQSR		
Apply BQSR	haplotypecaller	Haplotyper
Variant Calling		

Table 1. Stages of the variant calling pipeline and their Parabricks* and Sentieon equivalents for performance comparisons.

The side-by-side competitive performance of Sentieon vs. Parabricks on a variety of computing platforms is shown in **Figure 2** and **Table 2**. The platforms and pricing details are shown in **Table 3**. 3rd Gen Intel Xeon Scalable processors deliver competitive performance, with the 4th Gen Intel Xeon Scalable processor (formerly code-named Sapphire Rapids) giving the best overall performance. However, performance is only part of the story. Cost-per-genome and power consumption must also be considered.

The cost-per-genome is substantially lower for the Intel Xeon processor ($1.54) compared to the NVIDIA A100 Tensor Core processor ($4.59) (**Table 3**). If the 4th Gen Intel Xeon Scalable processor has similar AWS EC2* pricing, the cost-per-genome falls to less than a dollar ($2.1635/h * 26.8 minutes = $0.97). It is also worth noting that the 4th Gen Intel Xeon Scalable processors used in these benchmarks are pre-release hardware, so the performance of the final product could improve.

In terms of power consumption, the two Intel Xeon Platinum 8352M processors in the c6i.metal instance require 370W, whereas the eight NVIDIA A100 Tensor Core processors in the p4d.24xlarge instance require 3,200W. The best Parabricks performance requires 8.6x the power and 3.0x the cost, but only delivers 1.5x the performance of the 3rd Gen Intel Xeon 8352M processor.

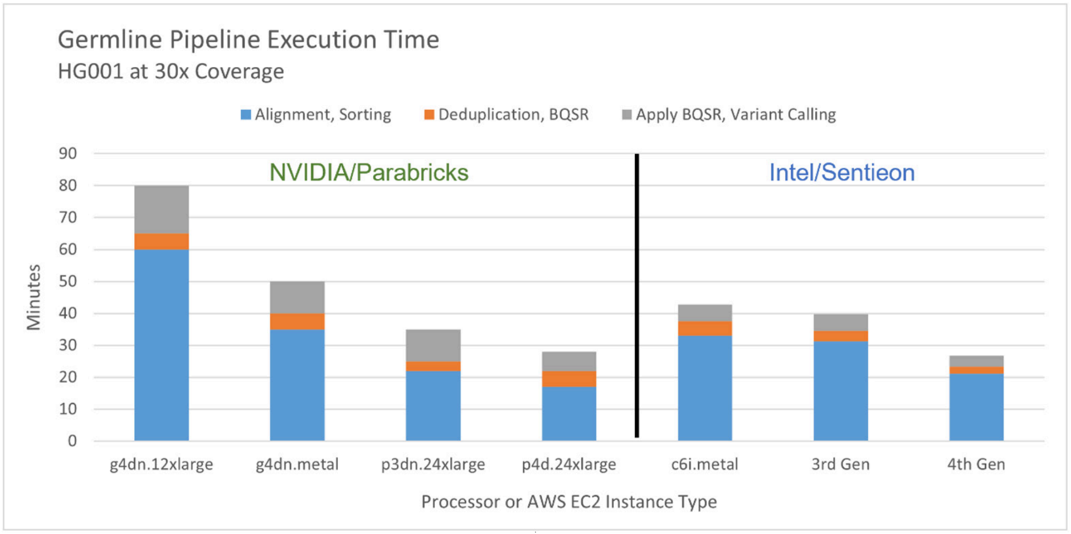


Figure 2. Side-by-side competitive performance of Sentieon vs. Parabricks* on a variety of cloud and on-premises platforms. Note that the Parabricks times are taken from the published results or approximated from visual inspection of Figure 1.

Performance measurements were performed by Sentieon in March 2022. The 3rd Gen Intel Xeon Scalable processor-based system is a two-socket 2.4 GHz Intel Xeon Platinum 8368 processor (152 cores, HyperThreading enabled), 256 GB DDR4-3200 memory, and 1 TB Intel® 660p and 2 TB Intel DC P4510 SSDs. The 4th Gen Intel Xeon Scalable processor-based system is an Intel® pre-production platform with two 4th Gen Intel Xeon Scalable processors (formerly code-named Sapphire Rapids), >40 cores, Hyper-Threading enabled), Intel pre-production BIOS, 256 GB DDR memory (16(1DPC)/16 GB/4800 MT/s), and 1 TB Intel D3-S4610 SSD. Ubuntu Linux* 20.04 was installed on both systems. Performance varies by use, configuration, and other factors so results may vary.

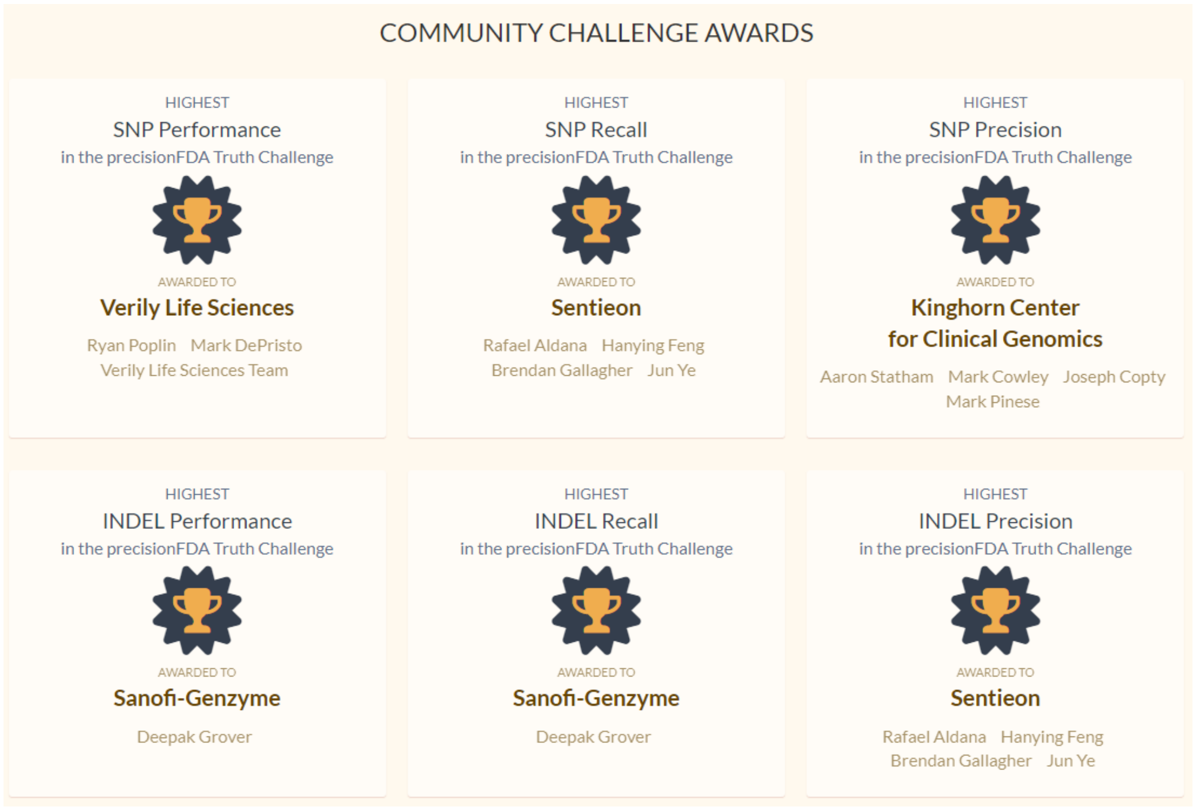
Pipeline Stages	NVIDIA Clara Parabricks*				Sentieon		
	AWS EC2* Instance Type				Intel® Xeon® Scalable Processors		
	g4dn.12xlarge	g4dn.metal	p3dn.24xlarge	p4d.24xlarge	c6i.metal	3rd Gen	4th Gen
Alignment, sorting	60	35	22	17	33.0	31.3	21.1
Deduplication, BQSR	5	5	3	5	4.5	3.2	2.2
Apply BQSR, variant calling	15	10	10	6	5.2	5.3	3.5
Total time	80	50	35	28	42.7	39.8	26.8

