IISWC'22

Overcoming the challenges when viewing oneAPI as a performance workload

Paul Petersen

Intel Fellow

Notices & Disclaimers

- Intel technologies may require enabled hardware, software or service activation.
- No product or component can be absolutely secure.
- Your costs and results may vary.
-

Agenda

- What is oneAPI?
- Viewing oneAPI as a workload

Accelerators are mainstream

- The world has concluded accelerators are here
  - Intel has led this trend with SVMS architectures and our XPU strategy
- To succeed, we need to deliver the right software environment
  - Simplify accelerator software development without sacrificing performance
- Our solution is **oneAPI**
  - A cross-industry, open, standards-based unified programming model delivering a common developer experience across CPU and XPU architectures
  - Three elements: an open specification, open-source implementations and Intel-enhanced products

What is oneAPI?

Open-source software stack

Built with industry standard components
(CLANG, LLVM, SPIR-V)

Intel LLVM | oneAPI Open-Source Projects

An open community

github/oneAPI-TAB

Technical Advisory Boards	Global Support	Implementations
SYCL	US Dept of Energy	Intel
oneMKL	Silicon Pearl	NVidia, AMD
oneDNN	Fugaku	Xilinx, ARM

An open specification

oneAPI
oneAPI.io

Building on other open standards
(SYCL 2020, OpenMP, BLAS, ...)

creative commons

Intel Products

oneAPI
software.intel.com/oneapi

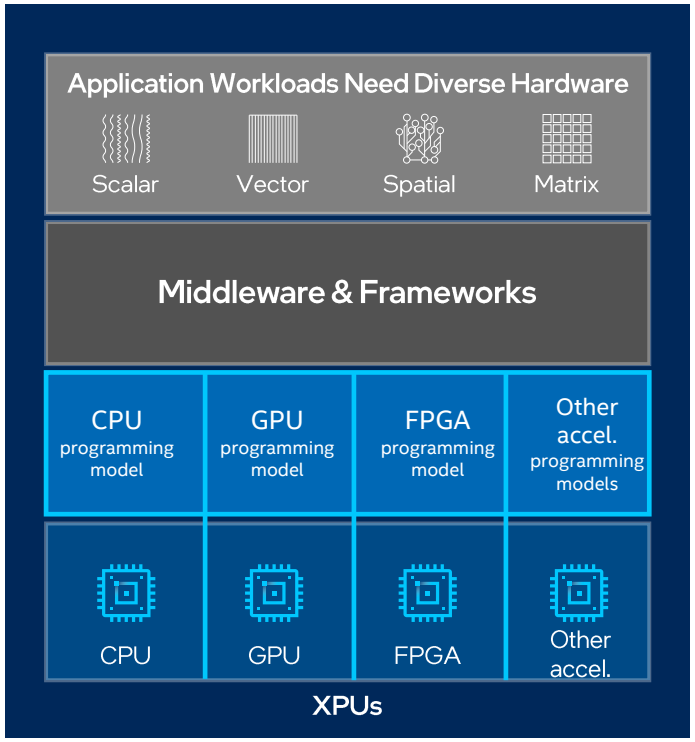
Intel® oneAPI Toolkits
Release 2022.2

Available on

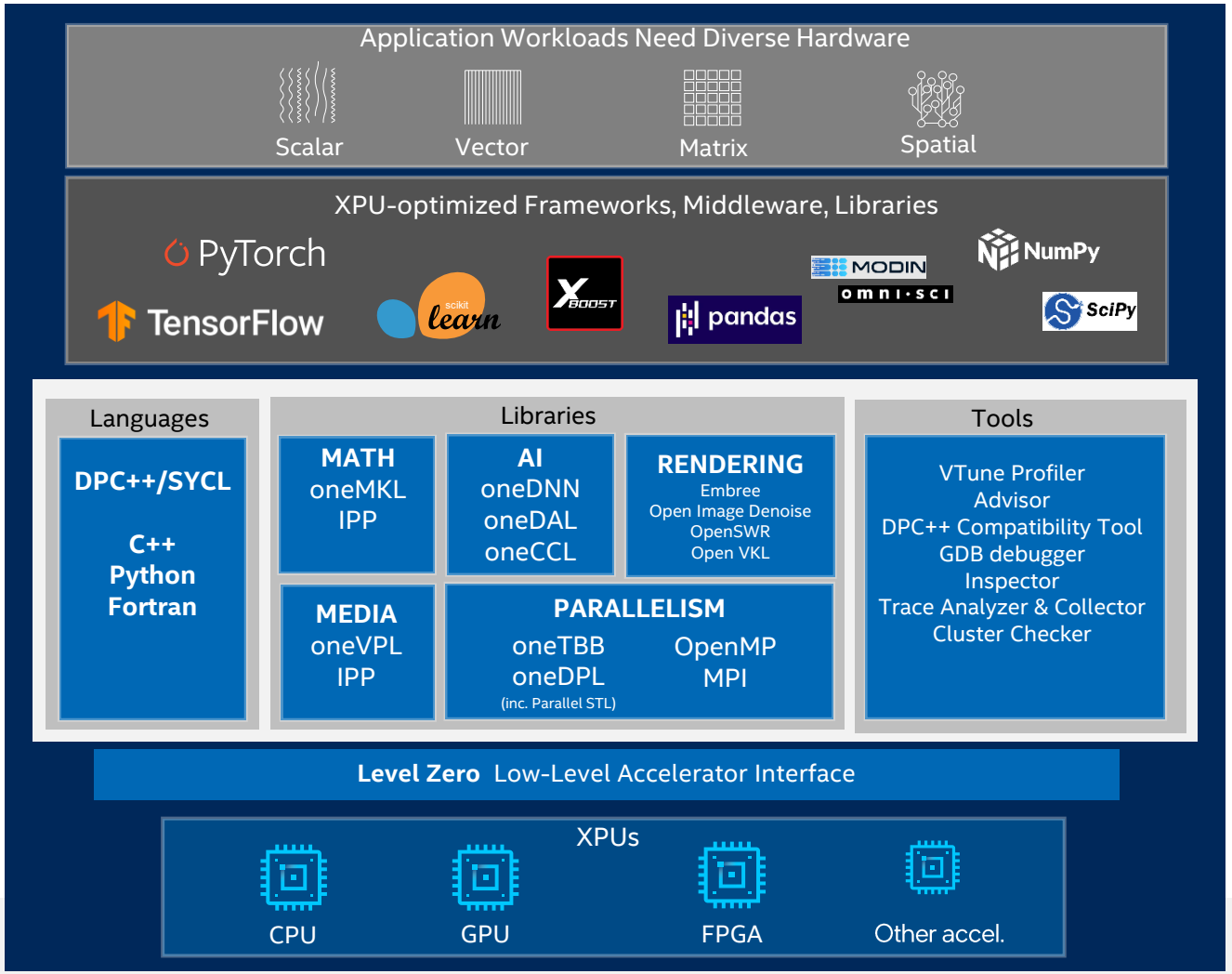
apt | GitHub | yum/dnf | intel | zypper | nuget | CONDA | Maven | Spack

oneAPI enables cross-platform XPU programming

Programming challenges for multiple architectures



oneAPI open specification provides a standard programming language, libraries, and hardware abstraction layer



SYCL 2020: *Khronos standard ready to support accelerators*

Based on modern C++ / Decoupled from OpenCL

Unified address space

```
sycl::malloc_host
sycl::malloc_device
sycl::malloc_shared
```

