using Intel® VTune and Intel® Advisor on GPUs:

Stephen Blair-Chappell

External Intel Certified oneAPI Instructor

7th June 2023

intel®

Introduction

About Me

Stephen Blair-Chappell is an independent software consultant and is an Intel certified oneAPI instructor. He was formerly the Technical Director at Bayncore where he led a team of consultants providing HPC and AI training on Intel Architecture. For 18 years he was a Technical Consulting Engineer at Intel helping their strategic customers in software optimisation and code modernisation. He is author of the book "Parallel Programming with Intel Parallel Studio XE".

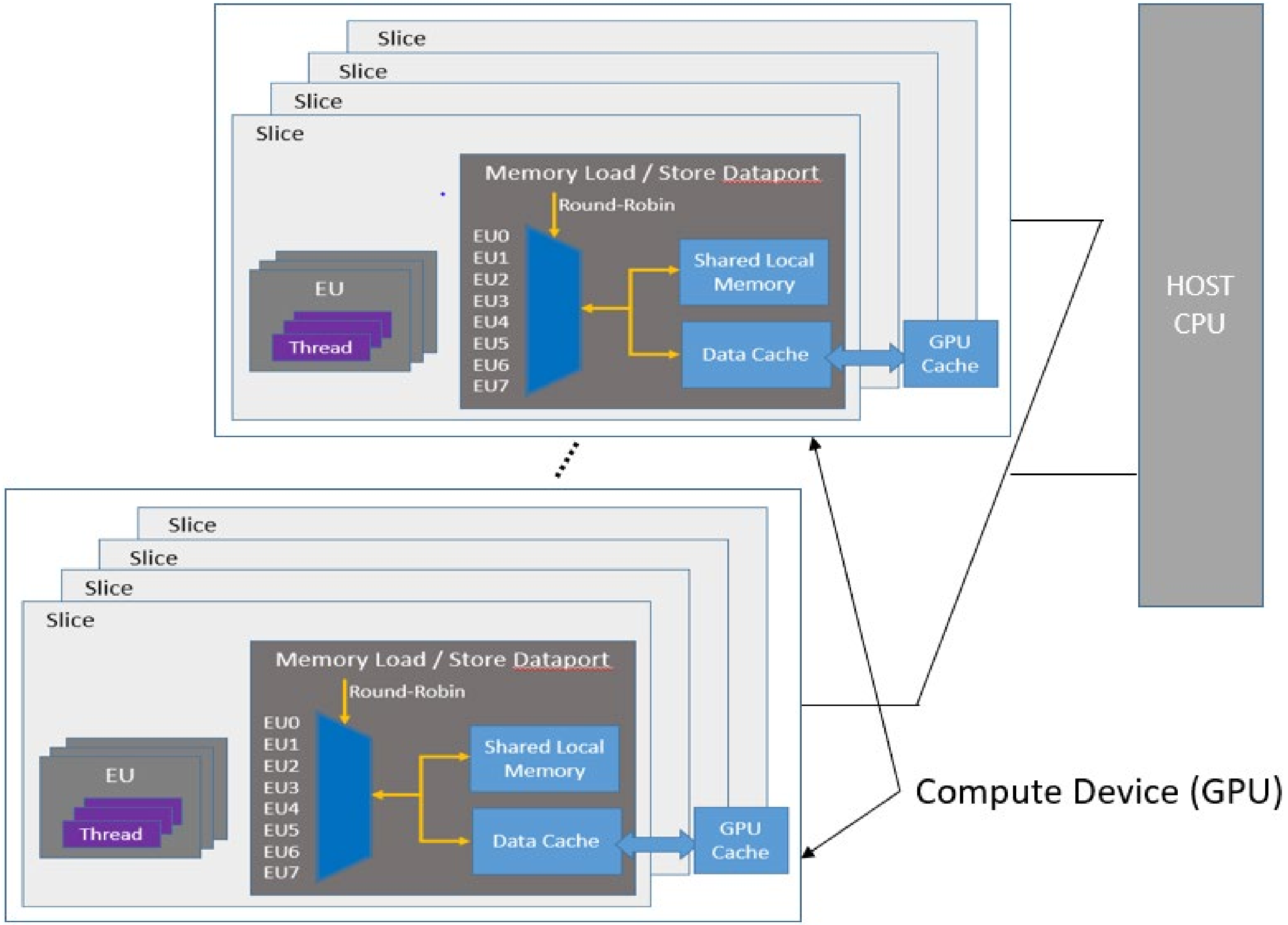
stephen-at-sbcnow-dot-co-dot-uk

Today’s session is about getting visibility

Agenda

- Occupancy
- GPU Offload Modelling with Advisor
- GPU Offload Analysis with Intel Vtune
- GPU Roofline Analysis with Intel Advisor
- Experience with Lammps
- Q & A

Host with accelerator GPUs



Programmers' perspective: Three things to consider

- **Offload the code to the device**

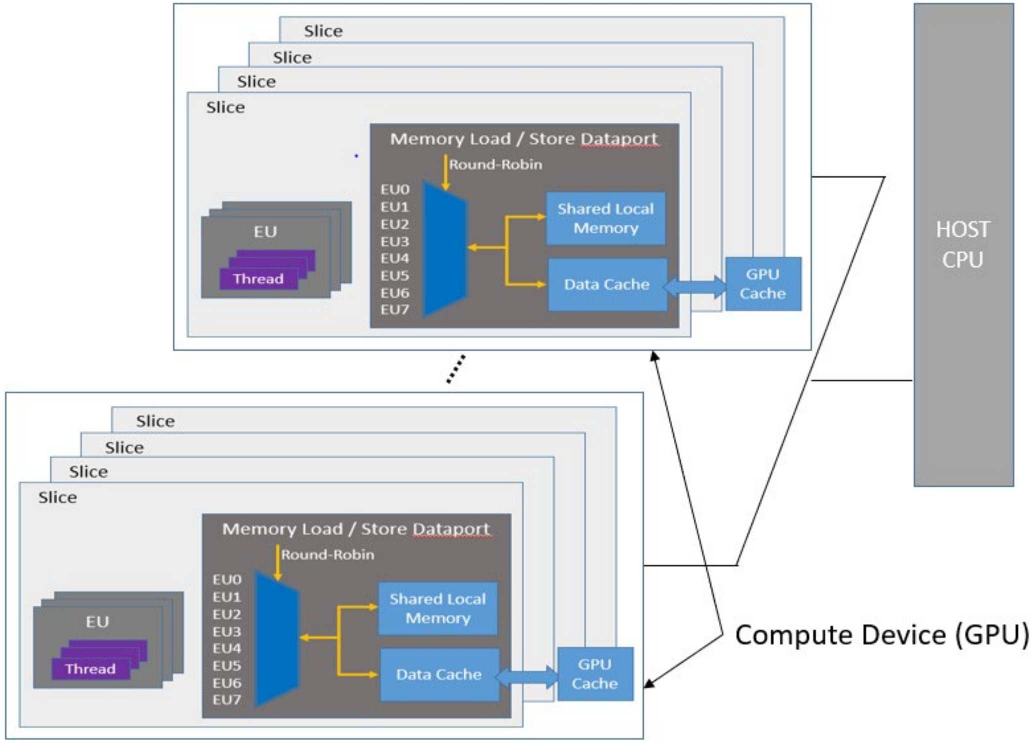
  Choosing appropriate kernel code

- **Manage the transfer of Data**

  (a) Mitigating against communication & memory latency
  (b) Re-use of data

- **Implement Parallelism**

  Maximising Occupancy



Programmers' perspective: Three things to consider

- **Offload the code to the device**

  Choosing appropriate kernel code

- **Manage the transfer of Data**

  (a) Mitigating against communication & memory latency

  (b) Re-use of data

- **Implement Parallelism**

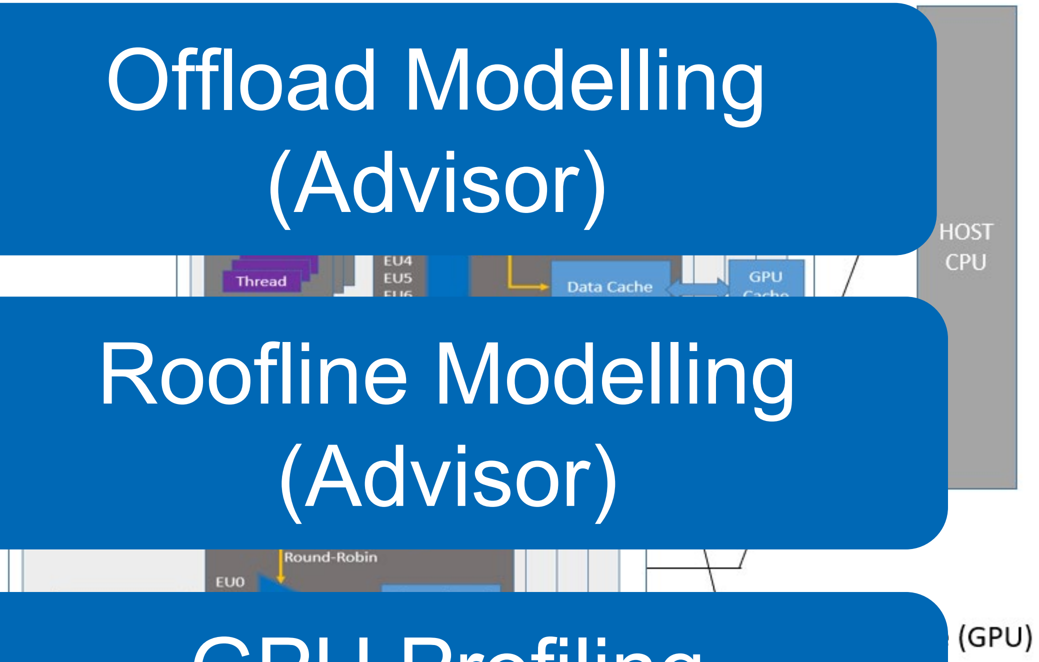
  Maximising Occupancy

Today's examples

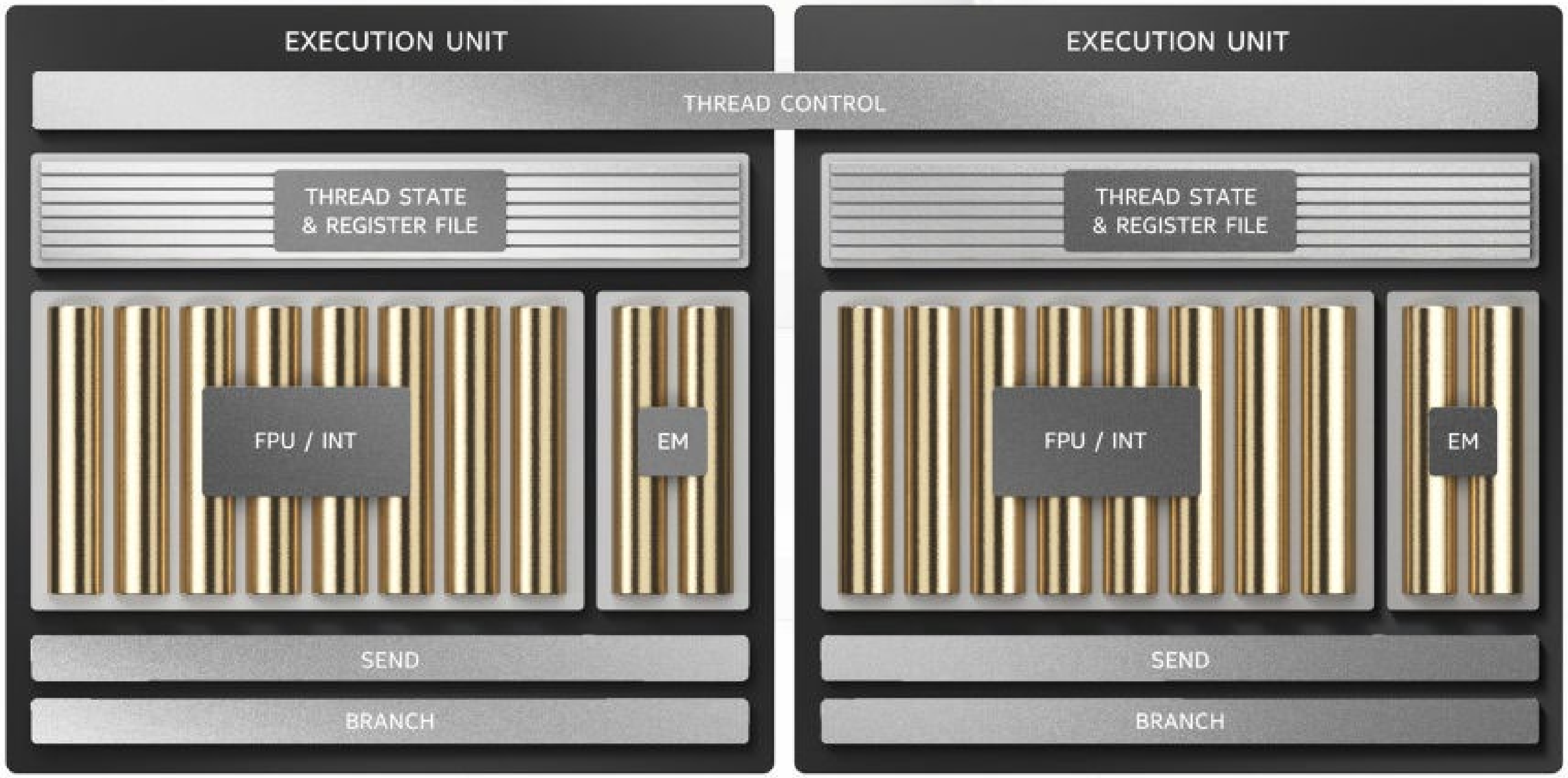
Offload Modelling (Advisor)

Roofline Modelling (Advisor)

GPU Profiling VTune



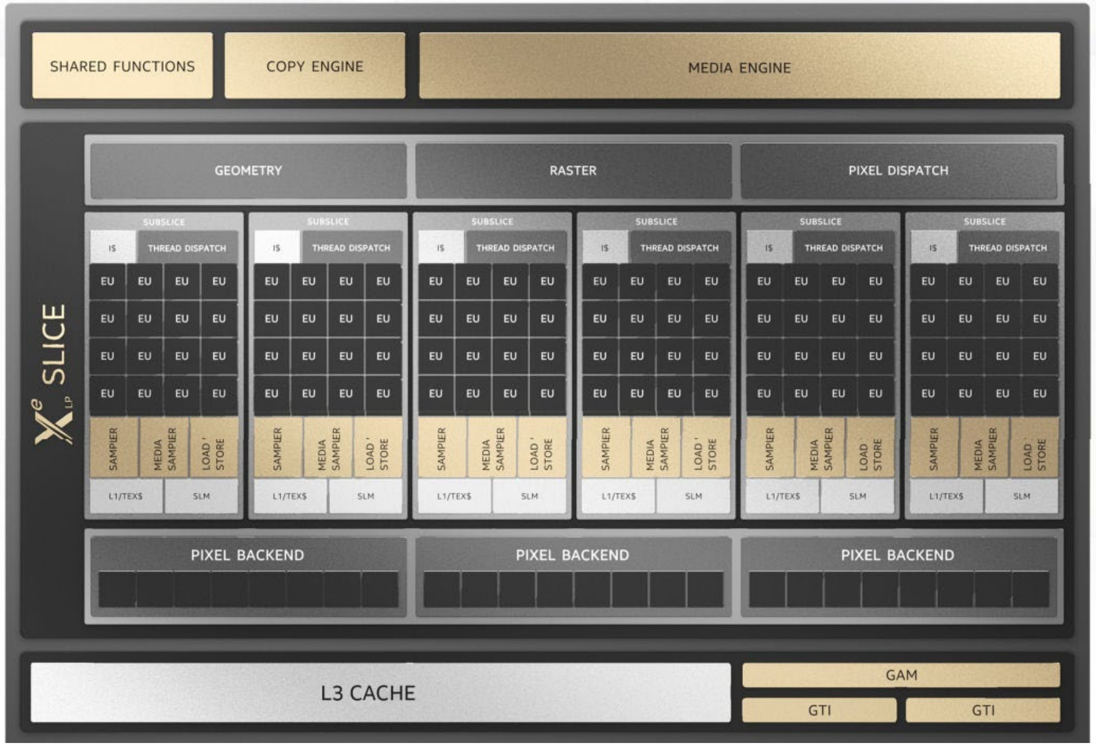
GPU Occupancy



Intel® Iris® Xe-LP GPU Compute Throughput Rates (Ops/clock/EU)

FP32	FP16	INT32	INT16	INT 8
8	16	8	16	32 (DP4A)

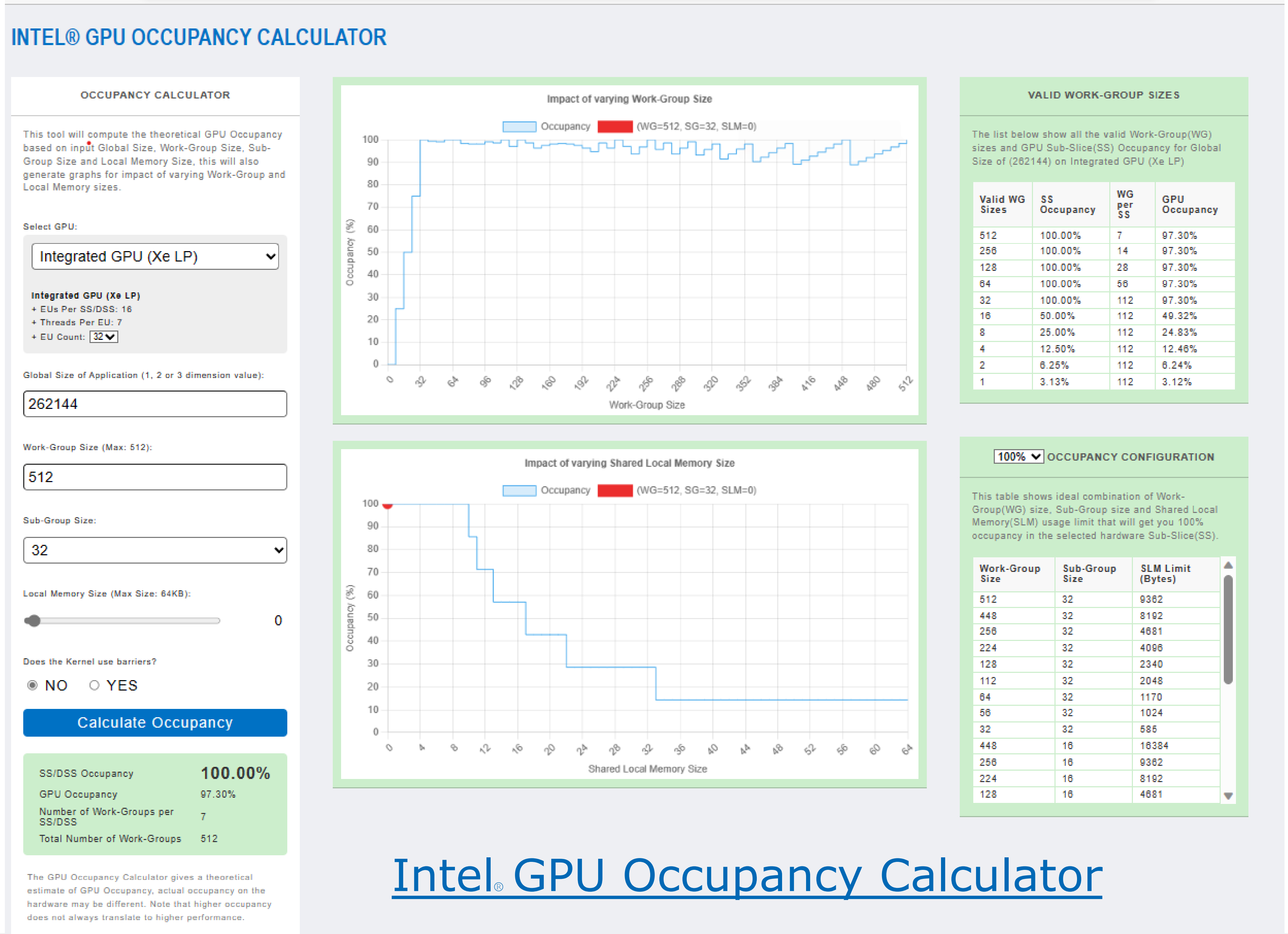
X^e-LP configuration



X^e-LP (TGL) GPU

	VEs	Threads	Operations	Maximum Work Group Size
Each X^e-core	16	$7 \times 16 = 112$	$112 \times 8 = 896$	512
Total	$16 \times 6 = 96$	$112 \times 6 = 672$	$896 \times 6 = 5376$	512

- 6 Cores
- 16 Vector Engines/core [Total 96]
- 7 Threads
- 16 FP16 ops/clock/VE



Maximum Occupancy Calculator

[Intel® GPU Occupancy Calculator]

https://github.com/oneapi-src/oneAPI-samples/tree/master/Tools/GPU-Occupancy-Calculator

Language concepts for max occupancy

SYCL	OpenMP
Work-item	Thread
Sub-group	Team
Work-Group	League

oneAPI Debug Tools (intel.com)

Tool	Description
Environment variables	Gather diagnostic information from the OpenMP and SYCL runtimes at program execution with no modifications to your program.
Onetrace	From Profiling Tools Interfaces for GPU (PTI for GPU). Used to debug backend errors and for performance profiling on both the host and device.
Intercept Layer for OpenCL™ Applications	Used to debug backend errors and for performance profiling on both the host and device (has wider functionality comparing with onetrace).
Intel® Distribution for GDB*	Used for source-level debugging of the application, typically to inspect logical bugs, on the host and any devices you are using (CPU, GPU, FPGA emulation).
Intel® Inspector	locate and debug memory and threading problems, including those that can cause offloading to fail.
In-application debugging	printf; looking at output of apps etc, etc
Intel® Advisor	Use to ensure Fortran, C, C++, OpenCL™, and SYCL applications realize full performance potential on modern processors.
Intel® VTune™ Profiler	Use to gather performance data either on the native system or on a remote system.

