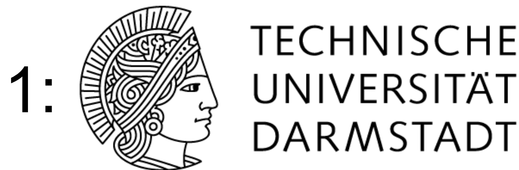
Altis-SYCL: Migrating Altis Benchmarking Suite from CUDA to SYCL for GPUs and FPGAs

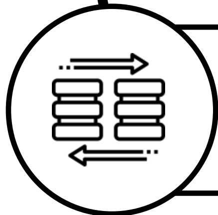
Christoph Weckert[1], Leonardo Solis-Vasquez[1], Julian Oppermann[1], Andreas Koch[1], Oliver Sinnen[2]

1: 

2:

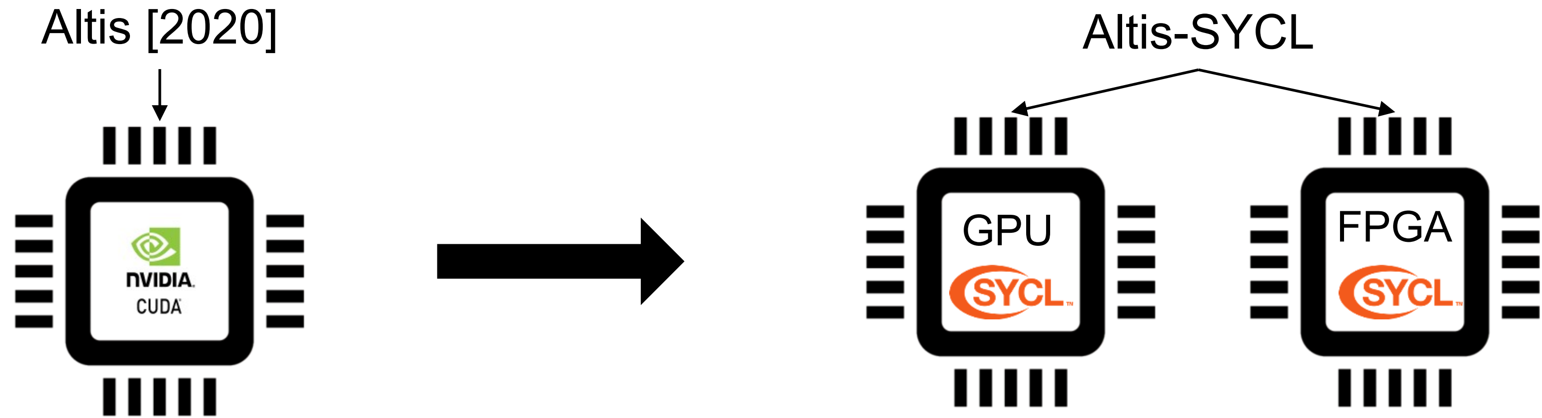
Contents

Introduction to Altis


Migrating from CUDA to SYCL for GPUs

Porting and Optimizing for FPGAs

Altis-SYCL
Benchmark Suite for GPUs and FPGAs



Our contribution

Set of practical development guidelines for deploying CUDA applications on HPC systems supporting SYCL

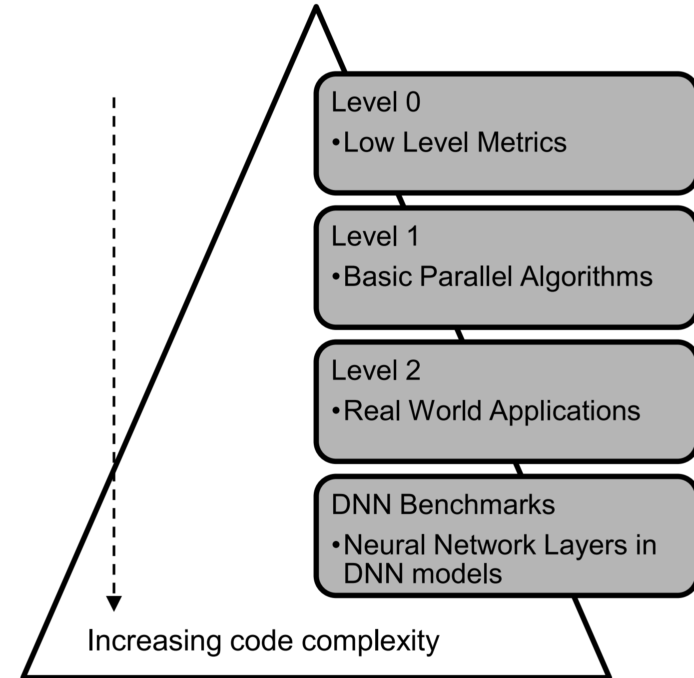
The Altis Benchmark Suite
B. Hu and C. J. Rossbach [ISPASS 2020]

Collection of a wide-range of GPGPU applications written in CUDA

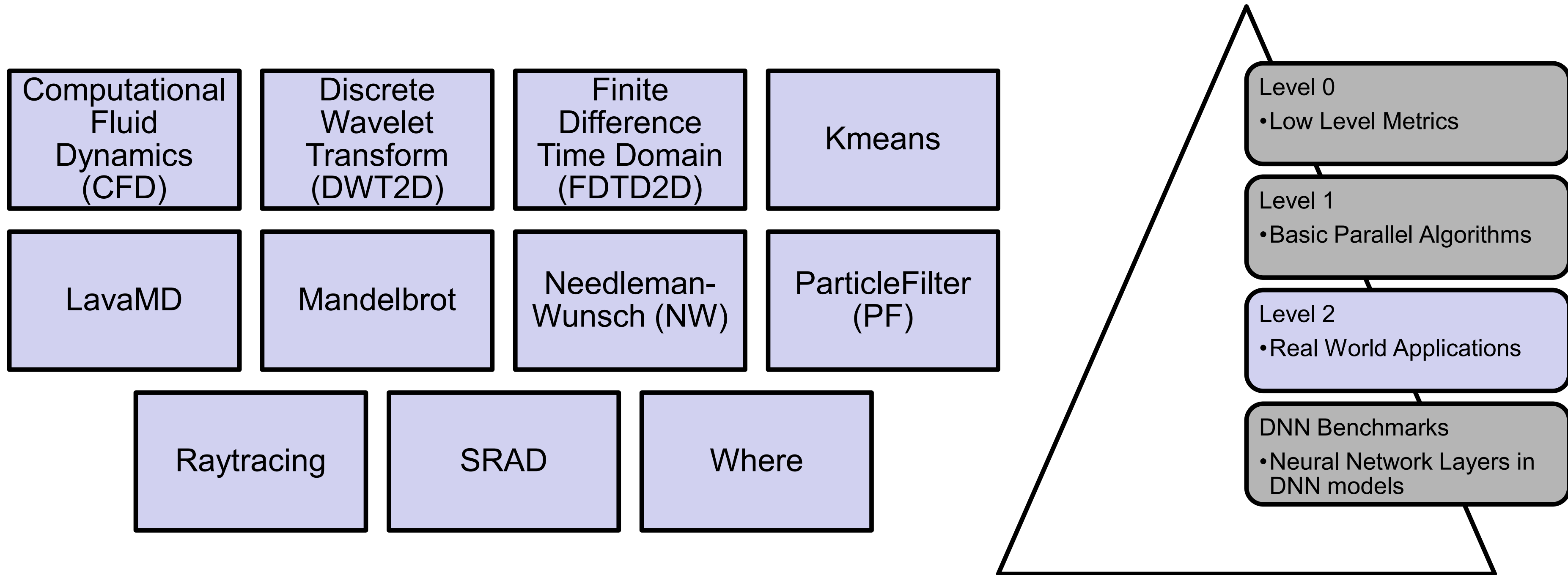
- Grouped into four categories

Aims to better represent modern GPGPU workloads

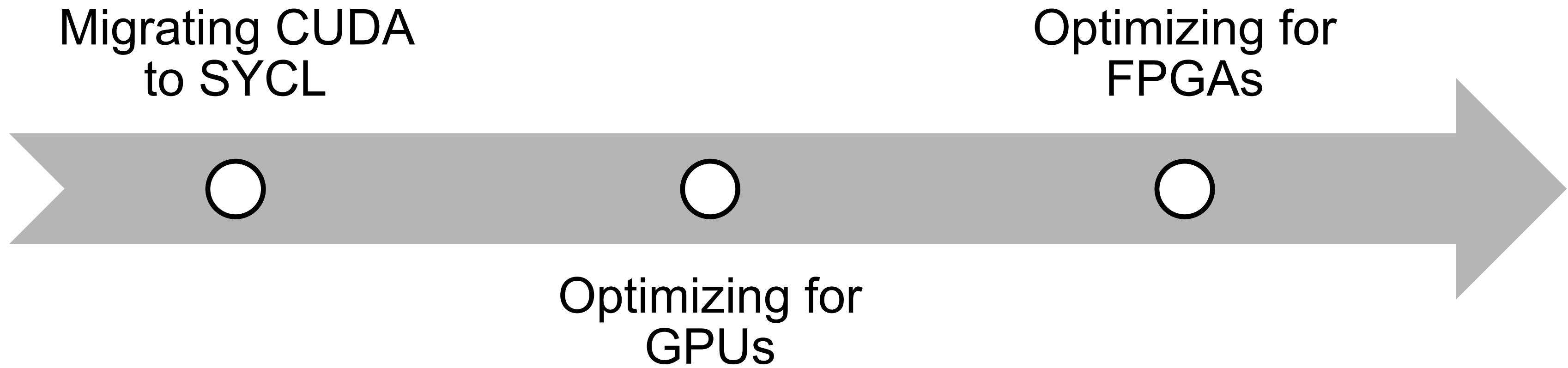
- By adopting and extending applications from Rodinia and SHOC