Atomic ops on non-atomic objects

```
sycl::atomic_ref
```

Subgroups (similar to CUDA warp)
Subgroup and work group algorithms

```
Any_of, all_of, non_of, reduce, exclusive_scan,
inclusive_scan, shift_left, shift_right, select, permute
```

Reductions

```
sycl::reduction
```

SYCL 1.2.1

```
float *sum = (float *)malloc(sizeof(float));
float *data = (float *)malloc(N*sizeof(float));
const size_t tc = N/ITEMS_PER_THREAD;
const size_t itb = ITEMS_PER_THREAD*BS;
float *output = (float *)malloc(tc*sizeof(float));
buffer<float, 1> buf(data, N);
buffer<float, 1> ans(output, tc);
q.submit([&](handler &h) {
  auto buf_acc = buf.get_access<access::mode::read>(h);
  auto ans_acc = ans.get_access<access::mode::discard_write>(h);
  h.parallel_for<class SumK>(nd_range<1>(thread_count, BS),
    [=](nd_item<1> it) {
      const size_t st = it.get_group(0)*itb+it.get_local_id(0);
      float lsum = 0;
      for (size_t i = st; i <st+itb; i += it.get_local_range(0)) {
        lsum += buf_acc[i];
      }
      ans_acc[it.get_global_id(0)] = lsum;
    });
}).wait();
auto ans_acc = ans.get_access<access::mode::read>();
*sum = 0;
for (size_t i = 0; i < tc; ++i) { *sum += ans_acc[i]; }
```

SYCL 2020

```
float* sum = malloc_shared<float>(1, q);
float* data = malloc_shared<float>(N, q);


q.parallel_for(N, reduction(sum, std::plus<>()),
  [=](size_t i, auto& sum) {
    sum += data[i];
});
```

KHRONOS GROUP · SYCL · oneAPI · codeplay · ComputeCpp · hipSYCL

OpenMP 5.x: *directive based parallel accelerator programming*

Hierarchical parallelism

```
#pragma omp target teams distribute
for (...) {
    #pragma omp parallel for
    for ( ...) {
        for (...) {
            #pragma omp simd
            for (...) {
                for (...) {

                }
            }
        }
    }
}
```

Unified Shared Memory

```
#pragma omp requires unified_address
 A = omp_target_alloc_shared(...);
```

Or explicit control of data movement

```
int *arr_host = malloc(...);
int *arr_device = omp_target_alloc_device(...);
#pragma omp target is_device_ptr(arr_device)
#pragma omp target map(tofrom: arr_host[0:N])
```

oneAPI Libraries

Domain	Name	Description	Open Spec	Open Source
Parallel Programming	oneDPL	Data Parallel C++ Library including Parallel STL	Yes	Yes
	oneTBB	Threading Building Blocks	Yes	Yes
AI & ML	oneDNN	Deep Neural Networks	Yes	Yes
	oneCCL	Collective Communications	Yes	Yes
	oneDAL	Data Analytics and Machine learning	Yes	Yes
Math	oneMKL	Math Kernels: linear algebra, FFT, random numger generation	Yes	Partial
Video	oneVPL	Video Processing: encode, decode, transcode	Yes	Yes
Ray Tracing	Embree, VKL, OID, OSPRay	Geometric & Volumetric Ray Tracing, Image Denoise, Scalable Rendering	Yes	Yes

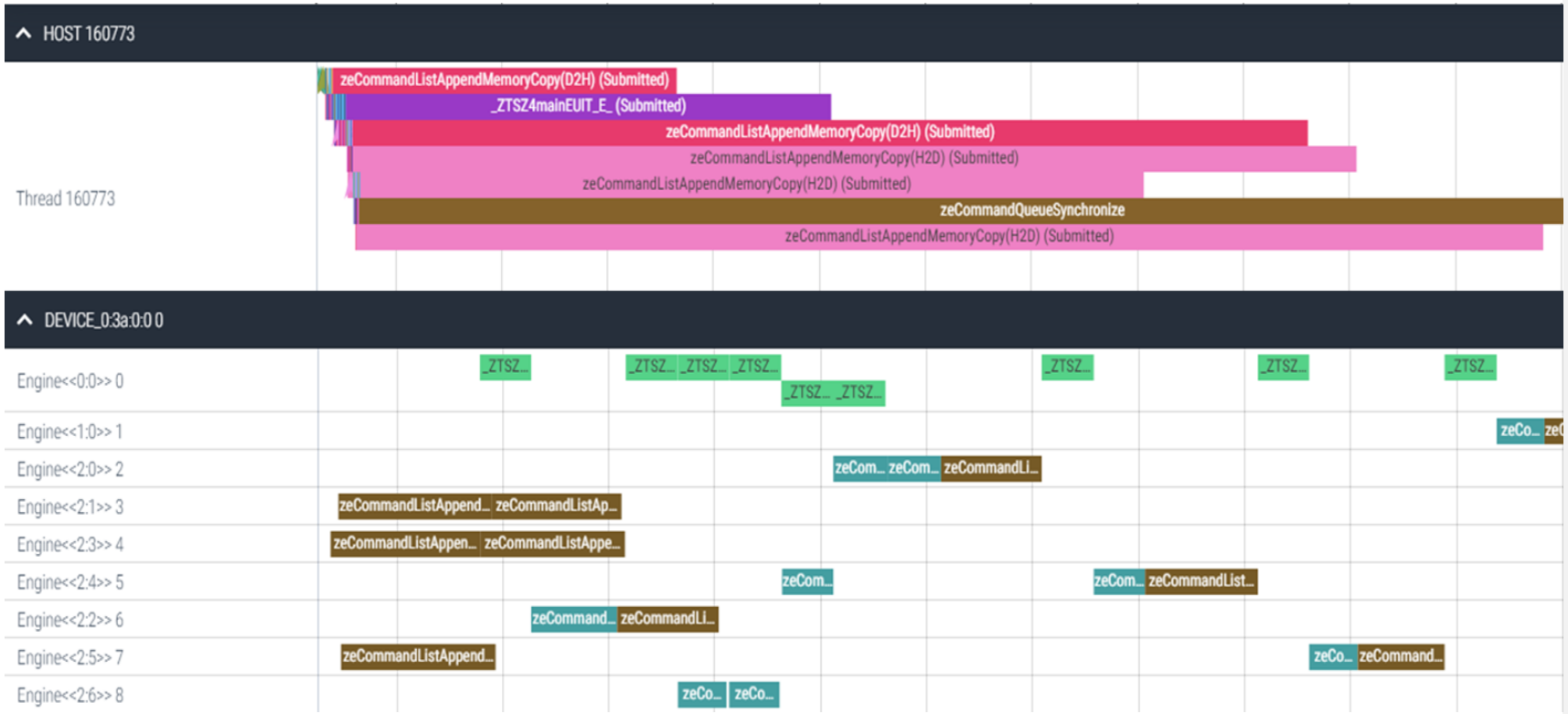
A Few Basics to Consider for Accelerator Programming

- CPU Behavior
  - Is the CPU runtime and driver overhead too high?
- CPU+GPU Behaviors
  - Are we overlapping actions across the CPU and GPU?
- GPU Behavior
  - **Occupancy**: Are the compute units fully utilized or occupied?
  - **Stall**: Are the compute units stalled? What causes the stalls?
  - **ALU Utilization**: Is ALU kept busy enough or is it saturated?