Table 2. Execution time (in minutes) for Parabricks* and Sentieon on a variety of cloud and on-premises platforms. Note that the Parabricks times are taken from the published results or approximated from visual inspection of Figure 1.

AWS EC2* instance	Processor	On-Demand Price (USD/ hour)	Spot Price (USD/ hour)	Cost-per-Genome (USD)
c6i.metal	Intel® Xeon® Platinum 8352M	$5.44	$2.1635	$2.1635/h * 42.7 minutes = $1.54
g4dn.12xlarge	NVIDIA T4 Tensor Core	$3.912	$1.2439	$1.2439/h * 80 minutes = $1.66
g4dn.metal	NVIDIA T4 Tensor Core	$7.824	$2.3472	$2.3472/h * 50 minutes = $1.96
p3dn.24xlarge	NVIDIA Tesla V100	$31.212	$9.3636	$9.3626/h * 35 minutes = $5.46
p4d.24xlarge	NVIDIA A100 Tensor Core	$32.7726	$9.8318	$9.8318/h * 28 minutes = $4.59

Table 3. AWS EC2* price comparisons for Intel® and NVIDIA instances. Prices are for on-demand and spot instances in the U.S. West (Oregon) region as of March 11, 2022. (See Amazon EC2 Pricing.)

The Parabricks blog cited above reports variant calling accuracy (F1 scores) comparable to GATK. Sentieon, however, is a consistent winner in the PrecisionFDA Truth Challenge administered by the U.S. Food and Drug Administration (**Figure 3**). The HG001 benchmark comes from this challenge. In the more recent PrecisionFDA Truth Challenge V2, Sentieon competed against 19 other teams and won four of the 12 categories. Parabricks was not among the entries.



COMMUNITY CHALLENGE AWARDS

HIGHEST
SNP Performance
in the precisionFDA Truth Challenge
AWARDED TO
Verily Life Sciences
Ryan Poplin Mark DePristo
Verily Life Sciences Team

HIGHEST
SNP Recall
in the precisionFDA Truth Challenge
AWARDED TO
Sentieon
Rafael Aldana Hanying Feng
Brendan Gallagher Jun Ye

HIGHEST
SNP Precision
in the precisionFDA Truth Challenge
AWARDED TO
Kinghorn Center for Clinical Genomics
Aaron Statham Mark Cowley Joseph Copty
Mark Pinese

HIGHEST
INDEL Performance
in the precisionFDA Truth Challenge
AWARDED TO
Sanofi-Genzyme
Deepak Grover

HIGHEST
INDEL Recall
in the precisionFDA Truth Challenge
AWARDED TO
Sanofi-Genzyme
Deepak Grover

HIGHEST
INDEL Precision
in the precisionFDA Truth Challenge
AWARDED TO
Sentieon
Rafael Aldana Hanying Feng
Brendan Gallagher Jun Ye

Figure 3. Results from the first PrecisionFDA Truth Challenge, showing Sentieon winning two of the six categories.

Sentieon does not use proprietary programming languages like CUDA, thus avoiding vendor lock-in. The software is written in standard C++. It is also optimized to take advantage of the vector processing capability of modern processors. Sentieon uses algorithmic improvements rather than expensive, power-hungry hardware to achieve performance. It supports and optimizes for all short- and long-read sequencing platforms, and it is a consistent winner in the FDA's open challenges. This demonstrates that Sentieon on Intel Xeon Scalable processors is the leadership platform for genome secondary analysis.

intel® software

Break Free of Code Boundaries

Experience the power of cross-architecture programming in the Intel® DevCloud for oneAPI.

Demo

Run our Mandelbrot demo on different architectures to see cross-architecture performance for yourself.

