Profiling Heterogeneous Computing Performance with VTune Profiler

Vladimir Tsymbal, vladimir.tsymbal@intel.com (presenter)

Alexandr Kurylev, alexandr.kurylev@intel.com

intel

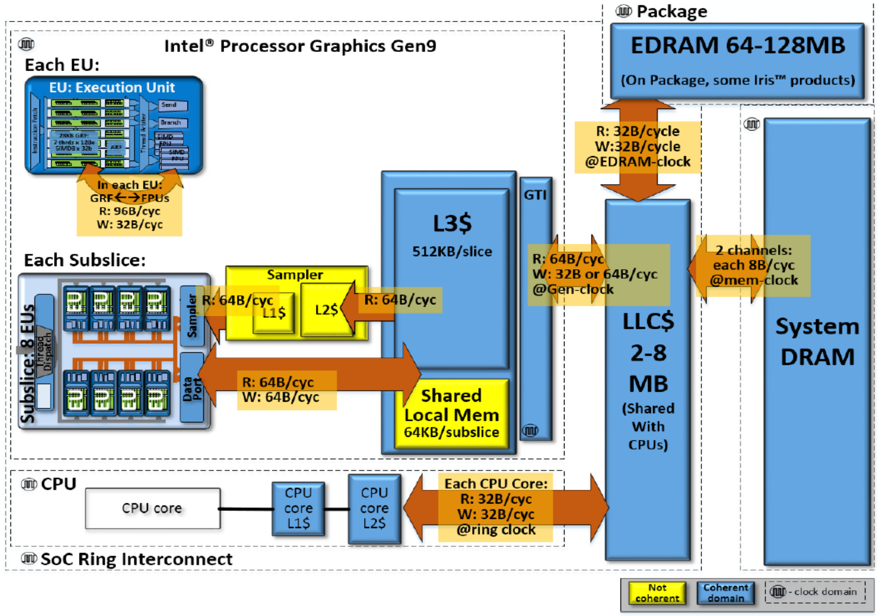
IWOCL & SYCLcon

9TH INT'L. WORKSHOP ON OPENCL & SYCL

Contents

- Intel GPU and Microarchitecture
- GPU Development Environment and Runtimes
- CPU or GPU bound? VTune Offload Analysis
- GPU Task Efficiency
- Computing Tasks and Data Transfer
- Applications to offload to GPU. SYCL/DPC++, OpenCL, OpenMP
- In-kernel Analysis
- Memory Stalls in GPU Microarchitecture
- GPU Instructions Count
- Basic Blocks Latency and Memory Latency
- Platform Analysis

Intel GPU and Microarchitecture



Intel® Gen9 HD Graphics

- Embedded to Coffee Lake SoC and newer
- Up to 48EUs x 7thr, up to 883 GFLOPS (SP)
- 2 SIMD-4 FPUs of 32-bit FP or INT data

Intel® Iris® Xe MAX discrete GPU

- 6 DSS x 16EUs (96 EUs x 7thr).
- VRAM 68 GB/s
- PCIe3x16 card, 2456 GFLOPS (SP)

GPU Development Environment and Runtimes

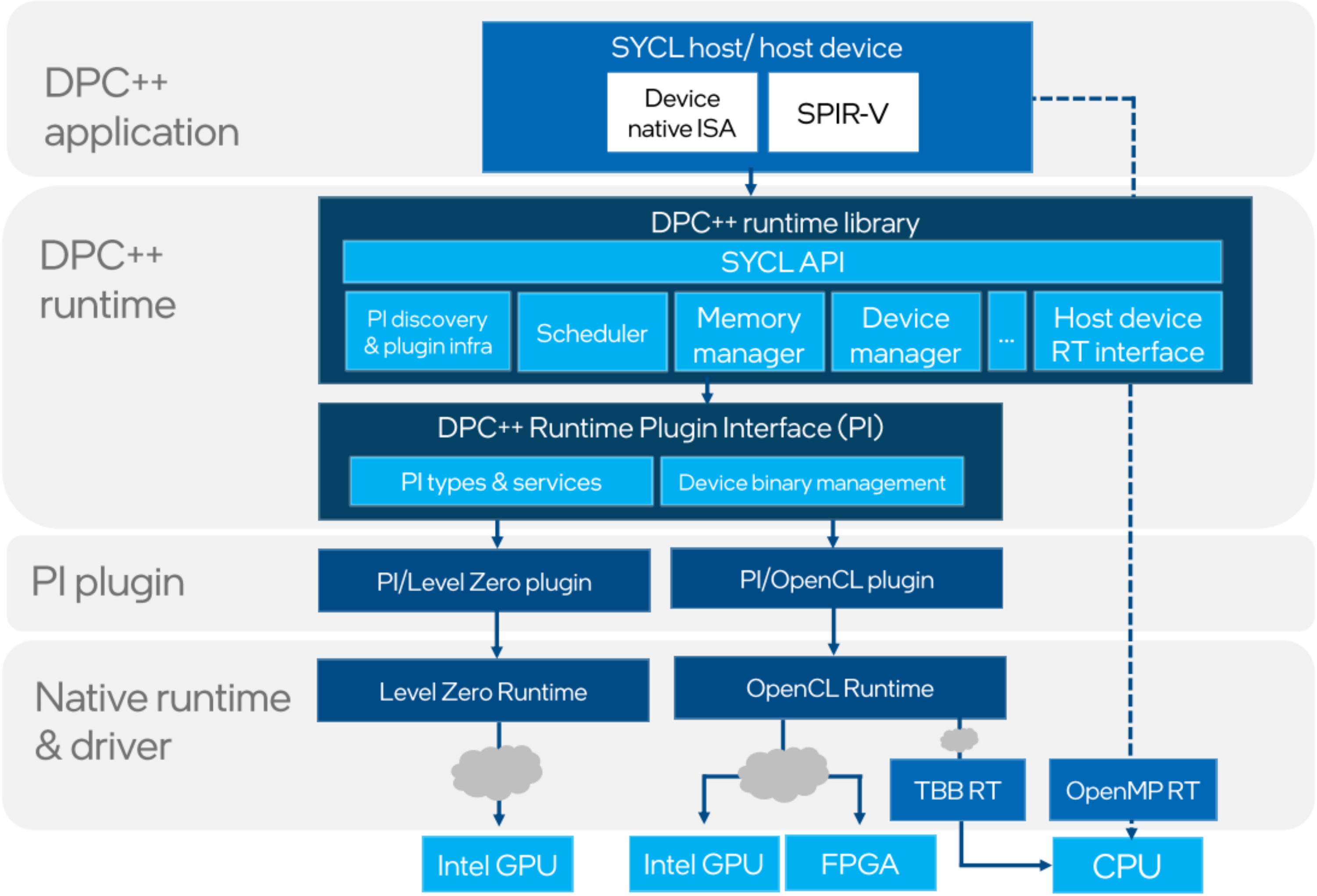
Several high-level languages for Media and GPGPU programming