Advisor Offload Modelling

intel®

Offload Modelling – doing it by hand

1. Run code on CPU and find hotspots

2. Examine results – decide which hotspots are suitable for offloading

3. Implement offload using SYCL or OpenMP

4. Run code on CPU and GPU and see if there is a speed up

CPU

CPU — GPU

Potentially time-consuming.
Possibly no return-on-investment

Offload Modelling – with Advisor

1 Run code on CPU using Advisor (multiple stage profiling collection)

CPU

2 Examine results – decide which hotspots are suitable for offloading

3 Implement offload using SYCL or OpenMP

4 Run code on CPU and GPU and see if there is a speed up

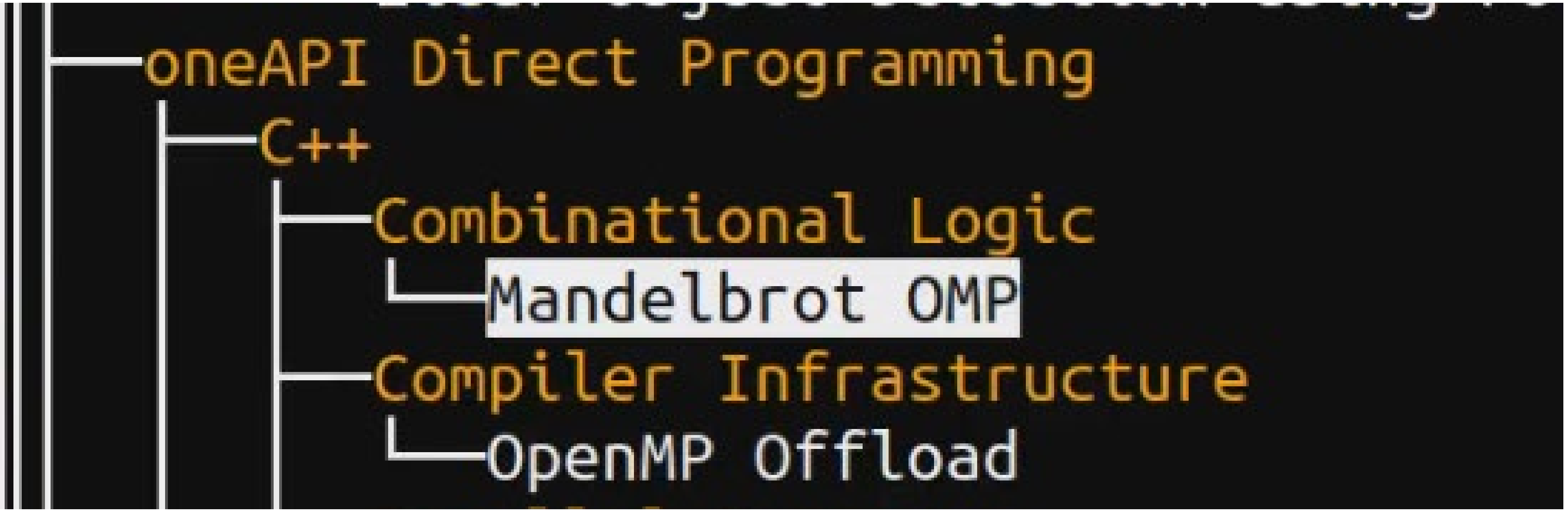
CPU — GPU

Advisor Simulates all these stages

CPU

Report

Demo – Our Example
oneapi-cli



make CXX=icpx EXTRA_CFLAGS=-g

Using accuracy presets to control modelling

- Default (Medium Accuracy)
```
advisor --collect=offload --config=gen12_tgl
--project-dir=./cpu2gpu_offload_modeling --
./release/Mandelbrot 1
```

- Low Accuracy
```
advisor --collect=offload -accuracy=low
--config=gen12_tgl --project-
dir=./cpu2gpu_offload_modeling --
./release/Mandelbrot 1
```

- Getting list of steps
```
advisor --collect=offload --dry-run
--config=gen12_tgl --project-
dir=./cpu2gpu_offload_modeling --
./release/Mandelbrot 1
```

Low	Medium	High
5-10x overhead	15-50x overhead	50-80x overhead
Survey Trip Count Offload Modelling	Survey Trip Count Offload Modelling	Survey Trip Count Dependency analysis Offload Modelling
L1 Cache	L1 Cache + Host-Device data	L1 Cache + Host-Device data

Steps to Offload Projection with Advisor

1. Run a **Survey**: get a list of hotspots

```
advisor -collect survey …
```

- Sampling
- Binary Static Analysis
- Compiler & debug info

2. Run a **Trip Count**: count loop iteration

```
advisor -collect=tripcounts -target-device=gen9_gt2 .
```

- Trip count
- Cache simulation

3. Perform a **dependency analysis** [optional for quick modelling]

```
advisor -collect dependencies . . .
```

- Check memory accesses
- Loop selection heuristic

4. **Model** the Performance

```
advisor -collect projection -no-assume-dependencies . . .
```

- Generate HTML report

Expensive Steps

20

DRY RUN LOWACCURACY==

advisor: The 'offload' is a special batch mode for data collection. It runs several analyses one by one.
advisor --collect=survey --auto-finalize --static-instruction-mix --project-dir=./cpu2gpu_offload_modeling -- ./03-run.sh
advisor --collect=tripcounts --flop --auto-finalize --target-device=gen12_tgl --project-dir=./cpu2gpu_offload_modeling -- ./03-run.sh
advisor --collect=projection **--no-assume-dependencies** --config=gen12_tgl --project-dir=./cpu2gpu_offload_modeling -- ./03-run.sh

DRY RUN MEDIUM ACCURACY==

advisor: The 'offload' is a special batch mode for data collection. It runs several analyses one by one.
advisor --collect=survey --auto-finalize --static-instruction-mix --project-dir=./cpu2gpu_offload_modeling -- ./03-run.sh
advisor --collect=tripcounts --flop --stacks --auto-finalize **--cache-simulation=single** **--data-transfer=light** --target-device=gen12_tgl --project-dir=./cpu2gpu_offload_modeling -- ./03-run.sh
advisor --collect=projection **--no-assume-dependencies** --config=gen12_tgl --project-dir=./cpu2gpu_offload_modeling -- ./03-run.sh

DRY RUN HIGH ACCURACY==

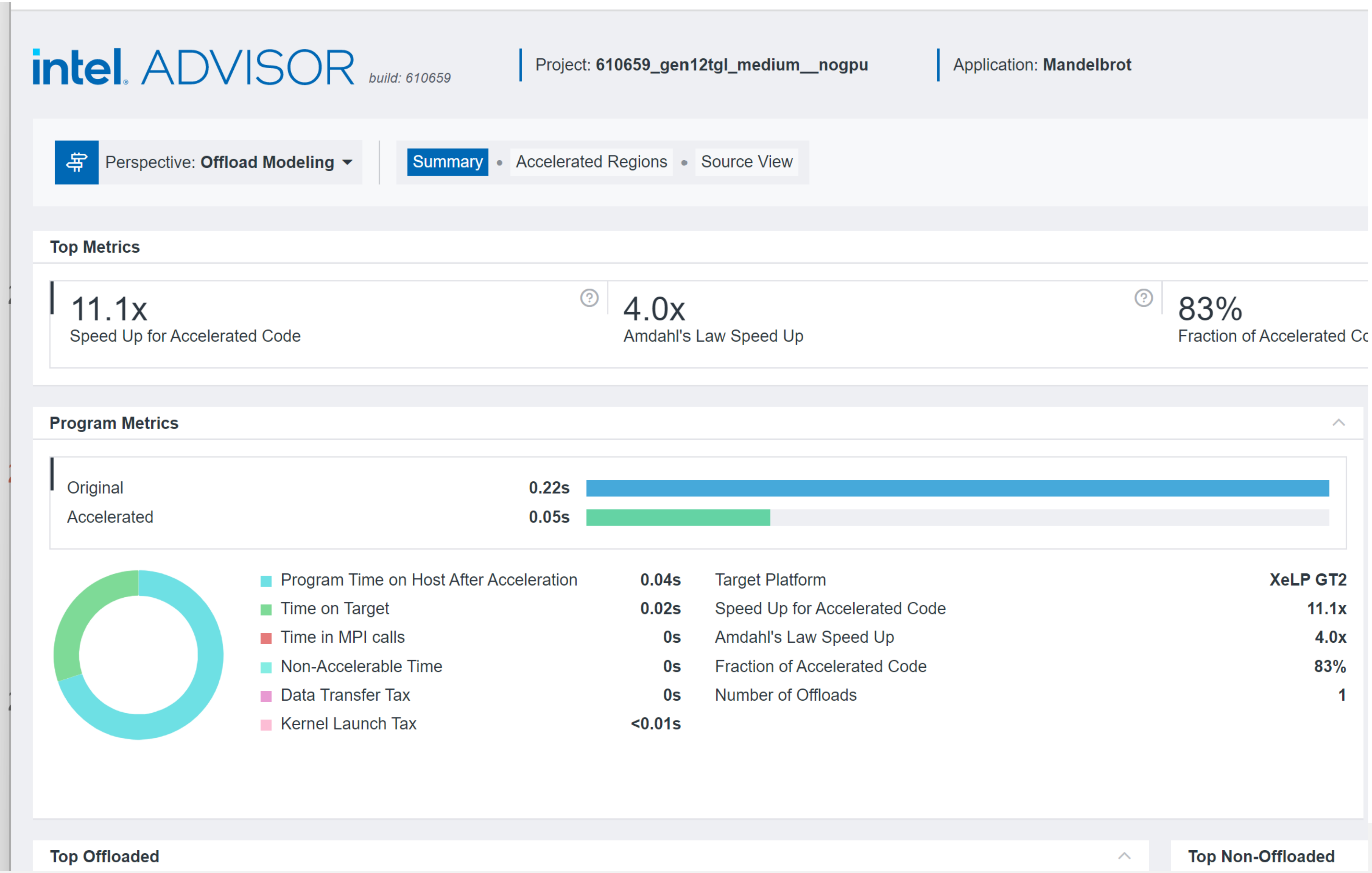
advisor: The 'offload' is a special batch mode for data collection. It runs several analyses one by one.
advisor --collect=survey --auto-finalize --static-instruction-mix --project-dir=./cpu2gpu_offload_modeling -- ./03-run.sh
advisor --collect=tripcounts --flop --stacks --auto-finalize **--cache-simulation=single** **--data-transfer=medium** --target-device=gen12_tgl --project-dir=./cpu2gpu_offload_modeling -- ./03-run.sh
advisor **--collect=dependencies --filter-reductions --loop-call-count-limit=16 --select=markup=gpu_generic --project-dir=./cpu2gpu_offload_modeling -- ./03-run.sh**
advisor --collect=projection --config=gen12_tgl --project-dir=./cpu2gpu_offload_modeling -- ./03-run.sh

LOW

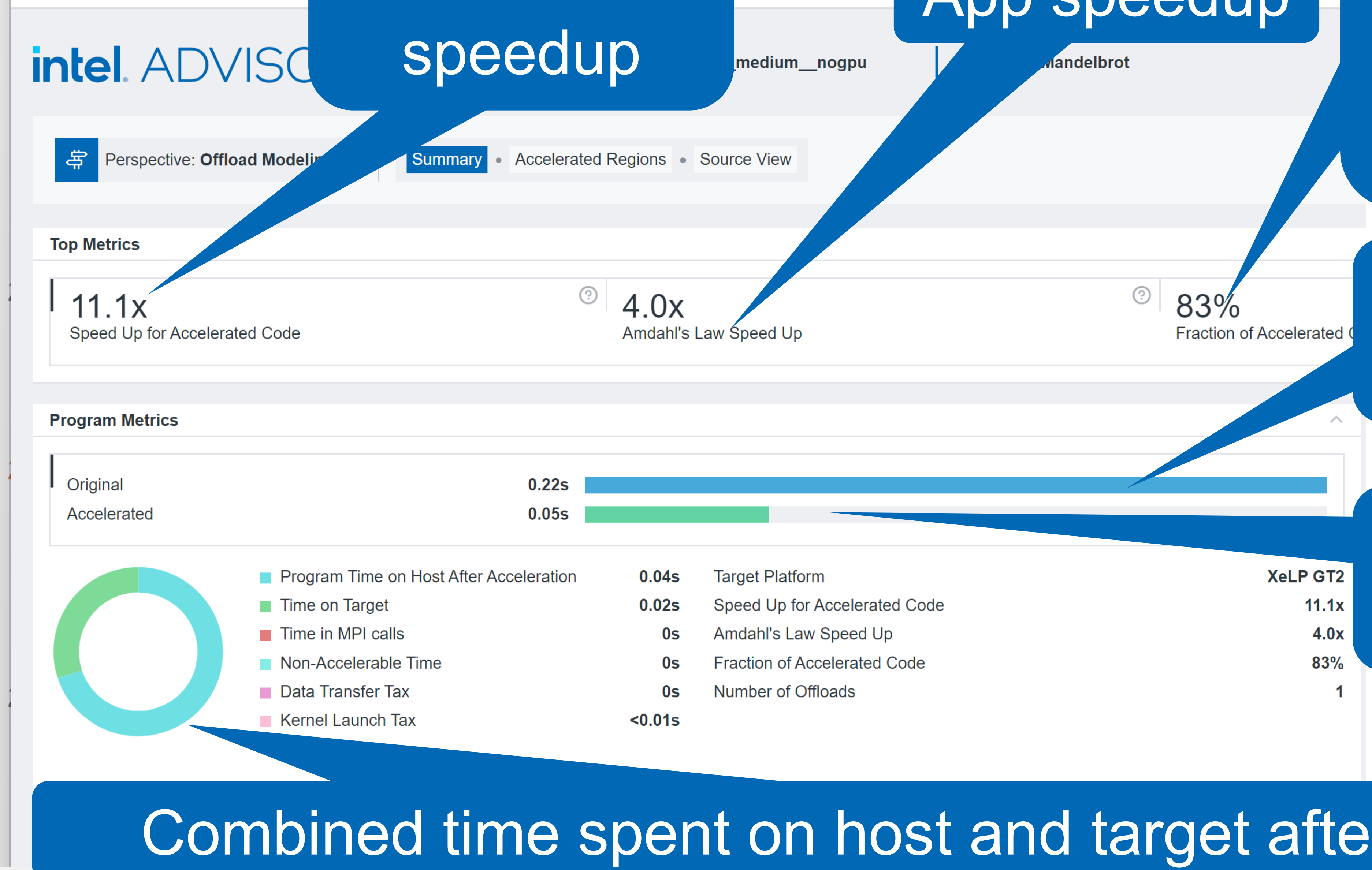
MEDIUM

HIGH

Summary


intel ADVISOR build: 610659
Project: 610659_gen12tgl_medium__nogpu
Application: Mandelbrot
Perspective: Offload Modeling
Summary
Accelerated Regions
Source View
Top Metrics
11.1x
Speed Up for Accelerated Code
4.0x
Amdahl's Law Speed Up
83%
Fraction of Accelerated Cc
Program Metrics
Original 0.22s
Accelerated 0.05s
Program Time on Host After Acceleration 0.04s
Time on Target 0.02s
Time in MPI calls 0s
Non-Accelerable Time 0s
Data Transfer Tax 0s
Kernel Launch Tax <0.01s
Target Platform XeLP GT2
Speed Up for Accelerated Code 11.1x
Amdahl's Law Speed Up 4.0x
Fraction of Accelerated Code 83%
Number of Offloads 1
Top Offloaded
Top Non-Offloaded

Summary



Accelerated code speedup

App speedup

% of time candidate code took in original app

Original App Time

New App Time

Combined time spent on host and target after aceleration

Loop Offload (from Summary Page)

Top Five offload candidates

Top Offloaded

Loop/Function	Execution Time	Speed-Up	Bounded By	Data Transfer
[loop in serial_mandelbrot at mandelbrot.cpp:56]	CPU 182.0ms GPU 16.4ms	11.128x	Compute	0B

Top Five non-offload (only in CPU-GPU)

Top Non-Offloaded

Loop/Function	Execution Time	Bounded By	Why Not Offloaded	Data Transfer
[loop in stbi_zlib_compress at stb_image_write.h:885]	CPU 28.0ms GPU 266.0ms	Trip Counts	Not profitable: The Number of Loop Iterations is not enough to fully utilize Target Device capabilities	2.43MB

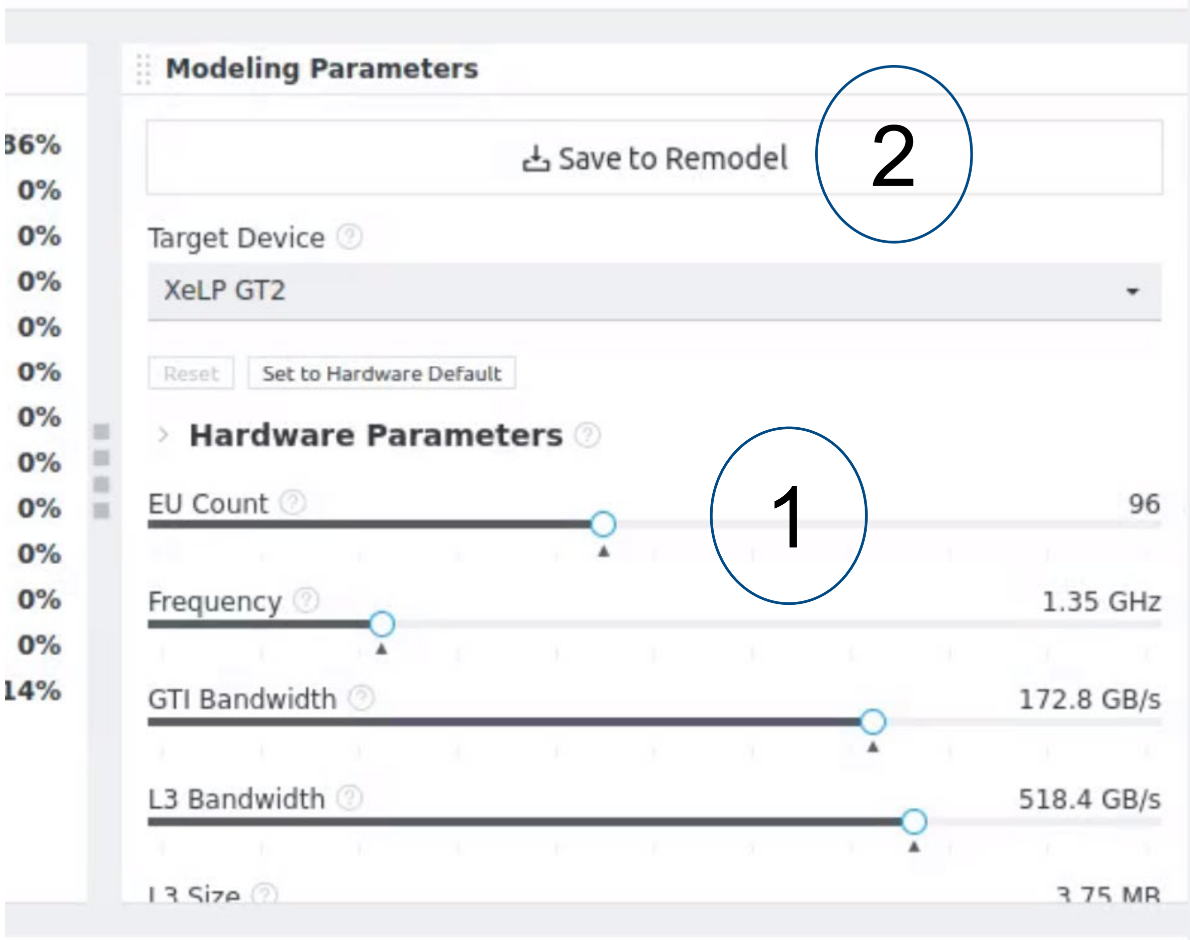
GPU-GPU modelling

- Add the –gpu flag
- Runs code on ACTUAL GPU and models against new GPU.
- Use when upgrading from one GPU to another one.
- NB: use –accuracy=low

Comparison of CPU-GPU and GPU-GPU modelling

CPU-GPU	GPU-GPU
Survey	Survey (on GPU)
Trip Count	Trip Count (Characterization - num Floats and Integer operations)
Dependency Check	x
Model Offloading	Model Offloading

Re-modelling for a different GPU
- once you already have a set of results



1. Adjust Values
2. Save parameters
3. Re-run modelling

e.g.: advisor -c=projection
--custom-config=config.toml
--config=gen12_tgl
--project-dir=/...