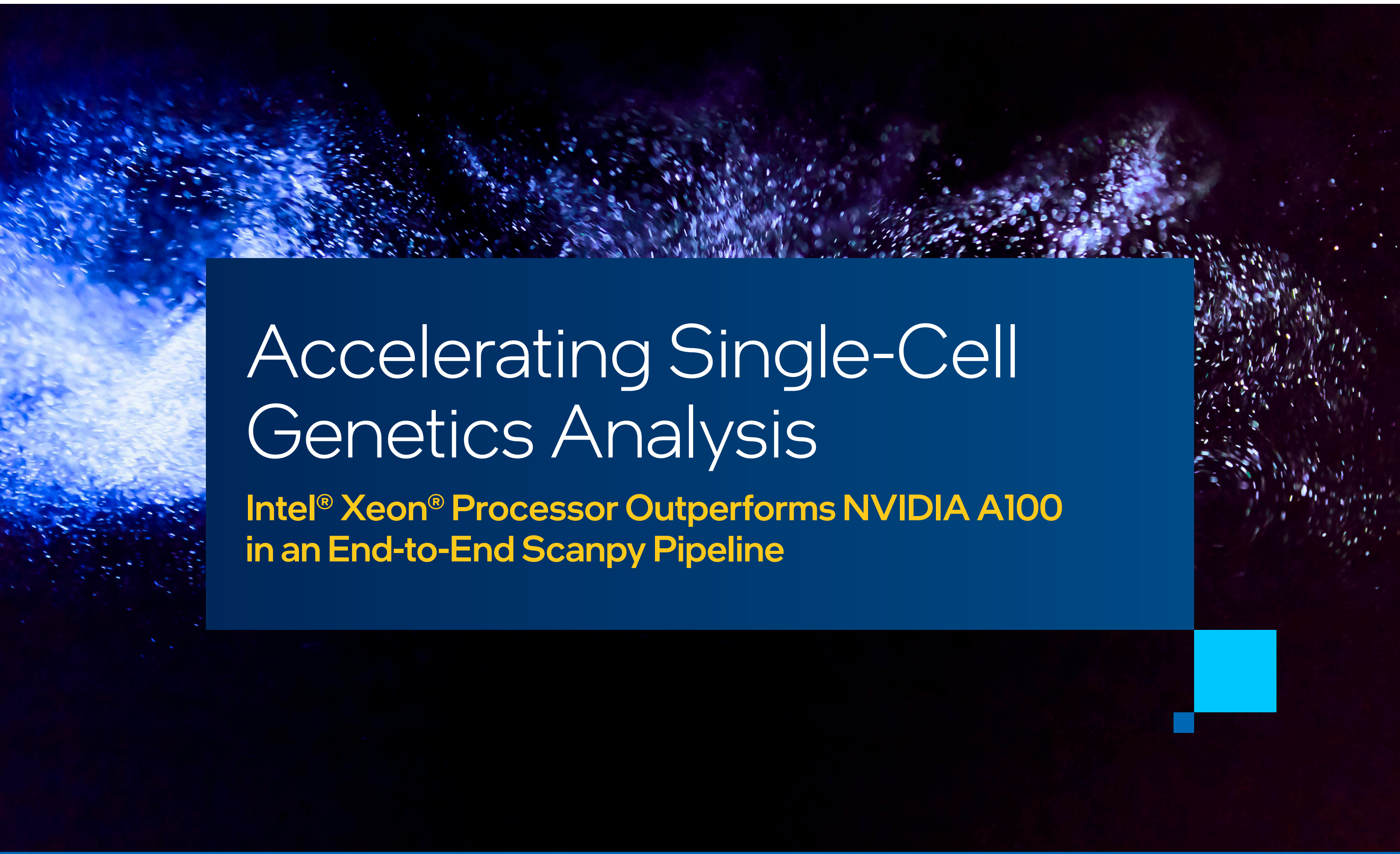
Learn

Get hands-on experience with Data Parallel C++ with 25 Jupyter notebooks loaded with code samples.

Develop

Plan and test future-ready applications on the latest Intel CPUs, GPUs, and FPGAs.

GET STARTED NOW >



Accelerating Single-Cell Genetics Analysis

Intel® Xeon® Processor Outperforms NVIDIA A100 in an End-to-End Scanpy Pipeline

Sanchit Misra, Senior Research Scientist, and Narendra Chaudhaury, Research Scientist, Intel Labs

Additional contributions from Padmanabhan Pillai, Senior Research Scientist, Bharat Kaul, Director of the Parallel Computing Lab, Henry Gabb, Senior Principal Engineer, Andrey Gorshkov, AI Software Engineering Manager, and Pavel Yakovlev, AI Frameworks Engineer, Intel Corporation; and Gurbinder Gill, Senior Software Engineer, Katana Graph

Increasing the resolution of measurement has always revolutionized fields; for example, the incredible scientific impact of the invention of microscope and telescope. Single-cell analysis is a good example of a similar revolution unfolding in biology. The human body is made up of nearly 40 trillion cells. Historically, these cells have been examined in bulk, sometimes millions of cells at a time, which cannot capture the differences across cells. Single-cell analysis looks at the individuality of cells. It is beginning to unravel the mystery of cell differentiation by identifying novel cell types, revealing mechanisms that make these cells different from each other, and demonstrating how cells respond to certain diseases or drugs. This relatively new field is already showing immense potential for biological discoveries with applications ranging from cancer to Covid-19 related research.

The amount of single-cell data is increasing at a rapid pace, thanks to the advancement of data measurement technologies. The size of individual datasets is increasing at a similar rate. Analysis of this data typically involves running a data science pipeline. Because the steps of the pipeline are often repeated with changes in parameters, it helps to have an interactive pipeline that can run in near real-time.

ScRNA-seq Analysis of 1.3 Million Mouse Cells in Just Seven and a Half Minutes on a Single Intel® Xeon® Processor

There are many kinds of single-cell analyses studying various aspects of cell-differentiation. Single-cell RNA-seq (scRNA-seq) analysis studies the differences in gene expression profiles across cells. It relies on single-cell RNA sequencing, which is an advanced technique that enables measurement of the gene expression of individual cells.

A typical workflow to do scRNA-seq analysis begins with a matrix that consists of the expression levels of the genes in each cell (**Figure 1**). In the data preprocessing steps, noise is filtered out and the data is normalized to obtain the activity of every human gene in each individual cell of the dataset. During this step, machine learning is often used to correct artifacts from data collection. Subsequently, dimensionality reduction is performed, followed by clustering to group cells with similar genetic activity, and then visualization of the clusters. With over 800,000 downloads, Scanpy is one of the most widely used toolkits for this analysis.

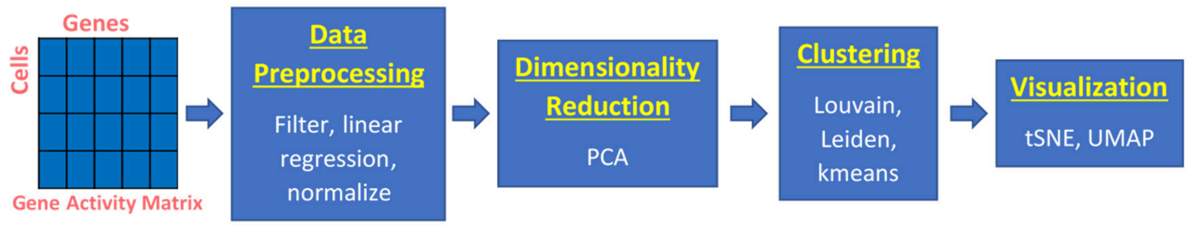


Figure 1. Pipeline showing the steps in analysis of single-cell RNA sequencing data, starting from gene activity matrix to visualization of different cell clusters.