- OpenCL™ Technology via Intel® Media SDK
- SYCL/Data Parallel C++ direct programming
- OpenMP offload to GPU
- Performance Libraries

Set of Intel Compilers based on LLVM technology

Intel Compiler	Target	OpenMP Support	OpenMP Offload Support	Included in oneAPI Toolkit
Intel® oneAPI DPC++/C++ Compiler ***dpcpp***	CPU, GPU, FPGA*	Yes	Yes	Base
Intel® oneAPI DPC++/C++ Compiler ***icx***	CPU GPU*	Yes	Yes	Base
Intel® Fortran Compiler ***ifx***	CPU, GPU*	Yes	Yes	HPC

Runtime Architecture



- Controlled via SYCL_BE env var:

PI_OPENCL

PI_LEVEL0

PI_CUDA for open-source compiler

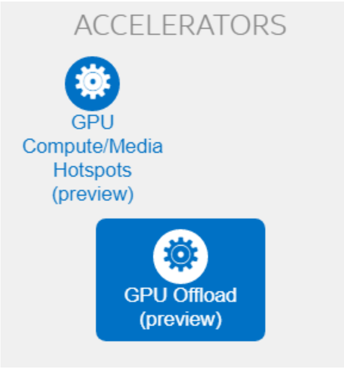
- Performance data comes from most of the levels

More info: tinyurl.com/dpcpp-pi

CPU or GPU bound? VTune Offload Analysis

All execution resources in focus

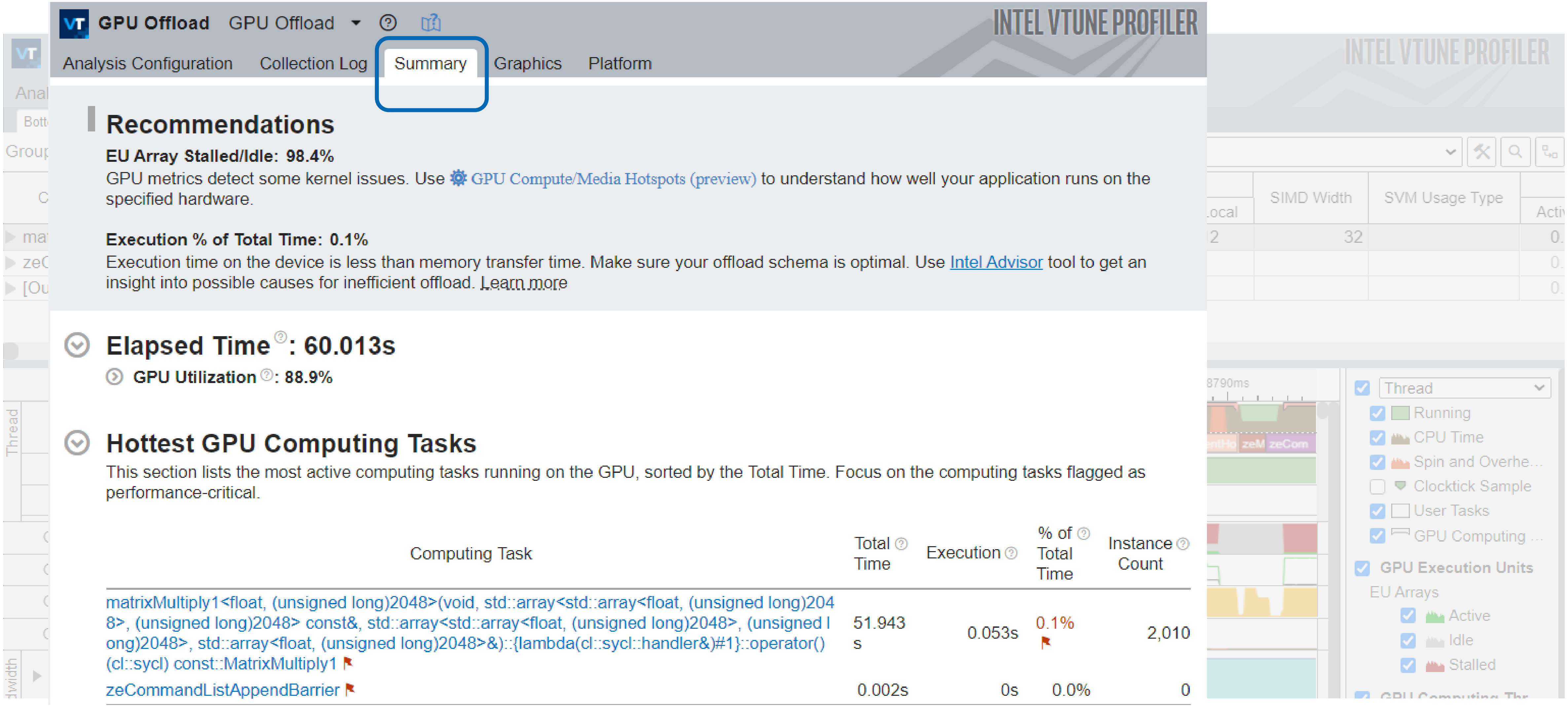
- Explore code execution on various CPU and GPU cores
- Correlate CPU and GPU activity
- Identify whether your application is GPU or CPU bound



Find kernels for further analysis

- Task level analysis
- Kernel efficiency
- Data transfer rates

GPU Task Efficiency



GPU Task Efficiency

GPU Offload GPU Offload

INTEL VTUNE PROFILER

Analysis Configuration | Collection Log | Summary | Graphics | Platform

Recommendations

EU Array Stalled/Idle: 98.4%

GPU metrics detect some kernel issues. Use GPU Compute/Media Hotspots (preview) to understand how well your application runs on the specified hardware.

Execution % of Total Time: 0.1%

Execution time on the device is less than memory transfer time. Make sure your offload schema is optimal. Use Intel Advisor tool to get an insight into possible causes for inefficient offload. Learn more

Elapsed Time: 60.013s

GPU Utilization: 88.9%

Hottest GPU Computing Tasks

This section lists the most active computing tasks running on the GPU, sorted by the Total Time. Focus on the computing tasks flagged as performance-critical.

Computing Task	Total Time	Execution	% of Total Time	Instance Count
matrixMultiply1<float, (unsigned long)2048>(void, std::array<std::array<float, (unsigned long)2048>, (unsigned long)2048> const&, std::array<std::array<float, (unsigned long)2048>, (unsigned long)2048>, std::array<float, (unsigned long)2048>&)::{lambda(cl::sycl::handler&)#1}::operator()(cl::sycl) const::MatrixMultiply1	51.943s	0.053s	0.1%	2,010
zeCommandListAppendBarrier	0.002s	0s	0.0%	0

GPU Task Efficiency

GPU Offload GPU Offload INTEL VTUNE PROFILER

Analysis Configuration Collection Log Summary Graphics Platform

Recommendations

EU Array Stalled/Idle: 98.4%
GPU metrics detect some kernel issues. Use GPU Compute/Media Hotspots (preview) to understand how well your application runs on the specified hardware.