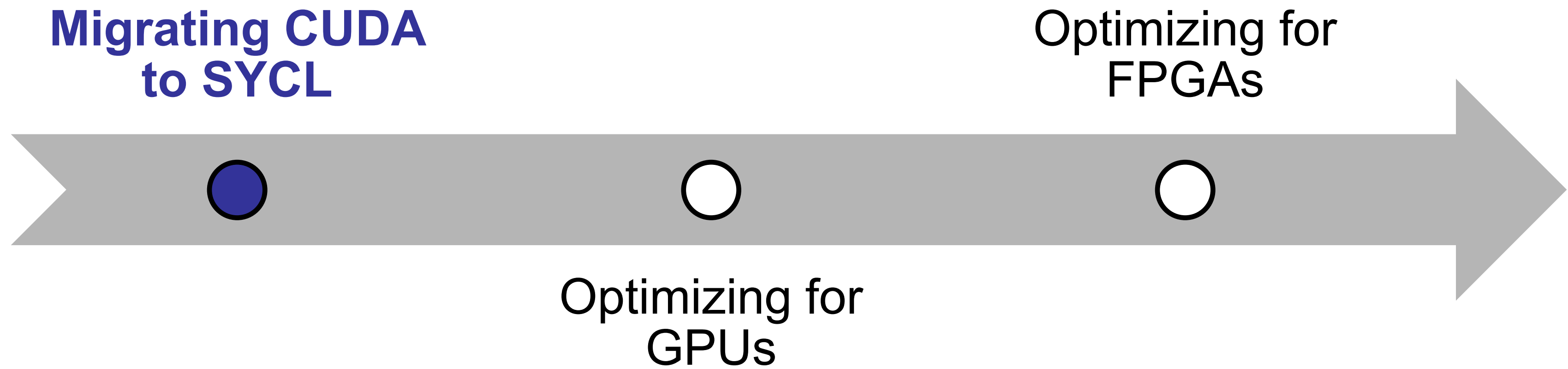
Our Focus: Altis' Level 2 Applications



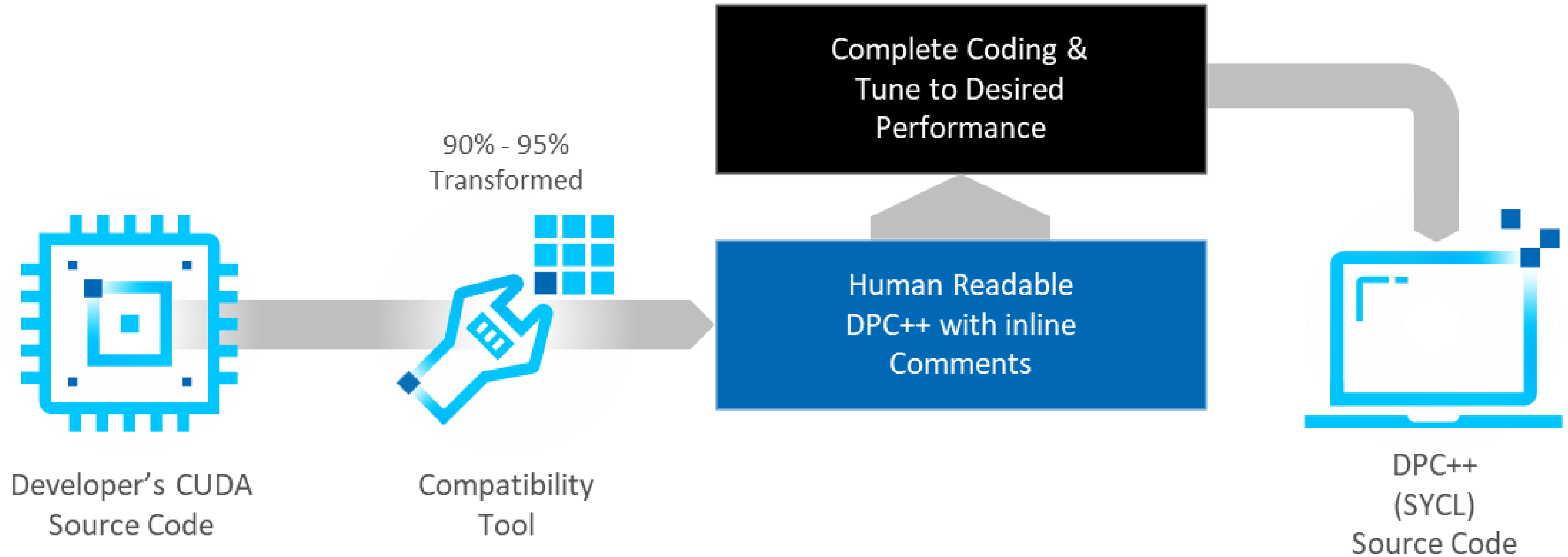
Migration Methodology for Altis-SYCL



Migration Methodology for Altis-SYCL



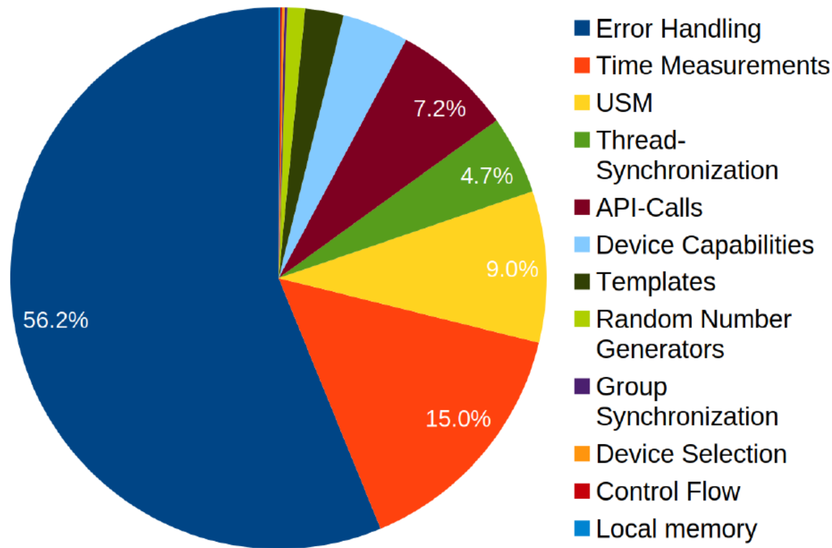
Intel’s DPC++ Compatibility Tool (DPCT) Usage Flow



[Intel]