Advantage: quicker than doing a completely new modelling

GPU Offload Analysis with Intel® VTune™

intel®

VTune GPU Profiling Recipe

[Intel® VTune™ Profiler Performance Analysis Cookbook]

[Profiling a SYCL* Application running on a GPU (intel.com)]

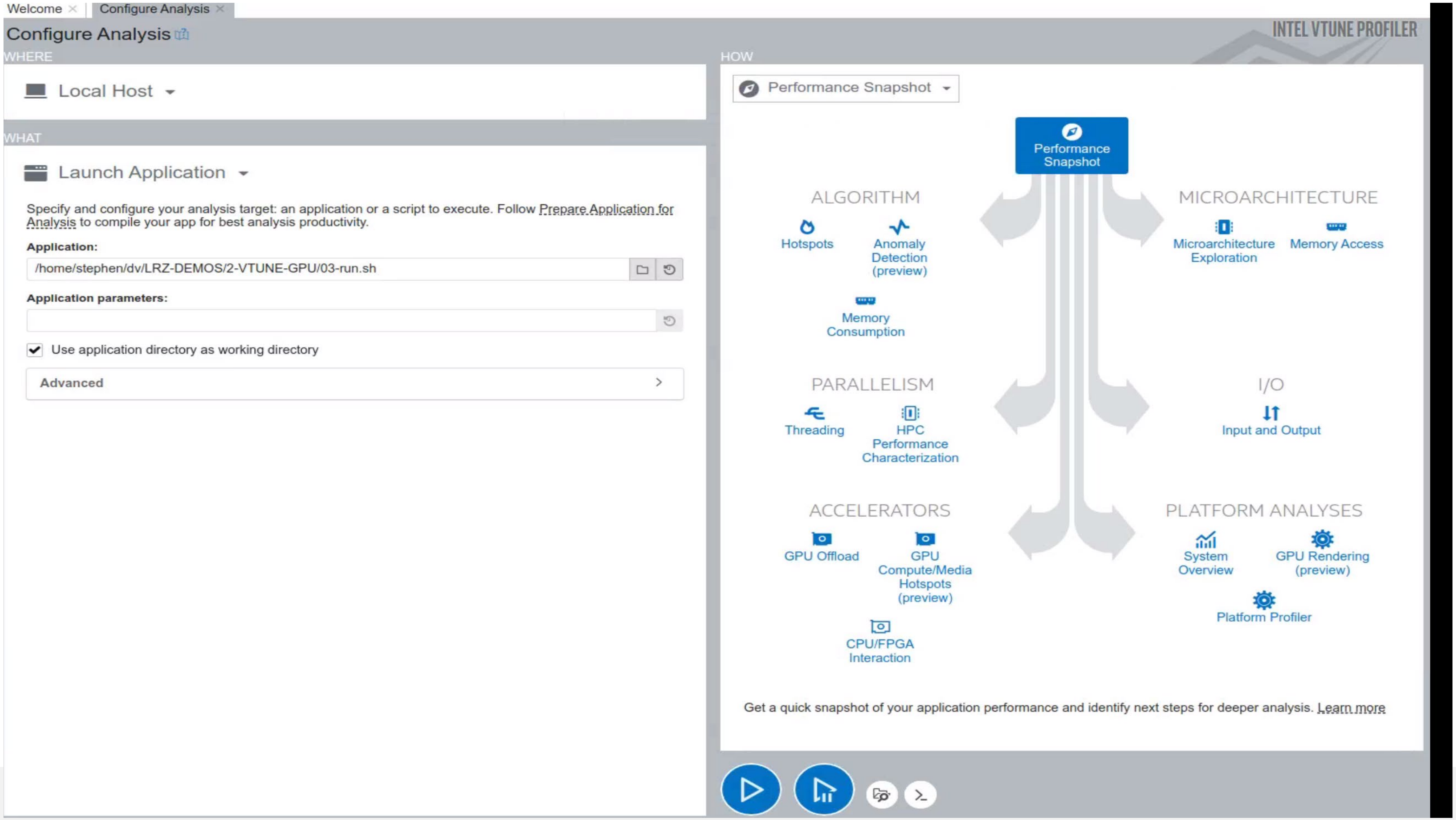
Two types of analysis

1. Build and Compile a SYCL Application
2. Run GPU Offload Analysis on a SYCL Application
3. Analyze Collected Data
4. Run GPU Compute/Media Hotspots Analysis
5. Analyze Your Compute Task

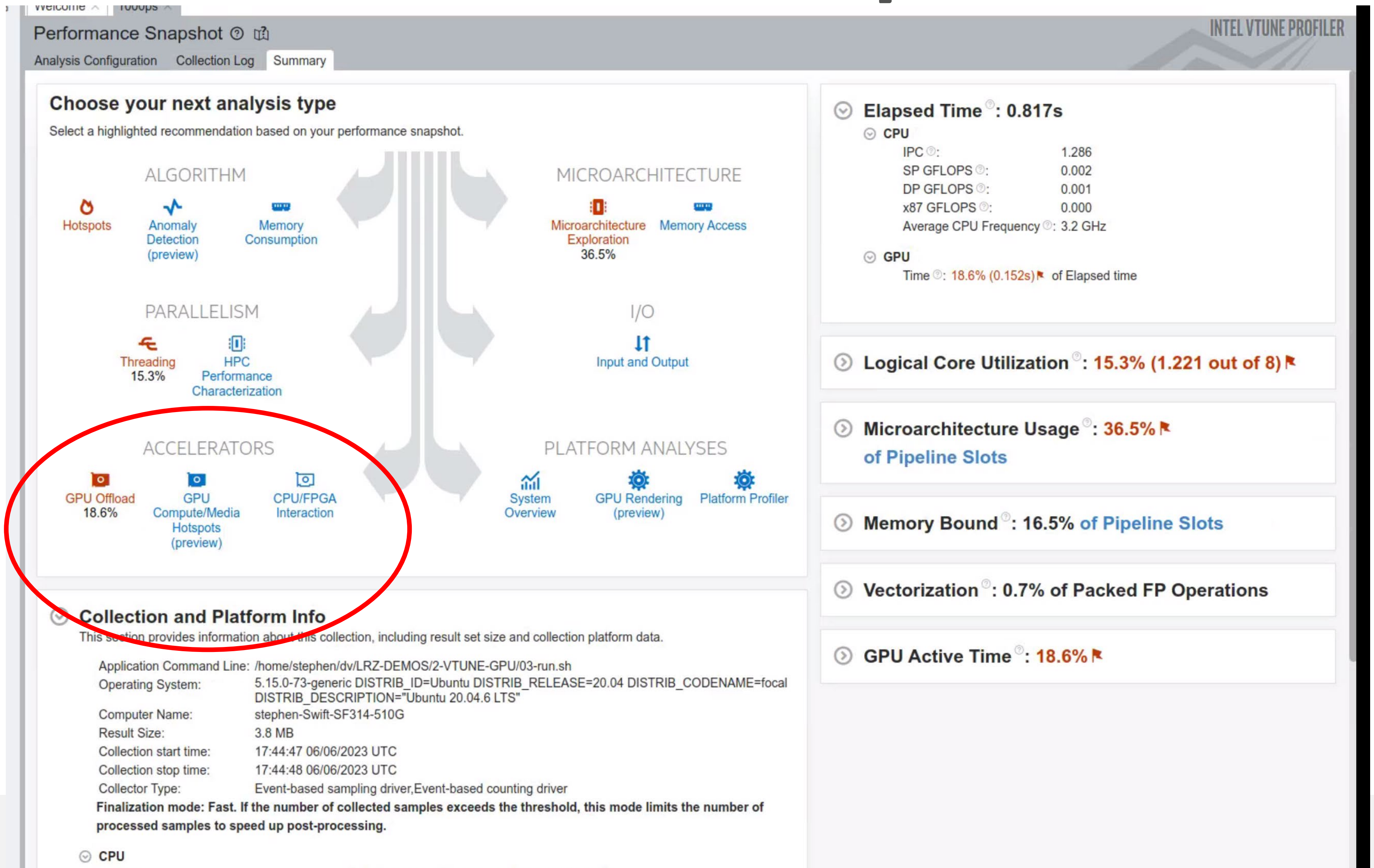
STEP 1: Build and Compile

- cd matrix_multiply_vtune
  mkdir build –p
  cd build
  cmake ..
  make
- icpx -g -O3 -fsycl -Wno-write-strings -w -D_Linux -MD -MT CMakeFiles/matrix.dpcpp.dir/src/matrix.cpp.o
  -MF CMakeFiles/matrix.dpcpp.dir/src/matrix.cpp.o.d
  -o CMakeFiles/matrix.dpcpp.dir/src/matrix.cpp.o
  -c /home/stephen/dv/LRZ-DEMOS/2-VTUNE-GPU/matrix_multiply_vtune/src/matrix.cpp

The Starting point – Running a Performance Snapshot



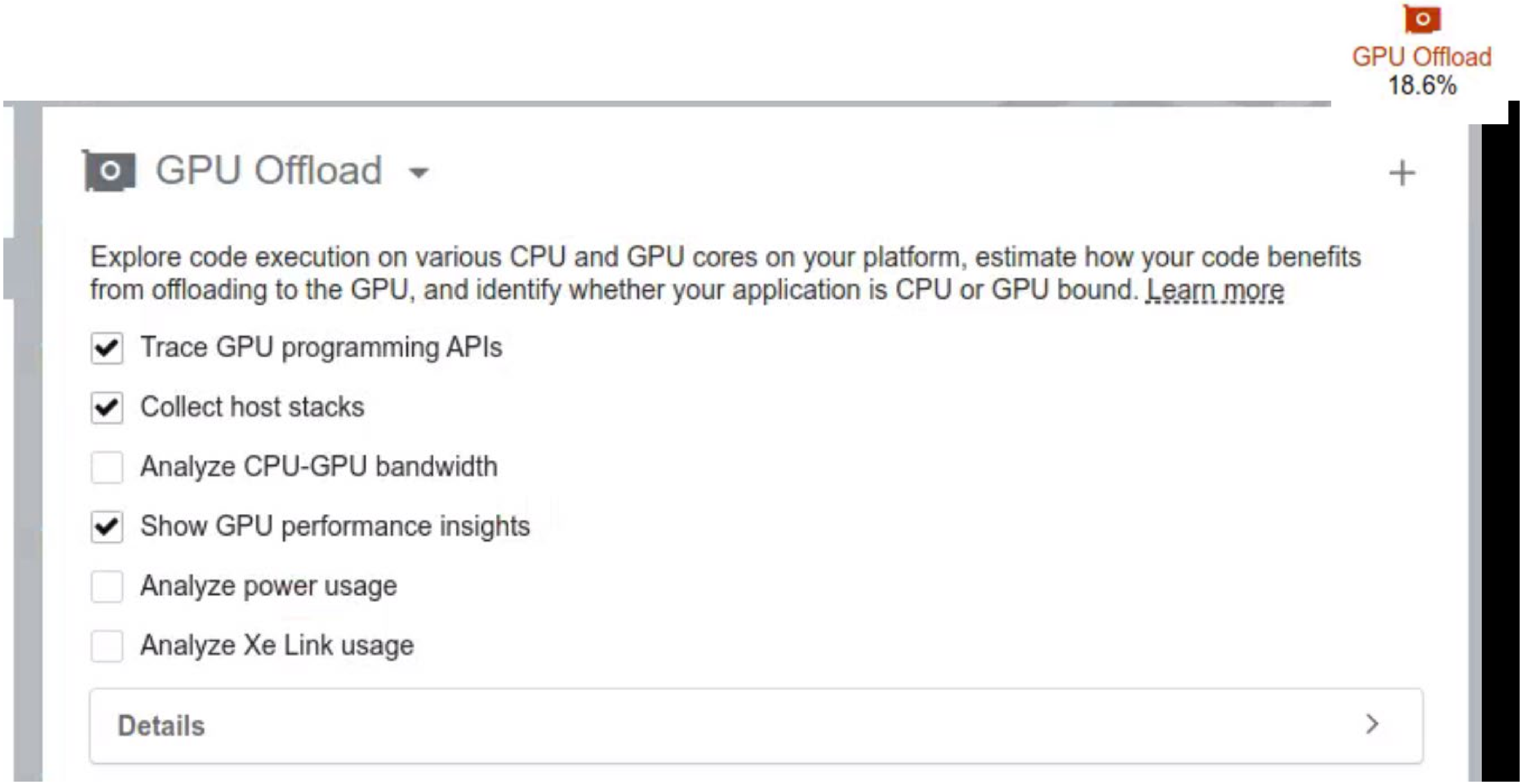
Choose one of the Accelerator Analysis Types [GPU Offload is best to do first]



GPU Offload Analysis

gpu-offload (GPU Offload Analysis)

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


GPU Offload
18.6%
GPU Offload
Explore code execution on various CPU and GPU cores on your platform, estimate how your code benefits from offloading to the GPU, and identify whether your application is CPU or GPU bound. Learn more
Trace GPU programming APIs
Collect host stacks
Analyze CPU-GPU bandwidth
Show GPU performance insights
Analyze power usage
Analyze Xe Link usage
Details

Command line for **GPU Offload Analysis**


```
vtune -collect gpu-offload

--app-working-dir=/home/stephen/dv/LRZ-DEMOS/2-VTUNE-GPU

-- /home/stephen/dv/LRZ-DEMOS/2-VTUNE-GPU/03-run.sh
```

Running a First Analysis -

Hottest CPU and GPU Tasks

GPU Offload
18.6%

Hottest Host Tasks

This section lists the most active tasks running on the host, sorted by the Task Time. Focus on performance-critical tasks first.

Host Task	Task Time	% of Elapsed Time	Task Count
zeModuleCreate	0.211s	7.6%	1
zeEventHostSynchronize	0.048s	1.7%	2
zeCommandListAppendMemoryCopyRegion	0.001s	0.0%	1
zeCommandQueueExecuteCommandLists	0.000s	0.0%	2
zeCommandListCreateImmediate	0.000s	0.0%	1
[Others]	0.000s	0.0%	9

*N/A is applied to non-summable metrics.

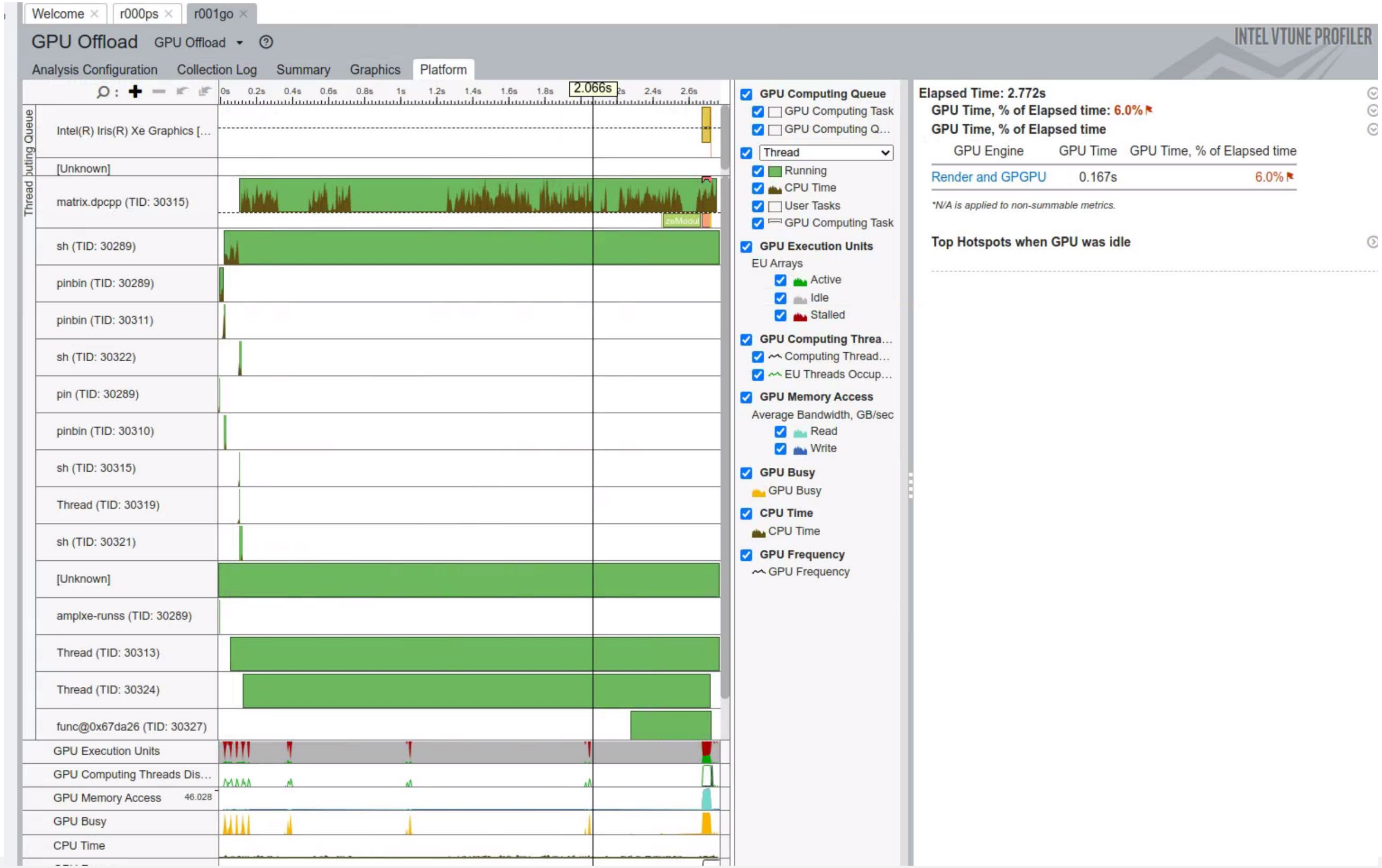
Hottest GPU Computing Tasks

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.

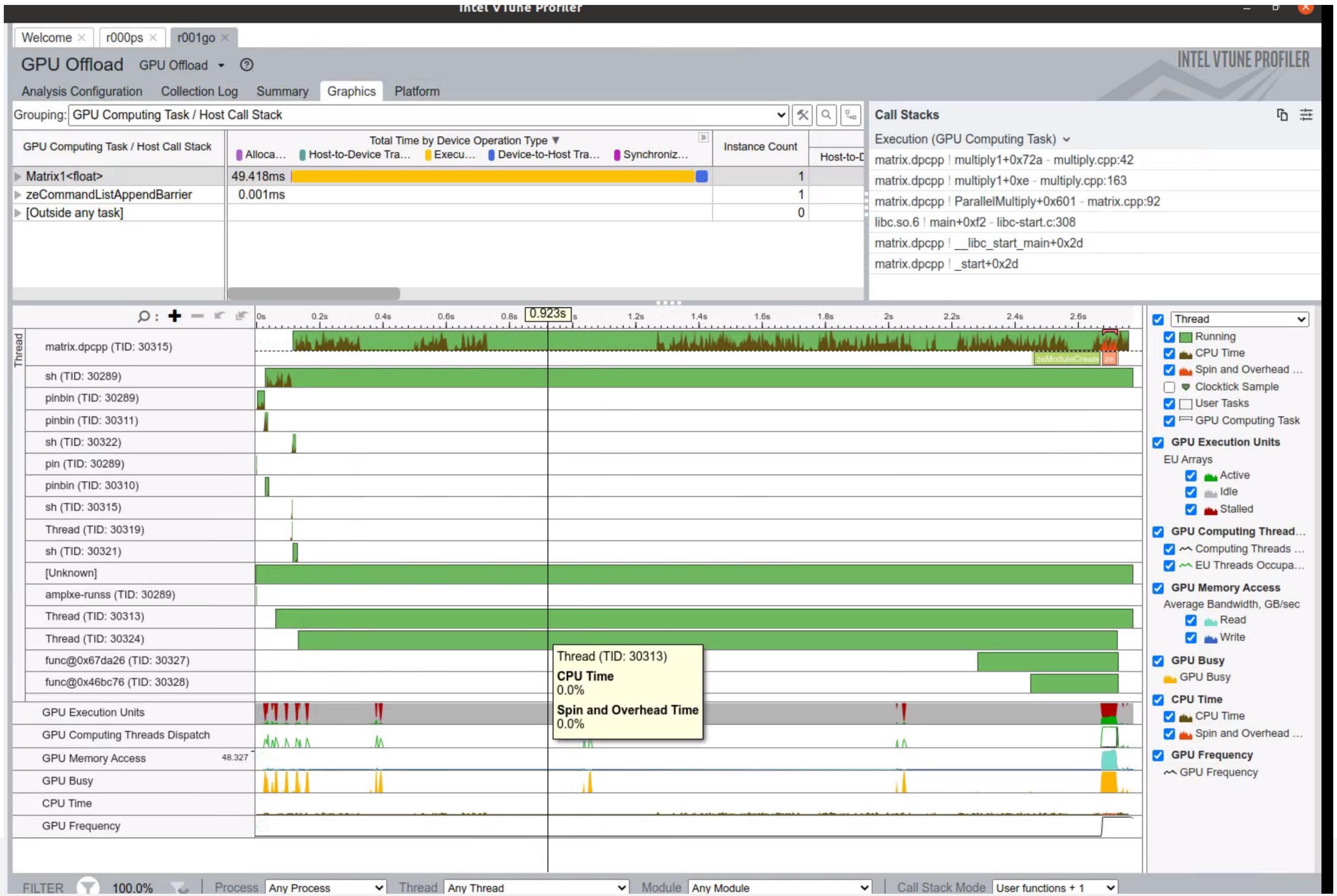
Computing Task	Total Time	Execution Time	% of Total Time	SIMD Width	Peak EU Threads Occupancy	EU Threads Occupancy	SIMD Utilization
Matrix1<float>	0.049s	0.048s	97.1%	32	100.0%	98.8%	100.0
zeCommandListAppendBarrier	0.000s	0s	0.0%				

*N/A is applied to non-summable metrics.