Execution % of Total Time: 0.1%
Execution time on the device is less than memory transfer time. Make sure your offload schema is optimal. Use Intel Advisor tool to get an insight into possible causes for inefficient offload. Learn more

Elapsed Time: 60.013s

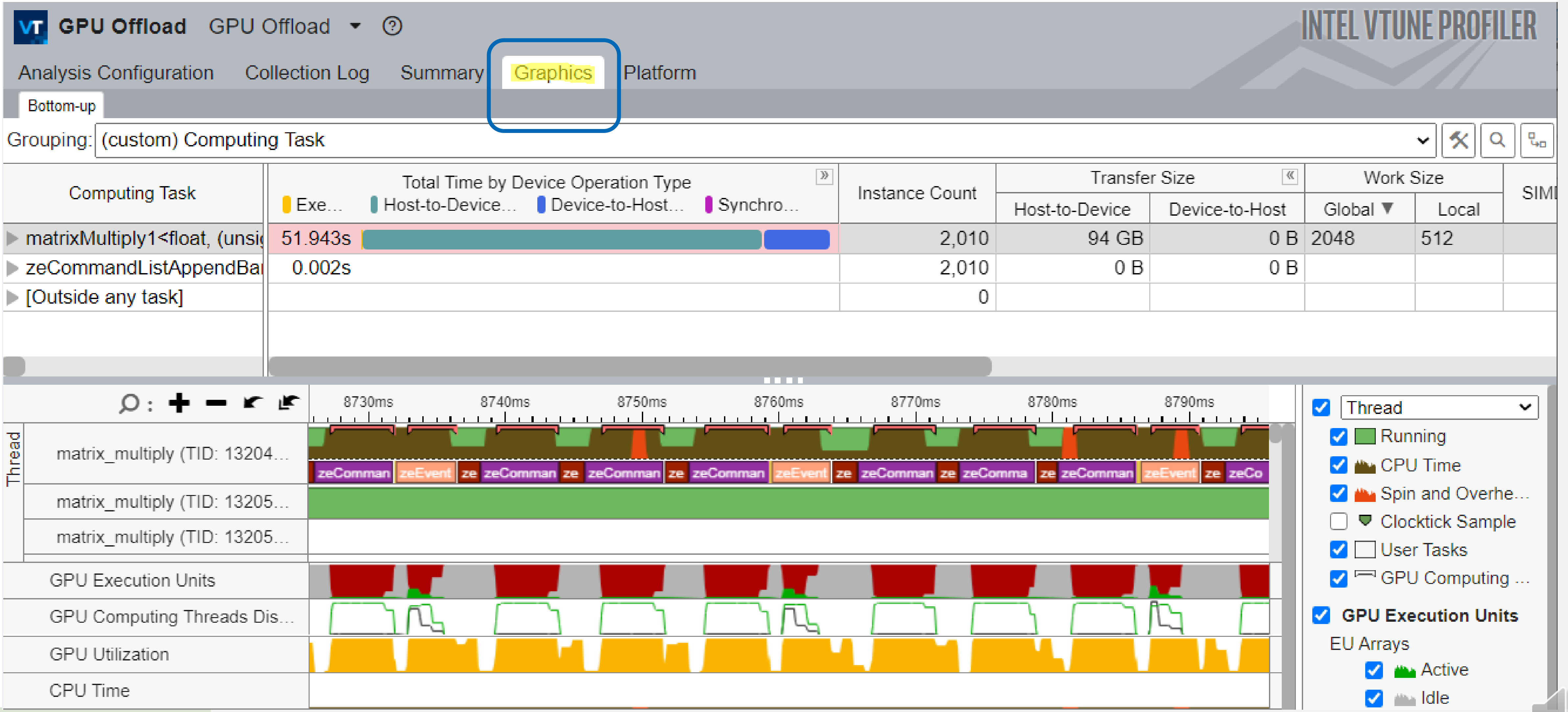
GPU Utilization: 88.9%

Hottest GPU Computing Tasks

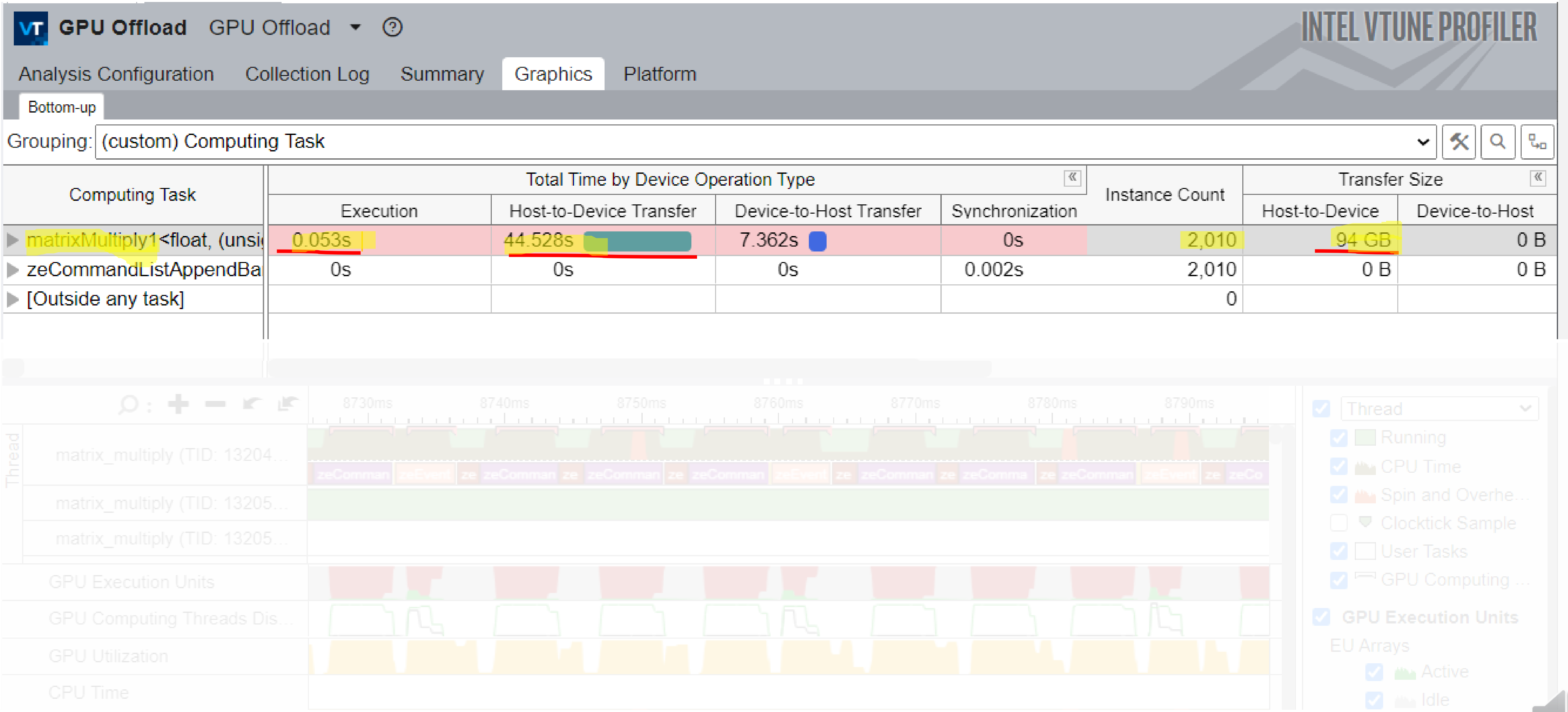
This section lists the most active computing tasks running on the GPU, sorted by the Total Time. Focus on the computing tasks flagged as performance-critical.