For a dataset consisting of 1.3 million mouse brain cells, the pipeline depicted in **Figure 1** would normally take nearly 5 hours on a single CPU instance (n1-highmem-64) on Google Cloud Platform* (GCP*) using the off-the-shelf (baseline) Scanpy implementation. For the same pipeline, NVIDIA has reported an end-to-end runtime of 686 seconds on a single NVIDIA A100 GPU using NVIDIA RAPIDS*.

At Intel Labs, we collaborated with the Intel® oneAPI Data Analytics Library (oneDAL) team and Katana Graph to accelerate the pipeline using better parallel algorithms and tuning the performance to the underlying architecture. While this is still a work-in-progress, **Table 1** and **Figure 2** report our current performance and cloud usage costs. These results were recently presented at Intel Investor Day 2022. The whole pipeline can now be finished on the same single CPU instance (n1-highmem-64) on GCP in just 626 seconds. This performance only gets better with the newer n2 instance types running 3rd Generation Intel® Xeon® Scalable processors (Ice Lake). We also reduced the memory requirement of the pipeline so that we can use the low-memory n2-highcpu-64 instances instead of high-memory n2-highmem-64 instances. On a single instance of n2-highcpu-64 on GCP, the whole pipeline finishes in just 459 seconds (7.65 minutes). This is nearly 40x faster than the 5-hour baseline that we started with. This is also nearly 1.5x faster than the NVIDIA* A100 GPU performance.

The speed-up and reduction in memory requirement has resulted in a significant reduction in cloud computing costs (**Table 1**). The n2-highcpu-64 instance on GCP costs only $0.29. This is nearly 66x cheaper than n1-highmem-64 running baseline Scanpy, and 2.4x cheaper than an NVIDIA A100 GPU.

Pipeline step	CPU n1-highmem-64 64 vCPUs (baseline Scanpy)	GPU a2-highgpu-1g Tesla A100 40GB GPU (GPU-accelerated Scanpy)	CPU n1-highmem-64 64 vCPUs (CPU-accelerated Scanpy)	CPU n1-highmem-64 64 vCPUs (CPU-accelerated Scanpy)	CPU n2-highcpu-64 64 vCPUs (CPU-accelerated Scanpy)
Data load + Preprocessing	1120	475	16.9	11.3	16.6
PCA	44	17.8	6.9	5.6	5.6
t-SNE	6509	37	216.2	175.6	172.8
k-means (single iteration)	148	2	11.1	7.8	8.1
KNN	154	62	73.8	60.0	64.2
UMAP	2571	21	167.4	100.8	96.2
Louvain clustering	1153	2.4	13.9	10.3	8.9
Leiden clustering	6345	1.7	52.8	36.4	34.5
Re-analysis of subgroup	255	17.9	23.9	20.8	19.2
Rest	39	49.2	42.7	33.6	32.8
End-to-end run time (seconds)	18338	686	625.7	462.1	458.8
On-demand Price (USD/hr)[1]	3.786	3.673	3.786	4.192	2.294
Total cost (USD)	19.284	0.700	0.658	0.538	0.292

Table 1. Execution time and cloud costs for scRNA-seq analysis of 1.3 million mouse brain cells on various GCP instances. The first two columns report published execution time and cloud costs of baseline Scanpy on a single CPU instance (n1-highmem-64) and GPU-accelerated Scanpy on a single GPU instance (a2-highgpu-1g). The last three columns report measured[2] execution time and cloud costs of CPU-accelerated Scanpy on single instances of two generations of CPU instance types (n1-highmem-64, n1-highmem-64, and n2-highcpu-64).

[1]As mentioned on this link on May 15, 2022: https://cloud.google.com/compute/vm-instance-pricing.
[2]Tests by Intel on May 25, 2022

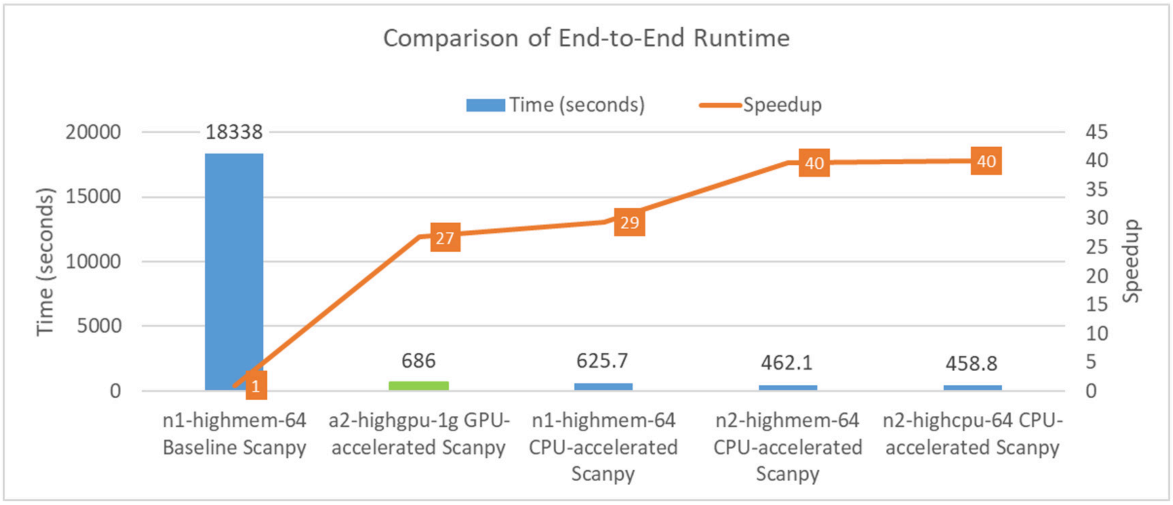


Figure 2. Execution time and speed-up for scRNA-seq analysis of 1.3 million mouse brain cells on various GCP instances. The chart uses (1) published execution time of baseline Scanpy on a single CPU instance (n1-highmem-64) and GPU-accelerated Scanpy on a single GPU instance (a2-highgpu-1g), and (2) measured[3] execution time of CPU-accelerated Scanpy on single instances of two generations of CPU instance types (n1-highmem-64, n2-highmem-64 and n2-highcpu-64). In addition, the line graph shows the speed-up over baseline Scanpy running on n1-highmem-64 instance.

