SIPEARL

An Overview of the Maturity of SYCL Implementations and Backends for AArch64

Thomas Roglin

Etienne Renault

AHUG 2025

AHUG
ARM HPC USER GROUP

RHEA1

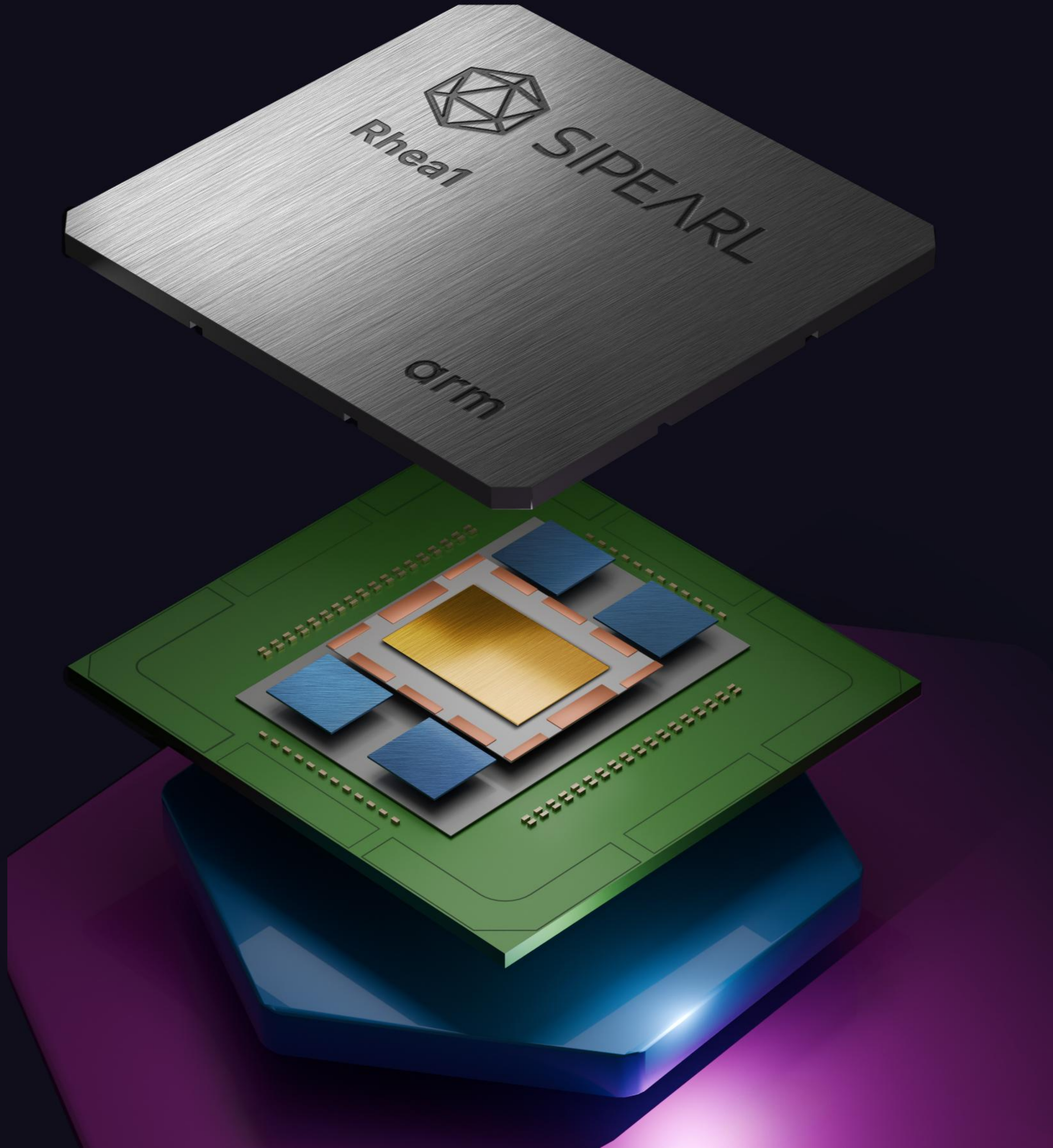
HPC and AI processor

Designed with

80 arm® Neoverse V1 cores with 2 x 256 SVE each

4 x HBM

4 x DDR5 interfaces



What is SYCL ?