Acknowledgement: Thanks to Zhiqiang Ma for providing this material

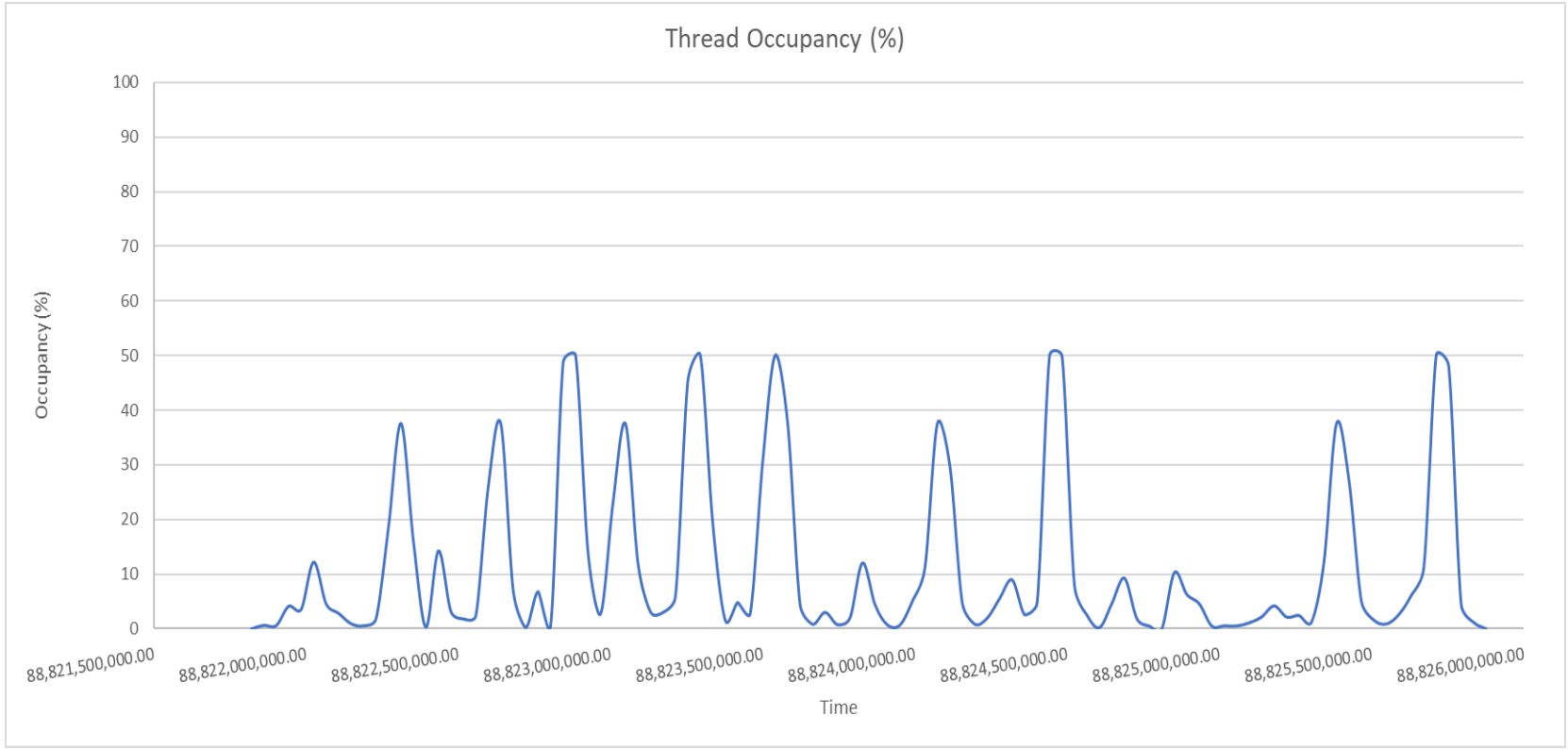
CPU+GPU Behaviors

- Overlap between CPU and GPU improves performance
- Reducing the overhead of CPU runtime stack improves efficiency
- Overlapping data transfer with compute reduces latency



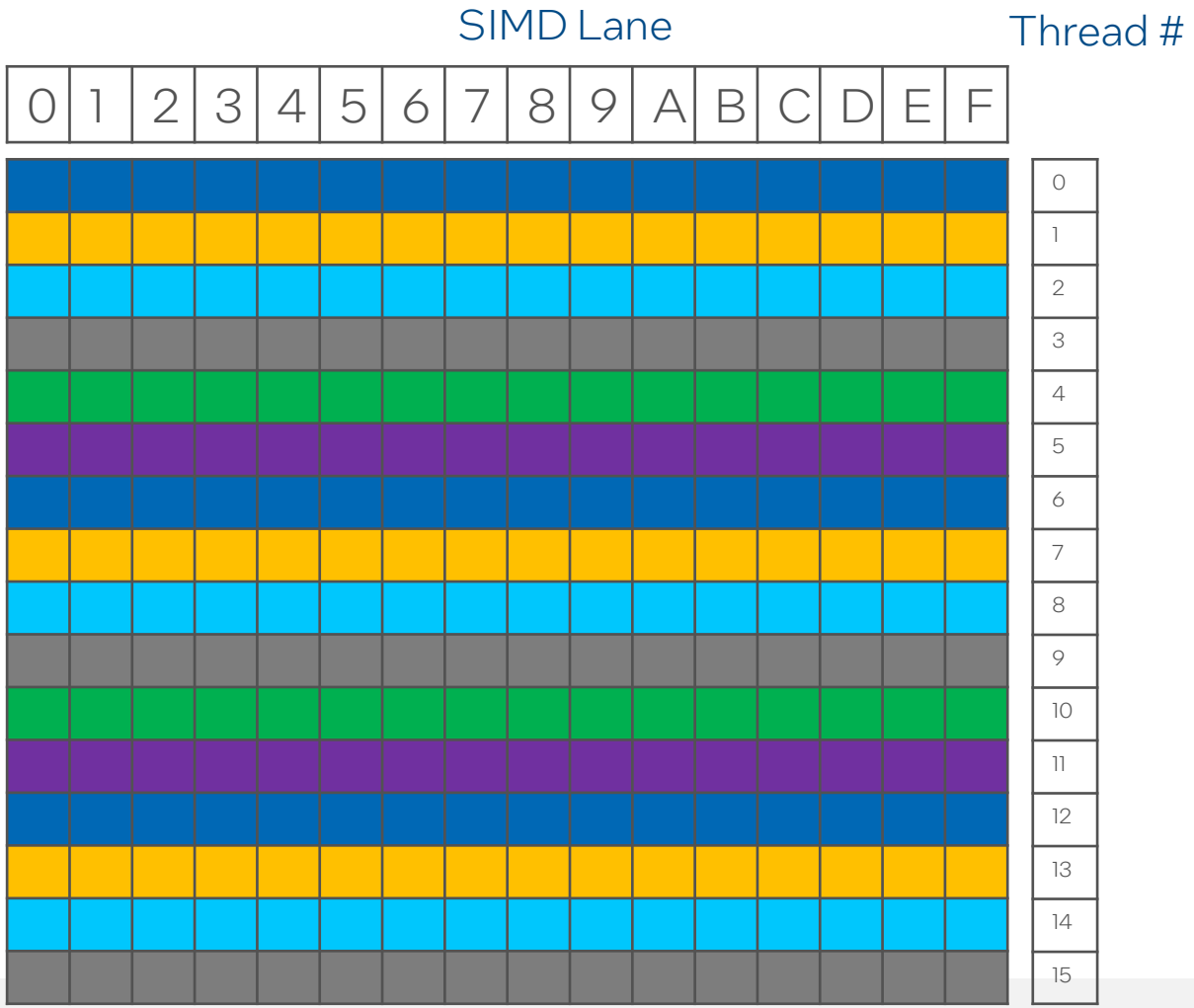
GPU: Low Thread Occupancy

- High occupancy is critical for instruction latency hiding
- Low occupancy results in low hardware resource utilization