Computing Task	Total Time	Execution	% of Total Time	Instance Count
matrixMultiply1<float, (unsigned long)2048>(void, std::array<std::array<float, (unsigned long)2048>, (unsigned long)2048> const&, std::array<std::array<float, (unsigned long)2048>, (unsigned long)2048>, std::array<float, (unsigned long)2048>&)::{lambda(cl::sycl::handler&)#1}::operator()(cl::sycl) const::MatrixMultiply1	51.943s	0.053s	0.1%	2,010
zeCommandListAppendBarrier	0.002s	0s	0.0%	0

Computing Tasks and Data Transfer

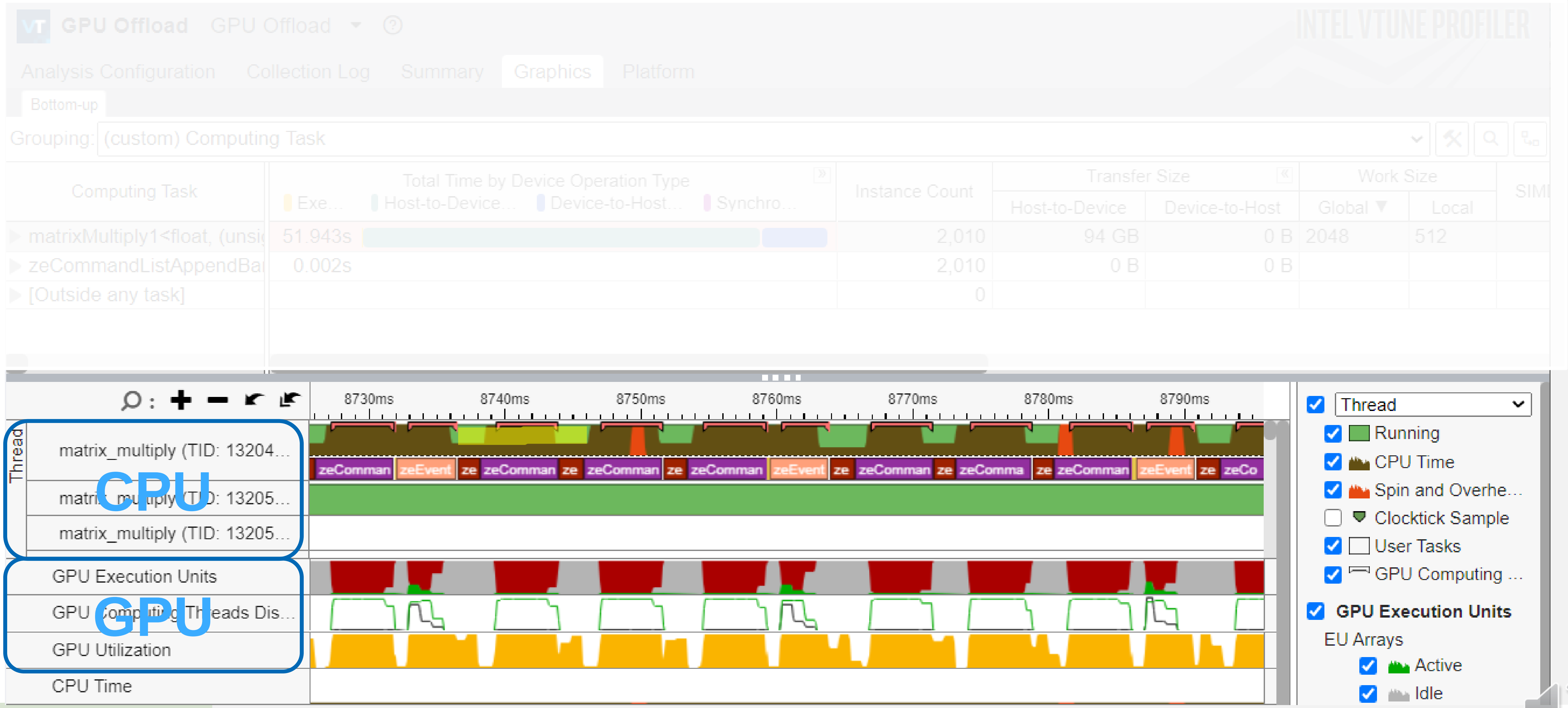


Computing Tasks and Data Transfer

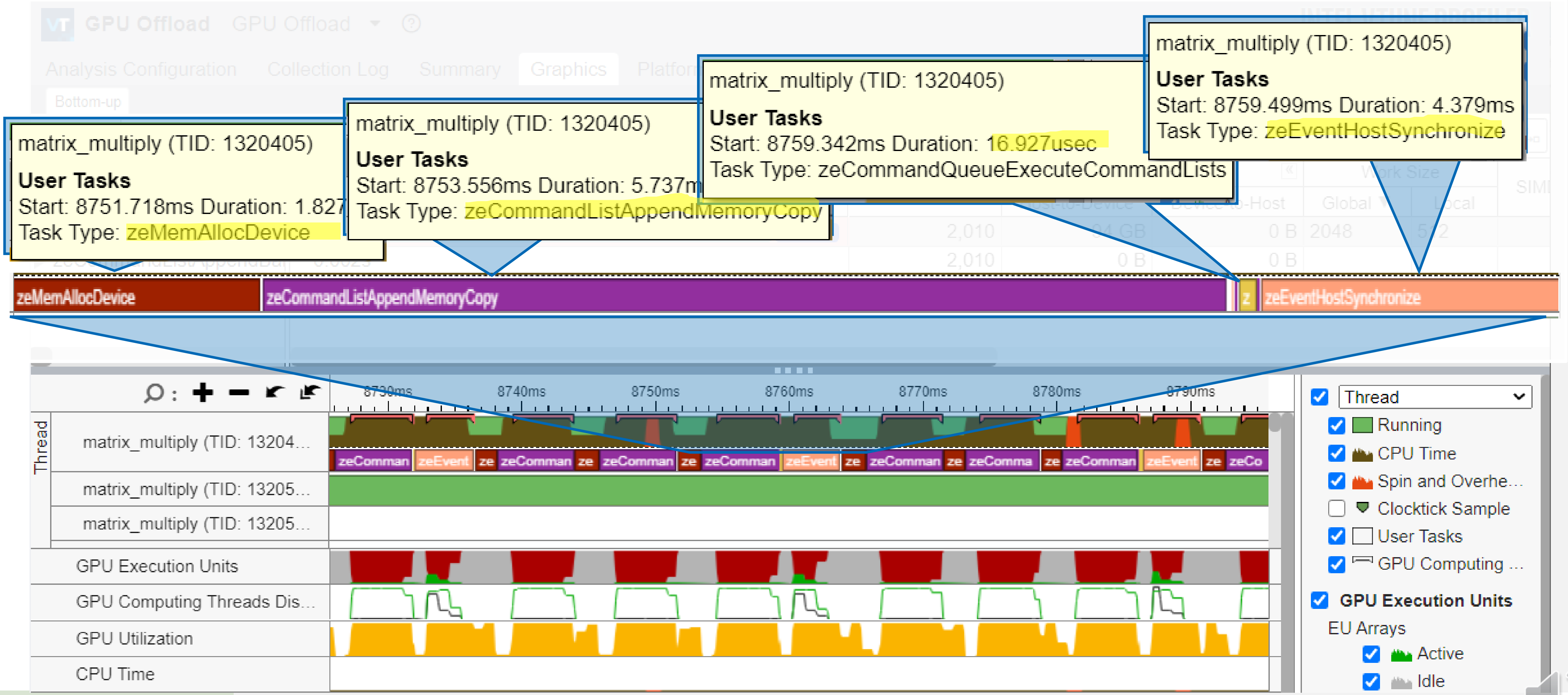


Computing Task	Total Time by Device Operation Type: Execution	Total Time by Device Operation Type: Host-to-Device Transfer	Total Time by Device Operation Type: Device-to-Host Transfer	Total Time by Device Operation Type: Synchronization	Instance Count	Transfer Size: Host-to-Device	Transfer Size: Device-to-Host
matrixMultiply1<float, (unsi	0.053s	44.528s	7.362s	0s	2,010	94 GB	0 B
zeCommandListAppendBa	0s	0s	0s	0.002s	2,010	0 B	0 B
[Outside any task]					0		

Computing Tasks and Data Transfer



Computing Tasks decomposition



Applications to offload to GPU. SYCL/DPC++

DPC++ “Hello World”: Vector Addition

```
int main() {
    float A[1024], B[1024], C[1024];
    {
        buffer<float, 1> bufA { A, range<1> {1024} };
        buffer<float, 1> bufB { B, range<1> {1024} };
        buffer<float, 1> bufC { C, range<1> {1024} };

        queue q;
        q.submit([&](handler& h) {
            auto A = bufA.get_access<dpc_r>(h);
            auto B = bufB.get_access<dpc_r>(h);
            auto C = bufC.get_access<dpc_w>(h);

            h.parallel_for(range<1> {1024}, [=](id<1> i) {
                C[i] = A[i] + B[i];
            });
        });
    }
    for (int i = 0; i < 1024; i++)
        std::cout << "C[" << i << "] = " << C[i] << std::endl;
}
```

Host code

Accelerator device code

Host code

Applications to offload to GPU. OpenCL

```
__kernel void vector_add(__global const float *x,
                         __global const float *y,
                         __global float *restrict z)
{
    // get index of the work item
    int index = get_global_id(0);
    // add the vector elements
    z[index] = x[index] + y[index];
}
```

Accelerator device code