SYCL is a C++-based heterogeneous programming model

Khronos group oversees SYCL standard

SYCL is a building block of UXL

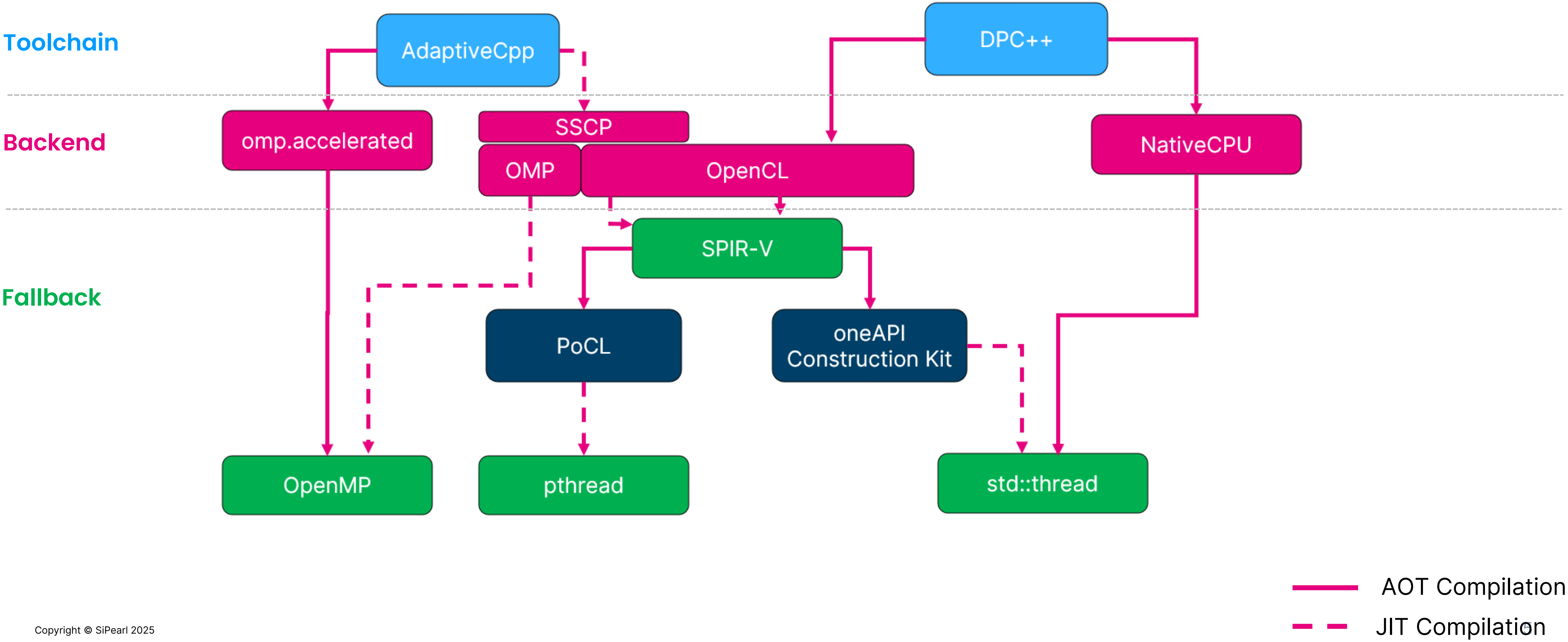
```
constexpr size_t N = 1024;

// Create SYCL queue
sycl::queue q;


// Allocate Unified Shared Memory
float* A = sycl::malloc_shared<float>(N, q);
float* B = sycl::malloc_shared<float>(N, q);
float* C = sycl::malloc_shared<float>(N, q);

sycl::range<1> global_range = {N};
sycl::range<1> local_range  = {64};
sycl::nd_range<1> kernel_nd_range = {global_range, local_range}


// Launch kernel
q.submit([&](sycl::handler& h) {
    h.parallel_for(kernel_nd_range, [=](sycl::nd_item<1> item) {
        size_t i = item.get_global_id(0);
        if (i < N) {
            C[i] = A[i] + B[i];
        }
    });
}).wait();
```