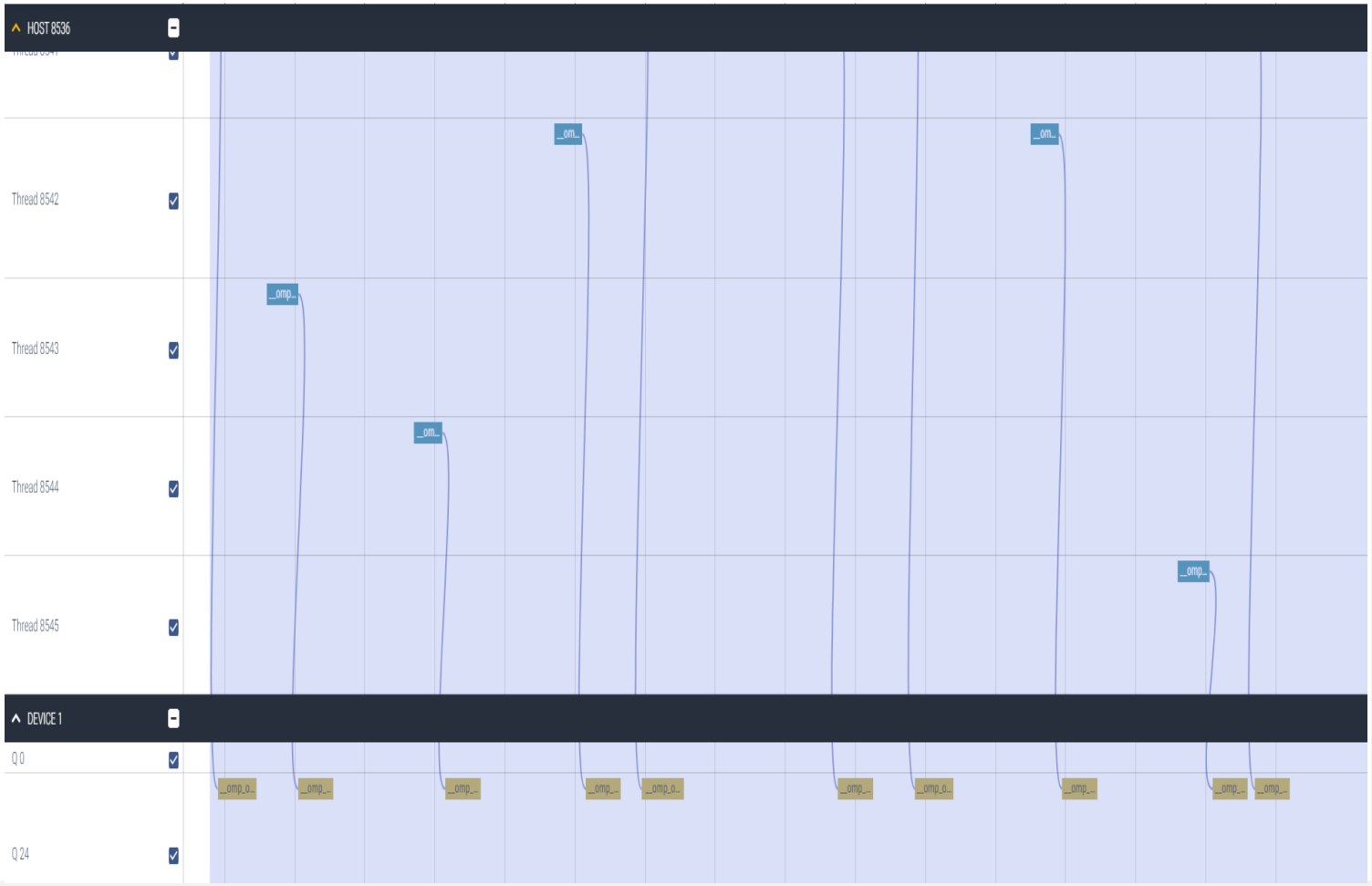
Small Kernels Tend to Have Low Occupancy

- Each cell is a work item which executes the kernel body
- Each SIMD lane executes one work item
- If SIMD width is 16, each thread has 16 SIMD lanes
- Kernels with small number of work items has small number of threads
- Combining multiple small kernels to a large one helps to improve occupancy



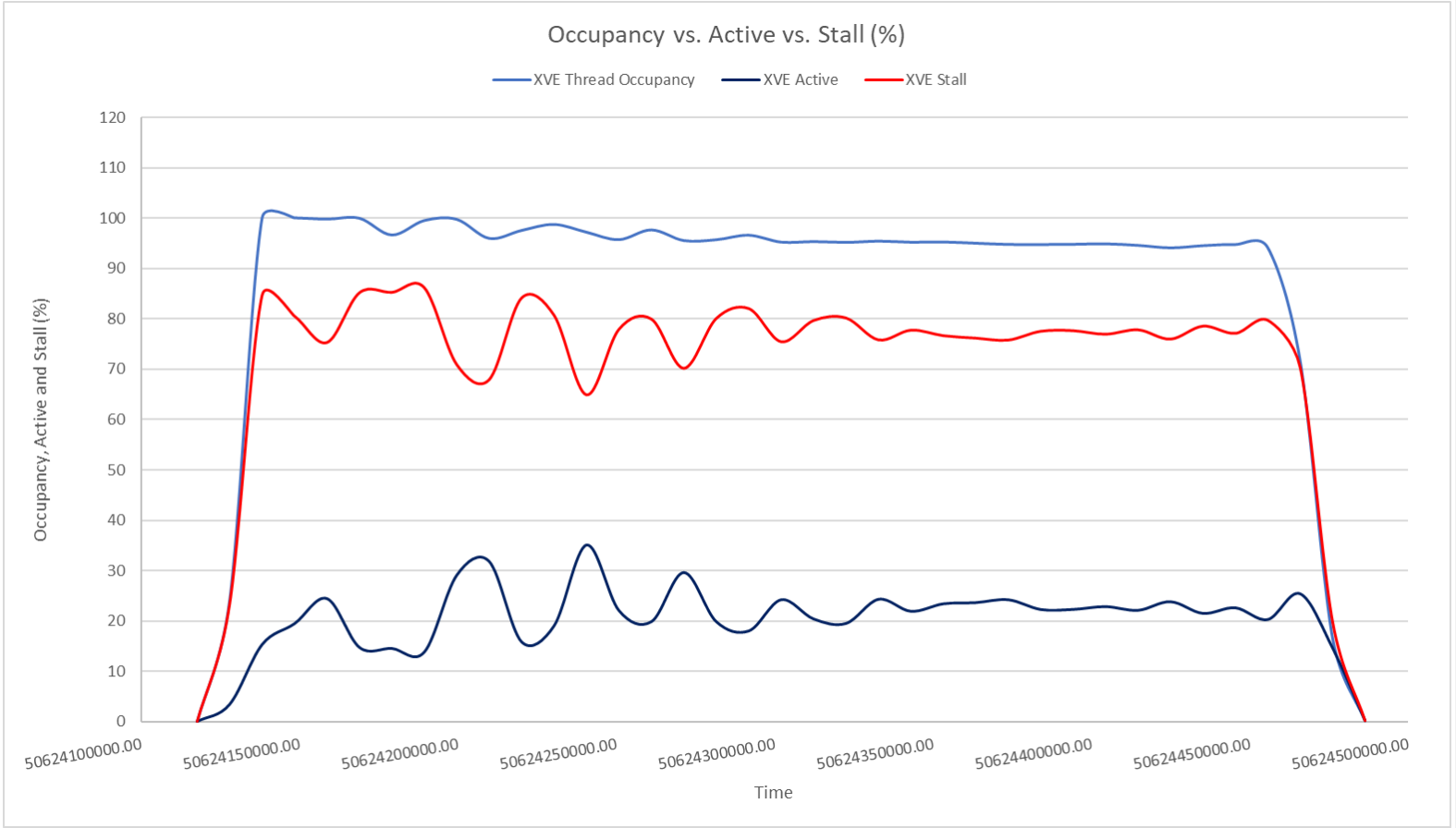
Kernel Execution: Sequential or Concurrent