Migrating CUDA to SYCL using DPCT

DPCT-inserted warnings domain distribution

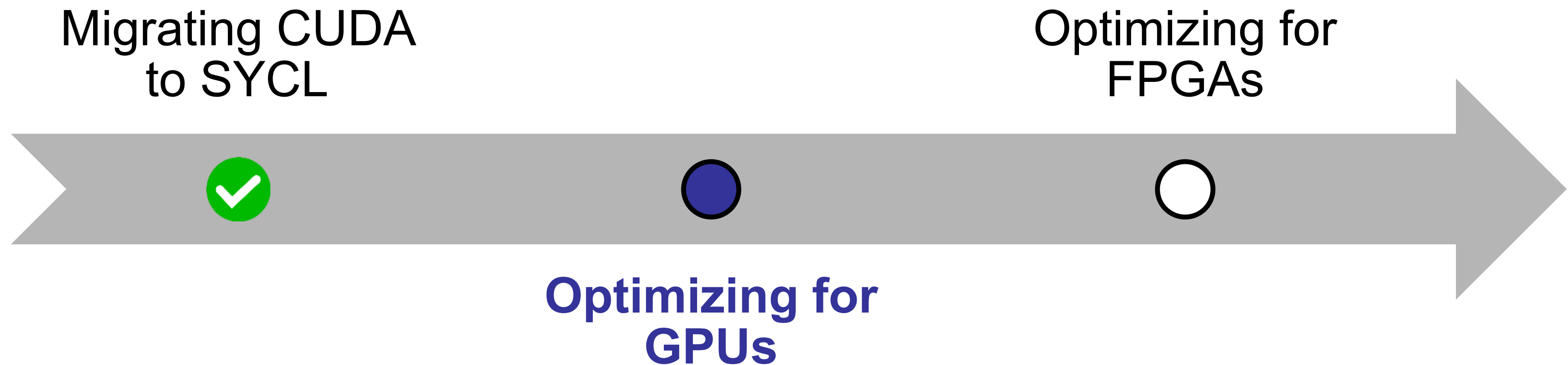


CUDA handles errors via function return, SYCL uses try-catch blocks instead

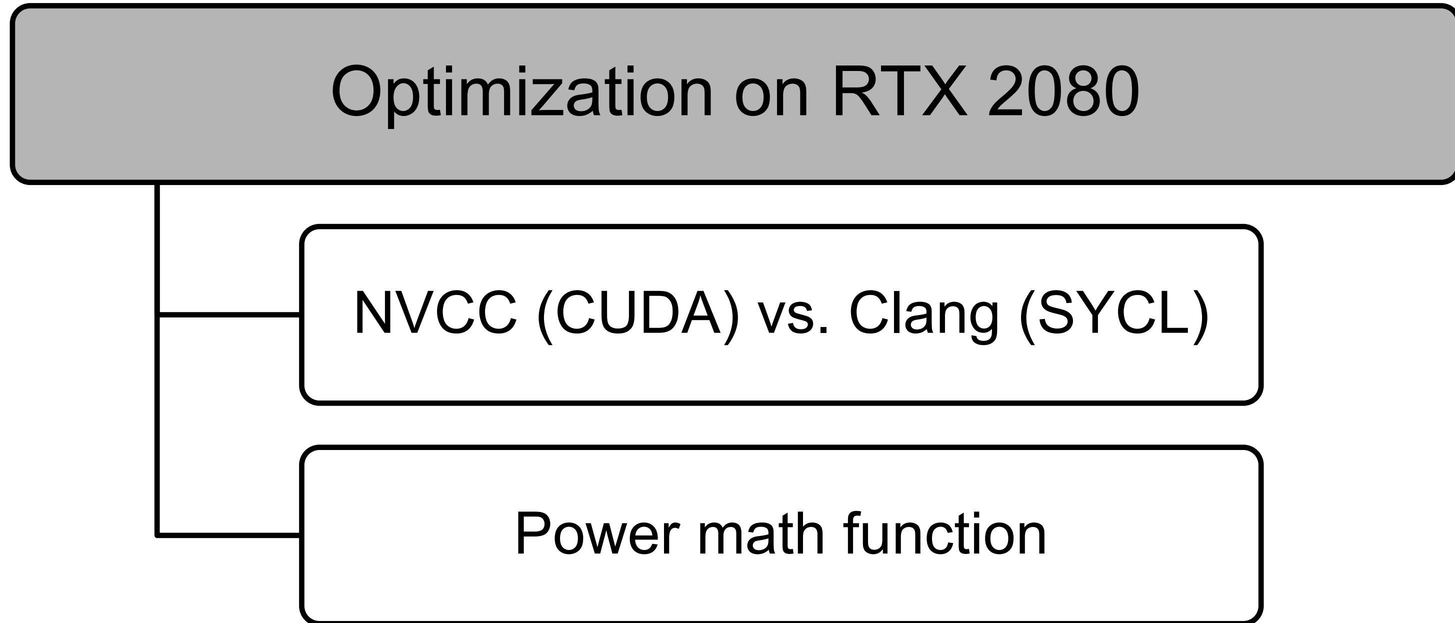
```
/*
DPCT1003:397: Migrated API does not return error code. (*, 0) is inserted.
You may need to rewrite this code.
*/
CUDA_SAFE_CALL((weights_GPU = sycl::malloc_device<double>(
                                 Nparticles, dpct::get_default_queue()),
                             0));
```

DPCT inserted a total of 2535 diagnostic references when translating Altis Level 2

Migration Methodology for Altis-SYCL



Optimizing for GPUs



NVCC (CUDA) vs. Clang (SYCL)

Different compilers
→
Different behavior

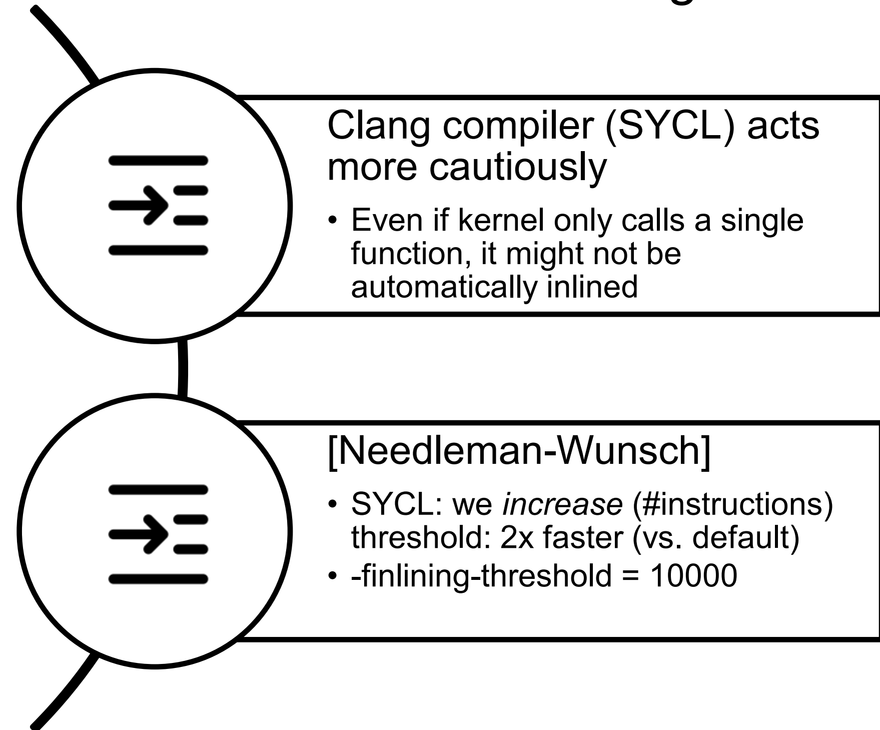
Loop Unrolling

- CUDA: might increase the performance
- SYCL: might have the *opposite* effect

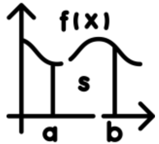
SYCL: we *disable* unrolling for the main loop → 3x faster (vs. unrolling)

NVCC (CUDA) vs. Clang (SYCL)

Different compilers
→
Different behavior

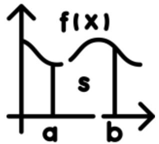


Power Math Function



[ParticleFilter Float]

- DPCT replaced: **pow (a, 2)** → **a x a**
- SYCL: 6x faster than CUDA



We apply above transformation back to the original CUDA

- CUDA vs. SYCL: performance is on par

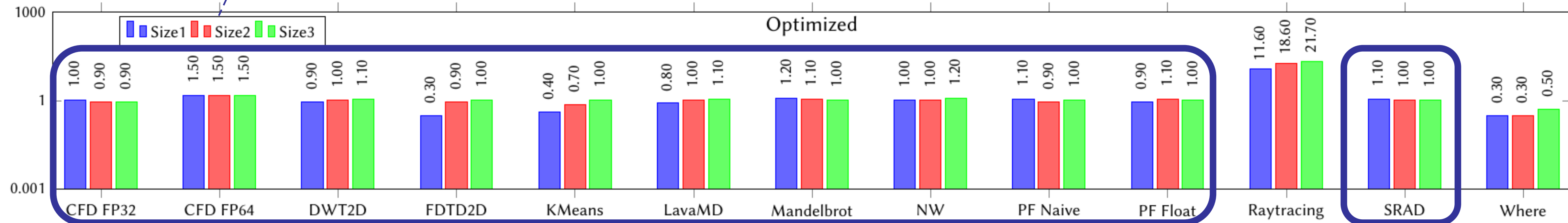
SYCL vs. CUDA
Speedup on RTX 2080 GPU

Predefined sizes in Altis:

Size 1 (small) → Size 2 → Size 3 (large)

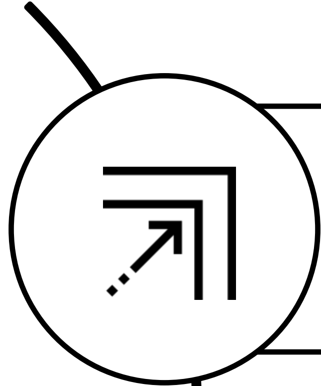
SYCL vs. CUDA
Speedup
(After optimizations)

Speedup values > 1
→
SYCL faster than CUDA



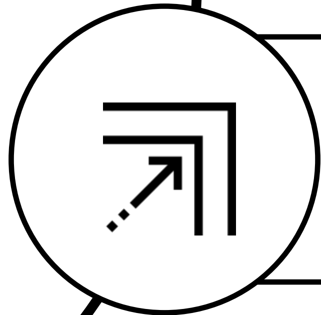
Most optimized
SYCL and CUDA versions
have comparable performance

Corner Case [Raytracing]

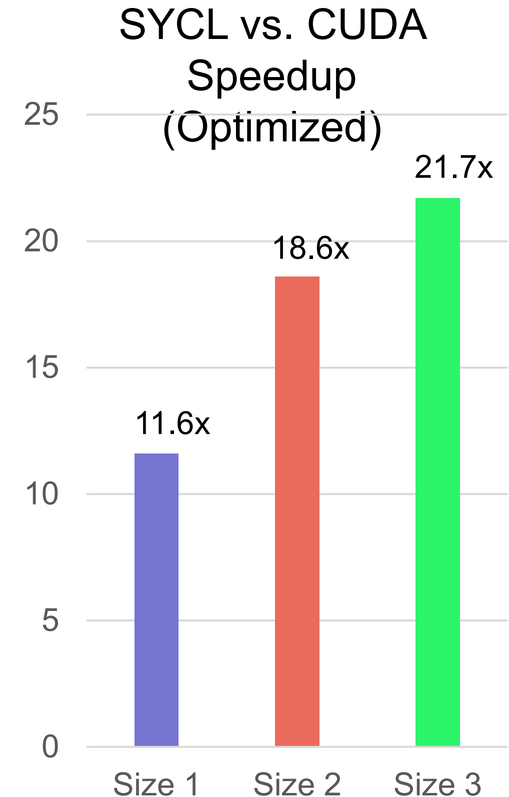


DPCT introduced a different RNG

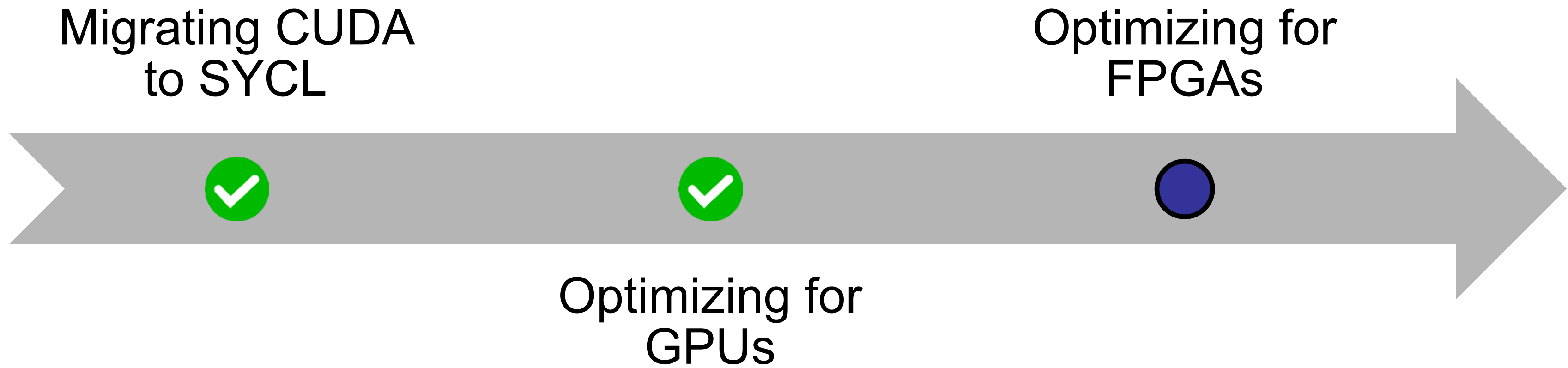
- cuRAND's XORWOW
- oneMKL's philox4x32x10