SYCL on AArch64

SYCL Implementation	Description	Purpose	Open-Source	AArch64 compatibility
Intel oneAPI DPC++	Led by Intel as part of the oneAPI	Production	✓ - ✗	✓ - ✗
AdaptiveCpp	Led by Heidelberg University.	Production	✓	✓
C-DaC ParaS compiler	Led by NSMIndia	Production	✗	?
SimSYCL	Led by University of Innsbruck	Debugging / Testing	✓	✓
triSYCL	Led by community	Testing	✓	✓
neoSYCL	Led by Tohoku-University	Production	✓	?

What is the best CPU configuration ?

Toolchain x Backend x Fallback



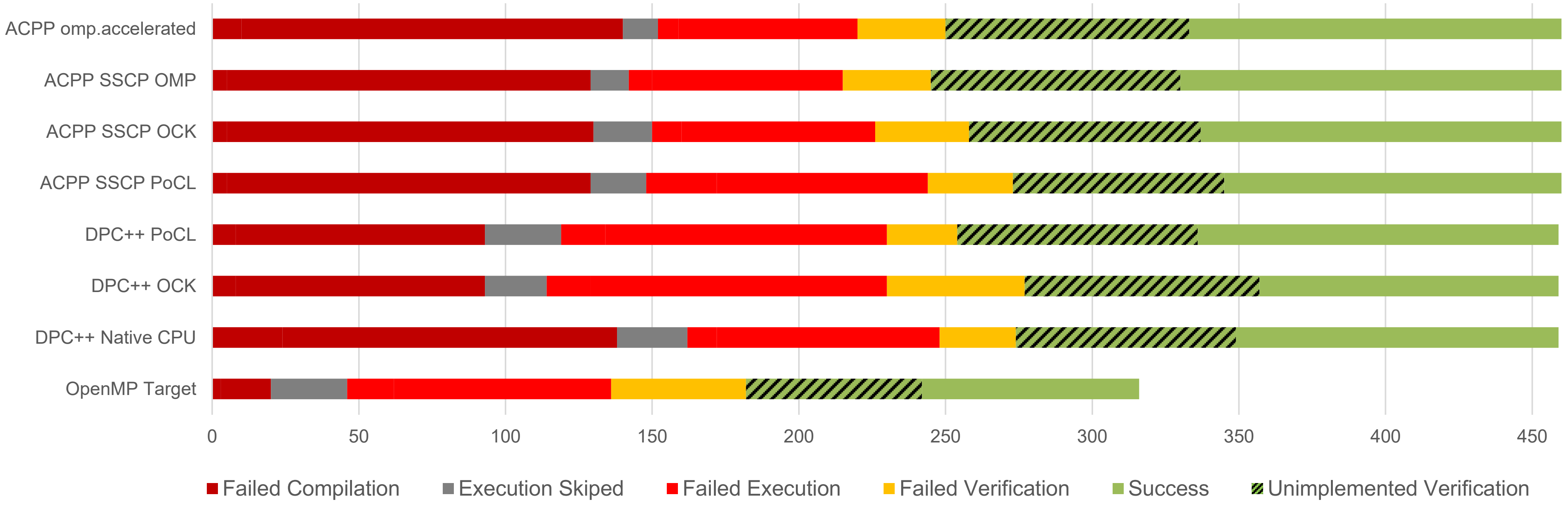
Experimental setup

HeCBench a suite of 400+ SYCL benchmarks

Machines	Micro Architecture	Frequency	NB Cores
Sapphire Rapids	X86_64	3.5 GHz	2x40
AWS Graviton 3	Neoverse-V1	2.6 GHz	1x64
Nvidia Grace	Neoverse-V2	3.3 GHz	2x72