The Platform Tab



The Graphics Tab



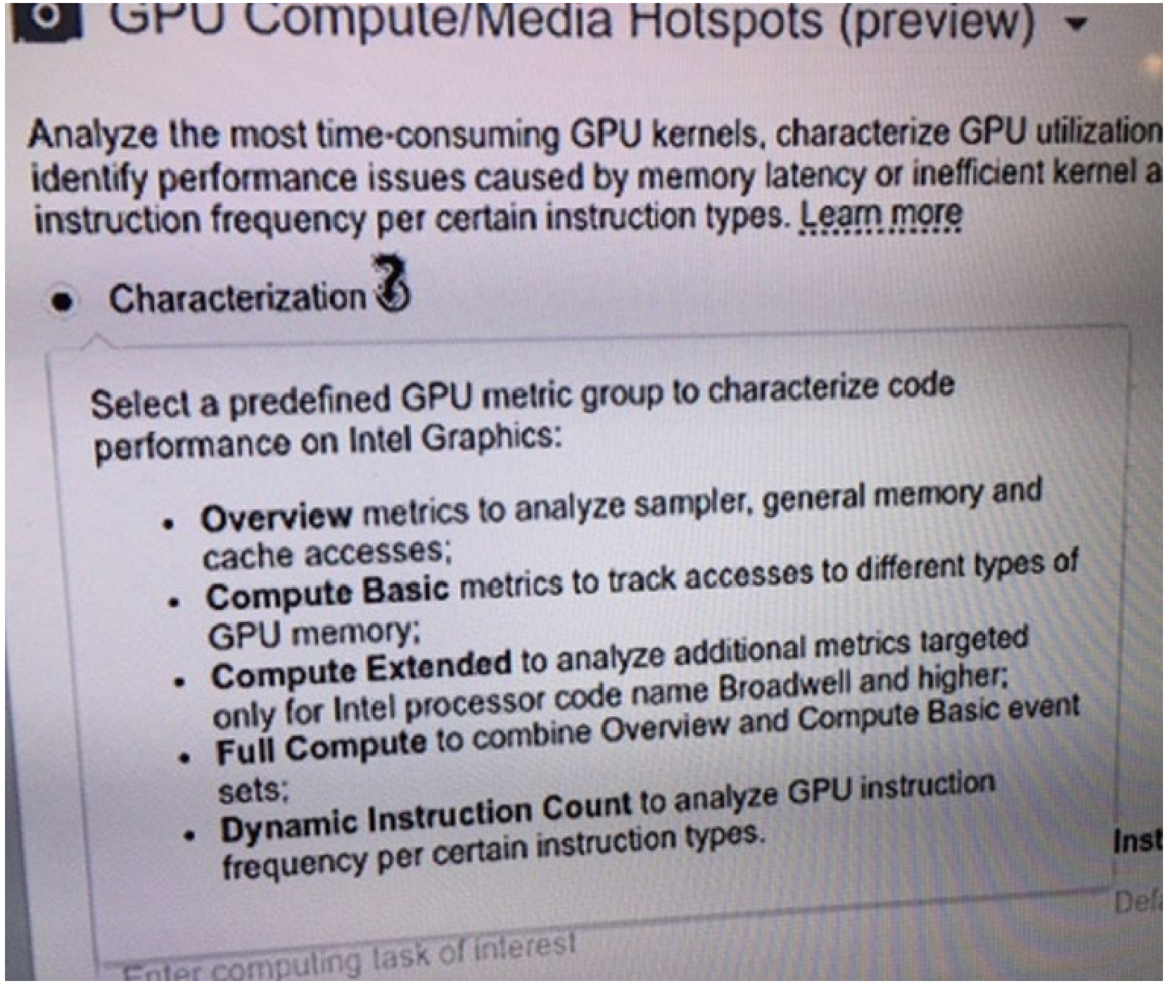
GPU Compute/Media Hotspots analysis

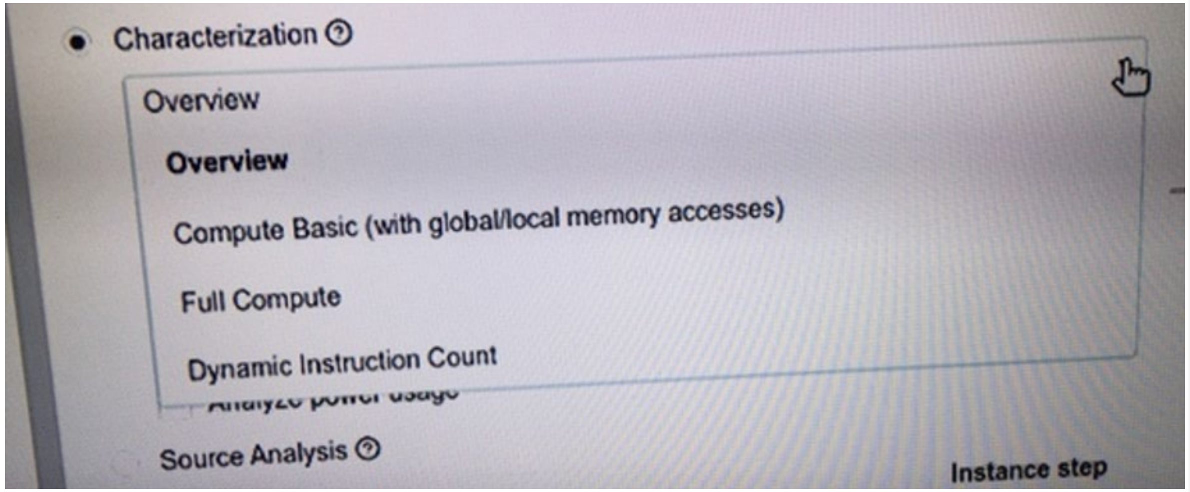
gpu-hotspots (GPU Compute/Media Hotspots analysis)

- Explore GPU kernels with high GPU utilization, estimate the effectiveness of this utilization, identify possible reasons for stalls or low occupancy and options.
- Explore the performance of your application per selected GPU metrics over time.
- Analyze the hottest SYCL* standards or OpenCL™ kernels for inefficient kernel code algorithms or incorrect work item configuration.

Configure Analysis

GPU
Compute/Media
Hotspots
(preview)





Running a **GPU Compute/Media Hotspots analysis**
*Characterization: **Overview***

vtune -collect **gpu-hotspots**

--app-working-dir=/home/stephen/dv/LRZ-DEMOS/2-VTUNE-GPU

-- /home/stephen/dv/LRZ-DEMOS/2-VTUNE-GPU/03-run.sh

Running a **GPU Compute/Media Hotspots analysis**
*Characterization: **Compute Basic***

vtune -collect **gpu-hotspots**

-knob characterization-mode=global-local-accesses

--app-working-dir=/home/stephen/dv/LRZ-DEMOS/2-VTUNE-GPU

-- /home/stephen/dv/LRZ-DEMOS/2-VTUNE-GPU/03-run.sh

Running a **GPU Compute/Media Hotspots analysis**
*Characterization: **Full Compute***

vtune -**collect gpu-hotspots**

-knob characterization-mode=full-compute

--app-working-dir=/home/stephen/dv/LRZ-DEMOS/2-VTUNE-GPU

-- /home/stephen/dv/LRZ-DEMOS/2-VTUNE-GPU/03-run.sh

Running a **GPU Compute/Media Hotspots analysis**
*Characterization: **Dynamic Instruction Count***

vtune -collect **gpu-hotspots**

-knob characterization-mode=instruction-count

--app-working-dir=/home/stephen/dv/LRZ-DEMOS/2-VTUNE-GPU

-- /home/stephen/dv/LRZ-DEMOS/2-VTUNE-GPU/03-run.sh

GPU Hotspots - Summary

GPU Compute/Media Hotspots (preview)

Elapsed Time: 2.912s

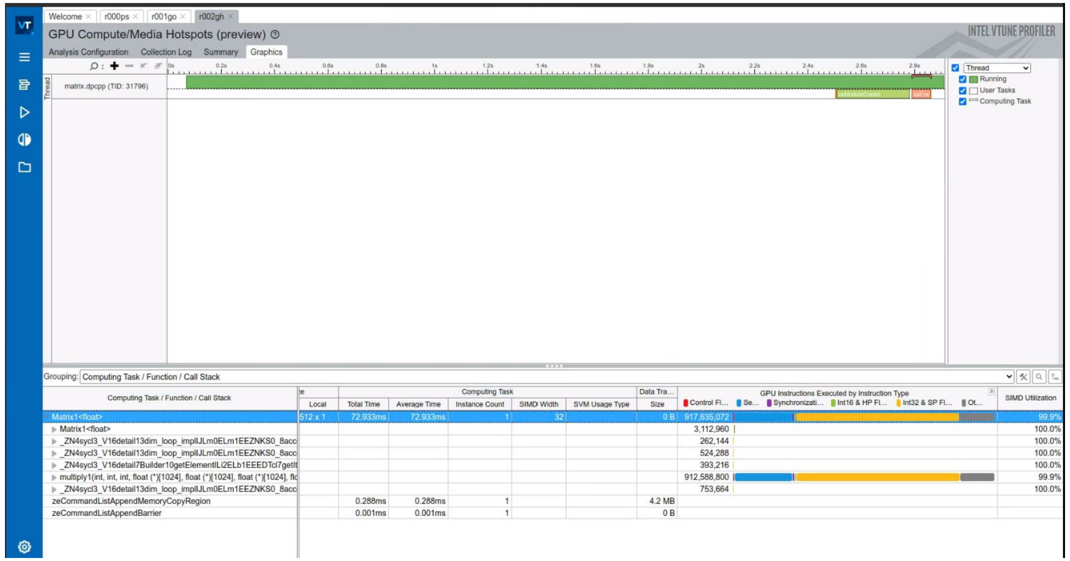
Hottest GPU Computing Tasks

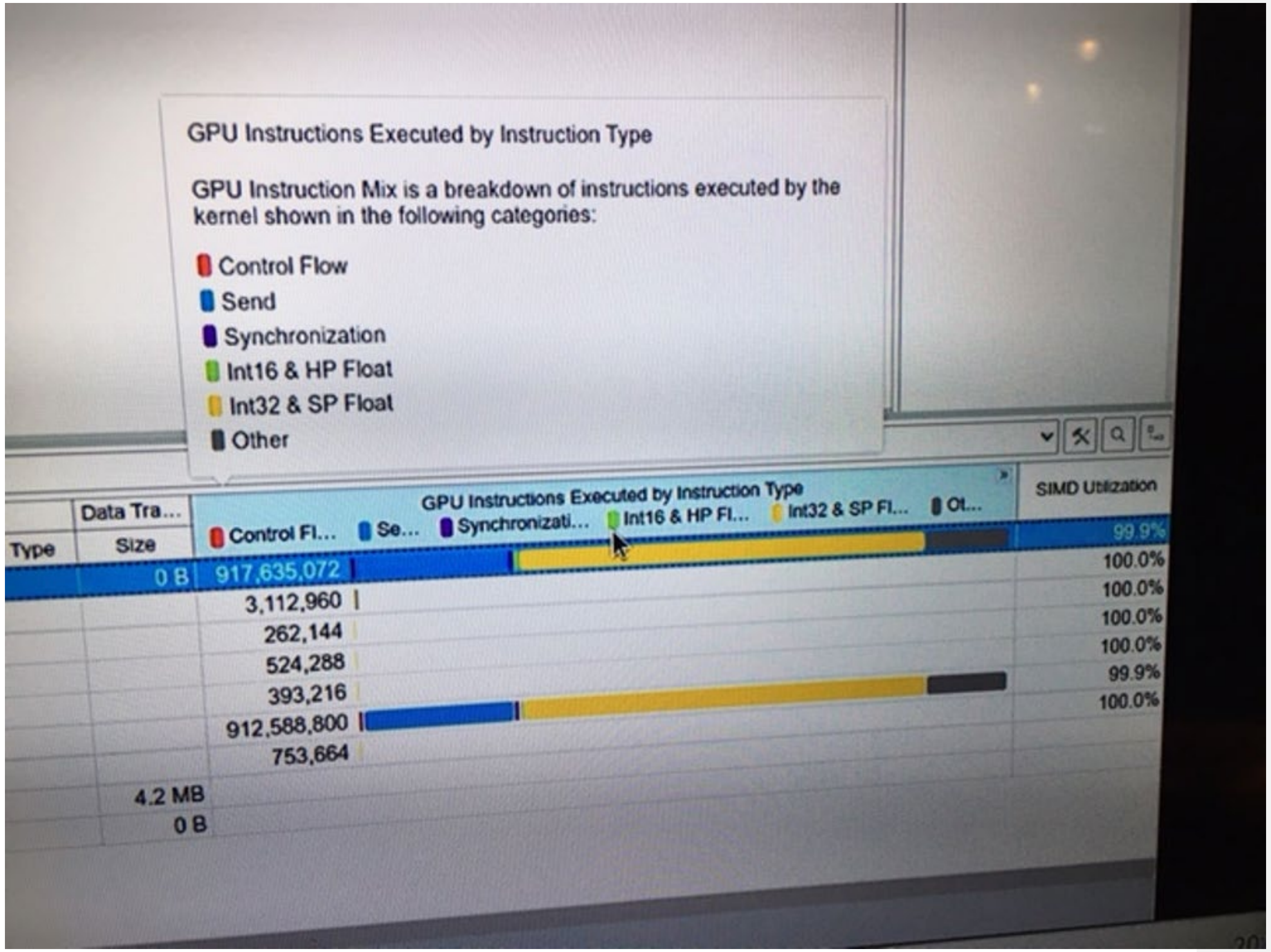
This section lists the most active computing tasks running on the GPU, sorted by the Total Time.

Computing Task	Total Time	Average Time	Instance Count	GPU Instructions Executed
Matrix1<float>	0.073s	0.073s	1	917,635,072

*N/A is applied to non-summable metrics.

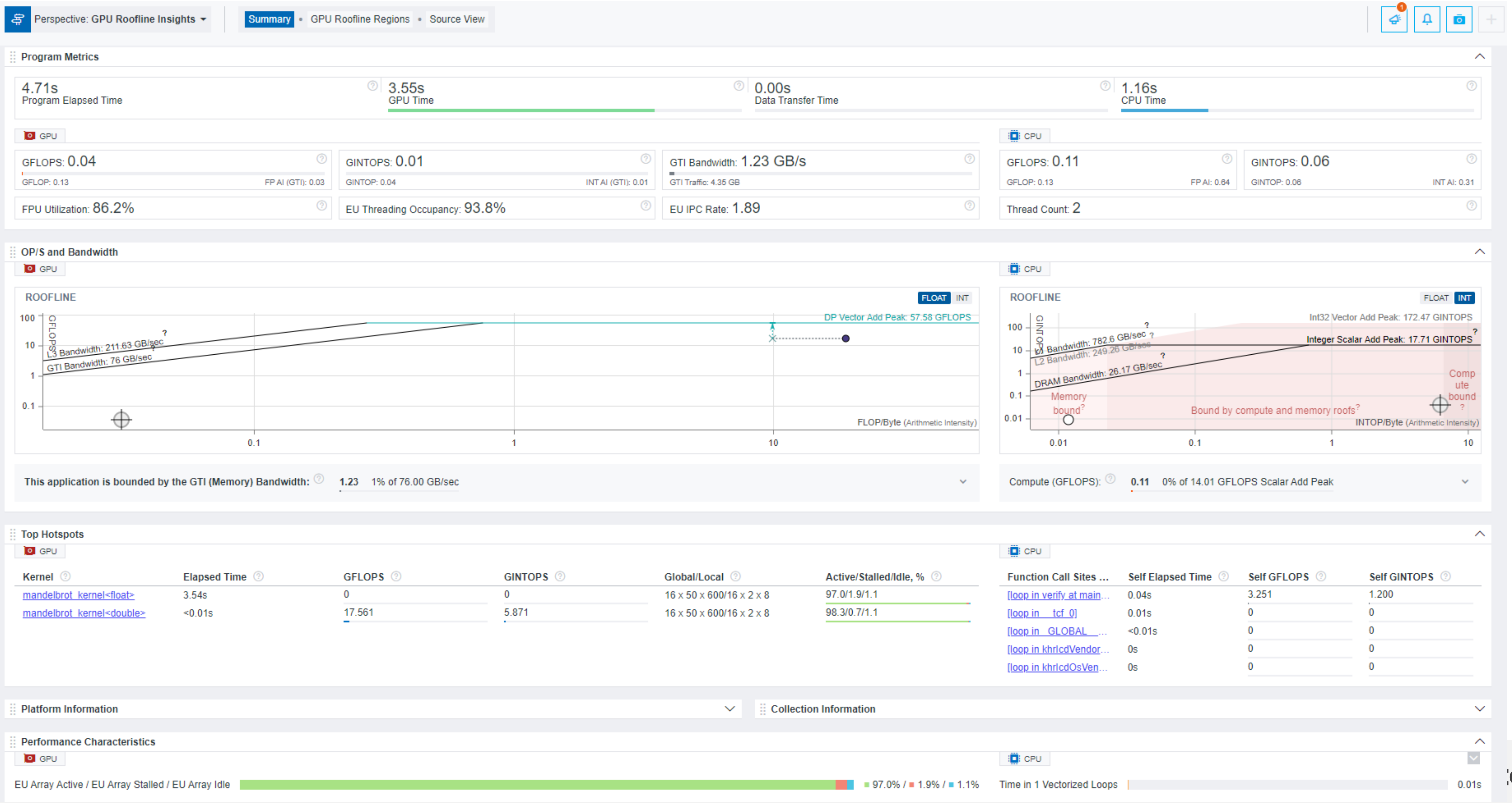
Graphics Tab
Instruction count – by Type





GPU Roofline Analysis with Intel® Advisor

Roofline Summary (GPU)

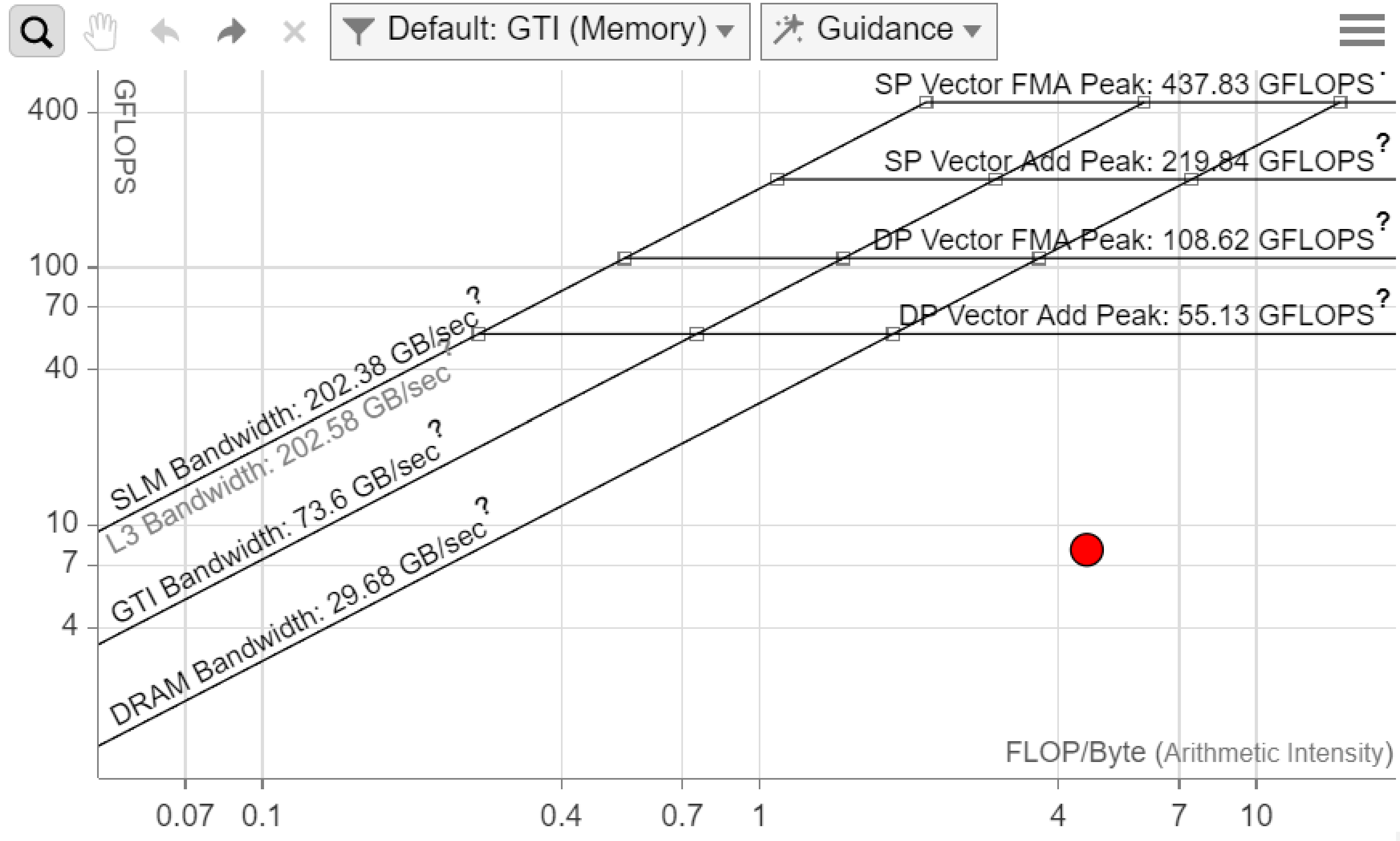


GPU Roofline differences to CPU Roofline

- The dots on the chart correspond to *OpenCL, OpenMP, Level Zero and SYCL kernels*, while in the CPU version, they correspond to individual loops.
- Some displayed information and controls (for example, thread/core count) are not relevant to GPU Roofline. For more information, see the table below.
- The GPU Roofline chart enables you to view arithmetic intensity of one kernel at multiple memory levels. To do so, double-click a dot representing this kernel or select it and press ENTER. The dots that appear on the Roofline chart correspond to different memory levels used to calculate arithmetic intensity. Hover over a dot to identify its arithmetic intensity. To show or hide certain dots from a chart, use the **Memory Level** drop-down filter.