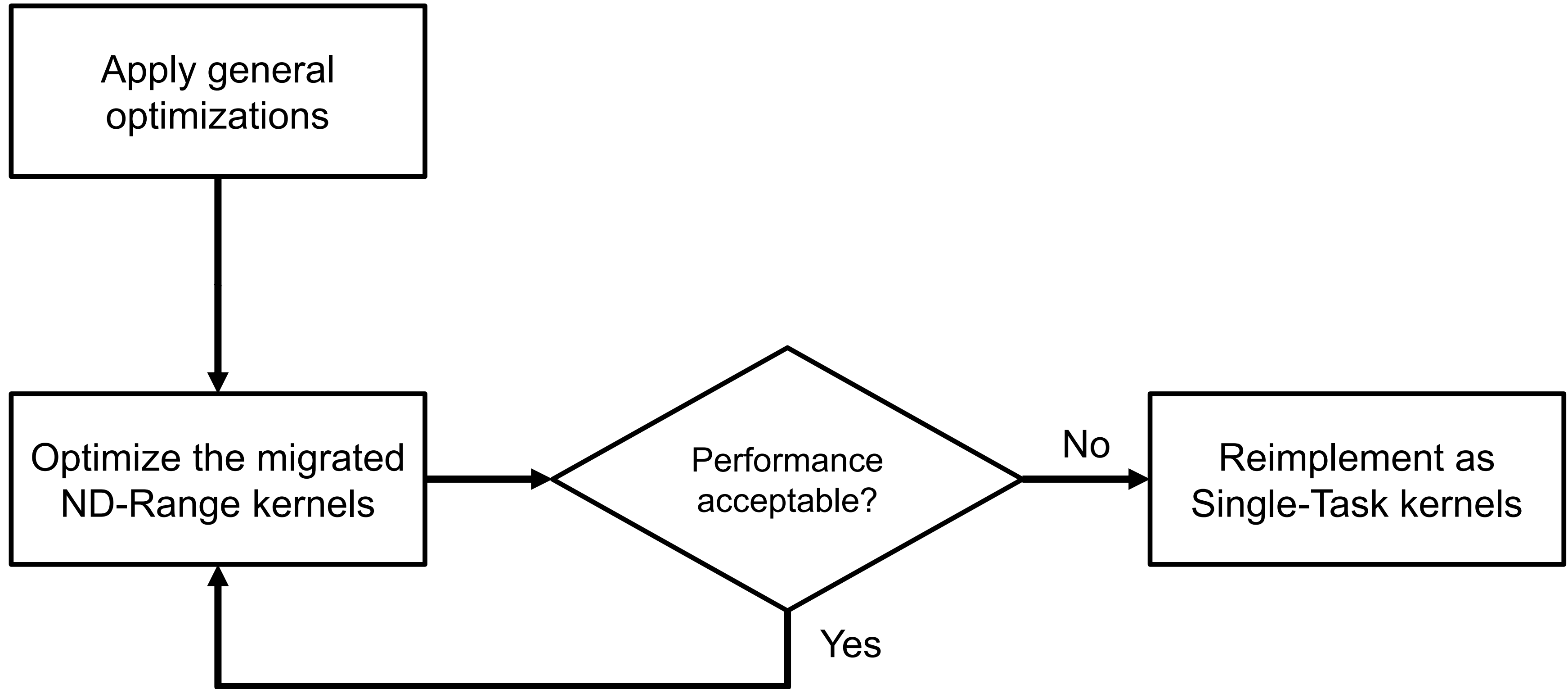
Significant code refactoring to cope with CUDA virtual functions



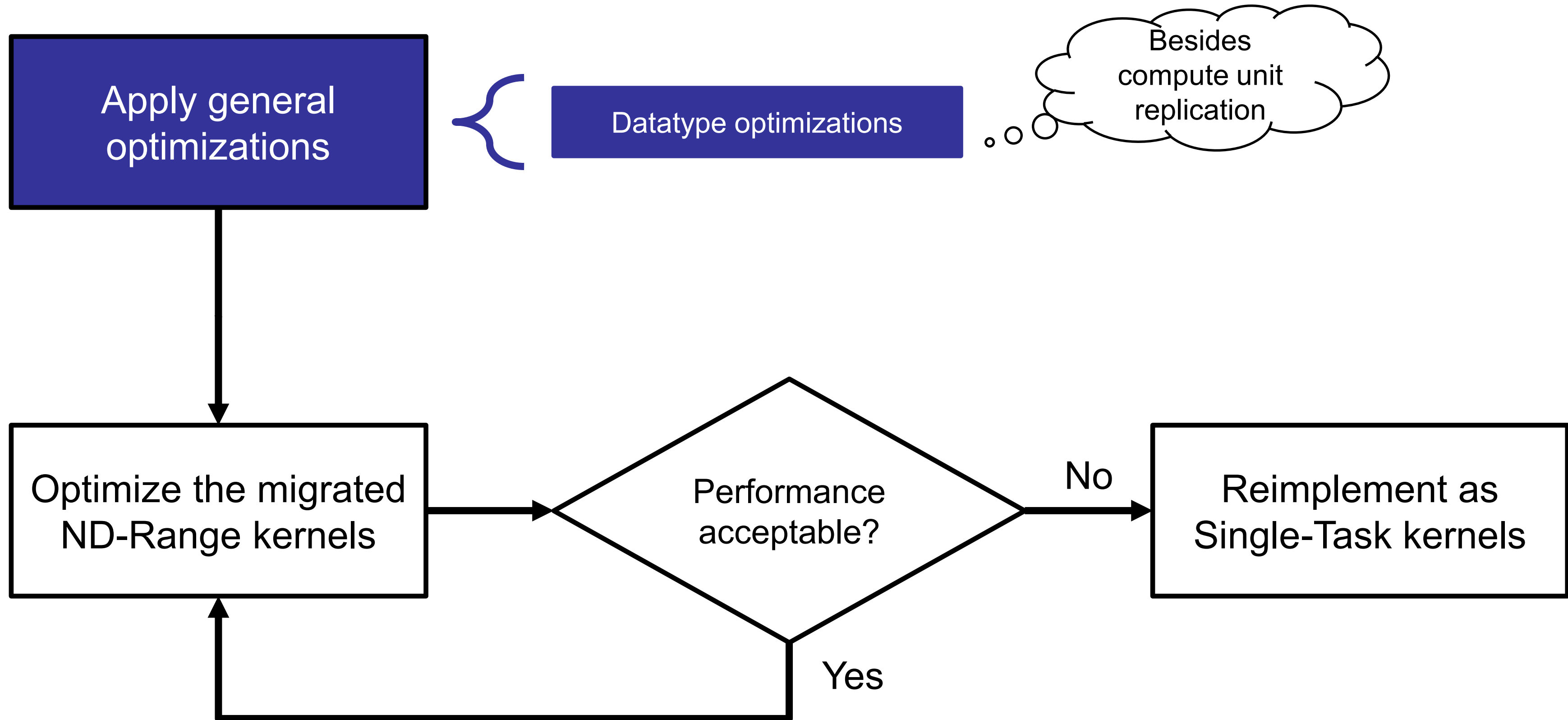
Migration Methodology for Altis-SYCL



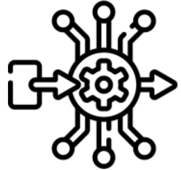
Optimization Methodology for FPGAs



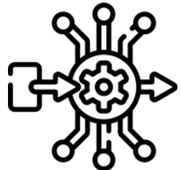
Optimization Methodology for FPGAs



Datatype Optimizations
[Raytracing] <material> class: Original



FPGA compiler might infer *inefficient* global and local memory systems



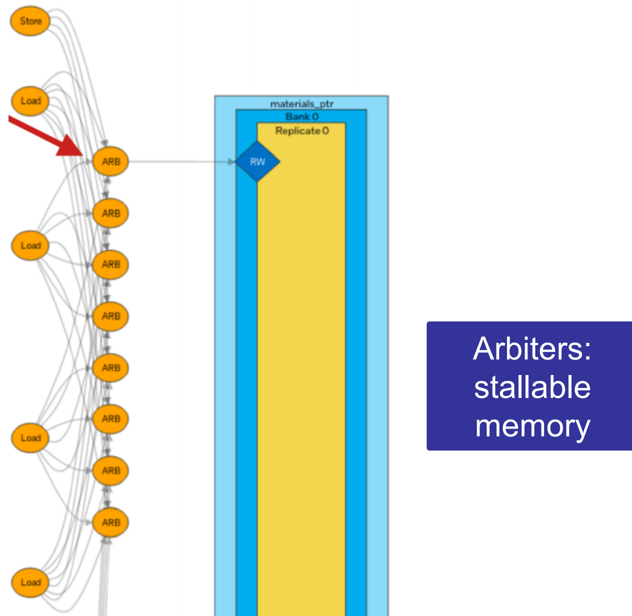
E.g., C++ classes or structs featuring multiple member variables of different types

```
class material {    /* Original */
public:
    enum type: uint8_t {metal, dielectric, lambertian};
    type m_type;
    vec3 m_albedo;        // lambertian and metal (lam)
    float m_fuzz;         // metal (met)
    float m_ref_idx;};    // dielectic (die)
```