Toolchain	Version
AdaptiveCpp	25.02.0
DPC++	v6.0.1 (Clang 19.0)
LLVM	20.1.3
PoCL	V7.0-RC2
oneAPI-construction-kit	df8de76 – V4.0
Intel OpenCL	2024.1

SYCL on AArch64



460 benchmarks in HeCBench

[320-366] benchmarks Compile on AArch64

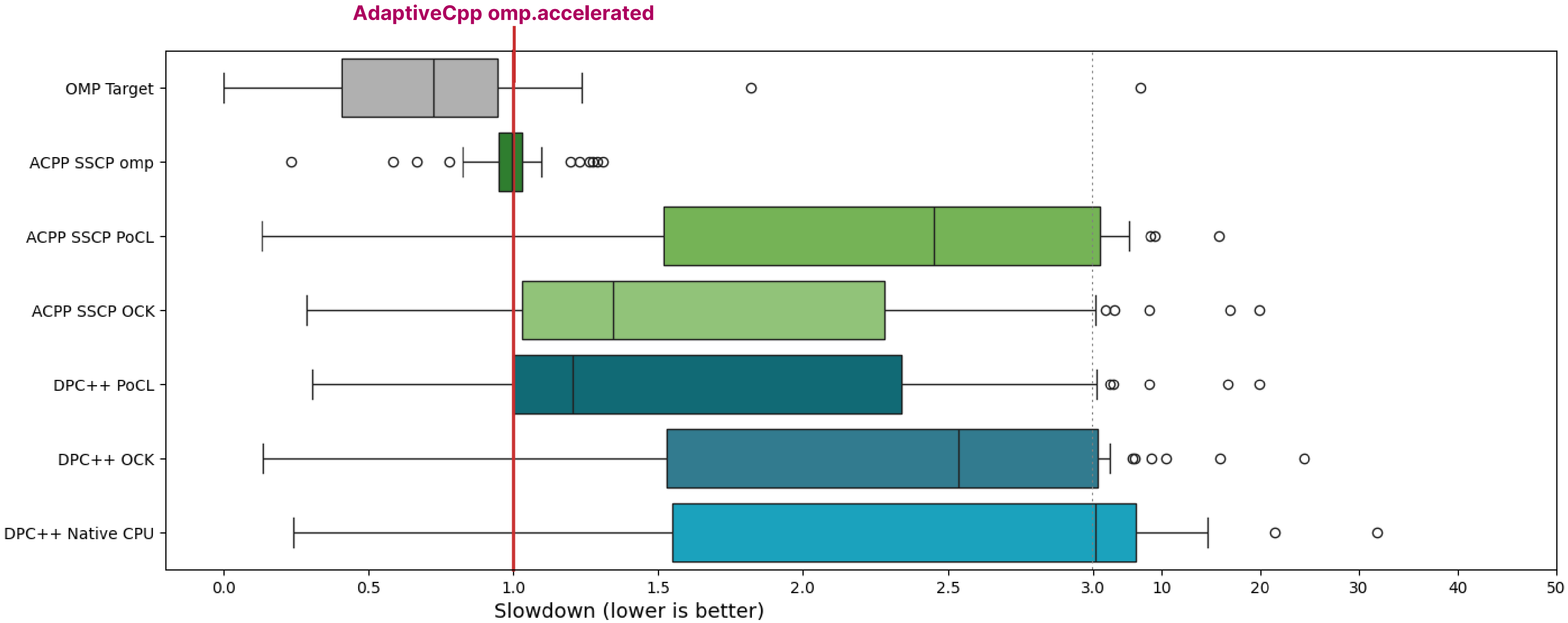
[224-248] benchmarks run on AArch64

[102-130] validates run on AArch64

40 cross validates OpenMP Target & every SYCL implementation

⚠ JIT not accounted for comparison ⚠

General overview on Graviton 3



Similar Performance on Neoverse-V2

OMP-target outperforms existing implementations

ACPP + OMP are the best SYCL configurations

POCL, OCK and **NativeCPU** backend exhibit poor performance

Performance analysis on Aarch64 backends

AdaptiveCpp SSCP OMP
force vectorization with metadata leading to non optimal decisions

HeCBench
Relies on constant work-group size optimized for GPU.