Top Tasks

This section lists the most active tasks in your application.

Task Type	Task Time	Task Count	Average Task Time
clBuildProgram	0.237s	1	0.237s
clCreateBuffer	0.118s	3	0.039s
clCreateKernel	0.016s	1	0.016s
clCreateContext	0.000s	1	0.000s
clCreateCommandQueueWithProperties	0.000s	1	0.000s

Applications to offload to GPU. OpenCL

```
void run() {
  cl_int status;
  const double start_time = getCurrentTimestamp();
  // Launch the problem for each device.
  scoped_array<cl_event> kernel_event(num_devices);
  scoped_array<cl_event> finish_event(num_devices);
  for(unsigned i = 0; i < num_devices; ++i) {
  // for the host-to-device transfer.
    cl_event write_event[2];
    status = clEnqueueWriteBuffer(queue[i], input_a_buf[i], CL_FALSE,
        0, n_per_device[i] * sizeof(float), input_a[i], 0, NULL, &write_event[0]);
    checkError(status, "Failed to transfer input A");
    status = clEnqueueWriteBuffer(queue[i], input_b_buf[i], CL_FALSE,
        0, n_per_device[i] * sizeof(float), input_b[i], 0, NULL, &write_event[1]);
    checkError(status, "Failed to transfer input B");
    // Set kernel arguments.
    unsigned argi = 0;
    status = clSetKernelArg(kernel[i], argi++, sizeof(cl_mem), &input_a_buf[i]);
    checkError(status, "Failed to set argument %d", argi - 1);
    status = clSetKernelArg(kernel[i], argi++, sizeof(cl_mem), &input_b_buf[i]);
    checkError(status, "Failed to set argument %d", argi - 1);
    status = clSetKernelArg(kernel[i], argi++, sizeof(cl_mem), &output_buf[i]);
    checkError(status, "Failed to set argument %d", argi - 1);
    const size_t global_work_size = n_per_device[i];

    status = clEnqueueNDRangeKernel(queue[i], kernel[i], 1, NULL,
        &global_work_size, NULL, 2, write_event, &kernel_event[i]);
    checkError(status, "Failed to launch kernel");
    // Read the result. This the final operation.
    status = clEnqueueReadBuffer(queue[i], output_buf[i], CL_FALSE,
        0, n_per_device[i] * sizeof(float), output[i], 1, &kernel_event[i], &finish_event[i]);
    // Release local events.
    clReleaseEvent(write_event[0]);    clReleaseEvent(write_event[1]);
  // Wait for all devices to finish.
  clWaitForEvents(num_devices, finish_event);
  // Release all events.
  for(unsigned i = 0; i < num_devices; ++i) {
```