How Was the Data Science Pipeline Accelerated?

Detailed below is a brief summary of the steps we took to improve the performance of this pipeline:

- To increase the efficiency of data preprocessing, we used a warm file cache and multithreading using Numba, a just-in-time (JIT) compiler. This improved the baseline preprocessing performance by more than 70x.
- We also used the Intel® Extension for Scikit-learn*, which has efficient implementations of K-means clustering, KNN (K Nearest Neighbor), and PCA (Principal Component Analysis).
- Scanpy originally used scikit-learn's tSNE (t-distributed Stochastic Neighbor Embedding) implementation, which was inefficient for Intel Xeon processors. We achieved nearly 40x speed-up of tSNE by building an efficient implementation through:
  - A shared-memory parallel implementation of the Barnes-Hut algorithm
  - Parallelization of quadtree building, sorting, and summarization steps using Morton codes
- Continuing our efforts, we optimized (Uniform Manifold Approximation and Projection) by:
  - Converting the Python* source code to C++
  - Creating an efficient AVX512/AVX2-based implementation for a pseudo-random number generator
  - Using Intel® oneAPI Math Kernel Library (oneMKL) for the eigenvalue computation step
- As part of our collaboration, Katana Graph provided efficient implementations of the Louvain and Leiden community detection algorithms that were integrated into the pipeline.

[3]Tests by Intel on May 25, 2022

These developments significantly reduce the time it takes to analyze large datasets, allowing researchers to complete their work 40x faster than baseline on Intel Xeon processors and 1.5x faster than on NVIDIA* A100 GPUs.

Conclusions

Single-cell analysis has applications in many areas: oncology, microbiology, neurology, reproduction, immunology, and digestive and urinary systems. Hopefully, reduced working time will allow for a much deeper understanding of different cells, paving the way for medical advances that could have great collective benefits. We are working on further refining the scRNA-seq analysis pipeline. Specifically, our efforts are focused on making further improvements in the tSNE, UMAP, and Leiden steps.

Configuration Details

GCP n1-highmem-64: 1-instance GCP n1-highmem-64: 64 vCPUs (Skylake), 416 GB total memory, bios: Google, ucode: 0x1, Ubuntu* 20.04, 5.13.0-1024-gcp

GCP n2-highmem-64: 1-instance GCP n2-highmem-64: 64 vCPUs (Ice Lake), 512 GB total memory, bios: Google, ucode: 0x1, Ubuntu 20.04, 5.13.0-1024-gcp

GCP n2-highcpu-64: 1-instance GCP n2-highcpu-64: 64 vCPUs (Ice Lake), 64 GB total memory, bios: Google, ucode: 0x1, Ubuntu 20.04, 5.13.0-1024-gcp

Sign up for future issues

Leveraging Tools for Cross-Platform Software Development

How to Build oneAPI for Linux* and Windows* using Windows Subsystem for Linux 2* and Visual Studio Code*

Tony Mongkolsmai, Technology Evangelist, Intel Corporation

Hello, world! As a 19-year software engineer and software architect at Intel in performance and parallelization tools and AI system platforms, I am very lucky and excited to share some of my experiences with you and get your thoughts about software, technology, and Intel. My first focus is trying to tackle the dilemmas of the modern software developer.

Why tackle this first? For as long as I can remember, I have always loved programming and solving problems. I started programming using BASIC when I was in first grade in 1985, when it was still slightly less common. I've been fortunate enough (or unfortunate enough?) to have learned a few dozen programming languages, programmed in several IDEs, and developed software in both Windows* and Linux* environments. I've also had to develop in rapidly evolving areas, like web/UI frameworks, cloud development, deep learning, etc.

One constant over this time is that engineers are always fighting their tools to be more productive. As a developer, I want to be as efficient as possible to get more work done or to free up my time to do something else in my life. So, this means a lot to me personally.

Narrowing the Scope

Unfortunately for us, there is no one modern software developer. When I write Go* code for microservices or Kubernetes*, I use a quite different toolchain than I do when I'm writing React/Angular/Vue/JavaScript* code. There are also a lot of times I need cross-platform support between Windows/Linux/Mac*.

Fortunately, there have been advances in cross-platform IDEs and support. I am a longtime Vi user (no offense, Emacs users; I know Emacs is more powerful), but I like modern IDEs. As an engineer, I love normalizing my workflow across platforms because it makes me more efficient. Luckily for me, the languages I typically use (C/C++, Python*, JavaScript, Go) all work reasonably well within Microsoft Visual Studio Code* (VSCode*), and recent updates to the Microsoft Window Subsystem for Linux* (WSL) also work well for cross-platform development.

Given that backdrop, I thought I'd see how well I can get some cross-platform oneAPI samples running on my Intel® Core™ i9 processor-based Alienware* R13 system (not purchased by Intel).

Setting up WSL and VSCode*

Setting up WSL was painless: I simply opened a Windows command prompt and ran:

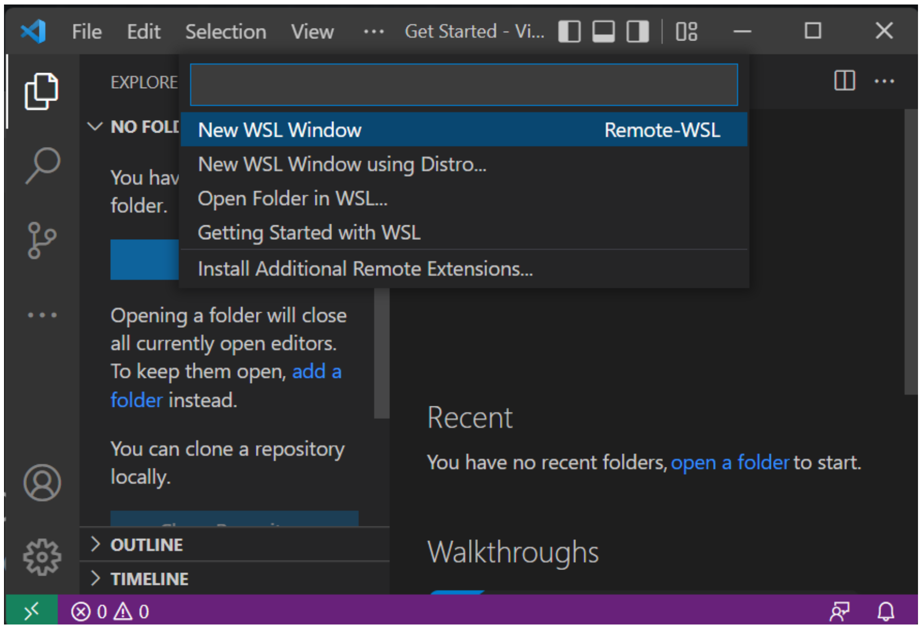
```
> wsl --install
```

This installed the Ubuntu* 20.04 distro for WSL after a quick, required reboot.