[Measure GPU Performance Using GPU Roofline (intel.com)]

A GPU Roofline



Cookbook Example

- Intel® Advisor Performance Optimization Cookbook
- Analyze a SYCL Application with GPU Roofline (intel.com)

1. Prerequisites.
2. Run GPU Roofline Insights perspective.
3. View GPU Roofline results.
4. Examine the Application Performance on GPU.
5. Explore detailed GPU metrics with Intel Advisor Python* API.
6. Alternative steps.

Example Results: https://cdrdv2.intel.com/v1/dl/getContent/720943

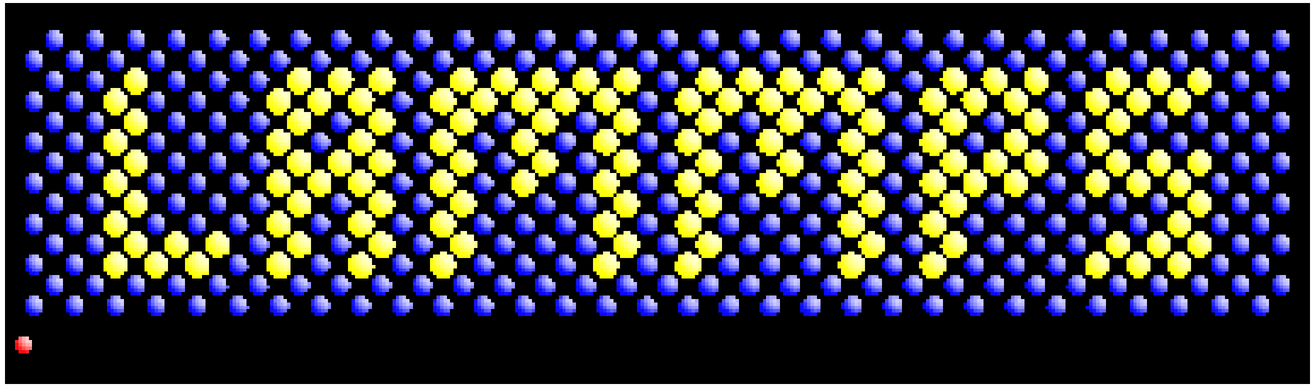
Lammps offloading

Stephen Blair-Chappell

stephen AT sbcnow DOT co DOT uk

Test Application

LAMMPS Molecular Dynamics Simulator



https://www.lammps.org/

Optimize Your GPU Application with the Intel® oneAPI Base Toolkit

The Workflow

Step 1: Choose Your GPU Hardware Access.

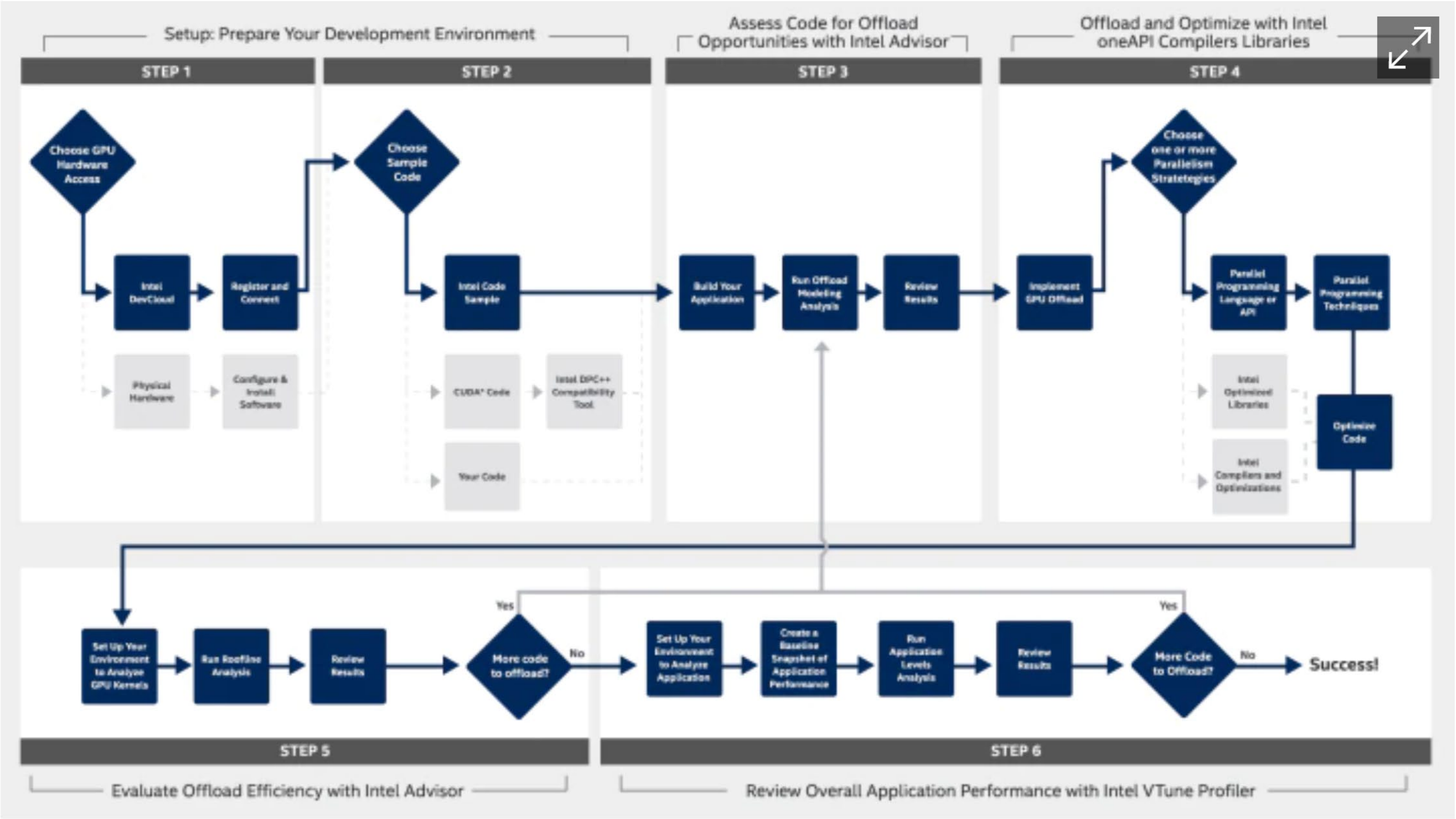
Step 2: Choose Your Sample Code.

Step 3: Assess Code for Offload Opportunities with Intel® Advisor.

Step 4: Offload and Optimize Code Using Intel® Compilers and Libraries.

Step 5: Evaluate Offload Efficiency with Intel Advisor.

Step 6: Review Overall Application Performance with Intel® VTune™ Profiler.



https://www.intel.com/content/www/us/en/developer/tools/oneapi/training/gpu-optimization-workflow.html

Machine 1 Spec - Workstation

```
u58345@s011-n001:~/ILDevCON/lammps-offload-OneAPI/LAB3/1-Model-Offload$ lscpu
Architecture:                    x86_64
CPU op-mode(s):                  32-bit, 64-bit
Byte Order:                      Little Endian
Address sizes:                   46 bits physical, 48 bits virtual
CPU(s):                          24
On-line CPU(s) list:             0-23
Thread(s) per core:              2
Core(s) per socket:              12
Socket(s):                       1
NUMA node(s):                    1
Vendor ID:                       GenuineIntel
CPU family:                      6
Model:                           85
Model name:                      Intel(R) Core(TM) i9-10920X CPU @ 3.50GHz
Stepping:                        7
CPU MHz:                         1200.315
CPU max MHz:                     4800.0000
CPU min MHz:                     1200.0000
BogoMIPS:                        6999.82
Virtualization:                  VT-x
L1d cache:                       384 KiB
L1i cache:                       384 KiB
L2 cache:                        12 MiB
L3 cache:                        19.3 MiB
NUMA node0 CPU(s):               0-23
Vulnerability Itlb multihit:     KVM: Mitigation: Split huge pages
Vulnerability L1tf:              Not affected
Vulnerability Mds:               Not affected
Vulnerability Meltdown:          Not affected
Vulnerability Spec store bypass: Mitigation; Speculative Store Bypass disabled via prctl and seccomp
Vulnerability Spectre v1:        Mitigation; usercopy/swapgs barriers and __user pointer sanitization
Vulnerability Spectre v2:        Mitigation; Enhanced IBRS, IBPB conditional, RSB filling
Vulnerability Srbds:             Not affected
Vulnerability Tsx async abort:   Mitigation; TSX disabled
Flags:                           fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acp
                                 i mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc art arch_per
                                 fmon pebs bts rep_good nopl xtopology nonstop_tsc cpuid aperfmperf pni pclmulqdq dtes64
                                  monitor ds_cpl vmx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid dca sse4_1 sse4_2 x2apic
                                  movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm 3dnowprefetch cp
                                 uid_fault epb cat_l3 cdp_l3 invpcid_single ssbd mba ibrs ibpb stibp ibrs_enhanced tpr_s
                                 hadow vnmi flexpriority ept vpid ept_ad fsgsbase tsc_adjust bmi1 avx2 smep bmi2 erms in
                                 vpcid cqm mpx rdt_a avx512f avx512dq rdseed adx smap clflushopt clwb intel_pt avx512cd
                                 avx512bw avx512vl xsaveopt xsavec xgetbv1 xsaves cqm_llc cqm_occup_llc cqm_mbm_total cq
                                 m_mbm_local dtherm ida arat pln pts hwp hwp_act_window hwp_epp hwp_pkg_req avx512_vnni
                                 md clear flush l1d arch capabilities
```

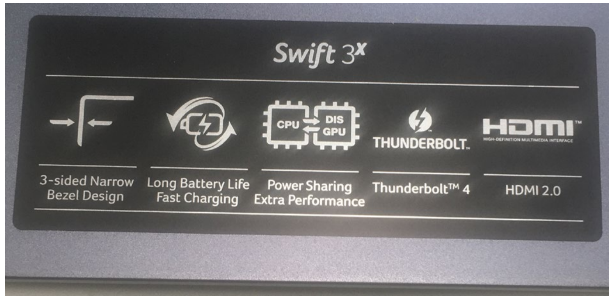
```
 Intel(R) Core(TM) i9-10920X CPU @ 3.50GHz
```

```
CPU(s):                          24
On-line CPU(s) list:             0-23
Thread(s) per core:              2
```

```
vx2 smep bmi2 erms in
wb intel_pt avx512cd
_llc cqm_mbm_total cq
_pkg_req avx512_vnni
```

Machine 2 Spec - Laptop

```
stephen@stephen-Swift-SF314-510G:~/dv/OneAPI-Package-1/LAMMPS$ lscpu
Architecture:                    x86_64
CPU op-mode(s):                  32-bit, 64-bit
Byte Order:                      Little Endian
Address sizes:                   39 bits physical, 48 bits virtual
CPU(s):                          8
On-line CPU(s) list:             0-7
Thread(s) per core:              2
Core(s) per socket:              4
Socket(s):                       1
NUMA node(s):                    1
Vendor ID:                       GenuineIntel
CPU family:                      6
Model:                           140
Model name:                      11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz
```

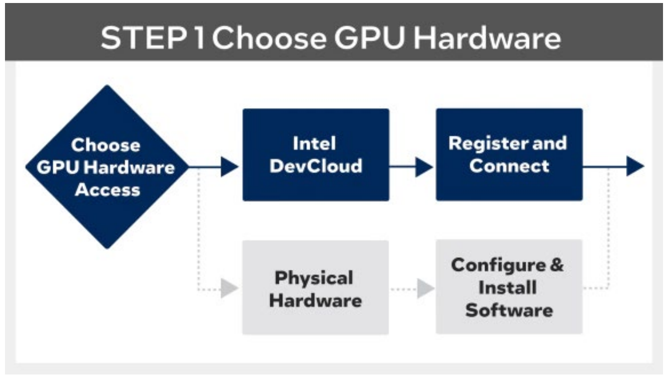


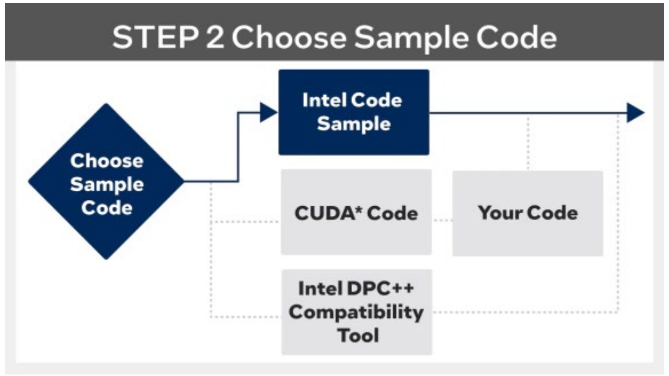

```
u58345@s011-n001:~/ILDevCON/lammps-offload-OneAPI/LAB3/1-Model-Offload$ lspci | grep VGA
1c:00.0 VGA compatible controller: Intel Corporation Device 4905 (rev 01)
6a:00.0 VGA compatible controller: Intel Corporation Device 4905 (rev 01)
```

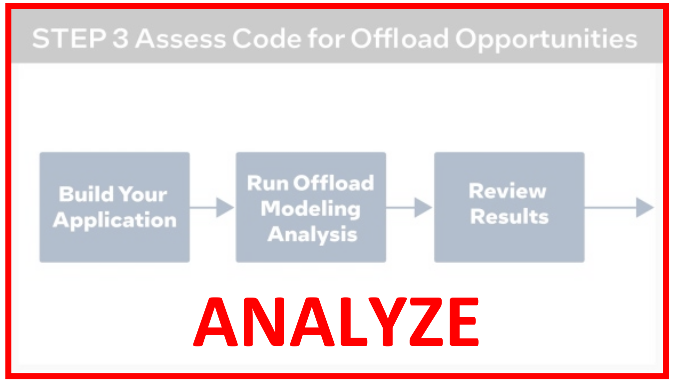
Graphics processor table

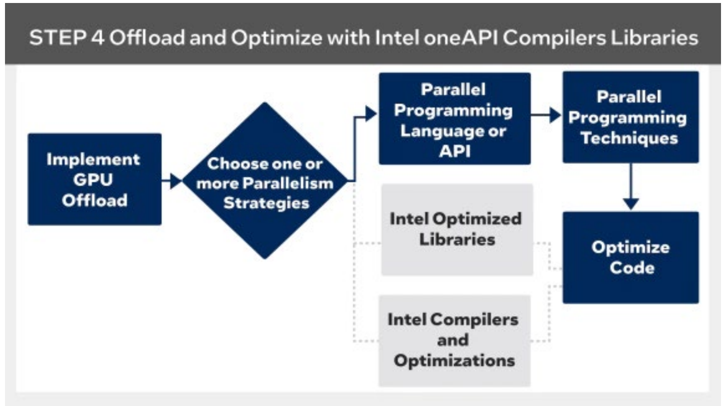
PCI IDs	Name	Architecture	Codename
4905	Intel® Iris® Xe MAX Graphics	Xe	DG1

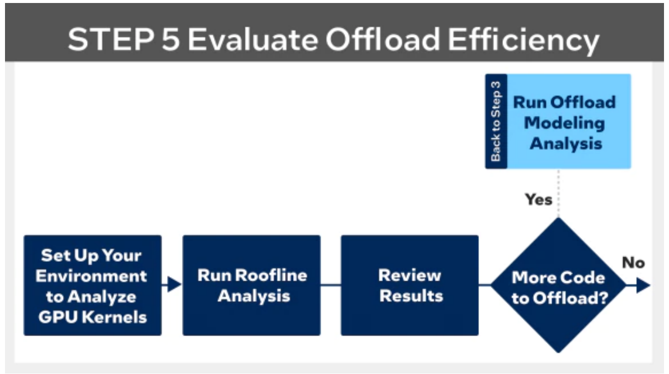
Analyze

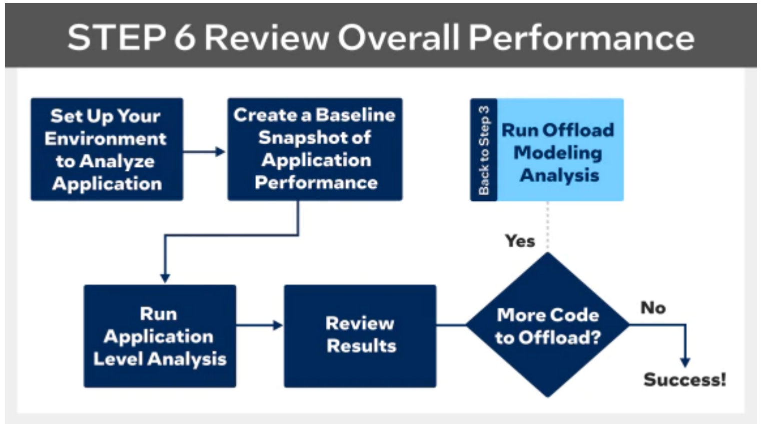






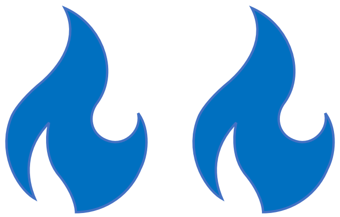






https://www.intel.com/content/www/us/en/developer/tools/oneapi/gpu-optimization-workflow.html

Goal of Analysis



1. Find the ‘hot spots’

2. if possible, predict benefit of offloading

Three Tools

APS – (Application Performance Snapshot)

Advisor - model offloading

VTune - Profiler

13

APS – (Application Performance Snapshot)

APS report

File | C:/Users/STEPHE~1/ONEDRI~1/DOCUME~1/MobaXterm/slash/stephenblaircha_desktopikul9t5/RemoteFiles/133800_2_1/aps_report_20210923_065855.html

Application Performance Snapshot

Application: *lmp*
Report creation date: *2021-09-23 06:58:02*
Number of ranks: *16*
Ranks per node: *16*
HW Platform: *Intel(R) microarchitecture code named Tigerlake*
Frequency: *2.42 GHz*
Logical Core Count per node: *8*
Collector type: *Event-based sampling driver,Event-based counting driver*

7.87 s CPU
Elapsed Time
0.06 s GPU

1 IPC Rate | 0 SP GFLOPS | 1.43 DP GFLOPS | 2.88 GHz Average CPU Frequency

Your application may underutilize the GPU.

Run a GPU Offload (Preview) or a GPU Compute/Media Hotspots (Preview) analysis with VTune Profiler to discover how to better utilize the GPU.