Corresponding implementation:
- Performs only a ***single*** write access to such object
- But inferred hardware contains ***three*** store ports

Datatype Optimizations
[Raytracing] <material> class: Original → Optimized

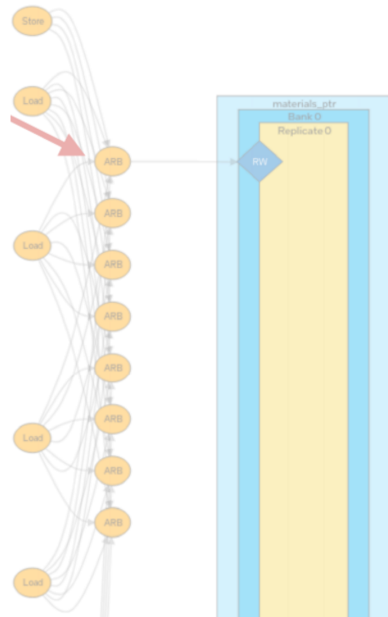
```
class material {    /* Original */
public:
    enum type: uint8_t {metal, dielectric, lambertian};
    type m_type;
    vec3 m_albedo;      // lambertian and metal (lam)
    float m_fuzz;       // metal (met)
    float m_ref_idx;};  // dielectic (die)
```



Datatype Optimizations
[Raytracing] <material> class: Original → Optimized

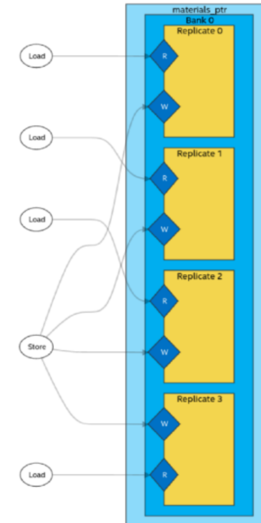
```
class material {    /* Original */
public:
    enum type: uint8_t {metal, dielectric, lambertian};
    type m_type;
    vec3 m_albedo;       // lambertian and metal (lam)
    float m_fuzz;        // metal (met)
    float m_ref_idx;};   // dielectic (die)
```

```
class material {    /* Optimized */
public:
//   data[0]: "fuzz" parameter
//   data[1]: "ref_idx" parameter
//   data[2:4]: "albedo" parameter
//   data[5]: material "type": met (0), die (1), lam (2)
//   data[6:7]: unused
     sycl::float8 data;};
```

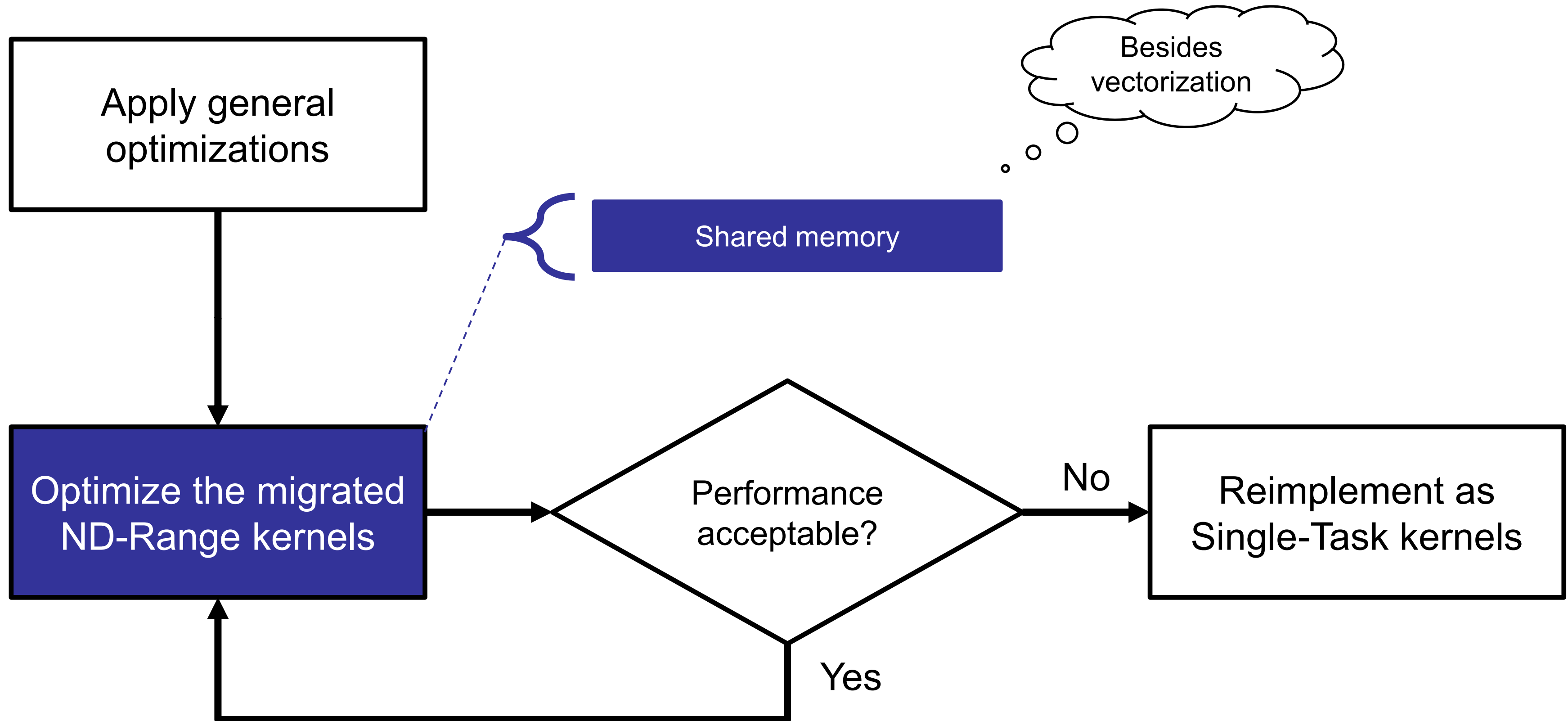


Fusing all class members into a single vector member

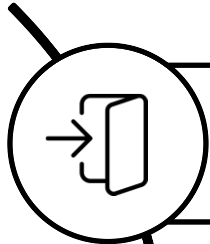
Arbiters: stallable memory → No arbiters: **stall-free** memory



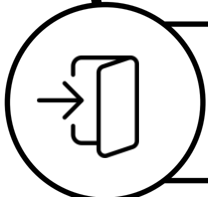
Optimization Methodology for FPGAs



SYCL Accessors Introduction

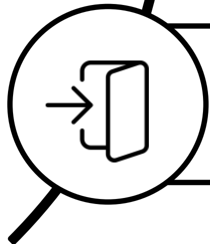


Standard method in SYCL for creating shared memory



DPCT: inserts SYCL accessors

- Dynamically sized
- Cannot be statically defined at compile time



Can cause issues when targeting FPGAs

```
65      sycl::buffer<int> results_buff{sycl::range(size)};
```

```
95      sycl::accessor results{results_buff, cgh, sycl::write_only,
96                             sycl::noinit};
```

```
100         cgh.parallel_for<mark_matches_cu<CU>>(
101             sycl::nd_range<1>(grid_per_cu_k3[CU], block_dim),
102             [=](sycl::nd_item<1> item_ct1)
103                 [[intel::kernel_args_restrict, intel::num_simd_work_items(16),
104                   intel::no_global_work_offset(1),
105                   sycl::reqd_work_group_size(1, 1, g_thread_cnt),
106                   intel::max_work_group_size(1, 1, g_thread_cnt)]] {
107                   const int tid = item_ct1.get_global_id(0) + offset;
108
109                   for (int i = tid; i < size; i += g_thread_cnt * grid_range)
110                     results[tid] = (d_arr[tid] < coverage) ? 1 : 0;
111                 });
```

[Where]

Shared Memory <group_local_memory_for_overwrite> Class

Allows the implementation of shared memories with user-defined sizes

- Replaces default SYCL accessors

Vendor and device specific

- Only for Intel FPGAs, not supported on CPUs/GPUs
- Available via oneAPI's FPGA Toolkit