For me the process was painless, but some of my colleagues mentioned that if you do run into issues, Microsoft has provided a handy troubleshooting guide that will guide you through the prerequisite steps to make your system WSL-ready.

The next step was setting up my VSCode. I initially tried to set up my VSCode in WSL using a Linux installation workflow, but it turns out that wasn't the proper way to do it. Following the instructions from Microsoft, I simply had to enable the Remote WSL extension in VSCode following the instructions.

After enabling the extension, it was as easy as asking for a new WSL Window (**Figure 1**), which popped up as a VSCode WSL-connected window.



You hav folder.

Opening a folder will close all currently open editors. To keep them open, add a folder instead.

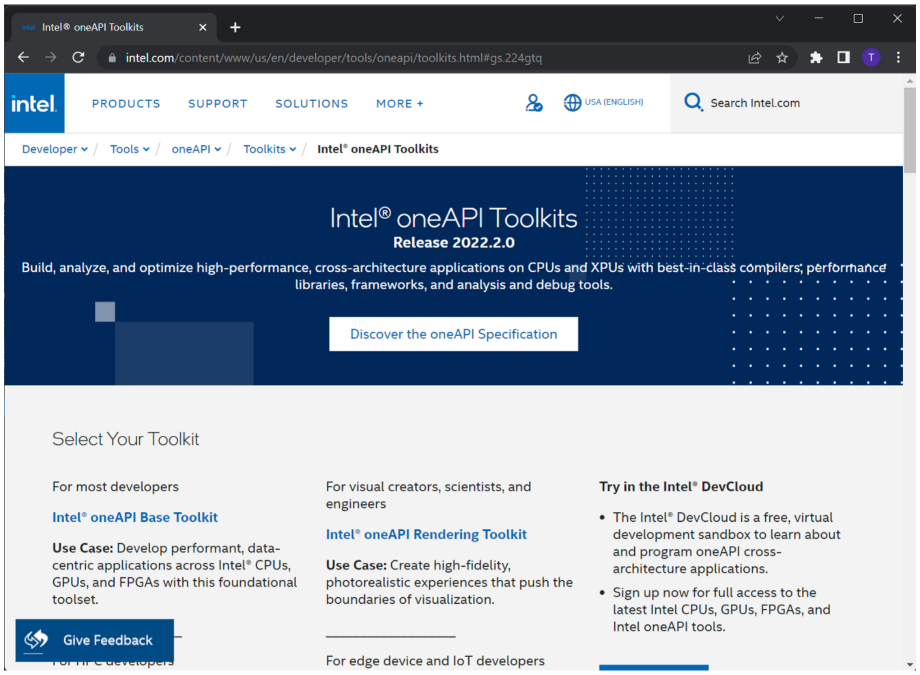
You can clone a repository locally.

You have no recent folders, open a folder to start.

Figure 1. Opening a new WSL window.

Installing the Intel® oneAPI Toolkits

Because I wanted to see if I could run both Linux and Windows oneAPI-based code on my system, I went to the Intel® oneAPI Toolkits release page (**Figure 2**), selected the toolkits that I wanted, and followed the instructions to install them on my system (**Figure 3**). I used an apt installation in a WSL terminal and the online installer for Windows.


Intel® oneAPI Toolkits
intel.com/content/www/us/en/developer/tools/oneapi/toolkits.html#gs.224gtq
intel
PRODUCTS
SUPPORT
SOLUTIONS
MORE +
USA (ENGLISH)
Search Intel.com
Developer / Tools / oneAPI / Toolkits / Intel® oneAPI Toolkits
Discover the oneAPI Specification
Give Feedback

Intel® oneAPI Toolkits
Release 2022.2.0

Build, analyze, and optimize high-performance, cross-architecture applications on CPUs and XPUs with best-in-class compilers, performance libraries, frameworks, and analysis and debug tools.

Select Your Toolkit

For most developers

Intel® oneAPI Base Toolkit

Use Case: Develop performant, data-centric applications across Intel® CPUs, GPUs, and FPGAs with this foundational toolset.

For HPC developers

For visual creators, scientists, and engineers

Intel® oneAPI Rendering Toolkit

Use Case: Create high-fidelity, photorealistic experiences that push the boundaries of visualization.

For edge device and IoT developers

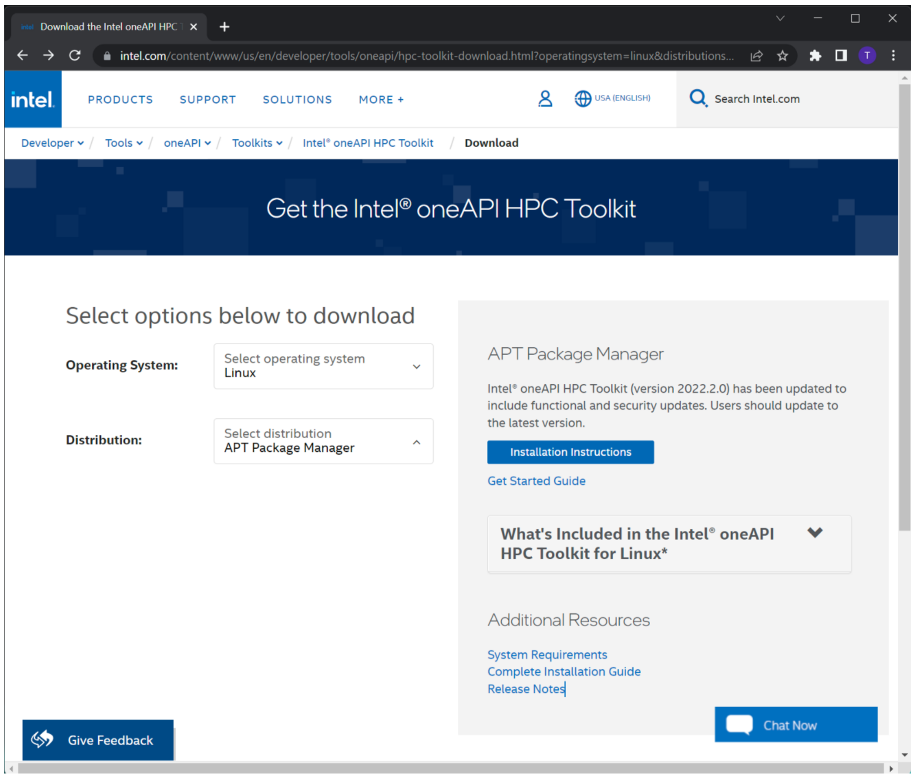
Try in the Intel® DevCloud

- The Intel® DevCloud is a free, virtual development sandbox to learn about and program oneAPI cross-architecture applications.
- Sign up now for full access to the latest Intel CPUs, GPUs, FPGAs, and Intel oneAPI tools.