Partial Host code

Applications to offload to GPU. OpenMP offload

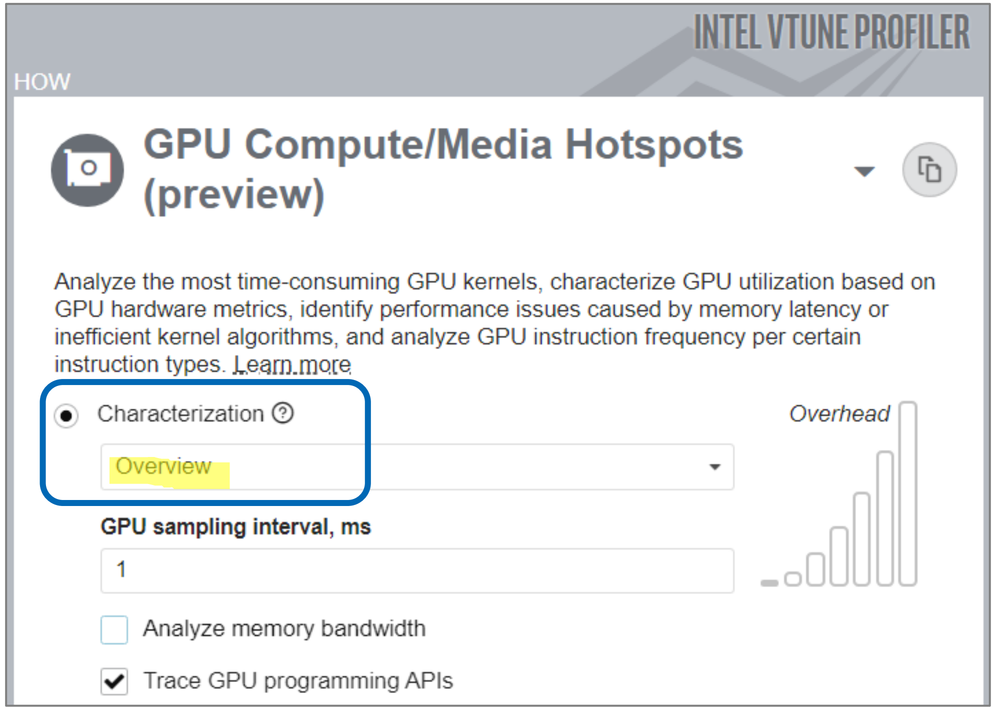
```
void __attribute__((noinline)) MatrixMulOpenMpGpuOffloading() {
  int i, j, k;

 // Each element of matrix a is 1.
  for (i = 0; i < M; i++)
    for (j = 0; j < N; j++) a[i][j] = 1.0f;

  // Each column of b is the sequence 1,2,...,N
  for (i = 0; i < N; i++)
    for (j = 0; j < P; j++) b[i][j] = i + 1.0f;

  // c is initialized to zero.
  for (i = 0; i < M; i++)
    for (j = 0; j < P; j++) c[i][j] = 0.0f;

// Parallelize on target device.
#pragma omp target teams distribute parallel for map(to : a, b) \
  map(tofrom : c) thread_limit(128)
  {
    for (i = 0; i < M; i++) {
      for (k = 0; k < N; k++) {
        // Each element of the product is just the sum 1+2+...+n
        for (j = 0; j < P; j++) {
          c[i][j] += a[i][k] * b[k][j];
        }
      }
    }
  }
```

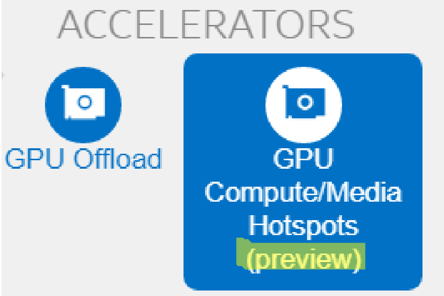
Host code

Accelerator device code

https://github.com/oneapi-src/oneAPI-samples/blob/master/DirectProgramming/DPC%2B%2B/DenseLinearAlgebra/matrix_mul/src/matrix_mul_omp.cpp

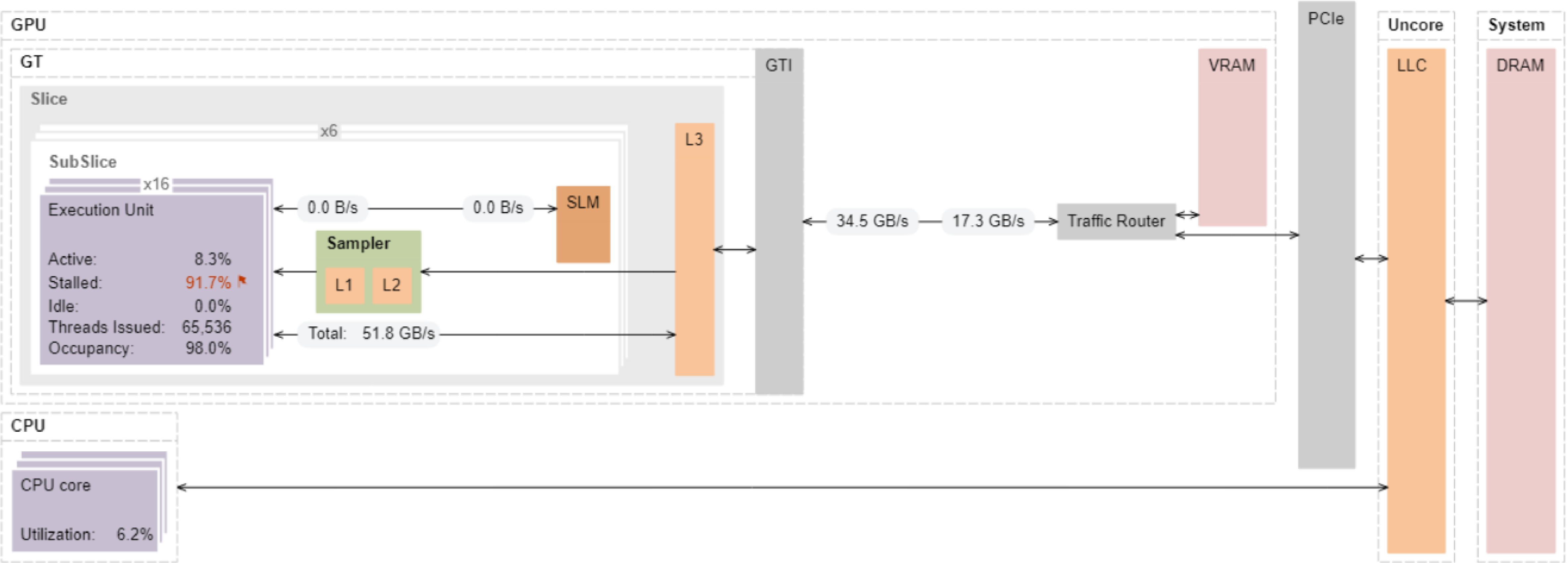
In-kernel Analysis





GPU Adapter / Computing Task	Work Size		Computing Task					Data Transferred	
	Global ▼	Local	Total Time	Average Time	Instance...	SIMD Width	SVM...	Size	Total, GB/s...
▼ DG1 [Iris Xe MAX Graphics]			2.167s	0.361s	6			64 MB	0.024
▶ matrixMultiply2<float, (unsigned long)2048>(vo	2048 x 2048	512 x 1	1.990s	1.990s	1	32		0 B	0.000
▶ zeCommandListAppendMemoryCopy			0.017s	0.006s	3			48 MB	2.963
▶ zeCommandListAppendMemoryCopyRegion			0.160s	0.160s	1			16 MB	0.105
▶ zeCommandListAppendBarrier			0.000s	0.000s	1			0 B	0.000