We apply this to all ND-Range kernels → reduces resource utilization

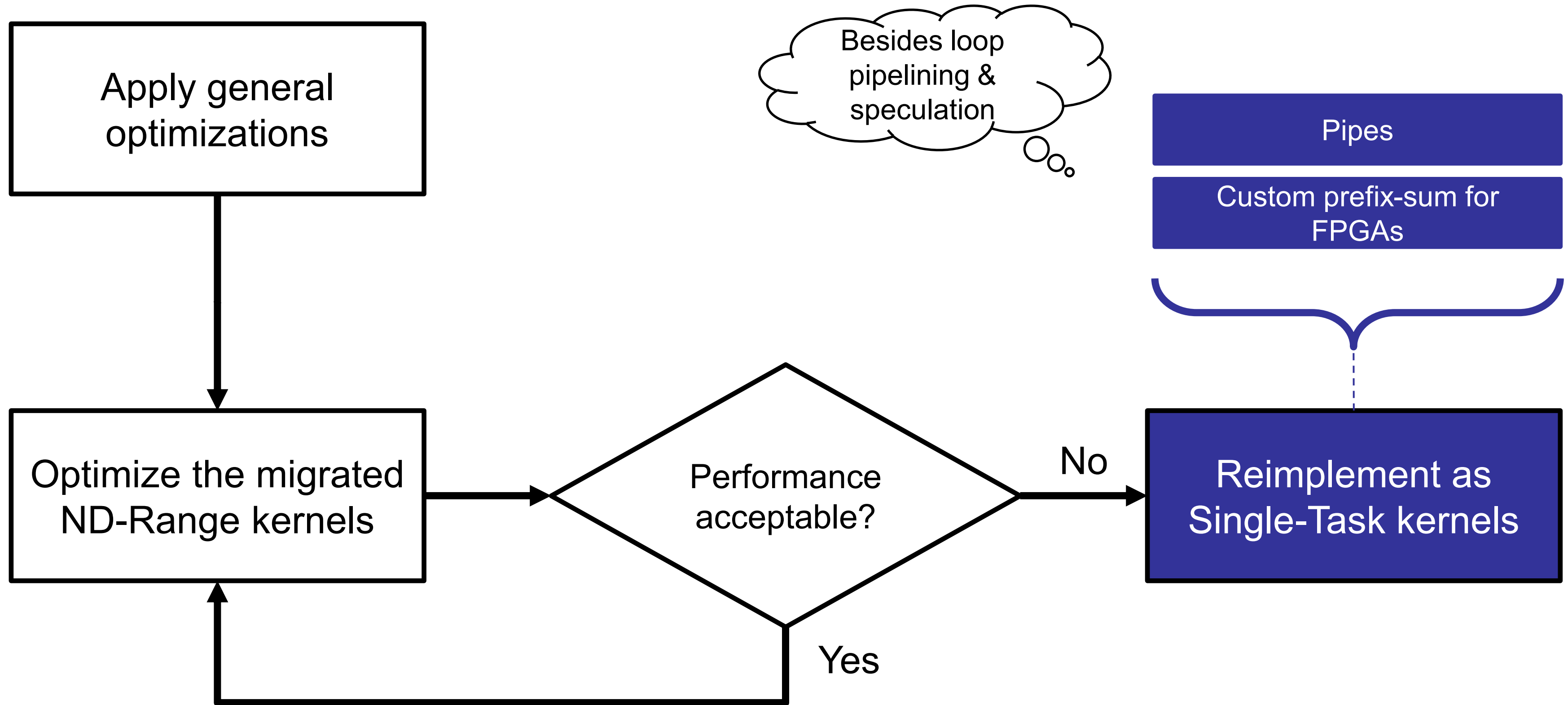
Can be used to allocate group-local memory at the kernel functor scope

```
template <typename T, typename Group>
multi_ptr<T, access::address_space::local_space>
group_local_memory_for_overwrite(Group g);
```

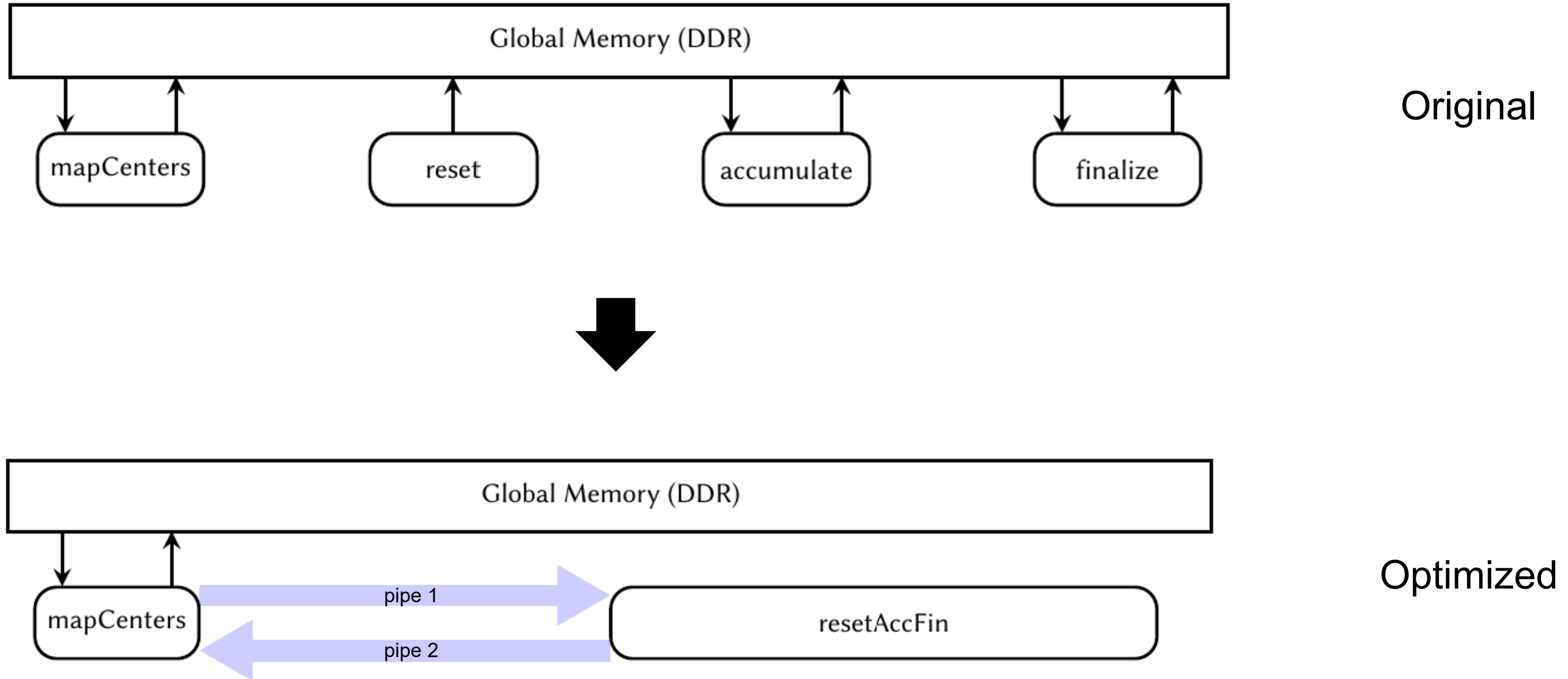
```
30  void kernel_gpu_cuda(par_str d_par_gpu, dim_str d_dim_gpu,
31                        sycl::device_ptr<box_str> d_box_gpu,
32                        sycl::device_ptr<FOUR_VECTOR> d_rv_gpu,
33                        sycl::device_ptr<fp> d_qv_gpu,
34                        sycl::device_ptr<FOUR_VECTOR> d_fv_gpu,
35                        sycl::nd_item<1> item_ct1) {
36     auto rA_shared_ptr =
37         group_local_memory_for_overwrite<FOUR_VECTOR[NUMBER_PAR_PER_BOX]>(
38             item_ct1.get_group());
```

[LavaMD]

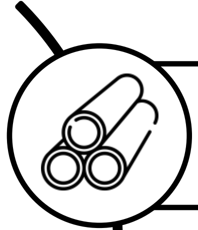
Optimization Methodology for FPGAs



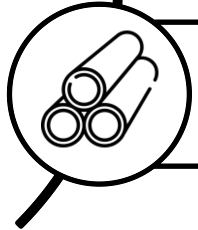
Pipes
[Kmeans]: Optimization Process



Pipes
[Kmeans]: Optimization Process

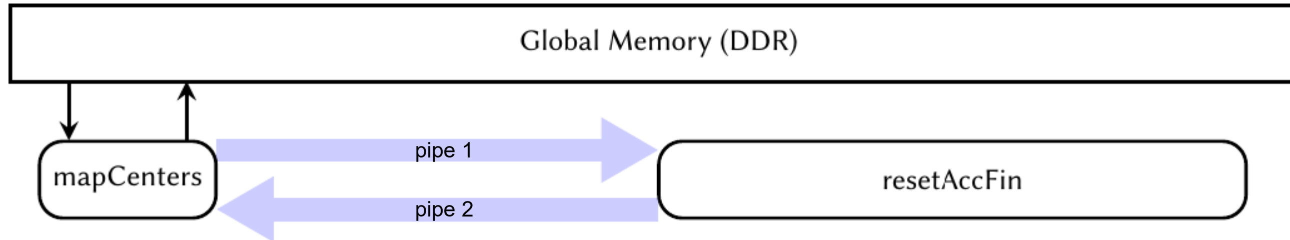


Dataflow to/from global memory is limited to <mapCenters> kernel only



Benefits of pipes

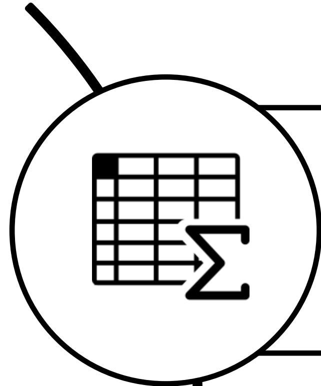
- Mapping of each data point is immediately passed between kernels



Performance improvement: ~510x

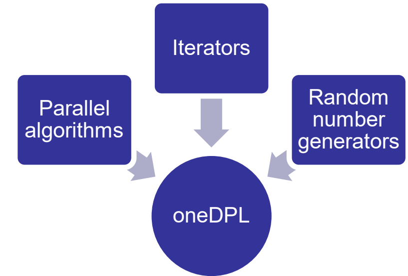
(pipes vs. no-pipes)