	Current run	Target	Tuning Potential
MPI Time	54.64%	<10%	
Memory Stalls	7.2%	<20%	
Vectorization	0%	>70%	
GPU Utilization when Busy	3.6%	>80%	

GPU Utilization when Busy

3.6%

EU State	% of EUs
Active	3.6%
Idle	83%
Stalled	13.4%

GPU Occupancy
9.1% of Peak Value

Memory Footprint

Resident
113.19 MB

Resident per Node
1811 MB

Virtual
2646.12 MB

Virtual Per Node
42338 MB

MPI Time

4.3 s
54.64% of Elapsed Time

MPI Imbalance
2.38 s
30.28% of Elapsed Time

TOP 5 MPI Functions	% of Elapsed Time
MPI_Bcast	28.84%
MPI_Wait	9.86%
MPI_Allreduce	7.87%
MPI_Init	5.4%
MPI_Sendrecv	1.69%

Memory Stalls

7.2% of Pipeline Slots

Cache Stalls
19.2% of Cycles

DRAM Stalls
7.5% of Cycles

DRAM Bandwidth	
Average	9 GB/s
Peak	8.9 GB/s
Bound	0%

Vectorization

0%

Instruction Mix

SP FLOPs
0% of uOps

DP FLOPs
6.9% of uOps
Packed:
0% from DP FP
128-bit:
0%
256-bit:
0%
512-bit:
0%
Scalar:
100% from DP FP

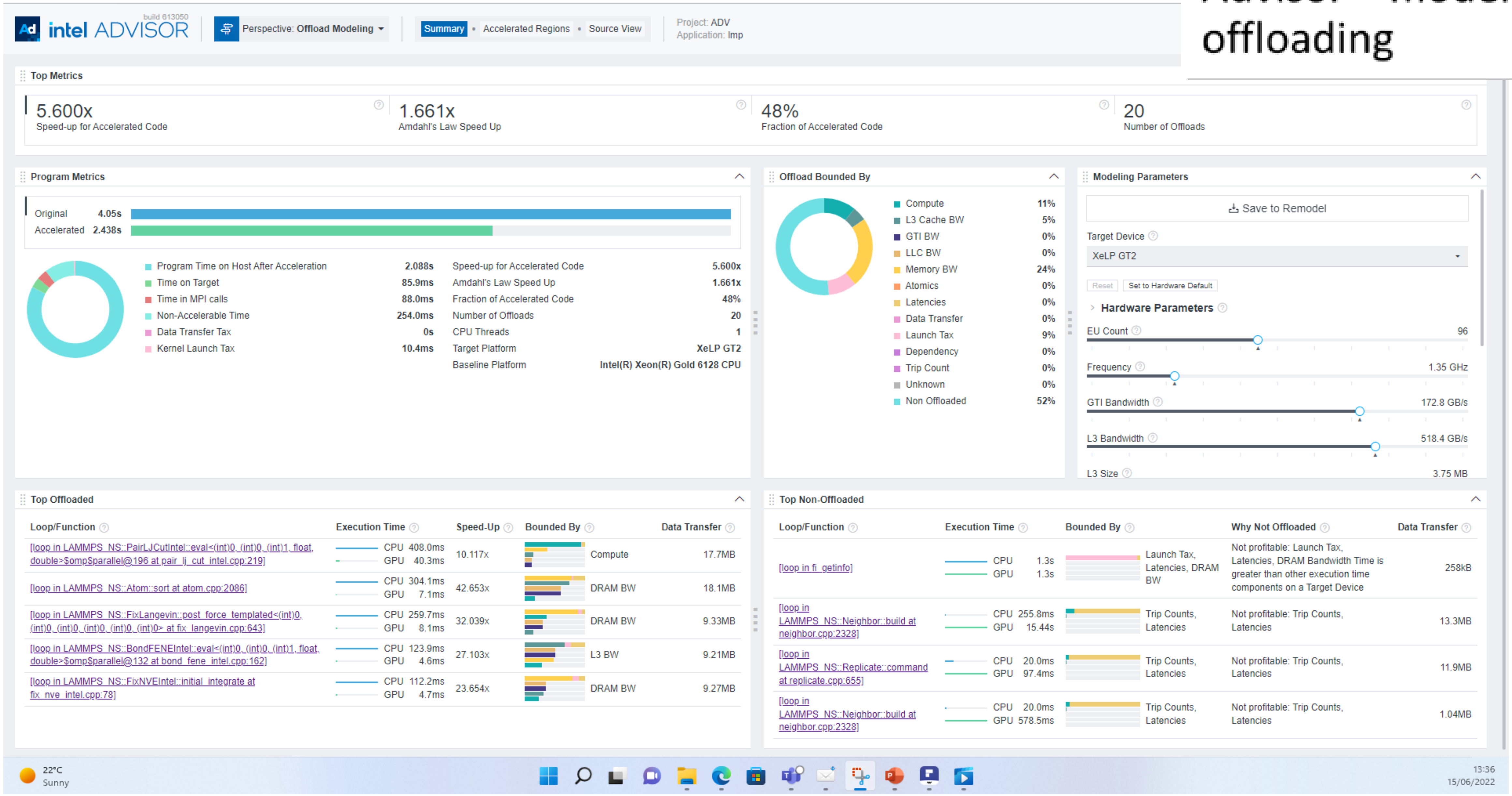
Non-FP
93.1% of uOps

FP Arith/Mem Rd Instr. Ratio
0.21

FP Arith/Mem Wr Instr. Ratio
0.73

LAB1

Advisor - model offloading



Steps to Offload Projection with Advisor

1. Run a **Survey**: get a list of hotspots

   ```
   advisor -collect survey …
   ```

- Sampling
- Binary Static Analysis
- Compiler & debug info

2. Run a **Trip Count**: count loop iteration

   ```
   advisor -collect=tripcounts -target-device=gen9_gt2 .
   ```

- Trip count
- Cache simulation

3. Perform a **dependency analysis** [optional for quick modelling]

   ```
   advisor -collect dependencies . . .
   ```

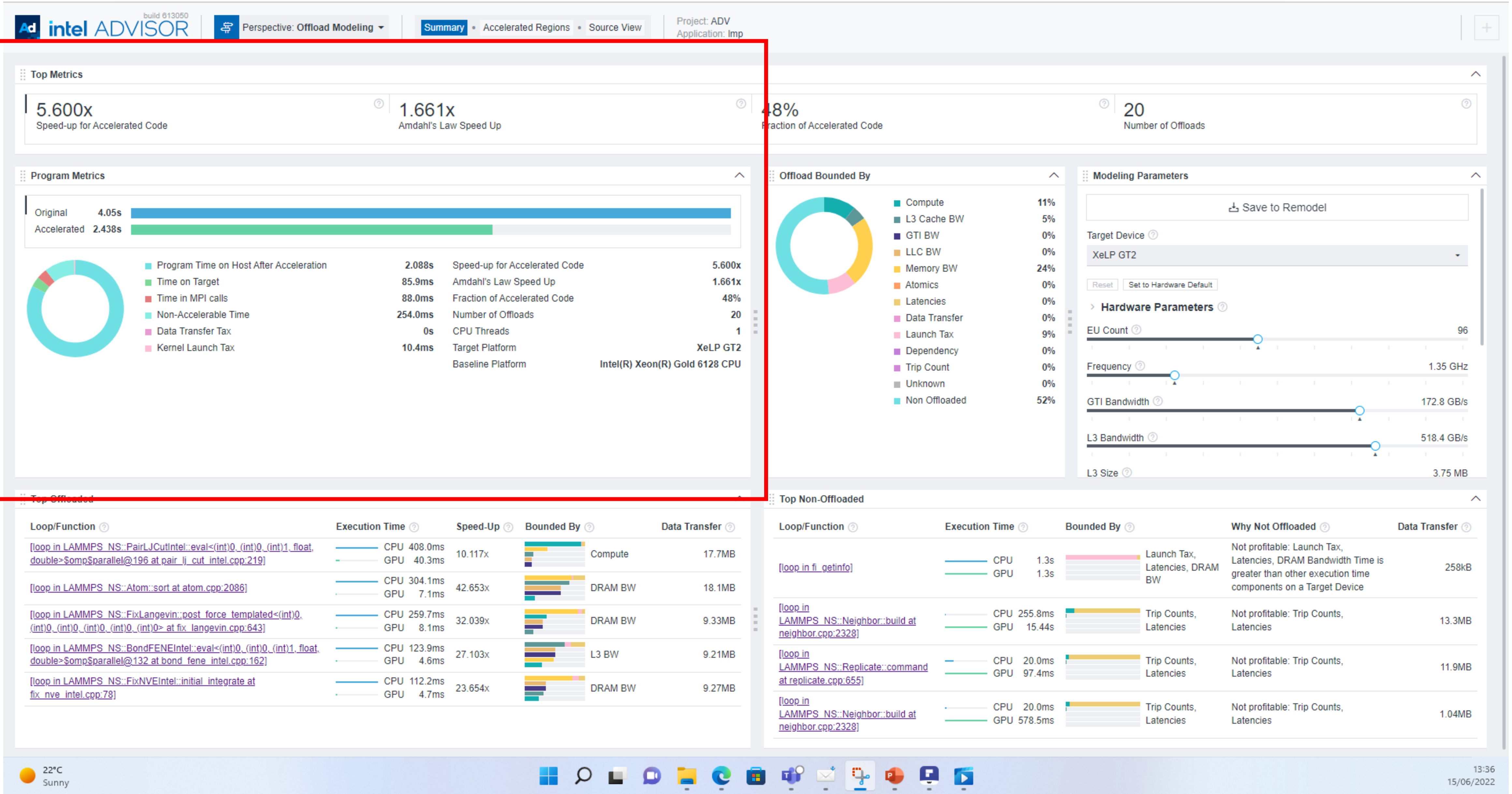
- Check memory accesses
- Loop selection heuristic

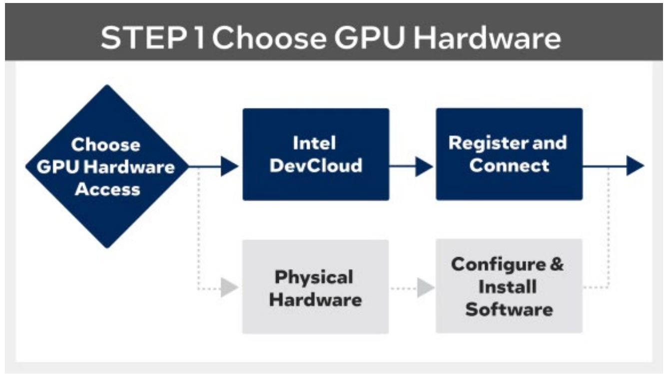
4. **Model** the Performance

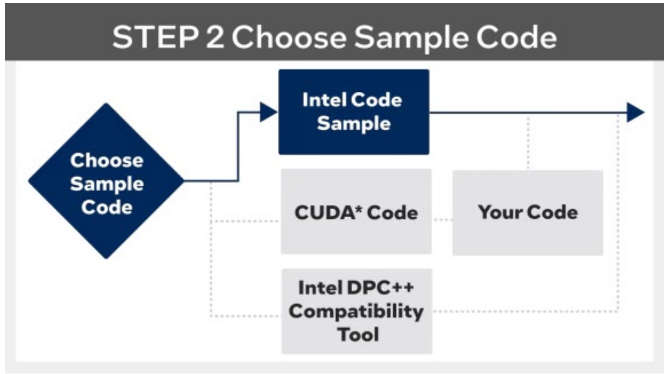
   ```
   advisor -collect projection -no-assume-dependencies . . .
   ```

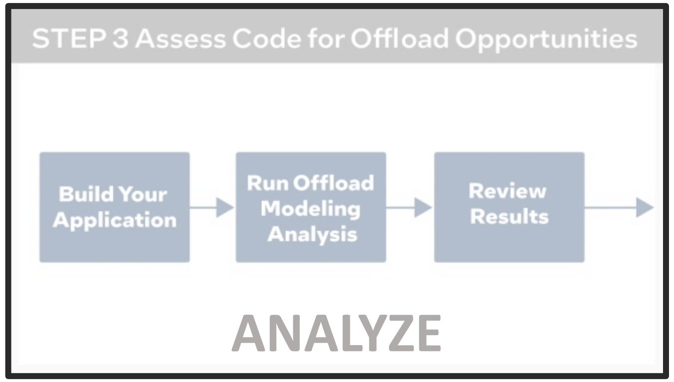
- Generate HTML report

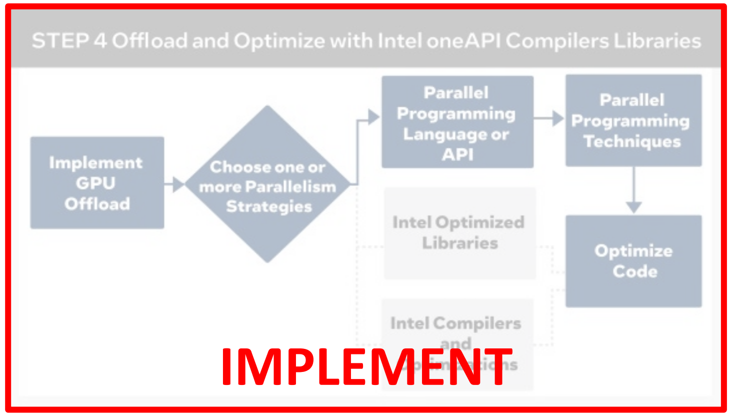
Expensive Steps

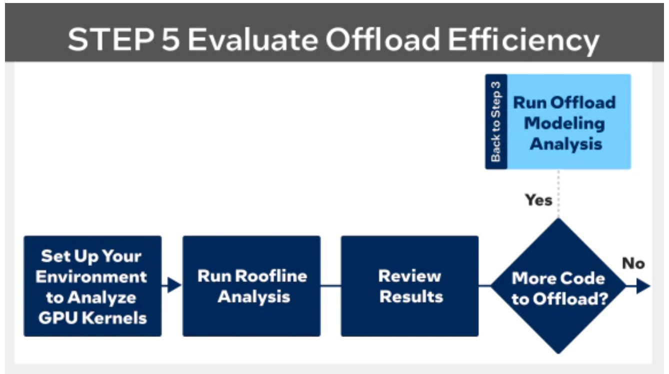

intel ADVISOR
build 613050
Perspective: Offload Modeling
Summary
Accelerated Regions
Source View
Project: ADV
Application: lmp
Top Metrics
5.600x
Speed-up for Accelerated Code
1.661x
Amdahl's Law Speed Up
48%
Fraction of Accelerated Code
20
Number of Offloads
Program Metrics
Original 4.05s
Accelerated 2.438s
Program Time on Host After Acceleration 2.088s
Time on Target 85.9ms
Time in MPI calls 88.0ms
Non-Accelerable Time 254.0ms
Data Transfer Tax 0s
Kernel Launch Tax 10.4ms
Speed-up for Accelerated Code 5.600x
Amdahl's Law Speed Up 1.661x
Fraction of Accelerated Code 48%
Number of Offloads 20
CPU Threads 1
Target Platform XeLP GT2
Baseline Platform Intel(R) Xeon(R) Gold 6128 CPU
Offload Bounded By
Compute 11%
L3 Cache BW 5%
GTI BW 0%
LLC BW 0%
Memory BW 24%
Atomics 0%
Latencies 0%
Data Transfer 0%
Launch Tax 9%
Dependency 0%
Trip Count 0%
Unknown 0%
Non Offloaded 52%
Modeling Parameters
Save to Remodel
Target Device
XeLP GT2
Reset
Set to Hardware Default
Hardware Parameters
EU Count 96
Frequency 1.35 GHz
GTI Bandwidth 172.8 GB/s
L3 Bandwidth 518.4 GB/s
L3 Size 3.75 MB
Top Offloaded
Loop/Function
Execution Time
Speed-Up
Bounded By
Data Transfer
[loop in LAMMPS_NS::PairLJCutIntel::eval<(int)0, (int)0, (int)1, float, double>ompparallel@196 at pair_lj_cut_intel.cpp:219]
CPU 408.0ms GPU 40.3ms 10.117x Compute 17.7MB
[loop in LAMMPS_NS::Atom::sort at atom.cpp:2086]
CPU 304.1ms GPU 7.1ms 42.653x DRAM BW 18.1MB
[loop in LAMMPS_NS::FixLangevin::post_force_templated<(int)0, (int)0, (int)0, (int)0, (int)0, (int)0> at fix_langevin.cpp:643]
CPU 259.7ms GPU 8.1ms 32.039x DRAM BW 9.33MB
[loop in LAMMPS_NS::BondFENEIntel::eval<(int)0, (int)0, (int)1, float, double>ompparallel@132 at bond_fene_intel.cpp:162]
CPU 123.9ms GPU 4.6ms 27.103x L3 BW 9.21MB
[loop in LAMMPS_NS::FixNVEIntel::initial_integrate at fix_nve_intel.cpp:78]
CPU 112.2ms GPU 4.7ms 23.654x DRAM BW 9.27MB
Top Non-Offloaded
Loop/Function
Execution Time
Bounded By
Why Not Offloaded
Data Transfer
[loop in fi_getinfo]
CPU 1.3s GPU 1.3s Launch Tax, Latencies, DRAM BW
Not profitable: Launch Tax, Latencies, DRAM Bandwidth Time is greater than other execution time components on a Target Device 258kB
[loop in LAMMPS_NS::Neighbor::build at neighbor.cpp:2328]
CPU 255.8ms GPU 15.44s Trip Counts, Latencies
Not profitable: Trip Counts, Latencies 13.3MB
[loop in LAMMPS_NS::Replicate::command at replicate.cpp:655]
CPU 20.0ms GPU 97.4ms Trip Counts, Latencies
Not profitable: Trip Counts, Latencies 11.9MB
[loop in LAMMPS_NS::Neighbor::build at neighbor.cpp:2328]
CPU 20.0ms GPU 578.5ms Trip Counts, Latencies
Not profitable: Trip Counts, Latencies 1.04MB
22°C
Sunny
13:36
15/06/2022

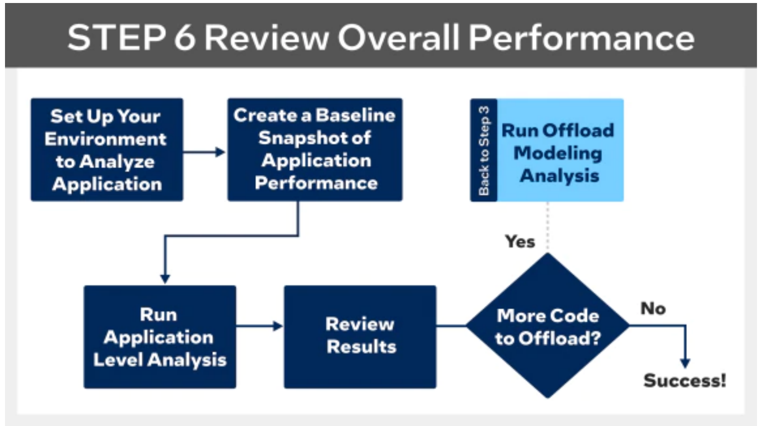












https://www.intel.com/content/www/us/en/developer/tools/oneapi/gpu-optimization-workflow.html

TWO TYPES OF COMPILER

Deprecated and will be removed from product release in the second half of 2023

- **CLASSIC** **[Same as was in Parallel Studio]**
  - icc
  - icpc
  - ifort

Offloading not supported

LLVM based [Totally new compilers]

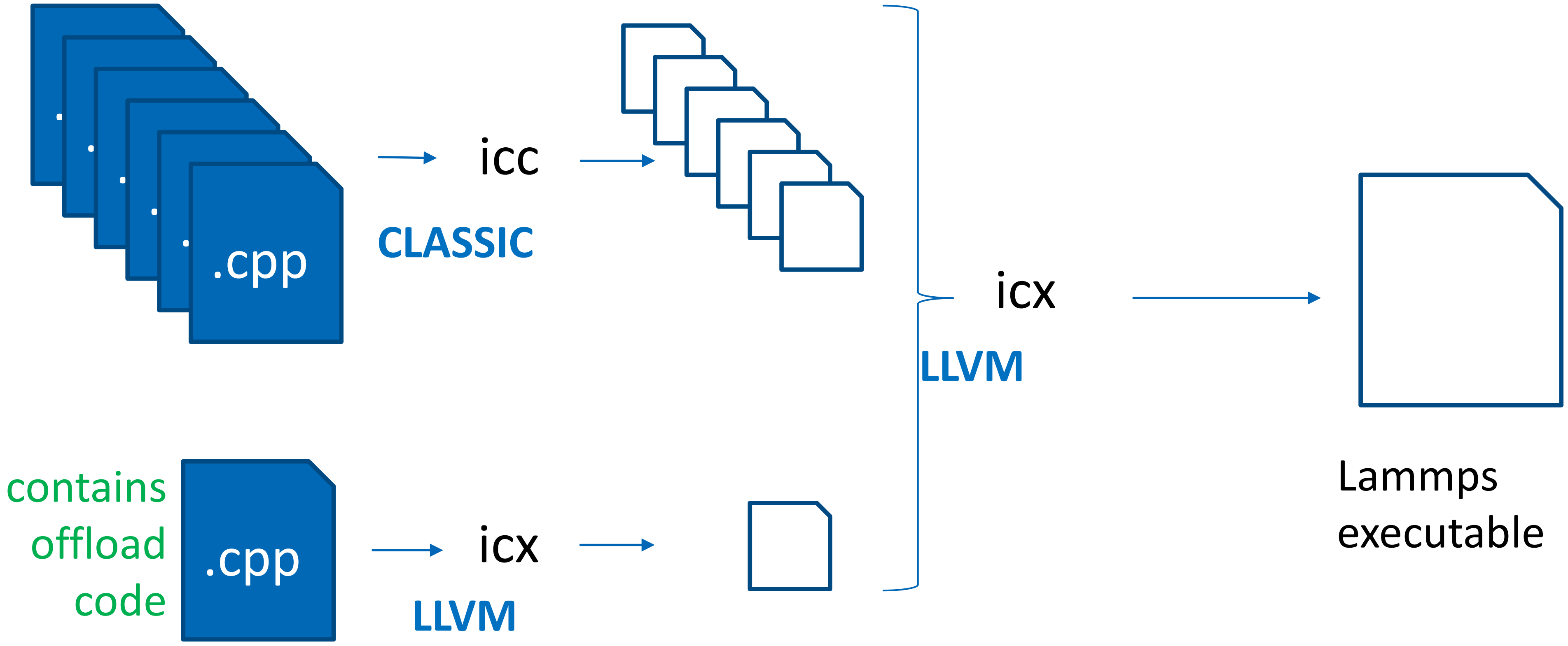
- icx
- dpcpp
- ifx

Offloading supported

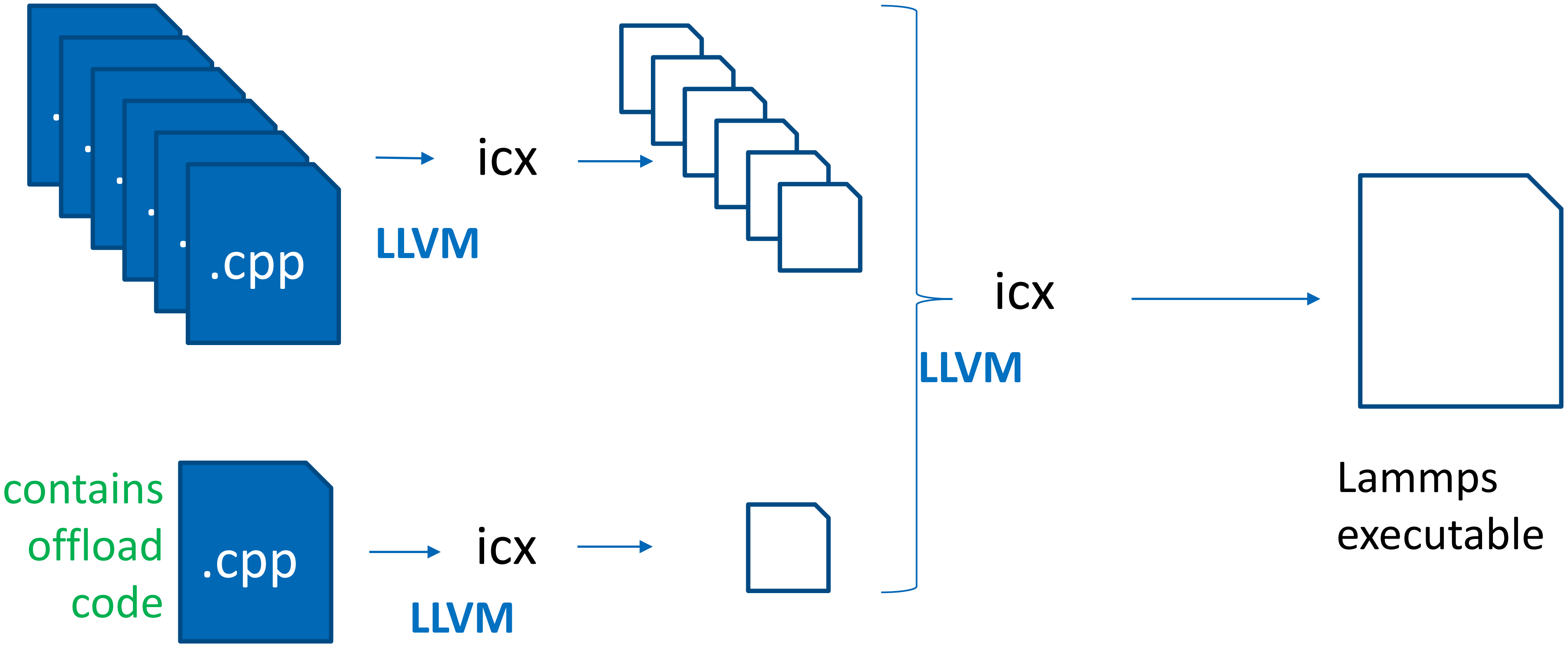
All objects are binary compatible

One approach . . .