- Sequential execution of small kernels cannot fully utilize compute resources
- Concurrent execution of independent kernels improves occupancy
- Shared local memory size used by workgroup is also a limiting factor of occupancy



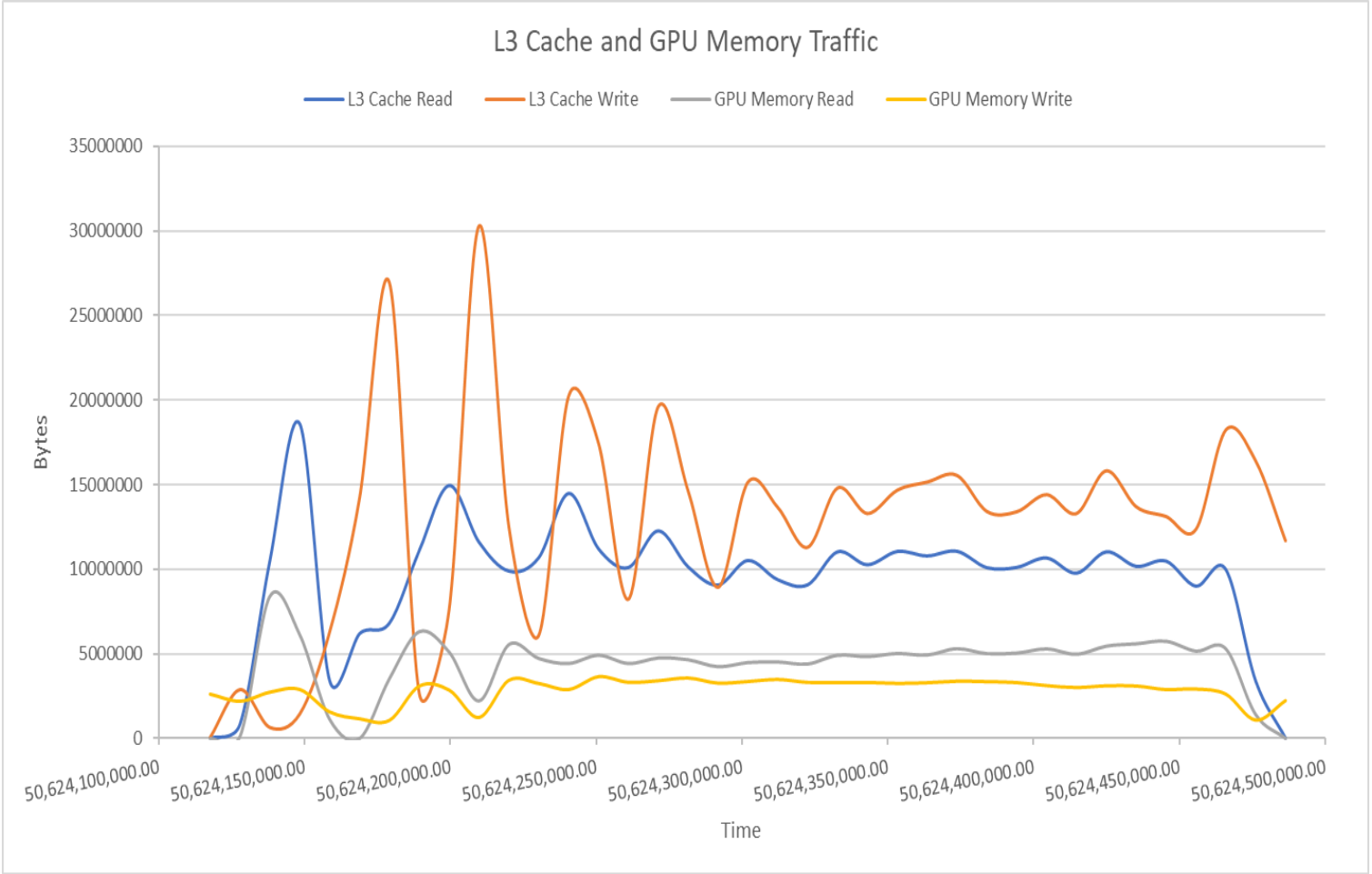
GPU: High Stall

- High occupancy does not necessarily translate to high performance
- A compute unit stalls if no instruction from any of the threads running on the same compute unit can be issued in a cycle
- Different threads on the same compute unit can be stalled for different reasons