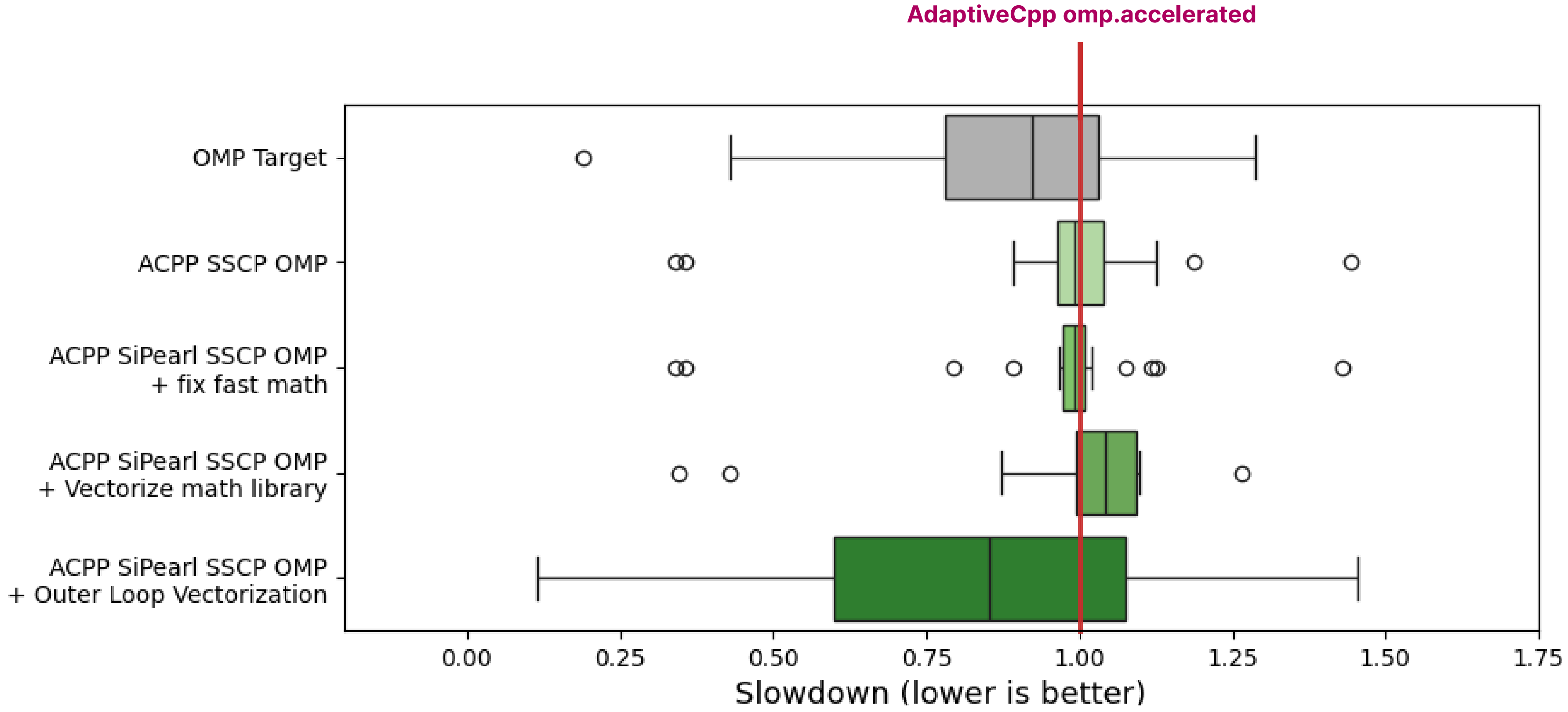
Up to to 1.5x~2x gains can be achieved by tweaking it

Comparison with X86

- Better performance for POCL and OCK backend
- DPC++ with Intel OpenCL backend outperform OpenMP Target and any other SYCL configuration. DPC++ performance is constrained by OpenCL implementation.