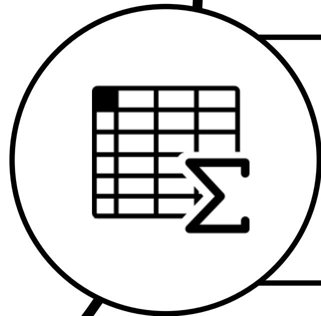
Custom Prefix-Sum for FPGAs



[Where]: prefix-sum

- CUDA: implementation from vendor library
- SYCL: DPCT incorporates oneDPL's prefix sum
- SYCL: ~2x slower vs. CUDA on RTX 2080





oneDPL does *not* provide an FPGA-optimized implementation →

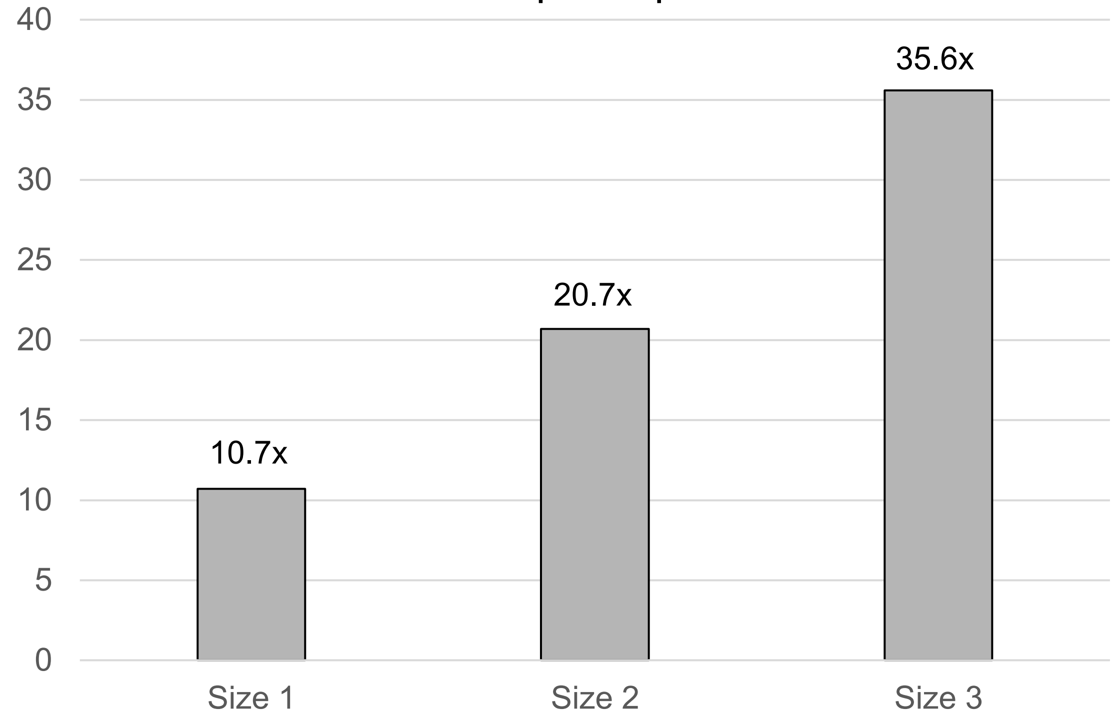
We develop a custom prefix-sum (Single-Task)

Performance improvement: ~100x

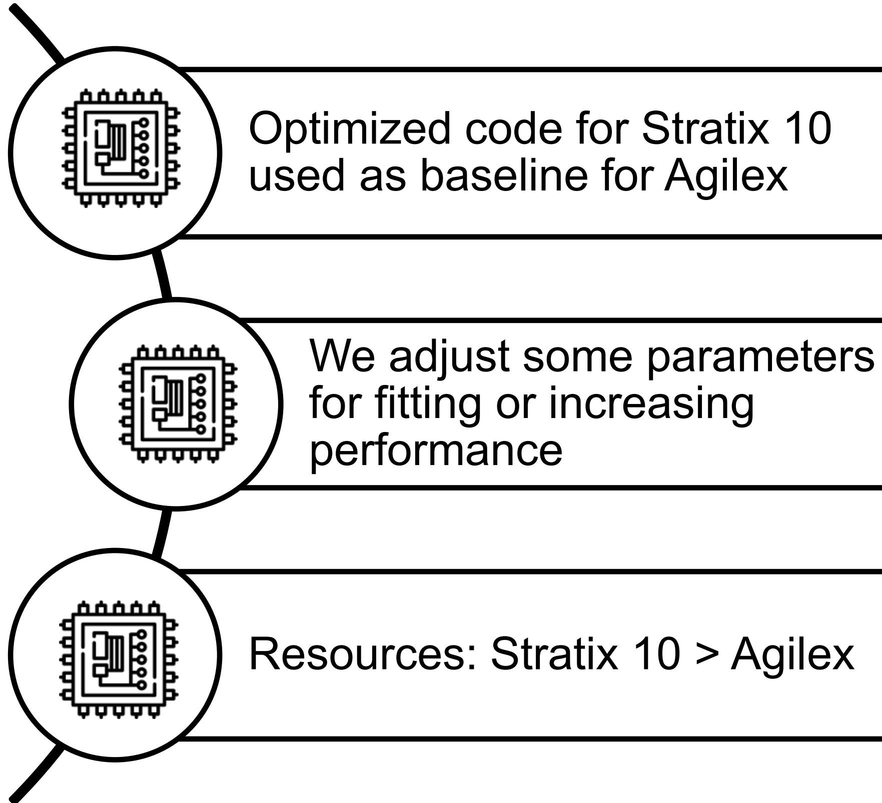
(Custom prefix-sum vs. oneDPL's implementation)

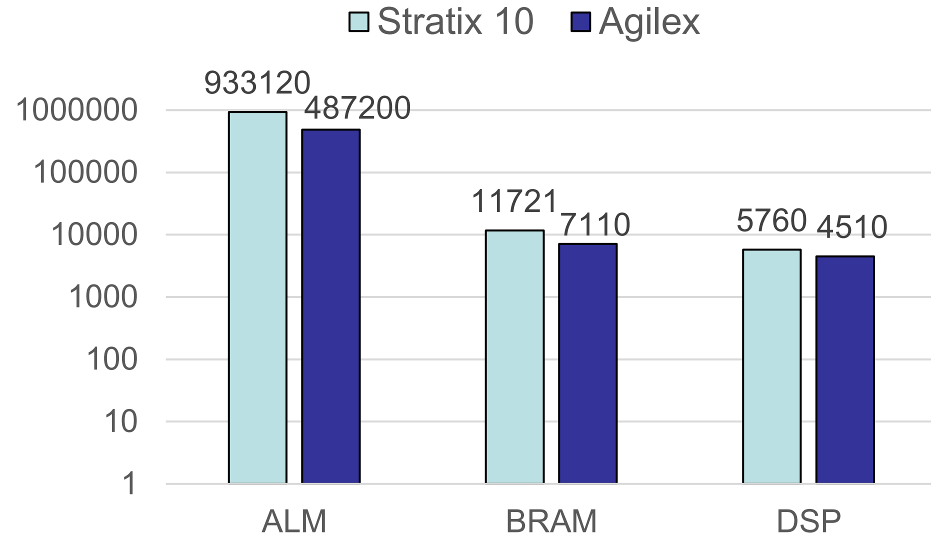
FPGA Optimized vs. FPGA Baseline Geometric Mean of Speedup on Stratix 10