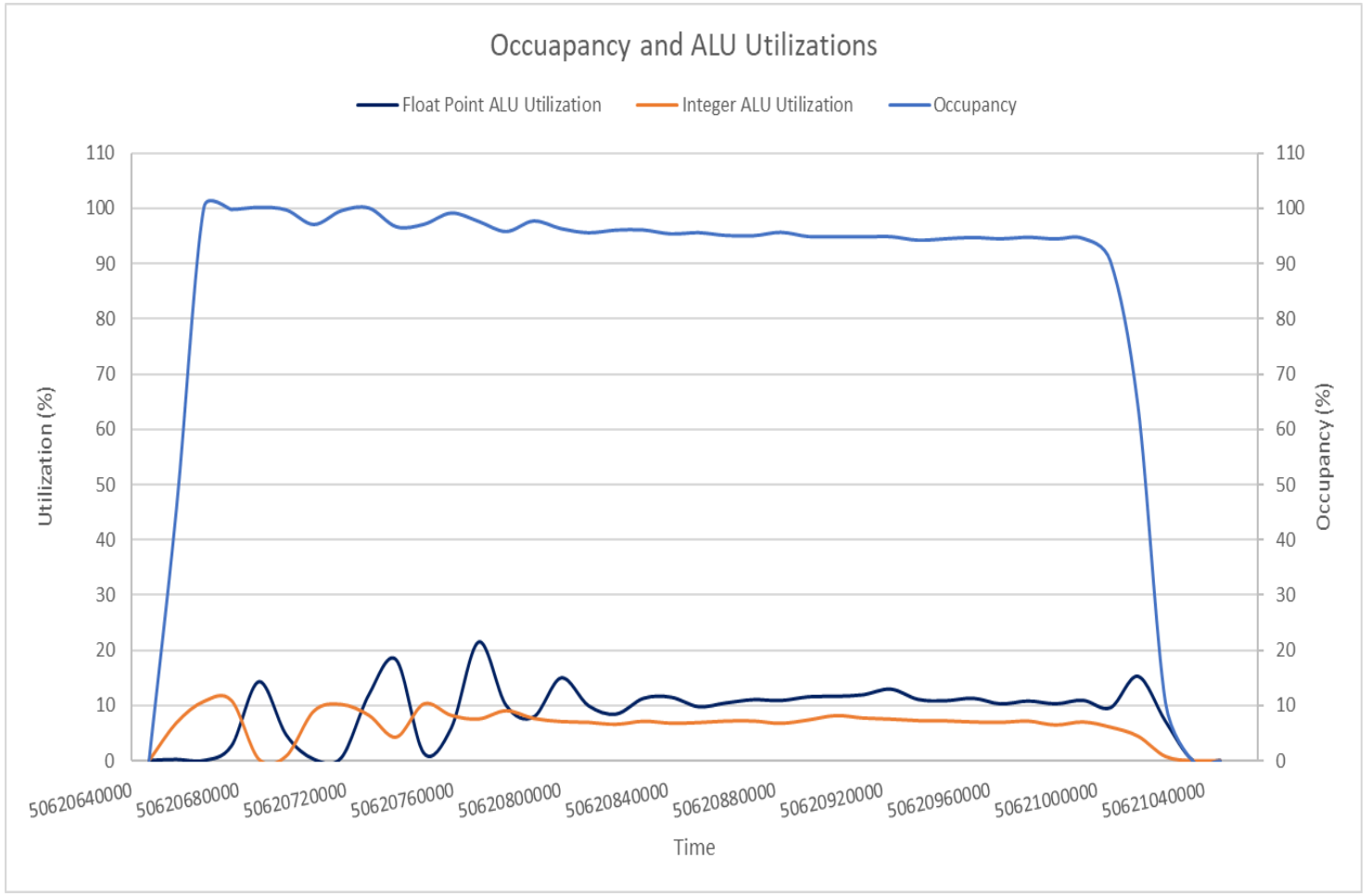
Memory Access Is Expensive

- Long memory access latency is a common stall reason
- Changing data structures and/or layouts for better locality helps to eliminate memory latency stalls
- Avoiding strided memory access also helps to reduce memory stalls



GPU: Low ALU Utilizations

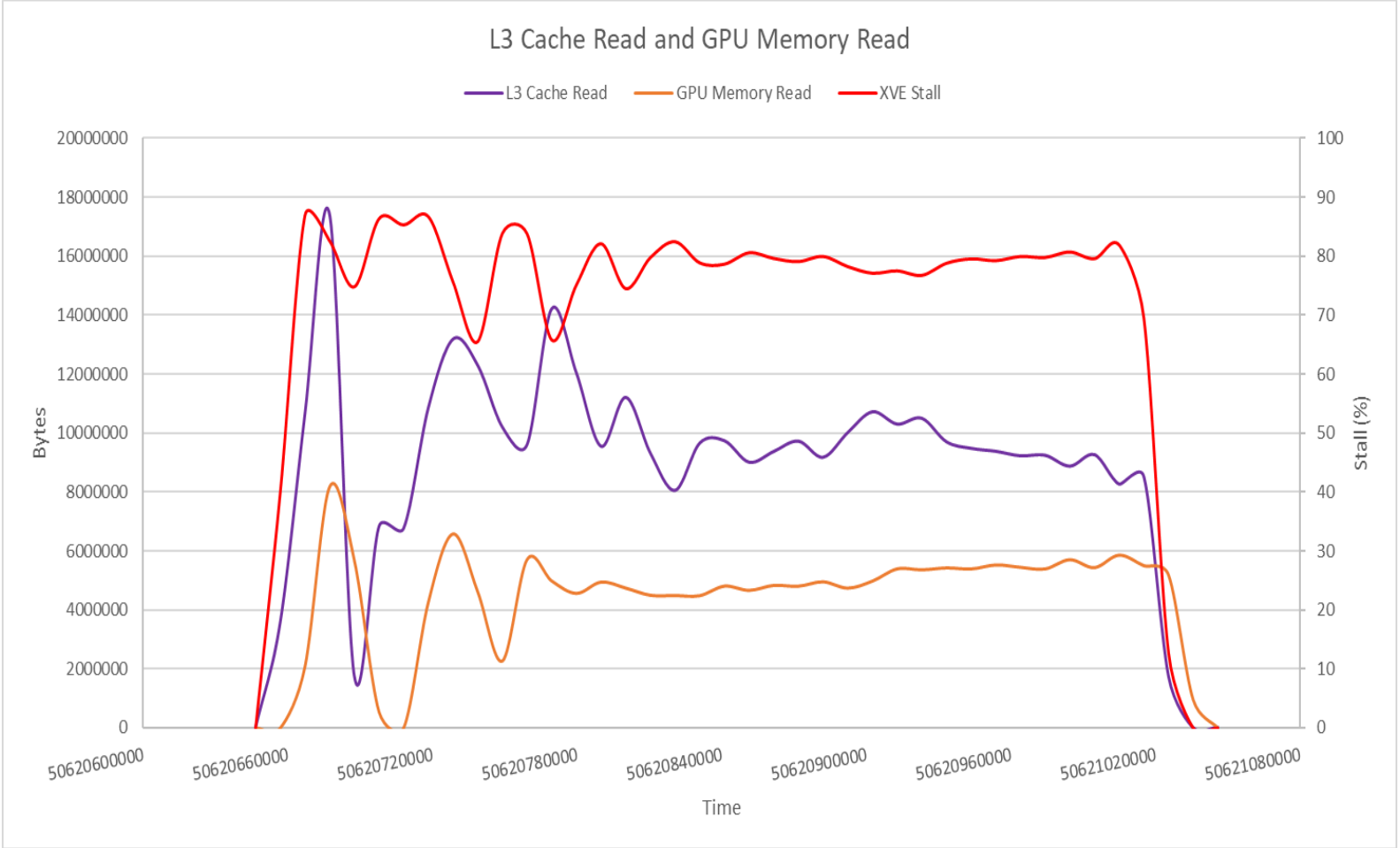
- A compute unit of modern GPUs have multiple ALUs
- ALU utilizations directly affect performance of compute intensive workloads
- Low ALU utilization often indicates optimization opportunities for compute intensive workloads



Is ALU Utilization Bottlenecked by Memory?