Improve AdaptiveCpp

Improvement of AdaptiveCPP on Neoverse-V2



Activate Fast-math flags forwarding to the JIT compilation

Improve vectorization with vectorize math library

Activate experimental Outer Loop Vectorization

Impact of HBM

Statistical Parametric Mapping Benchmark

-> Image registration calculations for the **Statistical Parametric Mapping** (SPM) system

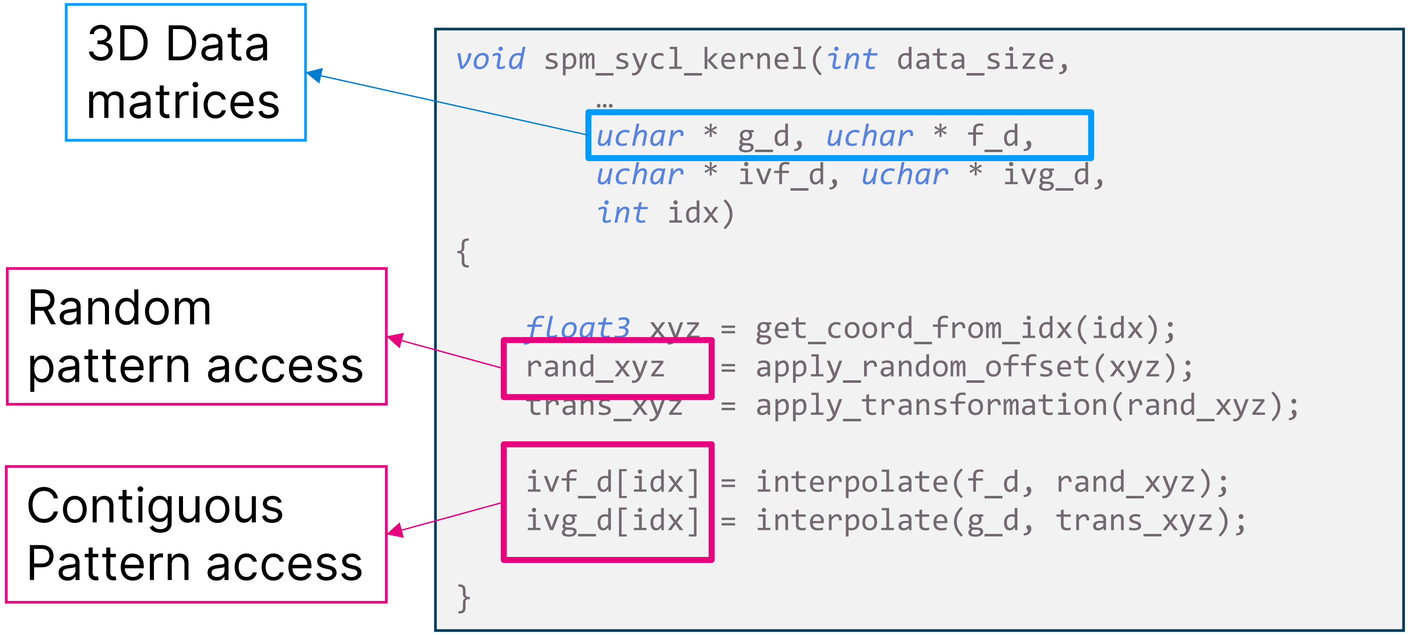
Each SYCL Work-item process an element of the 3D data grid :