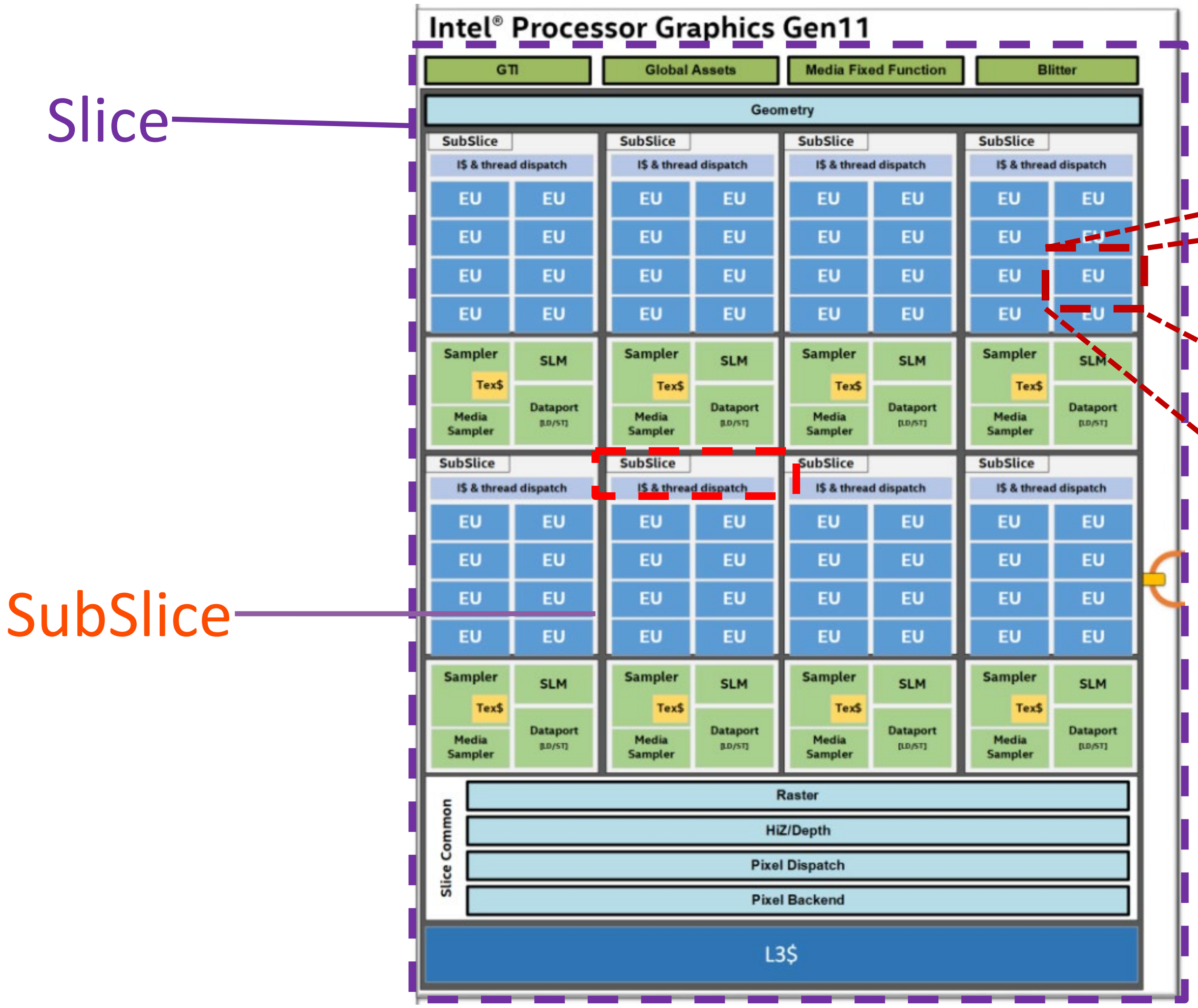
NB: icc deprecated mid 2023



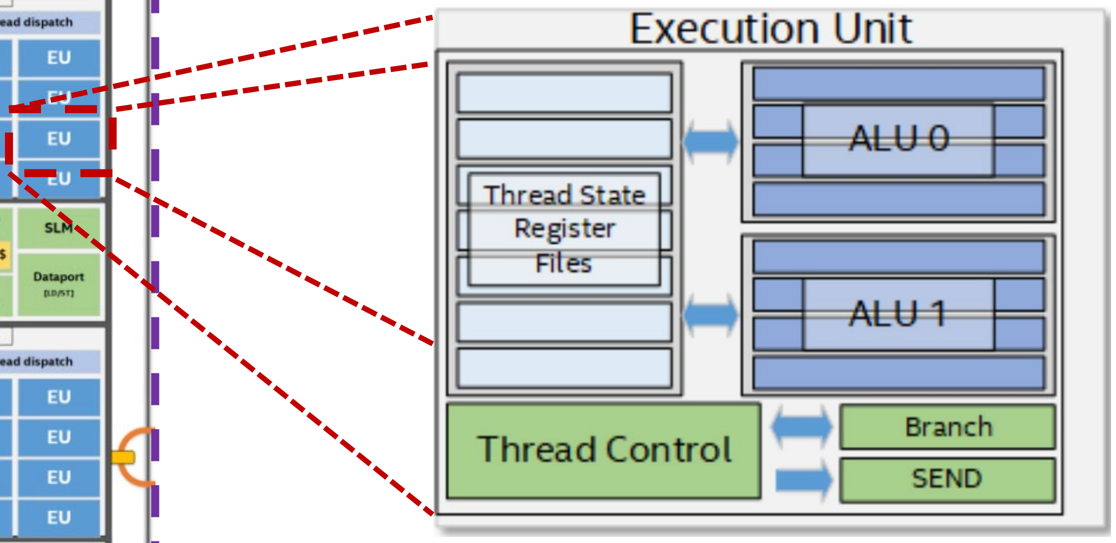
. . . or alternatively



GPU Architecture



Execution Unit with SIMD ALUs



GPU Architecture


```
#pragma omp target                    parallel for
```

Sends Code to target
but only on one sub-slice

OpenMP GPU Offload and OpenMP Constructs

- OpenMP GPU offload support all “normal” OpenMP constructs
  - E.g. parallel, for/do, barrier, sections, tasks, etc.
  - Not every construct will be useful
- Full threading model outside of a single GPU subslice **not** supported
  - No synchronization among subslices
  - No coherence and memory fence between among subslice L1 caches

GPU Architecture


```
#pragma omp target teams                parallel for
```

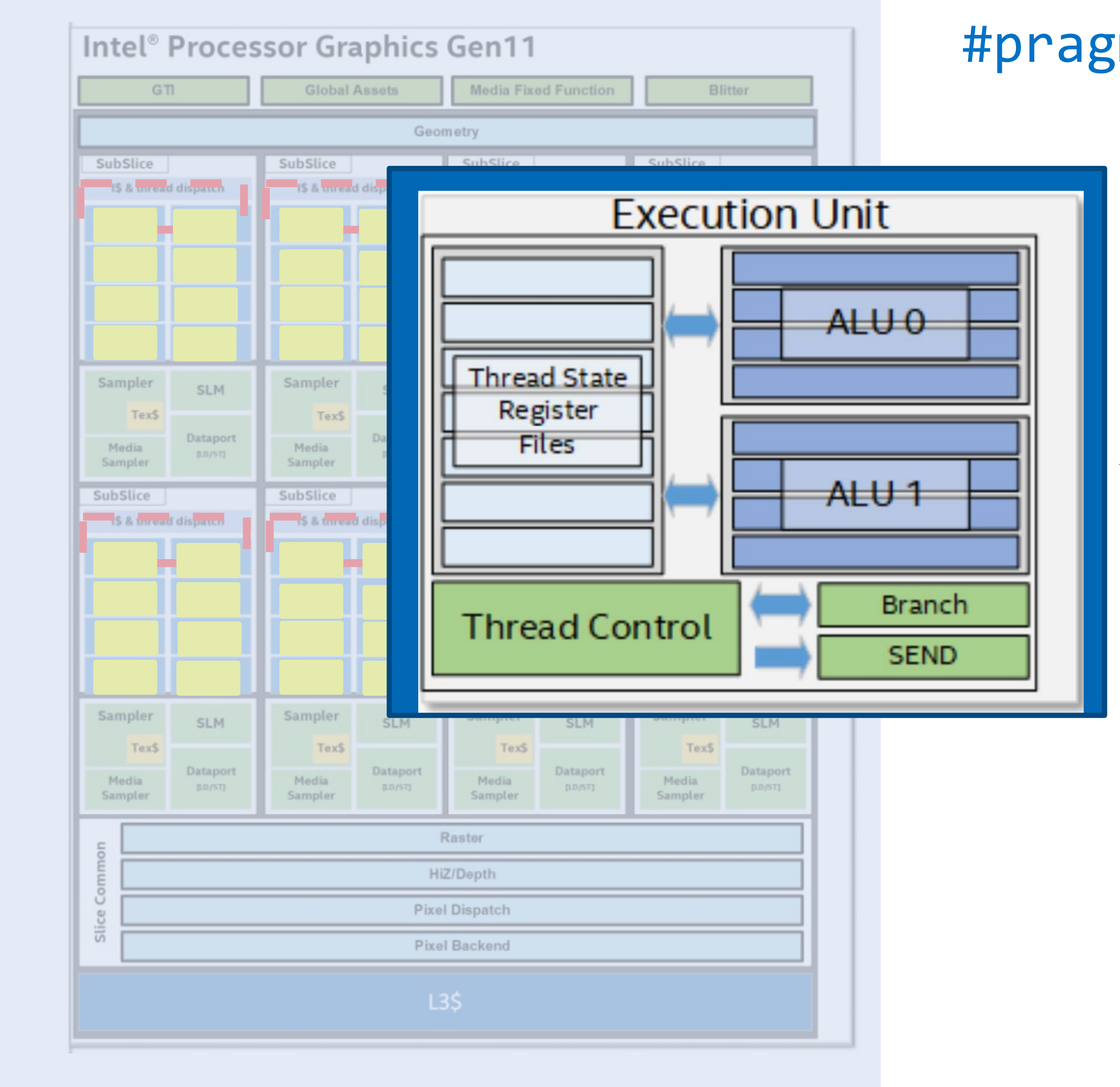
Share code across subslices
but only on one EU

GPU Architecture


```
#pragma omp target teams distribute parallel for
```

Distributes loop iterations across all EUs

GPU Architecture



`#pragma omp target teams distribute simd parallel`

Uses SIMD wide registers in execution units

Experiment in Lammps -Target

```
#if 1
      #pragma omp target map(to:ilist[iifrom:iito]) \
                         map(to:x[x_min:x_max]) \
                         map(tofrom:f[f_min:f_max]) \
                         map(from:ev_global[0:7]) \
                         map (tofrom:lj1,lj2,lj3,lj4,offset)
      #pragma omp teams distribute parallel for
 #endif
  // ------------------------ END SBC code -------------------


      for (int ii = iifrom; ii < iito; ii += iip) {
        const int i = ilist[ii];
        int itype, ptr_off;
        const FC_PACKED1_T * _noalias ljc12oi;
        const FC_PACKED2_T * _noalias lj34i;
        if (!ONETYPE) {
          itype = x[i].w;
          ptr_off = itype * ntypes;
```


intel

Experiment in Lammps –Indirect Indexes

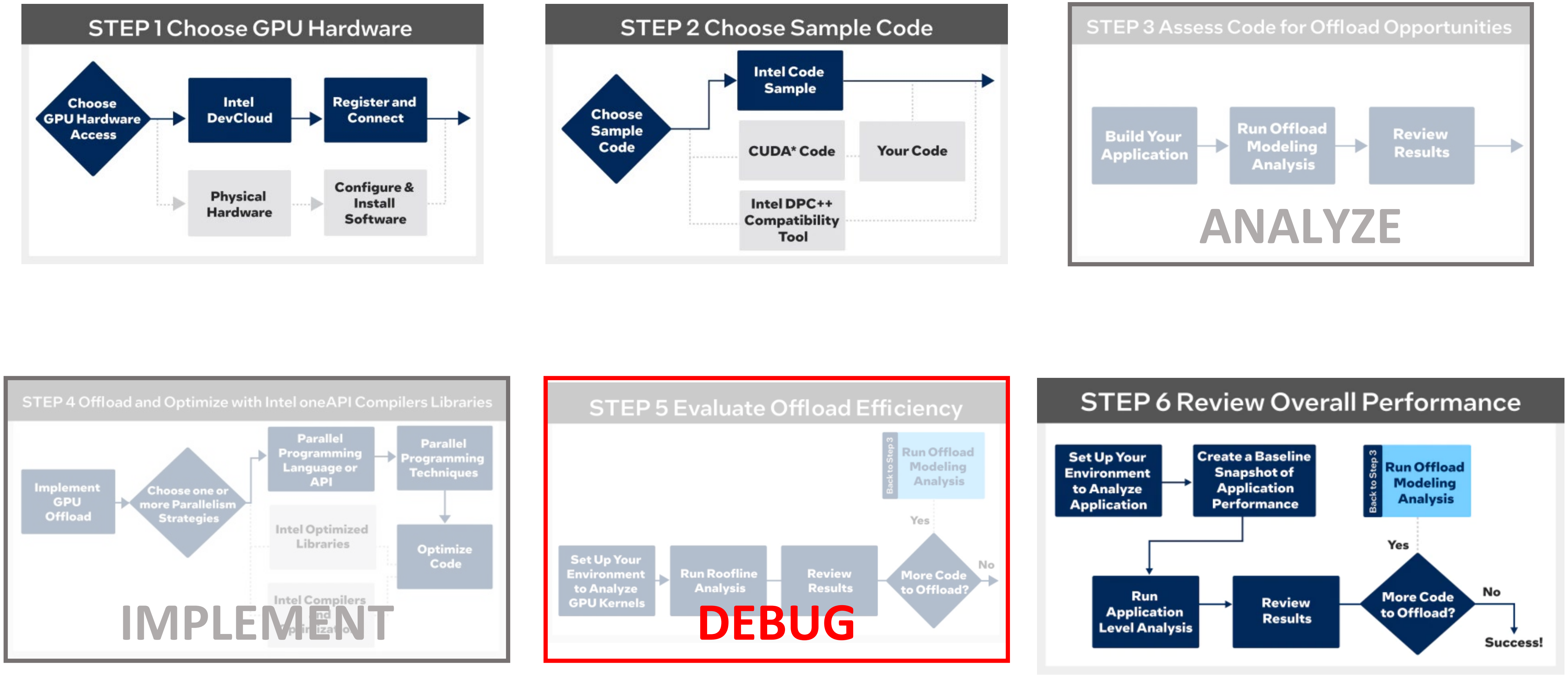
Start here   *pair_lj_cut_intel.cpp

```
// ---------------------- SBC code ------------------
  #if 1
     // find min an max of indirect indexes
     int i_min=0;
     int i_max=0;
     int jlist_min=0;
     int jlist_max=0;
     int j_min =0;
     int j_max=0;
     int x_min=0;
     int x_max=0;
     int f_min=0;
     int f_max=0;

     for (int ii = iifrom; ii < iito; ii += iip) {
       int i=0, itype=0, sbindex=0;

       // get i_min and i_max
       i = ilist[ii];
       i_min=std::min(i,i_min);
       i_max=std::max(i,i_max);


        const int * _noalias const jlist = firstneigh[i];
        int jnum = numneigh[i];

       // we also need the j_min and j_max  as this is used in x[j]
       for (int jj = 0; jj < jnum; jj++) {
         int j=0, jtype=0, sbindex=0;
         if (!ONETYPE) {
           sbindex = jlist[jj] >> SBBITS & 3;
           j = jlist[jj] & NEIGHMASK;
          } else
            j = jlist[jj];

          j_min=std::min(j,j_min);
          i_max=std::max(j,j_max);
        }
      } // ii
      x_min = f_min = std::min(i_min,j_min);
      x_max = f_max = std::max(i_max,j_max);

     #if 0
     printf("x_min: %d, x_max: %d ",x_min,x_max);
     #endif
  #endif

#if 1
     #pragma omp target map(to:ilist[iifrom:iito]) \
```

evCon/lammps-offload-OneAPI/LAB2/6-ICX-PATCHED/pair_lj_cut_intel.cpp   C/C++

```
// we also need the j_min and j_max  as this
for (int jj = 0; jj < jnum; jj++) {
  int j=0, jtype=0, sbindex=0;
  if (!ONETYPE) {
    sbindex = jlist[jj] >> SBBITS & 3;
    j = jlist[jj] & NEIGHMASK;
   } else
     j = jlist[jj];

   j_min=std::min(j,j_min);
   i_max=std::max(j,j_max);
 }
} // ii
x_min = f_min = std::min(i_min,j_min);
x_max = f_max = std::max(i_max,j_max);
```

Debug



https://www.intel.com/content/www/us/en/developer/tools/oneapi/gpu-optimization-workflow.html

oneAPI Debug Tools (intel.com)

Tool	Description
Environment variables	Gather diagnostic information from the OpenMP and SYCL runtimes at program execution with no modifications to your program.
Onetrace	From Profiling Tools Interfaces for GPU (PTI for GPU). Used to debug backend errors and for performance profiling on both the host and device.
Intercept Layer for OpenCL™ Applications	Used to debug backend errors and for performance profiling on both the host and device (has wider functionality comparing with onetrace).
Intel® Distribution for GDB*	Used for source-level debugging of the application, typically to inspect logical bugs, on the host and any devices you are using (CPU, GPU, FPGA emulation).
Intel® Inspector	locate and debug memory and threading problems, including those that can cause offloading to fail.
In-application debugging	printf; looking at output of apps etc, etc
Intel® Advisor	Use to ensure Fortran, C, C++, OpenCL™, and SYCL applications realize full performance potential on modern processors.
Intel® VTune ™ Profiler	Use to gather performance data either on the native system or on a remote system.