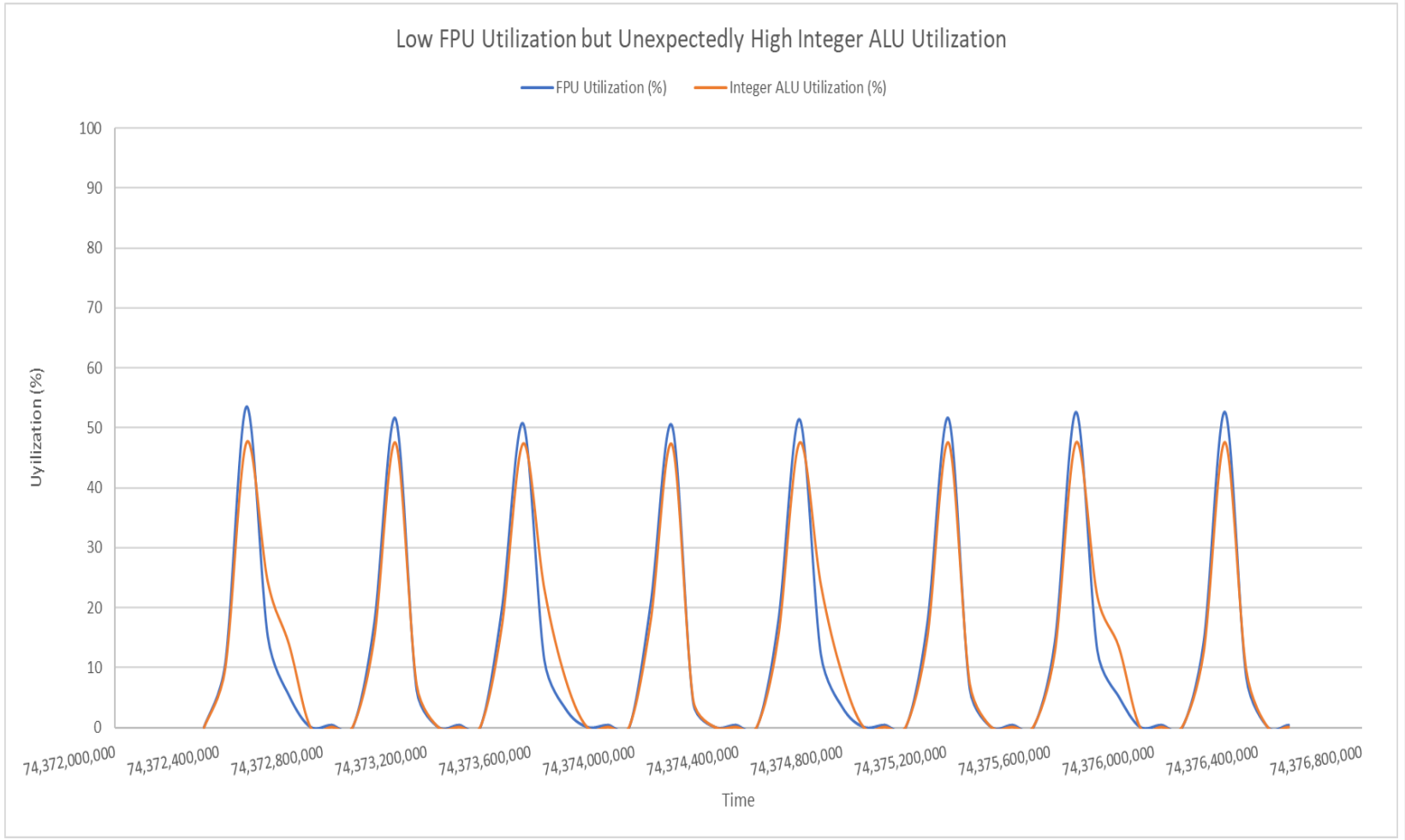
- One common reason of low ALU utilization is that data are not fed fast enough to ALUs because of slow memory access

- Loading data early helps to hide memory latency and pump data faster to ALUs



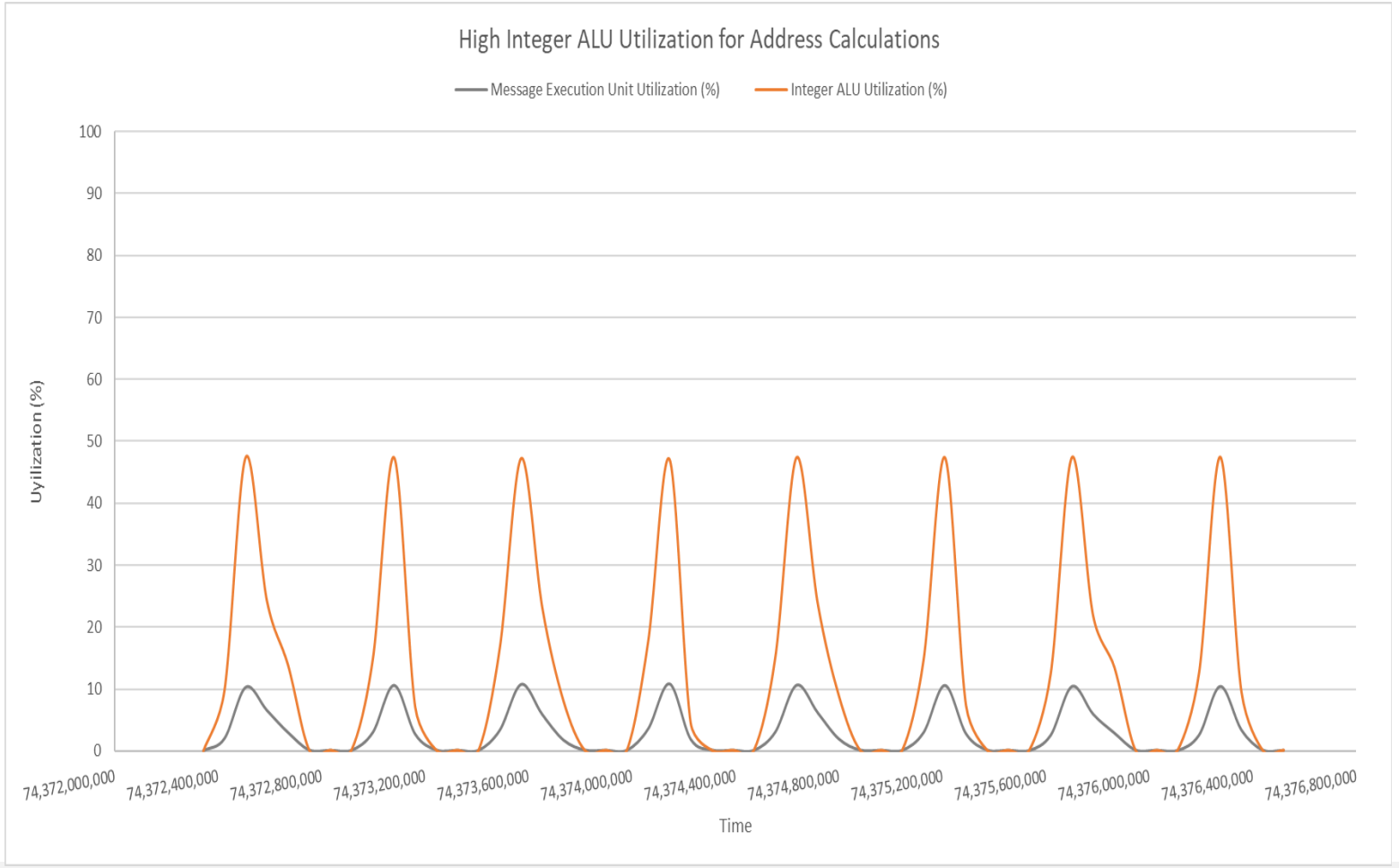
GPU: High ALU Utilization

- High ALU utilization does not always benefit performance
- For floating-point intensive workloads, high integer ALU utilization may hurt performance