Figure 2. Selecting the desired Intel® oneAPI toolkits.

*Other names and brands may be claimed as the property of others.
For more complete information about compiler optimizations, see our Optimization Notice.

Sign up for future issues


Download the Intel oneAPI HPC

intel.com/content/www/us/en/developer/tools/oneapi/hpc-toolkit-download.html?operatingsystem=linux&distributions...

intel.
PRODUCTS
SUPPORT
SOLUTIONS
MORE +
USA (ENGLISH)
Search Intel.com
Developer / Tools / oneAPI / Toolkits / Intel® oneAPI HPC Toolkit / Download

Get the Intel® oneAPI HPC Toolkit

Select options below to download

Operating System: Select operating system Linux

Distribution: Select distribution APT Package Manager

APT Package Manager

Intel® oneAPI HPC Toolkit (version 2022.2.0) has been updated to include functional and security updates. Users should update to the latest version.

What's Included in the Intel® oneAPI HPC Toolkit for Linux*

Additional Resources

Figure 3. Installing the Intel® oneAPI toolkits.

Running Some Code

Now that I theoretically had a working development environment, I cloned the oneAPI-samples repository from GitHub.

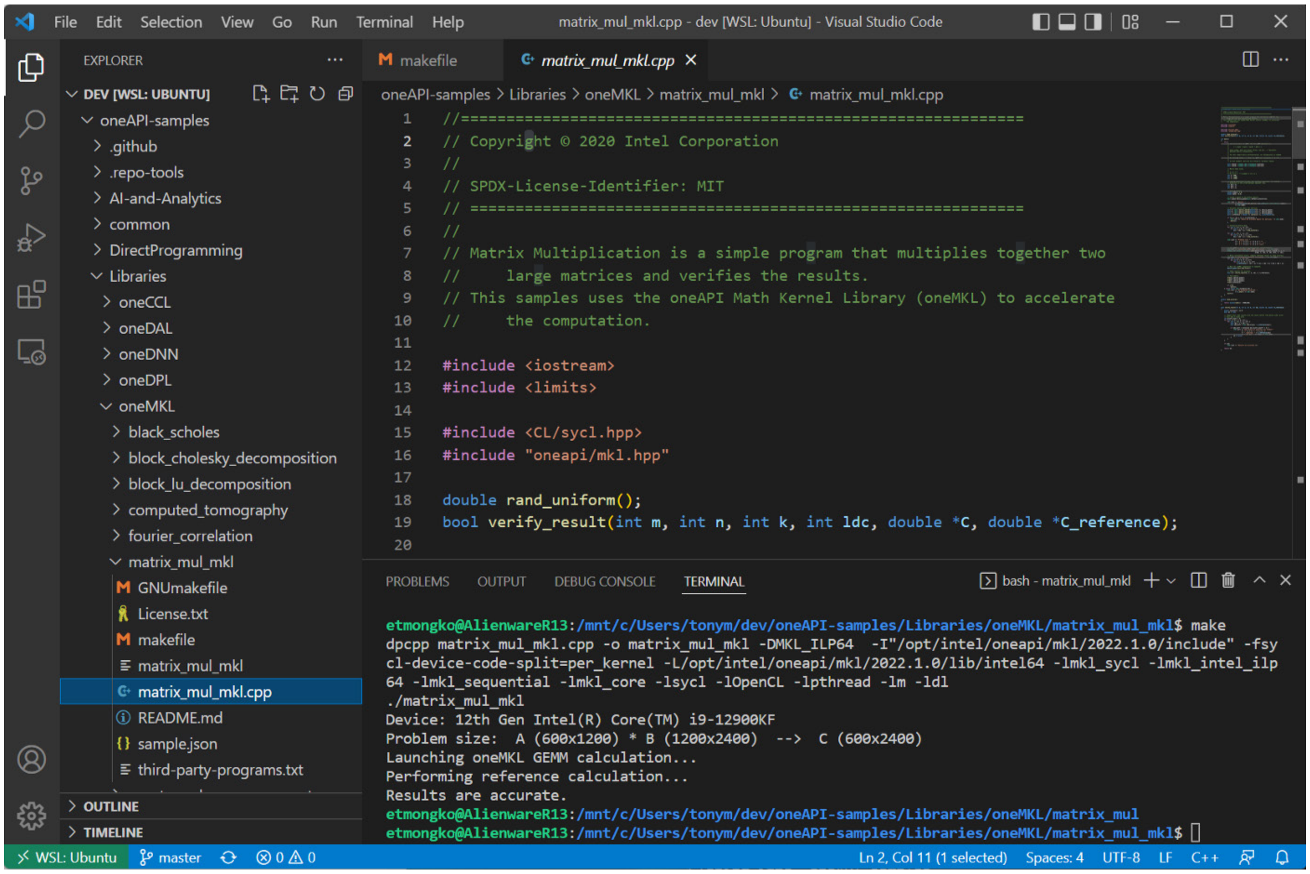
This code is designed to be both Windows and Linux compatible, so I loaded the Libraries > oneMKL > matrix_mul_mkl directory in my WSL VSCode window and opened a bash terminal. I set up the oneAPI environment and built the sample code:

```
> source /opt/intel/oneapi/setvars.sh
> make
```

This successfully built and ran the matrix multiplication example (**Figure 4**).

*Other names and brands may be claimed as the property of others.
For more complete information about compiler optimizations, see our Optimization Notice.