- ➢ **Add a random perturbation to the element coordinate**
- ➢ **Apply a transformation to the randomized coordinate**

Two Groups of matrix :

- *ivf_d* **and** *ivg_d -> Group A*

- g_d **and** *f_d -> Group B*



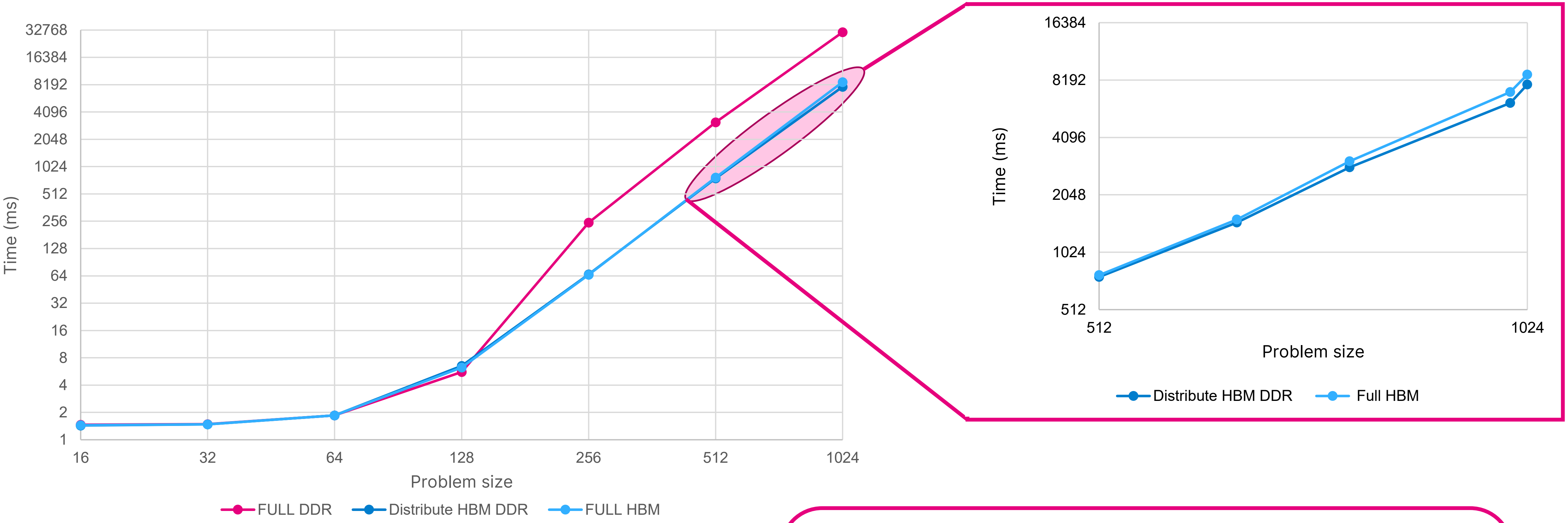
```
void spm_sycl_kernel(int data_size,
        …
        uchar * g_d, uchar * f_d,
        uchar * ivf_d, uchar * ivg_d,
        int idx)
{

    float3 xyz   = get_coord_from_idx(idx);
    rand_xyz     = apply_random_offset(xyz);
    trans_xyz    = apply_transformation(rand_xyz);

    ivf_d[idx] = interpolate(f_d, rand_xyz);
    ivg_d[idx] = interpolate(g_d, trans_xyz);

}
```

3D Data matrices

Random pattern access

Contiguous Pattern access

Memory configuration for SPM Benchmark



SiPearl upstreamed a SYCL property to make allocation numa-aware

- Allocating all memory on the HBM is not the optimal solution
- Prioritizing latency over throughput on matrix X is more optimal

Conclusion :

AdaptiveCpp is mature on AArch64

SiPearl helps the development of AdaptiveCpp

DPC++ is bounded to the performance of OpenCL

No SYCL implementation reaches performance of OMP on AArch64

Questions?