High Integer ALU Utilization for Address Calculations

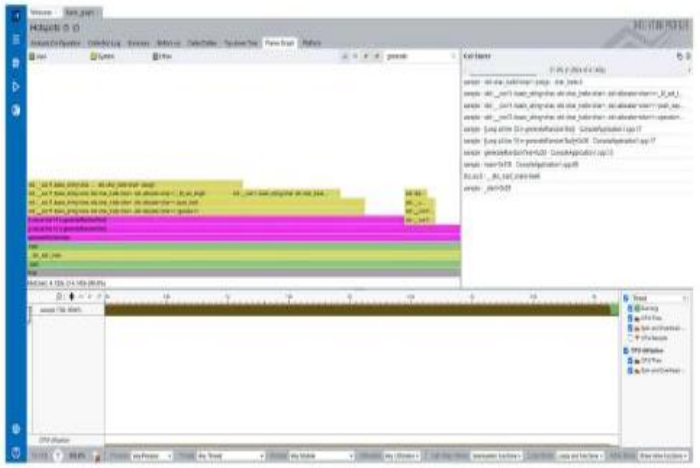
- Memory address calculations often contribute to integer ALU utilization
- Reducing address calculations often requires changes to data structures/layouts and/or even algorithms

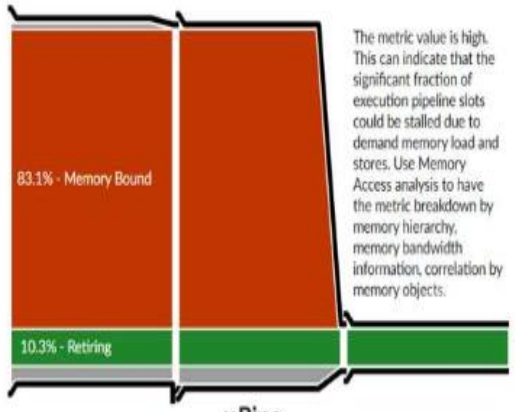


Tool Offerings

- Profiling Tools Interfaces for GPU (PTI for GPU)
  - https://github.com/intel/pti-gpu
- Intel® VTune™ Profiler
  - https://www.intel.com/content/www/us/en/developer/tools/oneapi/vtune-profiler.html

Intel® VTune Profiler





Algorithm Optimization

- Locate hot spots—the most time-consuming parts of your code.
- Visualize hot code paths and time spent in each function and with its callees with Flame Graph.

Analyze Hot Code Paths

Analyze Hot Spots

Microarchitecture and Memory Bottlenecks

- Identify the most significant hardware issues that affect the performance of your application with microarchitecture exploration analysis.
- Pinpoint memory-access-related issues such as cache misses and high-bandwidth problems.

Code-Tuning Methods for Intel® CPU Microarchitecture

Profile a Memory-Bound Application

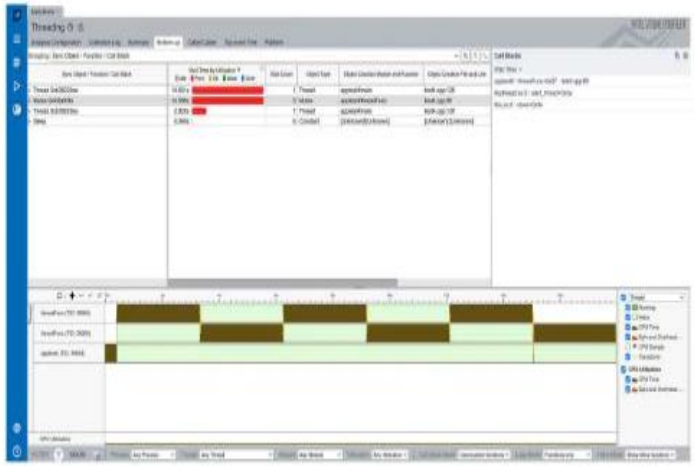
Accelerators and XPUs

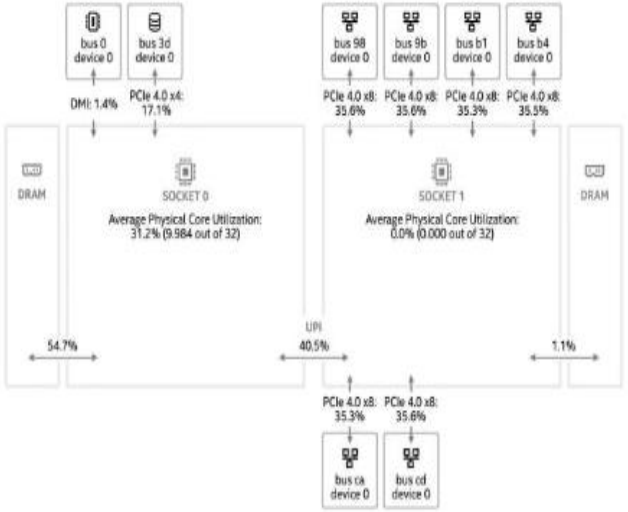
- Optimize GPU offload schema and data transfers for SYCL, OpenCL code, Microsoft DirectX*, or OpenMP* offload code. Identify the most time-consuming GPU kernels for further optimization.
- Analyze GPU-bound code for performance bottlenecks caused by microarchitectural constraints or inefficient kernel algorithms.
- Explore CPU and FPGA interactions, and FPGA use.

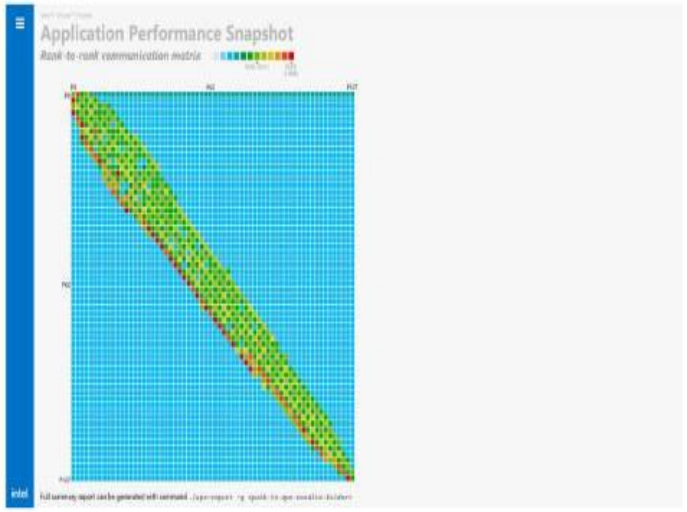
Optimize Software for Intel® GPUs

Profile OpenMP Offload Code on a GPU

Intel® VTune Profiler







Parallelism

- Examine how efficiently the code is threaded. Identify threading issues that impact performance.
- Evaluate compute-intensive or throughput HPC applications for efficient CPU use, vectorization, and memory use.

Method for OpenMP Code Analysis

Schedule Overhead in Intel® oneAPI Threading Building Blocks Applications

Platform and I/O

- Locate performance bottlenecks in I/O-intensive applications. Explore how effectively the hardware processes I/O traffic generated by external PCIe* devices or integrated accelerators.
- See a holistic view of system behavior for long-running workloads with Platform Profiler.
- Get a fine-grained overview for short-running workloads with System Overview.

Effective Use of Intel® Data Direct I/O Technology

Multi-Node

- Characterize performance aspects of large-scale message passing interface (MPI) and OpenMP workloads.
- Identify scalability issues and get recommendations for in-depth analysis.

Profile MPI Applications

Summary

- The world has concluded accelerators are here
  - Intel has led this trend with SVMS architectures and our XPU strategy
- To succeed, we need to deliver the right software environment
  - Simplify accelerator software development without sacrificing performance
- Our solution is **oneAPI**
  - A cross-industry, open, standards-based unified programming model delivering a common developer experience across CPU and XPU architectures
- oneAPI is also a workload, which has observable behaviors
  - Intel provides the tools necessary to collect the needed data

- Call to action: Interested? Join the community at oneapi.io

Thank You