Sign up for future issues



```
//==============================================================
// Copyright © 2020 Intel Corporation
//
// SPDX-License-Identifier: MIT
// =============================================================
//
// Matrix Multiplication is a simple program that multiplies together two
//     large matrices and verifies the results.
// This samples uses the oneAPI Math Kernel Library (oneMKL) to accelerate
//     the computation.

#include <iostream>
#include <limits>

#include <CL/sycl.hpp>
#include "oneapi/mkl.hpp"

double rand_uniform();
bool verify_result(int m, int n, int k, int ldc, double *C, double *C_reference);
```

```
etmongko@AlienwareR13:/mnt/c/Users/tonym/dev/oneAPI-samples/Libraries/oneMKL/matrix_mul_mkl$ make
dpcpp matrix_mul_mkl.cpp -o matrix_mul_mkl -DMKL_ILP64  -I"/opt/intel/oneapi/mkl/2022.1.0/include" -fsy
cl-device-code-split=per_kernel -L/opt/intel/oneapi/mkl/2022.1.0/lib/intel64 -lmkl_sycl -lmkl_intel_ilp
64 -lmkl_sequential -lmkl_core -lsycl -lOpenCL -lpthread -lm -ldl
./matrix_mul_mkl
Device: 12th Gen Intel(R) Core(TM) i9-12900KF
Problem size:  A (600x1200) * B (1200x2400)  -->  C (600x2400)
Launching oneMKL GEMM calculation...
Performing reference calculation...
Results are accurate.
etmongko@AlienwareR13:/mnt/c/Users/tonym/dev/oneAPI-samples/Libraries/oneMKL/matrix_mul
etmongko@AlienwareR13:/mnt/c/Users/tonym/dev/oneAPI-samples/Libraries/oneMKL/matrix_mul_mkl$ 
```

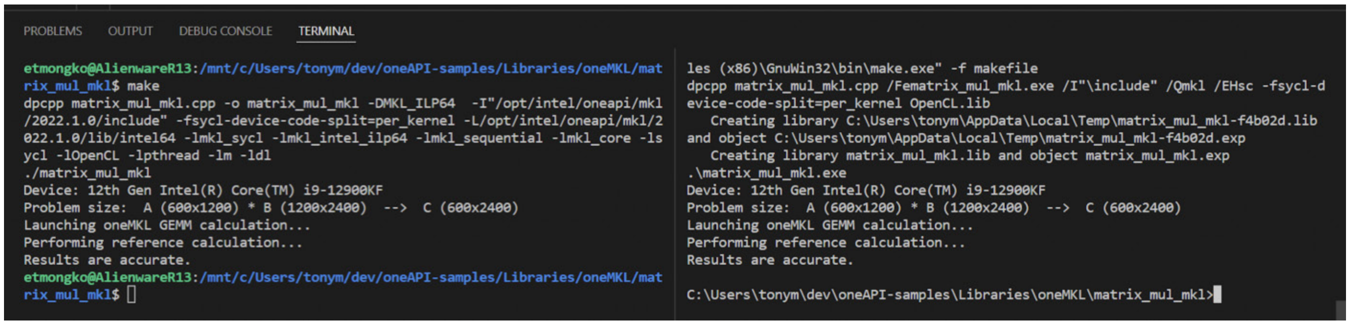
Figure 4. Building and running the example code.

Creating and running the sample using a Windows binary ended up being a little more work, as this was a new system. I had to install the Microsoft Visual Studio* 2022 Community Edition and GNU Make. After that, it again was as simple as loading up VSCode and running:

```
> C:\Program Files (x86)\Intel\oneAPI\setvars.bat
> make
```

But Wait, There's More!

I'm sure many of you know this, but this next thing was an amazing discovery for me: I was playing with VSCode (Windows, not WSL-based) and saw that I could select a WSL prompt for my split terminal (**Figure 5**)! Doing that allowed me to build and run the sample for both Windows and Linux in the same VSCode instance. Cross-platform development on a single system in a single IDE! I cannot express the endorphin high from the magic of this after having spent years dealing with the pain of cross-platform development.



```
etmongko@AlienwareR13:/mnt/c/Users/tonym/dev/oneAPI-samples/Libraries/oneMKL/mat
rix_mul_mkl$ make
dpcpp matrix_mul_mkl.cpp -o matrix_mul_mkl -DMKL_ILP64  -I"/opt/intel/oneapi/mkl
/2022.1.0/include" -fsycl-device-code-split=per_kernel -L/opt/intel/oneapi/mkl/2
022.1.0/lib/intel64 -lmkl_sycl -lmkl_intel_ilp64 -lmkl_sequential -lmkl_core -ls
ycl -lOpenCL -lpthread -lm -ldl
./matrix_mul_mkl
Device: 12th Gen Intel(R) Core(TM) i9-12900KF
Problem size:  A (600x1200) * B (1200x2400)  -->  C (600x2400)
Launching oneMKL GEMM calculation...
Performing reference calculation...
Results are accurate.
etmongko@AlienwareR13:/mnt/c/Users/tonym/dev/oneAPI-samples/Libraries/oneMKL/mat
rix_mul_mkl$ 
```

```
les (x86)\GnuWin32\bin\make.exe" -f makefile
dpcpp matrix_mul_mkl.cpp /Fematrix_mul_mkl.exe /I"\include" /Qmkl /EHsc -fsycl-d
evice-code-split=per_kernel OpenCL.lib
   Creating library C:\Users\tonym\AppData\Local\Temp\matrix_mul_mkl-f4b02d.lib
and object C:\Users\tonym\AppData\Local\Temp\matrix_mul_mkl-f4b02d.exp
   Creating library matrix_mul_mkl.lib and object matrix_mul_mkl.exp
.\matrix_mul_mkl.exe
Device: 12th Gen Intel(R) Core(TM) i9-12900KF
Problem size:  A (600x1200) * B (1200x2400)  -->  C (600x2400)
Launching oneMKL GEMM calculation...
Performing reference calculation...
Results are accurate.

C:\Users\tonym\dev\oneAPI-samples\Libraries\oneMKL\matrix_mul_mkl>
```

Figure 5. Running Windows* and Linux* in the same VSCode* instance.

The End (For Now)

The nice thing about all of this is that I was able to set all of this up on a new system in just a couple of hours. I'll be writing more on how well this works for Intel® Arc™ GPUs, discussing other developer workflows (like Jupyter Notebook*), diving into software topics like cloud computing and AI system platform development, and sharing my experiences as a software engineer in the technology industry. Thanks for reading!

intel software

THE PARALLEL UNIVERSE