Memory Stalls in GPU Microarchitecture



Grouping: GPU Adapter / Computing Task

GPU Adapter / Computing Task	EU Array			EU Threads Occupancy	Computing Threads Started	L3 Bandwidth, GB/sec	Shared Local Mem...		GPU Memory Bandw...		GPU Barriers	GPU Atomics
	Active	Stalled	Idle				Read	Write	Read	Write		
DG1 [Iris Xe MAX Graphics]	6.9%	83.8%	9.3%	88.5%	376,832	6.2%	0.0%	0.000	24.699	12.397	0.000	0.000
matrixMultiply2<float, (unsigned long)2048>(void, std:	8.3%	91.7%	0.0%	98.0%	65,536	8.7%	0.0%	0.000	34.539	17.279	0.000	0.000
zeCommandListAppendMemoryCopy	0.3%	99.2%	0.5%	96.5%	47,759	1.1%	0.0%	0.000	3.576	2.827	0.000	0.000
zeCommandListAppendMemoryCopyRegion	0.2%	99.4%	0.4%	92.8%	262,144	0.1%	0.0%	0.000	0.425	0.422	0.000	0.000

GPU Instructions Count

Computing Task / Function / Call Stack	GPU Instructions Executed by Instruction Type					SIMD Utilization
	Control Flow	Send	Int16 & HP Float	Int32 & SP Float	Other	
▼ matrixMultiply2<float, (unsigned long)20	33,554,432	1,611,005,952	33,423,360	7,919,239,168	1,159,462,912	99.2%
▶ matrixMultiply2<float, (unsigned long)2	0	131,072	0	131,072	786,432	100.0%
▶ __spirv_GlobalInvocationId_y	0	0	0	2,359,296	131,072	100.0%
▶ cl::sycl::accessor<float, (int)2, (cl::sycl	0	0	0	7,471,104	393,216	100.0%
▶ cl::sycl::accessor<float, (int)2, (cl::sycl	0	0	0	524,288	0	100.0%
▶ matrixMultiply2<float, (unsigned long)2	33,554,432	1,610,874,880	33,423,360	1,281,753,088	805,830,656	97.2%
▶ cl::sycl::accessor<float, (int)2, (cl::sycl	0	0	0	3,623,878,656	218,103,808	100.0%
▶ cl::sycl::accessor<float, (int)2, (cl::sycl	0	0	0	1,426,063,360	50,331,648	100.0%
▶ cl::sycl::accessor<float, (int)2, (cl::sycl	0	0	0	67,108,864	16,777,216	100.0%
▶ cl::sycl::accessor<float, (int)2, (cl::sycl	0	0	0	1,509,949,440	67,108,864	100.0%

Instructions decomposition for a Computing Task and underlaying functions

GPU Instructions Count

Analysis Configuration | Collection Log | Summary | Graphics | matrix_multiply.cpp

Source | Assembly

Source Line ▲	Source	GPU Instructions Executed by Instruction Type (Control Flow, Send, Int16 & HP Float, Int32 & SP Float, Other)
146		
147	`hdlr.parallel_for<class MatrixMultiply2>(matrixRange, [=]`	39,845,888
148	`{`	
149	`size_t i = id[0], j = id[1];`	
150		
151	`rc[i][j] = T{};`	917,504
152	`for (size_t k = 0; k < w; k++)`	302,120,960
153	`{`	
154	`rc[i][j] += ra[i][k] * rb[k][j];`	3,422,552,064
155	`}`	
156	`});`	
157	`});`	

Instructions decomposition for a source line

Basic Blocks Latency

Computing Task / Function / Call Stack	Work Size		Computing Task					Data Transfe...	Estimated GPU Cycles
	Global ▼	Local	Total Time	Average Time	Instance ...	SIMD Width	SVM...	Size	
▼ matrixMultiply2<float, (unsigned long)2048>(voic	2048 x 2048	512 x 1	199.224ms	199.224ms	1	32		0 B	100.0%
▶ cl::sycl::accessor<float, (int)2, (cl::sycl::access									0.1%
▶ cl::sycl::accessor<float, (int)2, (cl::sycl::access									0.0%
▶ matrixMultiply2<float, (unsigned long)2048>(s									44.6%
▶ cl::sycl::accessor<float, (int)2, (cl::sycl::access									14.2%
▶ cl::sycl::accessor<float, (int)2, (cl::sycl::access									20.1%
▶ cl::sycl::accessor<float, (int)2, (cl::sycl::access									0.7%

Define function calls that took most of GPU cycles

Basic Blocks Latency

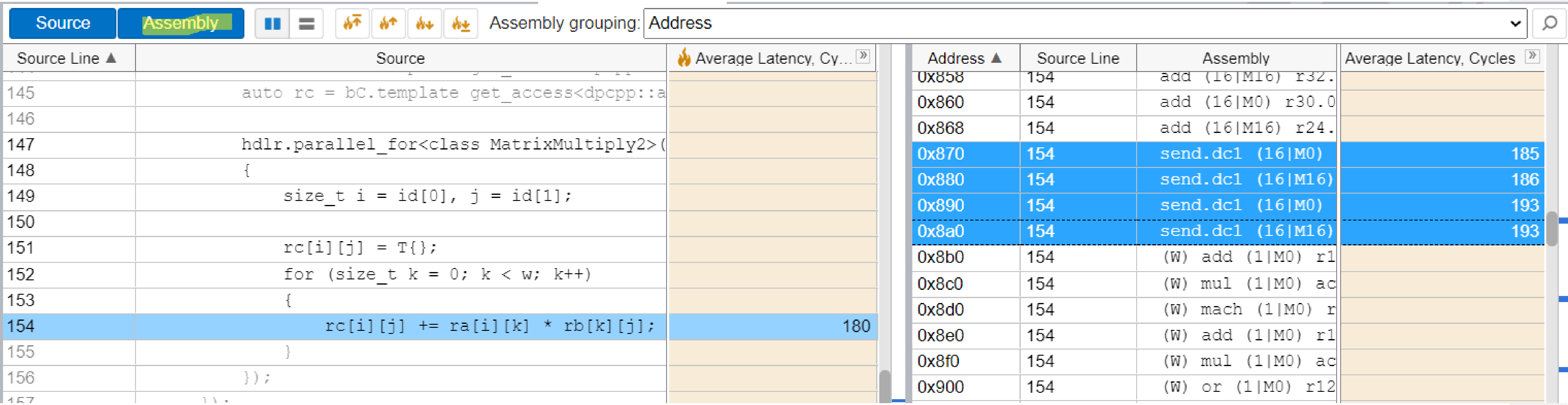
Computing Task / Function / Call Stack	Work Size: Global ▼	Work Size: Local	Computing Task: Total Time	Computing Task: Average Time	Computing Task: Instance ...	Computing Task: SIMD Width	Computing Task: SVM...	Data Transfe...: Size	Estimated GPU Cycles
▼ matrixMultiply2<float, (unsigned long)2048>(voic	2048 x 2048	512 x 1	199.224ms	199.224ms	1	32		0 B	100.0%
▶ cl::sycl::accessor<float, (int)2, (cl::sycl::access									0.1%
▶ cl::sycl::accessor<float, (int)2, (cl::sycl::access									0.0%
▶ matrixMultiply2<float, (unsigned long)2048>(s									44.6%
▶ cl::sycl::accessor<float, (int)2, (cl::sycl::access									14.2%
▶ cl::sycl::accessor<float, (int)2, (cl::sycl::access									20.1%
▶ cl::sycl::accessor<float, (int)2, (cl::sycl::access									0.7%