Example – offload on Laptop

```
====================================================================================
===
=   BAD TERMINATION OF ONE OF YOUR APPLICATION PROCESSES
=   RANK 5 PID 23953 RUNNING AT stephen-Swift-SF314-510G
=   KILLED BY SIGNAL: 9 (Killed)
====================================================================================
===
```

Runtime FAILS

```
export LIBOMPTARGET_DEBUG=4
export LIBOMPTARGET_INFO=-1
```

Get Visibility

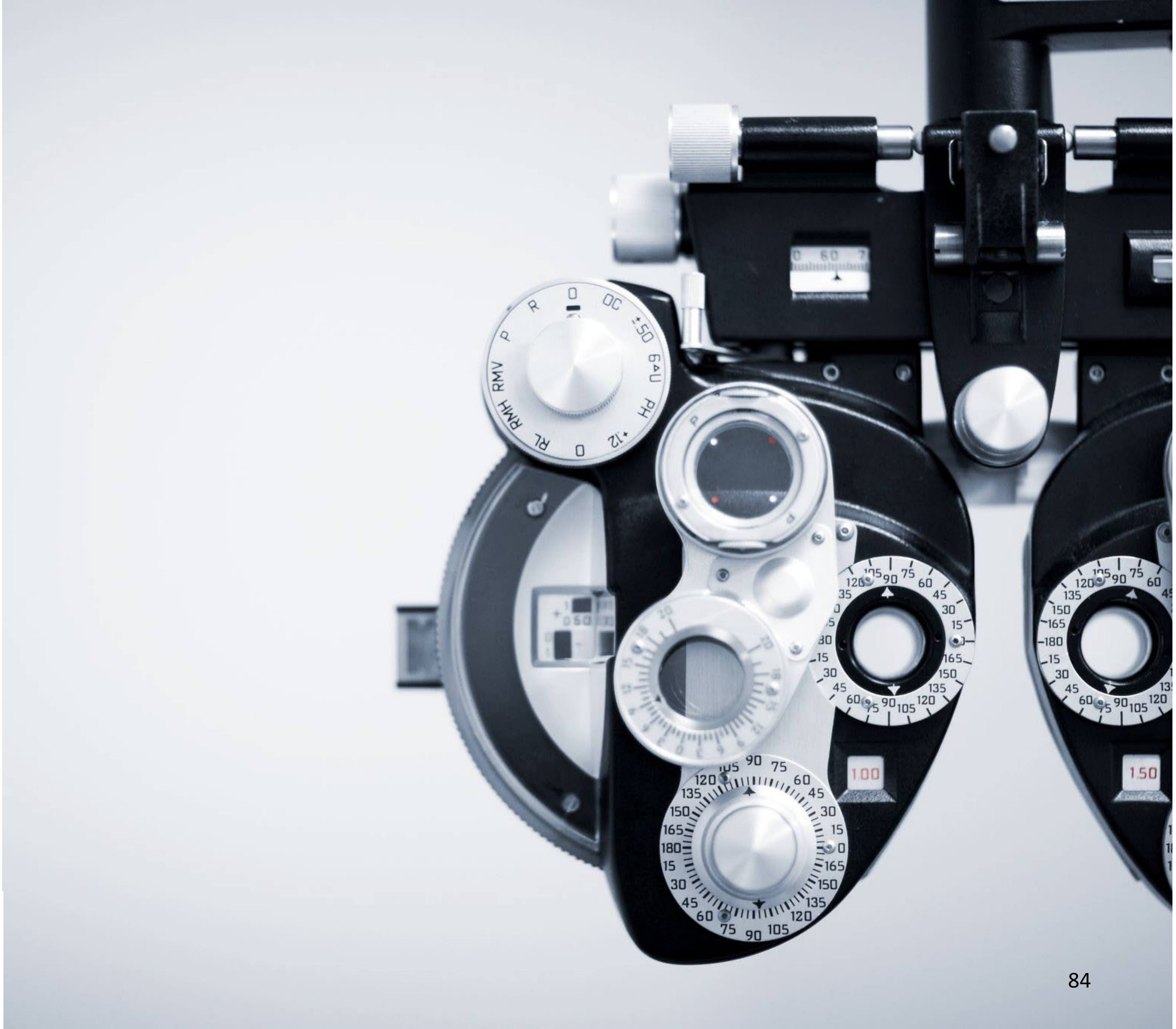
```
error: double type is not supported on this platform
in file: pair_lj_cut_intel.cpp:311
```

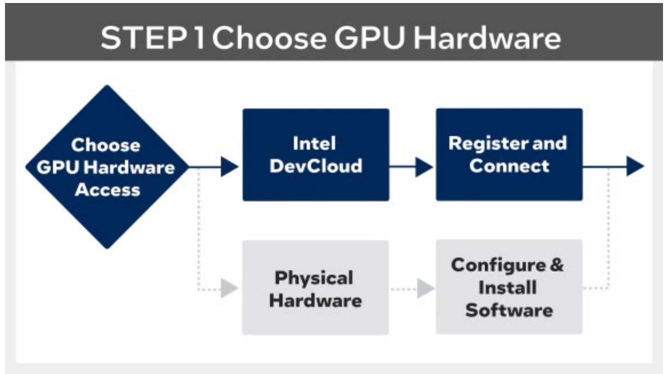
Get Visibility

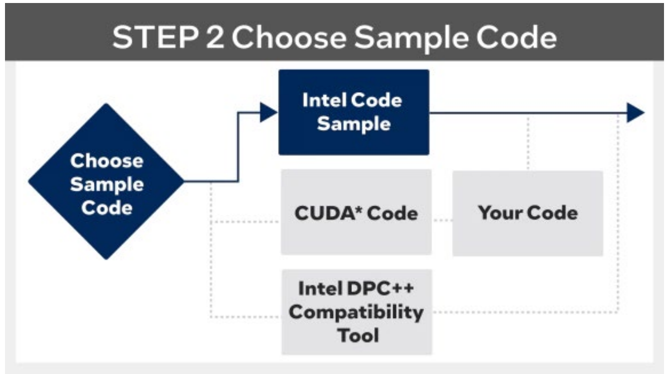
```
export OverrideDefaultFP64Settings=1
export IGC_EnableDPEmulation=1
```

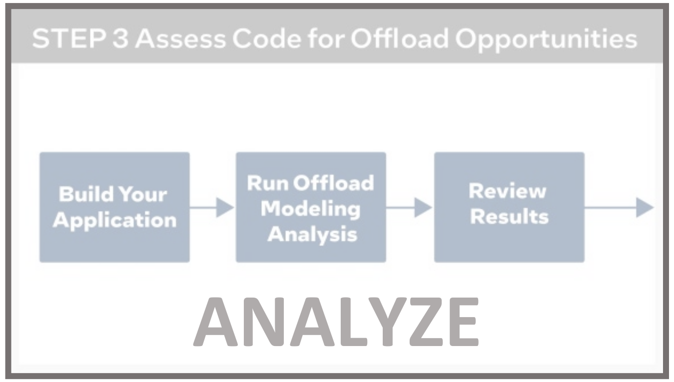
The Solution

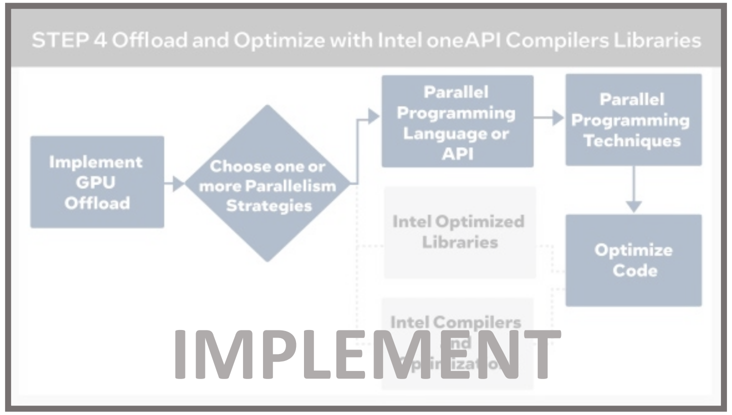
Tune

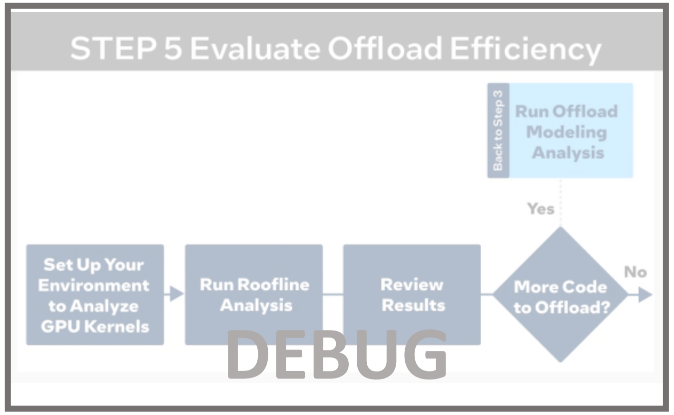


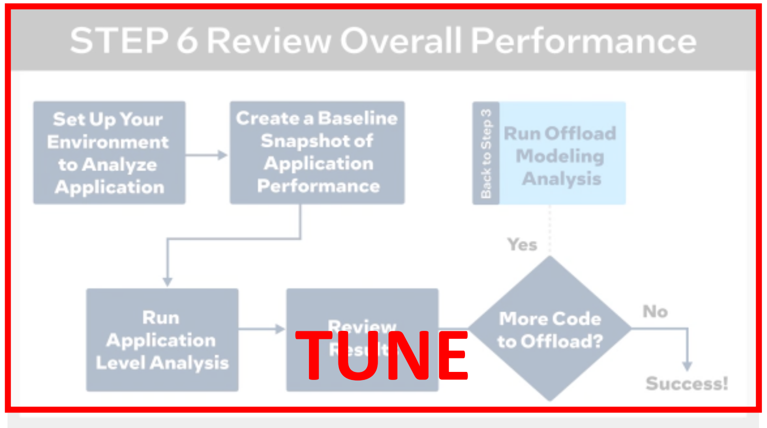












https://www.intel.com/content/www/us/en/developer/tools/oneapi/gpu-optimization-workflow.html

Run and collect VTune™ data

```
vtune -collect gpu_hotspots -result-dir vtune_data a.out
```

Various types of profiling data can be collected like `hotspots`, `memory-consumption`, `memory-access`, `threading` ...

Use the command line help to find out more:

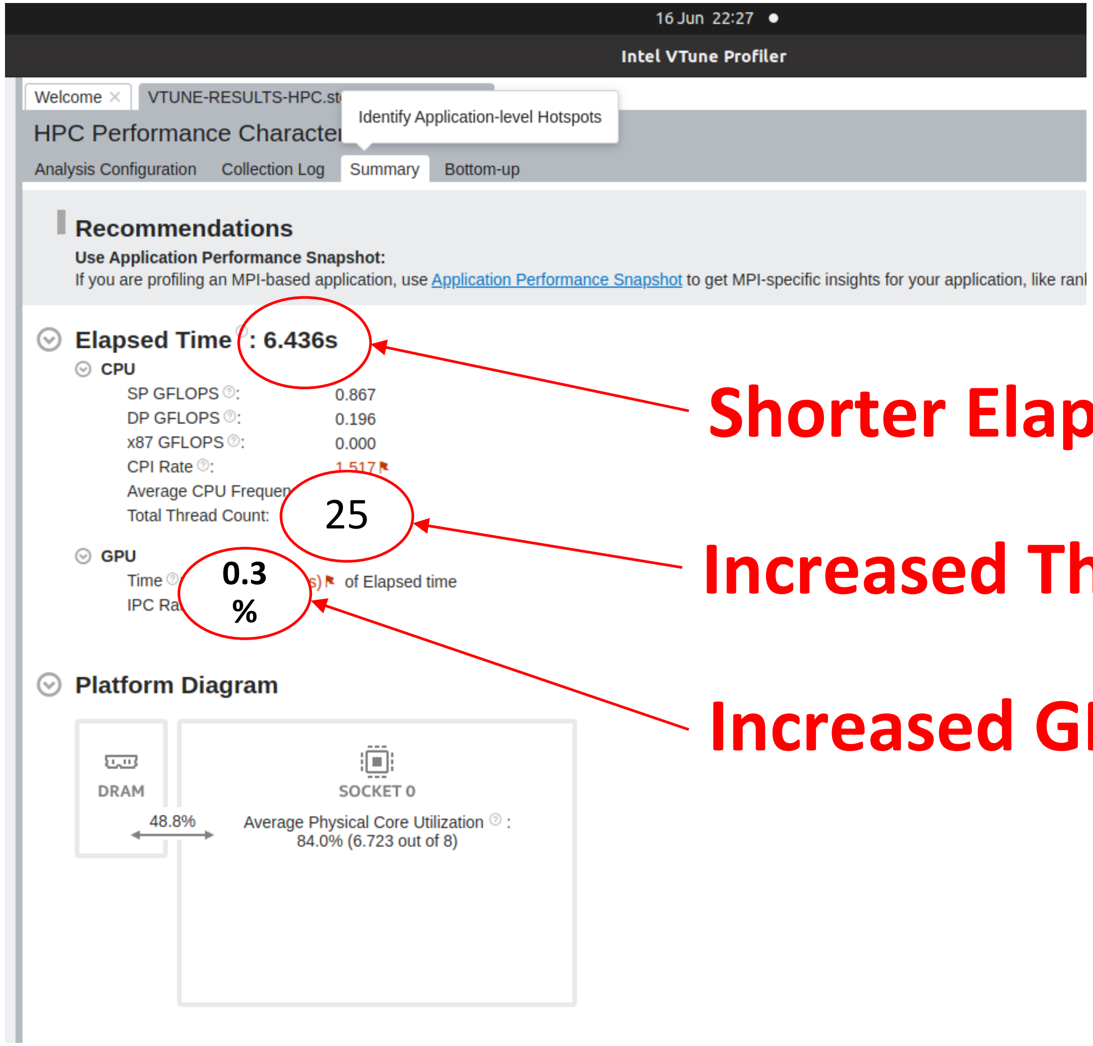
```
vtune --help -collect
```

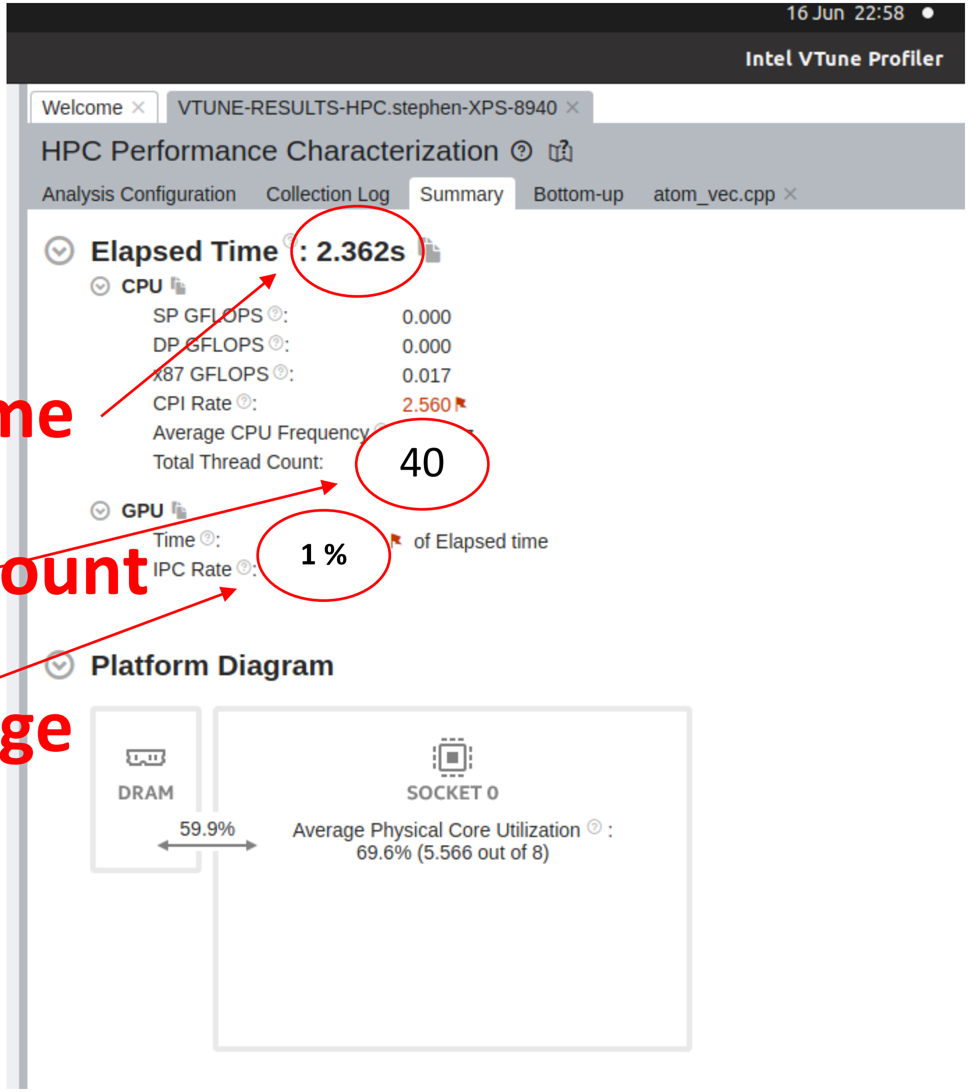
Generate html report for collected VTune™ data:

```
vtune -report summary -result-dir vtune_data -format html -report-output $(pwd)/summary.html
```

Various types of report can be generated like `summary`, `top-down`, `callstacks` ...

VTune Results - 1





Shorter Elapsed Time

Increased Thread Count

Increased GPU Usage

Reduced Core Utilization

Effective Physical Core Utilization: 84.0% (6.723 out of 8)

Effective Logical Core Utilization: 57.5% (9.197 out of 16)

MPI Imbalance: 0.003s (0.0%)

MPI Rank on the Critical Path:

MPI Busy Wait Time: 0.001s (0.0%)

Serial Time (outside parallel regions): 4.675s (72.8%)

Top Serial Hotspots (outside parallel regions)

This section lists the loops and functions executed serially in the master thread outside of any OpenMF
hotspot functions. Since the Serial Time metric includes the Wait time of the master thread, it may sign

Function	Module	Serial CPU Time
module_get_kallsym	vmlinux	0.534s
[Loop at line 76 in LAMMPS_NS::FixNVEIntel::initial_integrate]	lmp	0.259s
[Loop at line 633 in LAMMPS_NS::FixIntel::reduce_results<double>]	lmp	0.226s
[Loop at line 844 in .omp_outlined._debug__.35]	lmp	0.195s
[Loop at line 2238 in LAMMPS_NS::Neighbor::check_distance]	lmp	0.191s
[Others]	N/A*	2.705s

**N/A is applied to non-summable metrics.*

Parallel Region Time: 1.744s (27.2%)

Effective CPU Utilization Histogram

GPU Utilization when Busy: 10.5%

EU State:

Active: 10.5%

Stalled: 18.0%

Idle: 71.6%

Occupancy: 23.9% of peak value

Effective Physical Core Utilization: 69.6% (5.566 out of 8)

Effective Logical Core Utilization: 35.0% (5.604 out of 16)

Effective CPU Utilization Histogram

GPU Utilization when Busy: 27.9%

EU State:

Active: 27.9%

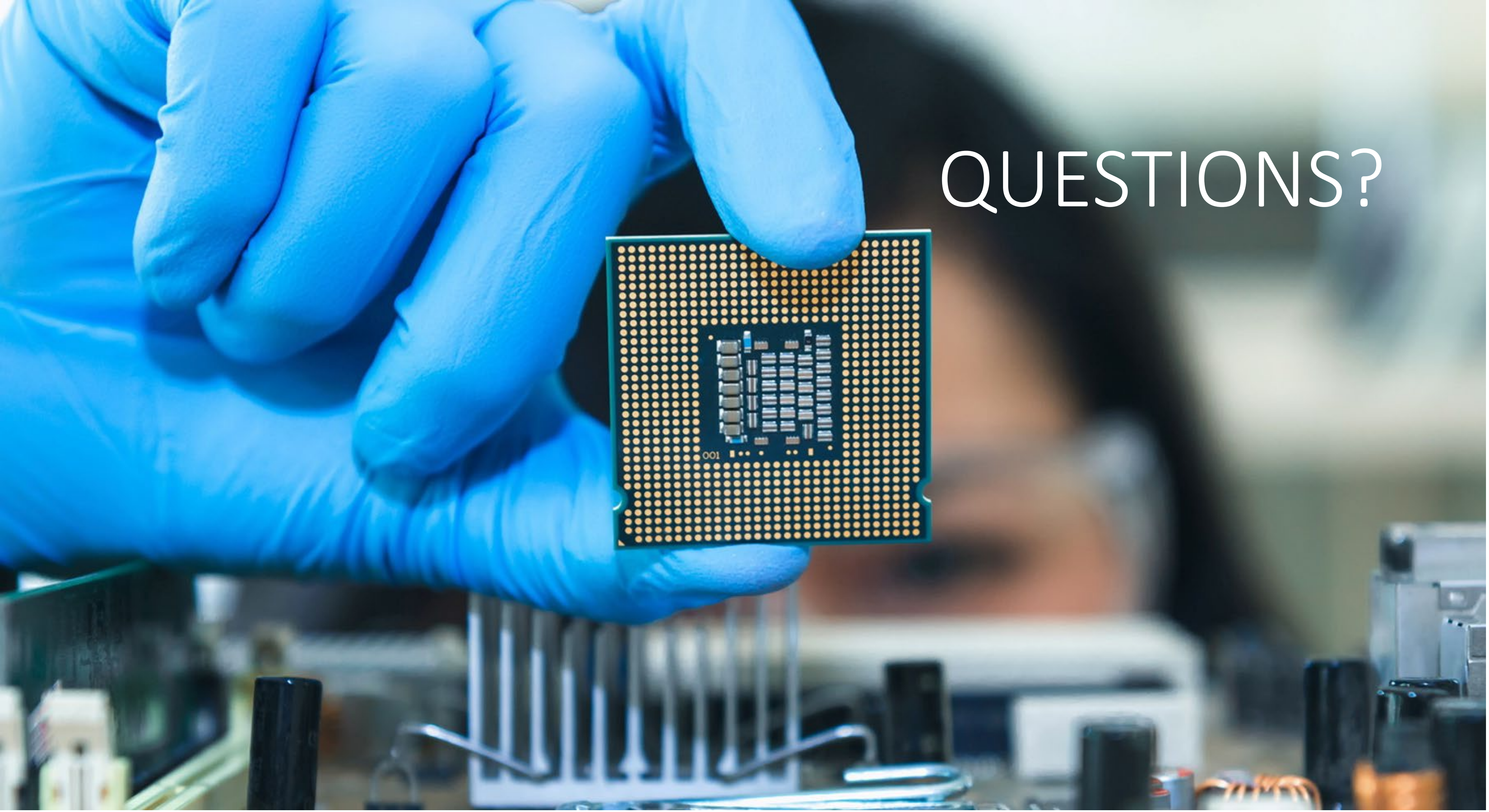
Stalled: 36.7%

Idle: 35.4%

Occupancy: 53.1% of peak value

INCREASED GPU Utilization when busy

QUESTIONS?



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.