$$Speedup = \frac{Runtime\ Baseline}{Runtime\ Optimized}$$



Stratix 10 → Agilex FPGA Retargeting

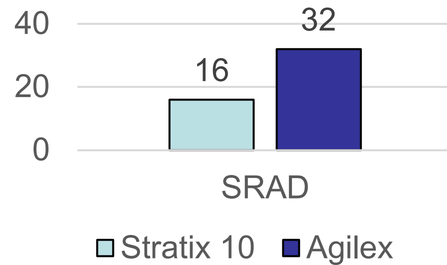




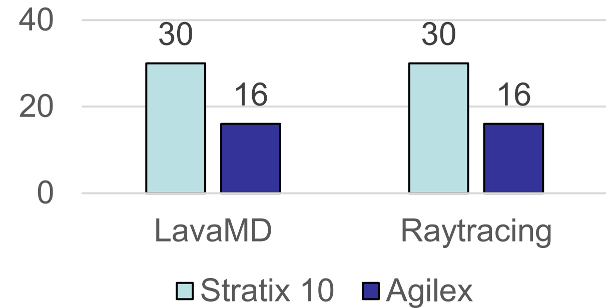
Δ = +47.7% Δ = +39.3% Δ = +21.7%

Stratix 10 → Agilex (Some) Parameter Adjustments

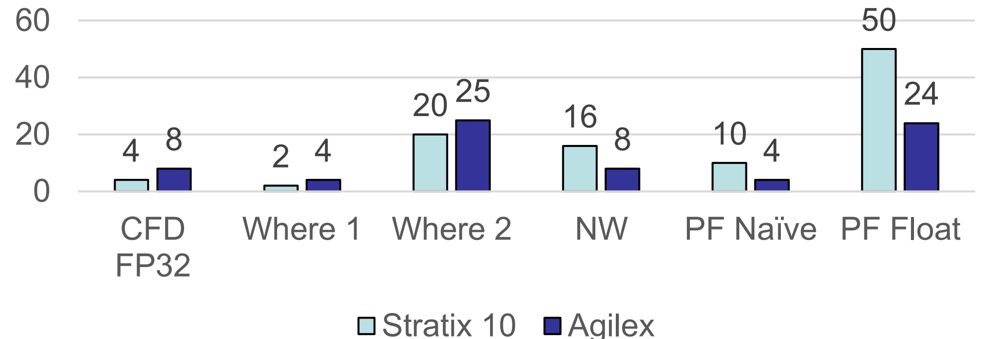
Increasing WG size



Reducing unrolling factor



Scaling up/down compute unit replication factor

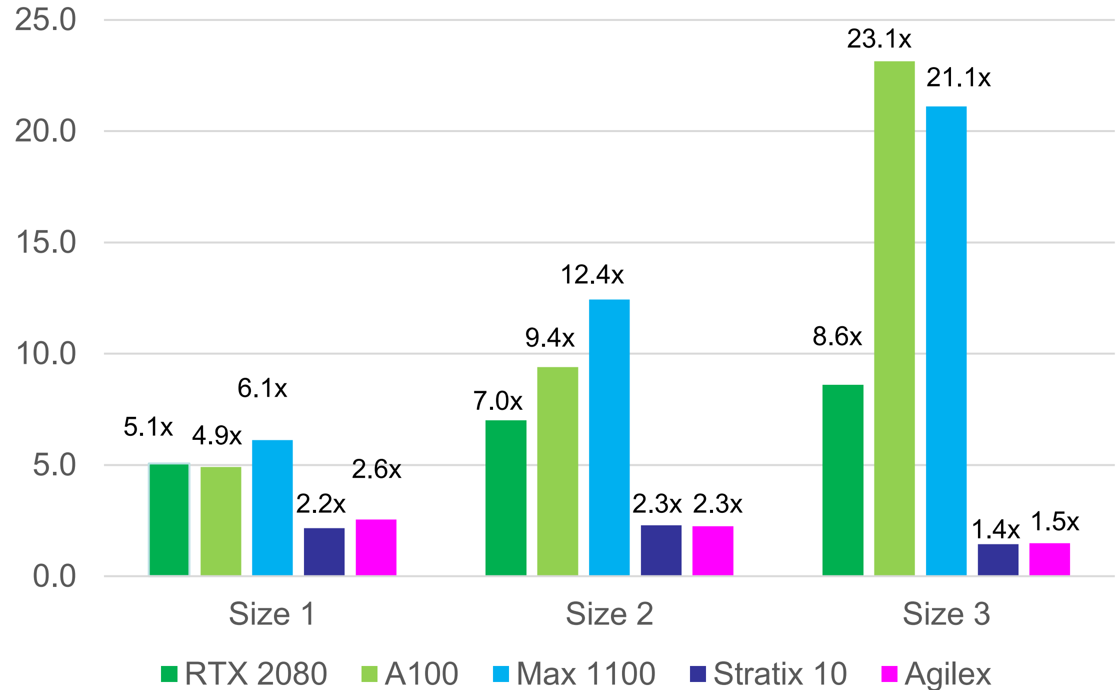


Evaluation Setup

	Device	Process [nm]	# Compute Units	Peak FP32 [TFLOP/s]	Peak Mem. BW [GB/s]
CPU	Xeon Gold 6128	14	6 Cores	1.1	128.0
GPUs	RTX 2080	12	46 SMs	10.1	448.0
	A100	7	108 SMs	19.5	1555.0
	Max 1100	10	56 X^e-cores	22.2	1229.0
FPGAs	Stratix 10	14	4713 DSPs	2.4 – 4.2	76.8
	Agilex	10	4510 DSPs	2.3 – 5.0	85.3

GPUs vs. FPGAs (wrt. CPU)
Geometric Mean of Relative Speedup

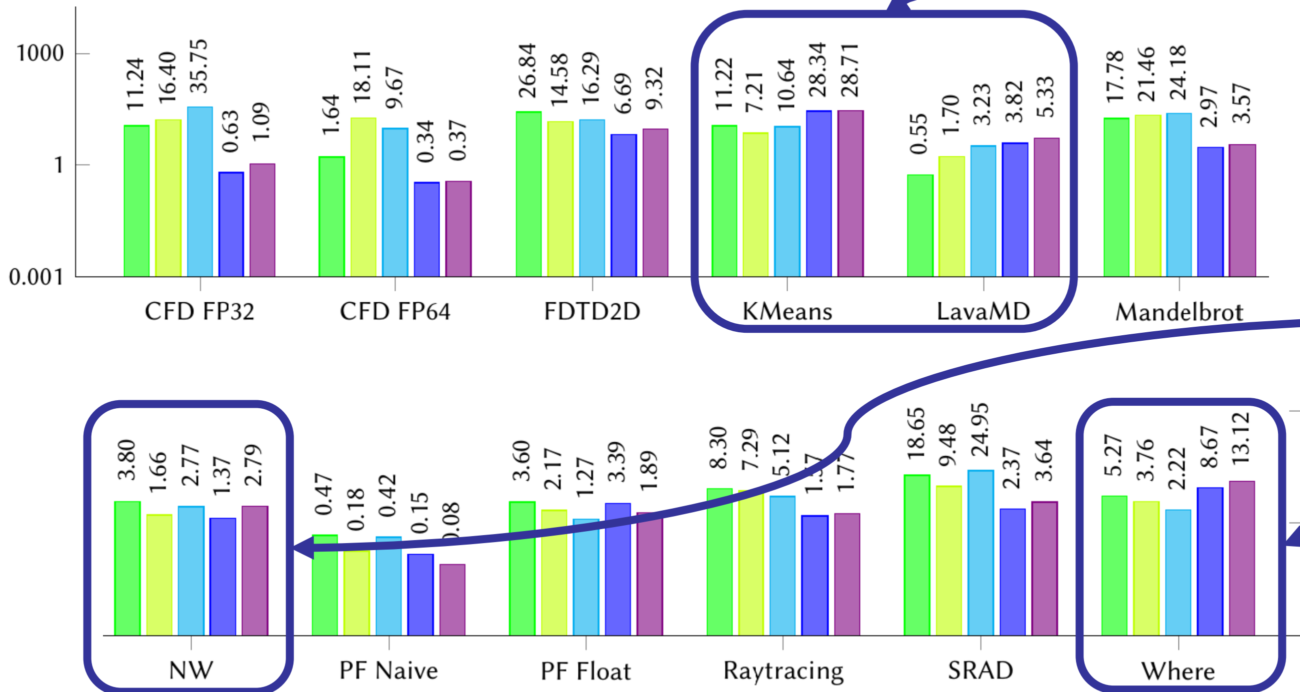
$$Speedup\ X = \frac{Runtime\ CPU}{Runtime\ X}$$



GPUs vs. FPGAs (wrt. CPU) Relative Speedup

$$Speedup\ X = \frac{Runtime\ CPU}{Runtime\ X}$$

Size 1



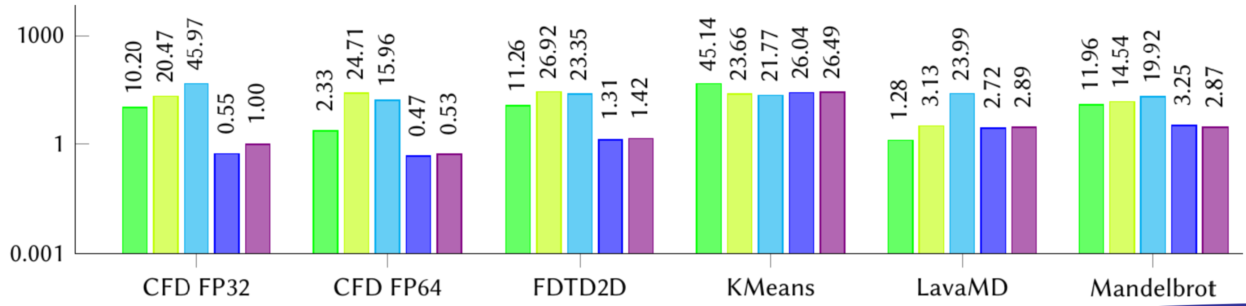
Comparable performance between FPGAs and GPUs

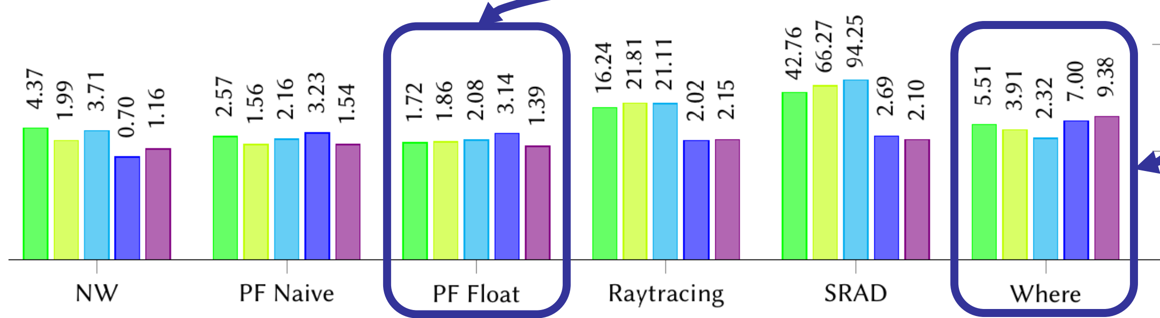
GPUs vs. FPGAs (wrt. CPU) Relative Speedup

$$Speedup\ X = \frac{Runtime\ CPU}{Runtime\ X}$$

GPUs vs. FPGAs (wrt. CPU)
Speedup

Size 2





Comparable performance between FPGAs and GPUs

Conclusions

Altis-SYCL: a modern and portable C++-based benchmark suite for GPUs and FPGAs

- Retains the advantages of Altis

Performance results

- Small input sizes: GPUs and FPGAs reach comparable performance for some applications
- Large input sizes: GPUs outperform FPGAs
  - Size 3: A100 vs. Agilex → geo. mean speedup: ~15x

Altis-SYCL: Migrating Altis Benchmarking Suite from CUDA to SYCL for GPUs and FPGAs

https://github.com/esa-tu-darmstadt/altis_sycl

Leonardo Solis-Vasquez

solis@esa.tu-darmstadt.de

Acknowledgements

Paderborn Center for Parallel Computing

Stratix 10 FPGA

Intel oneAPI Center of Excellence Research Award

Max 1100 GPU