Analysis Configuration | Collection Log | Summary | Graphics | matrix_multiply.cpp ×

Source | Assembly

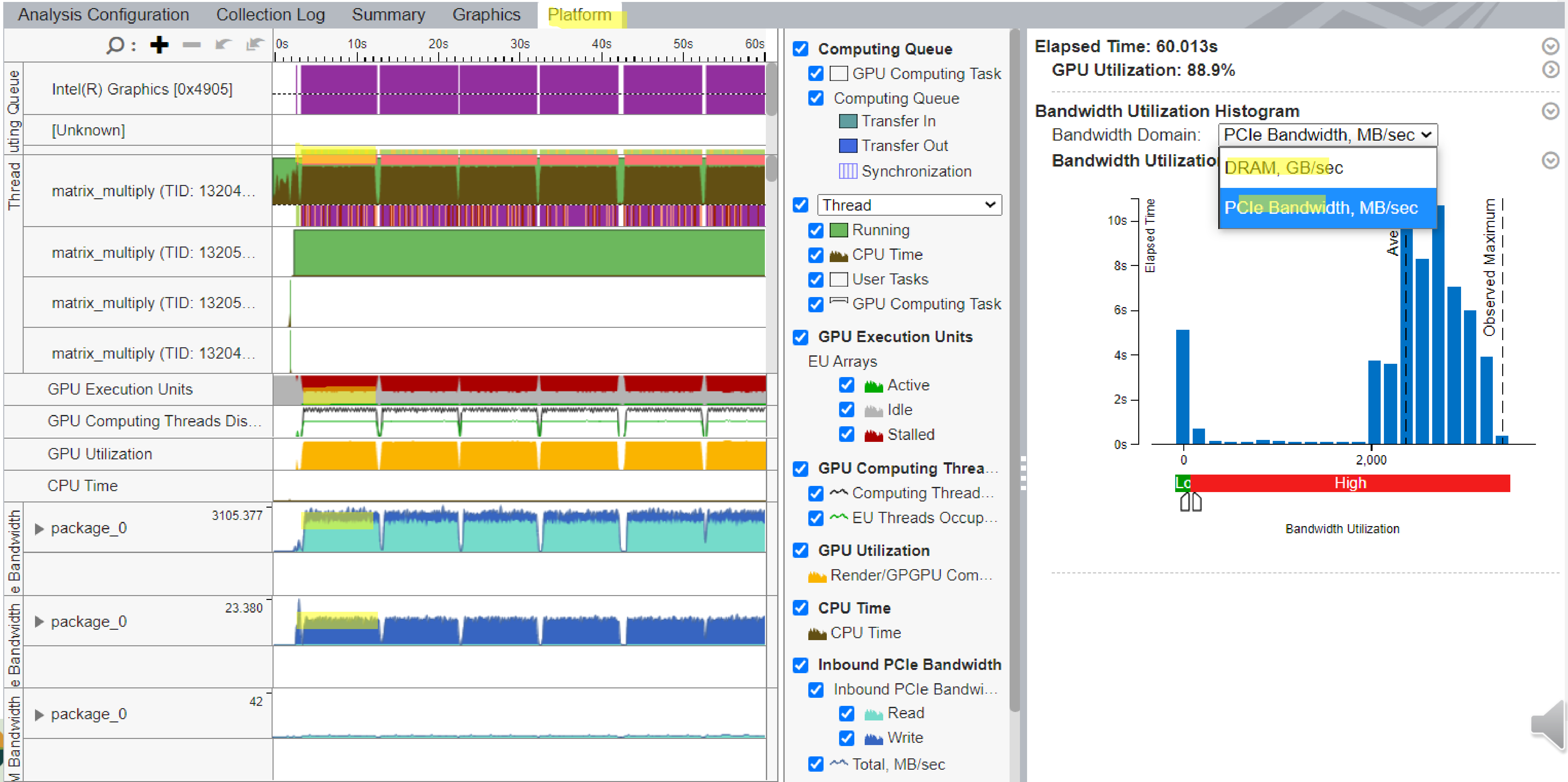
Source Line ▲	Source	Estimated GPU Cycles
145	`auto rc = bC.template get_access<dpcpp::access::mode::discard_write>(hdlr);`	
146		
147	`hdlr.parallel_for<class MatrixMultiply2>(matrixRange, [=](dpcpp::id<2> id)`	0.2%
148	`{`	
149	`size_t i = id[0], j = id[1];`	
150		
151	`rc[i][j] = T{};`	0.0%
152	`for (size_t k = 0; k < w; k++)`	2.6%
153	`{`	
154	`rc[i][j] += ra[i][k] * rb[k][j];`	41.9%
155	`}`	
156	`});`	

Memory Latency



Latencies per individual instructions

Platform Analysis



Quick References

Intel® VTune™ Profiler – Performance Profiler

- Product page – overview, features, FAQs…
- Training materials – Cookbooks, User Guide, Processor Tuning Guides
- Support Forum
- Online Service Center - Secure Priority Support
- What's New?

Additional Analysis Tools

- Intel® Advisor – Design and optimize for efficient vectorization, threading, memory usage, and accelerator offload. Roofline and flow graph analysis.
- Intel® Inspector – memory and thread checker/ debugger
- Intel® Trace Analyzer and Collector - MPI Analyzer and Profiler

Additional Development Products

- Intel® Software Development Products

Notices & Disclaimers

Performance varies by use, configuration, and other factors. Learn more at www.Intel.com/PerformanceIndex.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See configuration disclosure for details.

Your costs and results may vary.

Intel technologies may require enabled hardware, software or service activation.