14th International Workshop on OpenCL and SYCL

IWOCL 2026

A cross-vendor implementation of PopSift using SYCL

Mohammad Fadel Al Khafaji, University of Oslo

Mohammad Fadel Al Khafaji and Carsten Griwodz, University of Oslo. Håkon Kvale Stensland, Simula Research Laboratory

May 6-8, 2026 | Heilbronn, Germany | iwocl.org

A cross-vendor implementation of PopSift using SYCL

- The problem with PopSift
- The SIFT pipeline and why it is demanding on parallel hardware.
- The SYCL port
- Correctness and performance results across NVIDIA, AMD, and Intel hardware
- Key takeaways about portability versus performance,

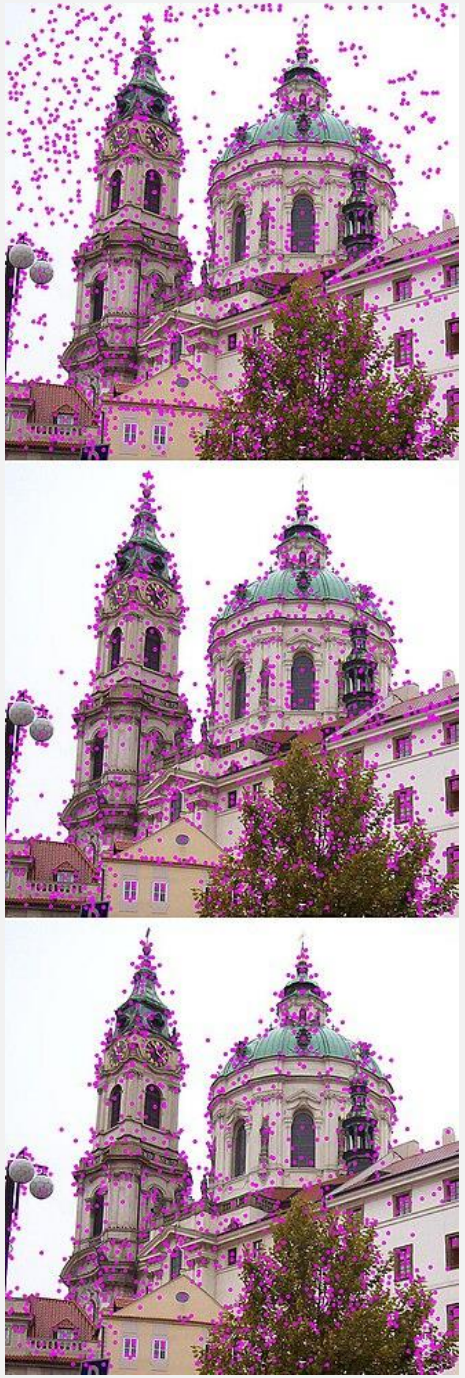
  and where this work can go next

UiO : Universitetet i Oslo

simula

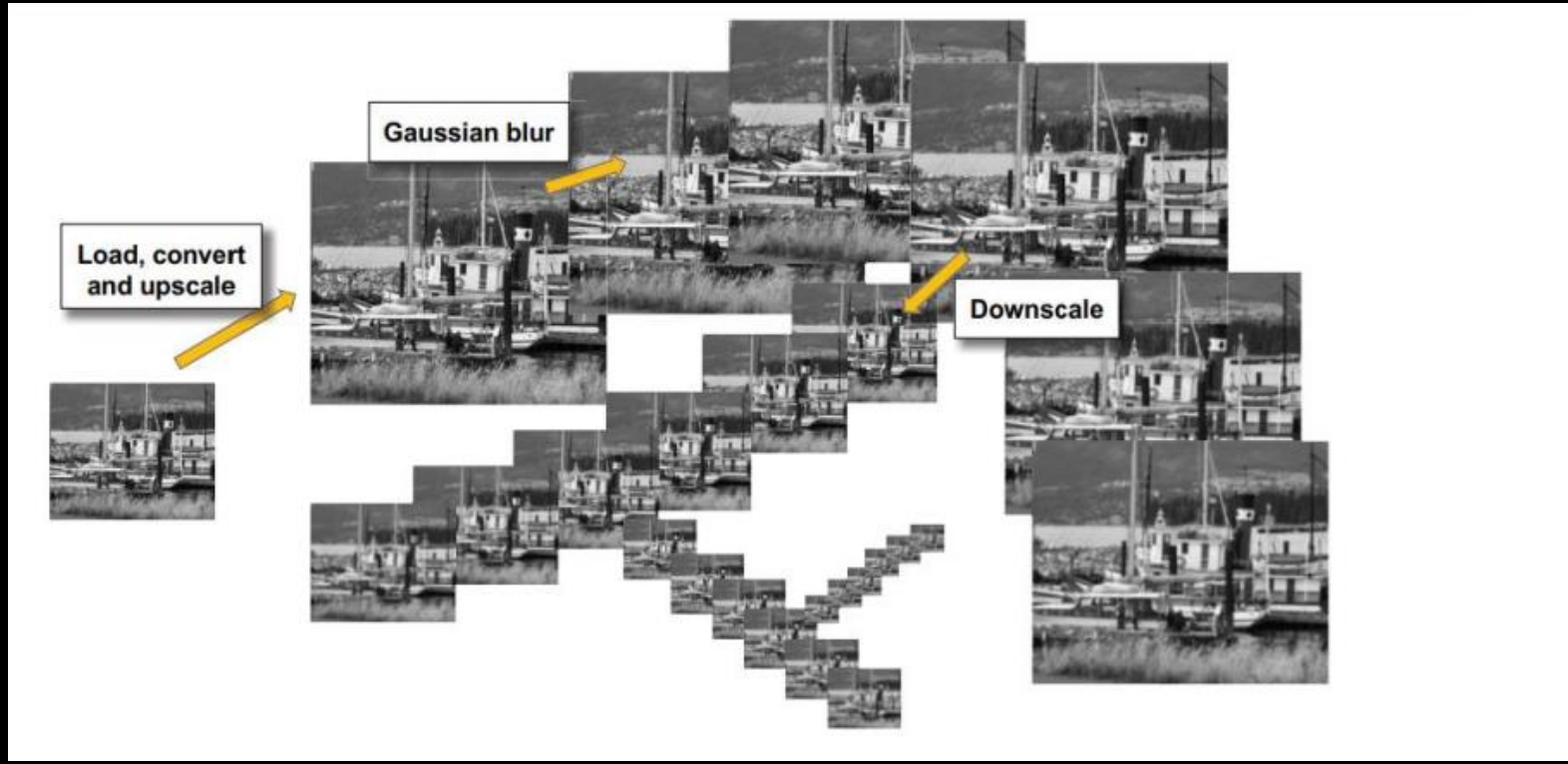
SIFT Algorithm

- The Scale Invariant Feature Transform (SIFT) introduced by D. Lowe
- Keypoint detection and descriptor extraction
- Widely used image-matching algorithms
- Compute-intensive
- PopSift (CUDA)
- PopSift-SYCL



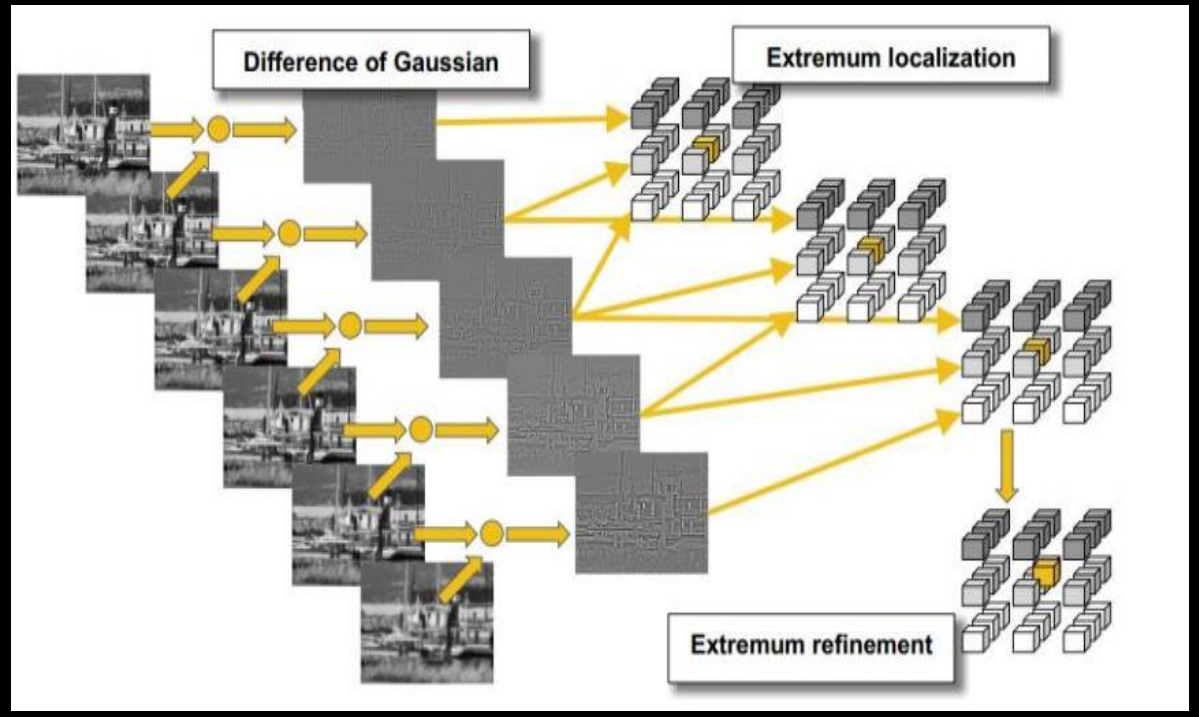
SIFT Algorithm

- Upscaling
- Gaussian pyramid



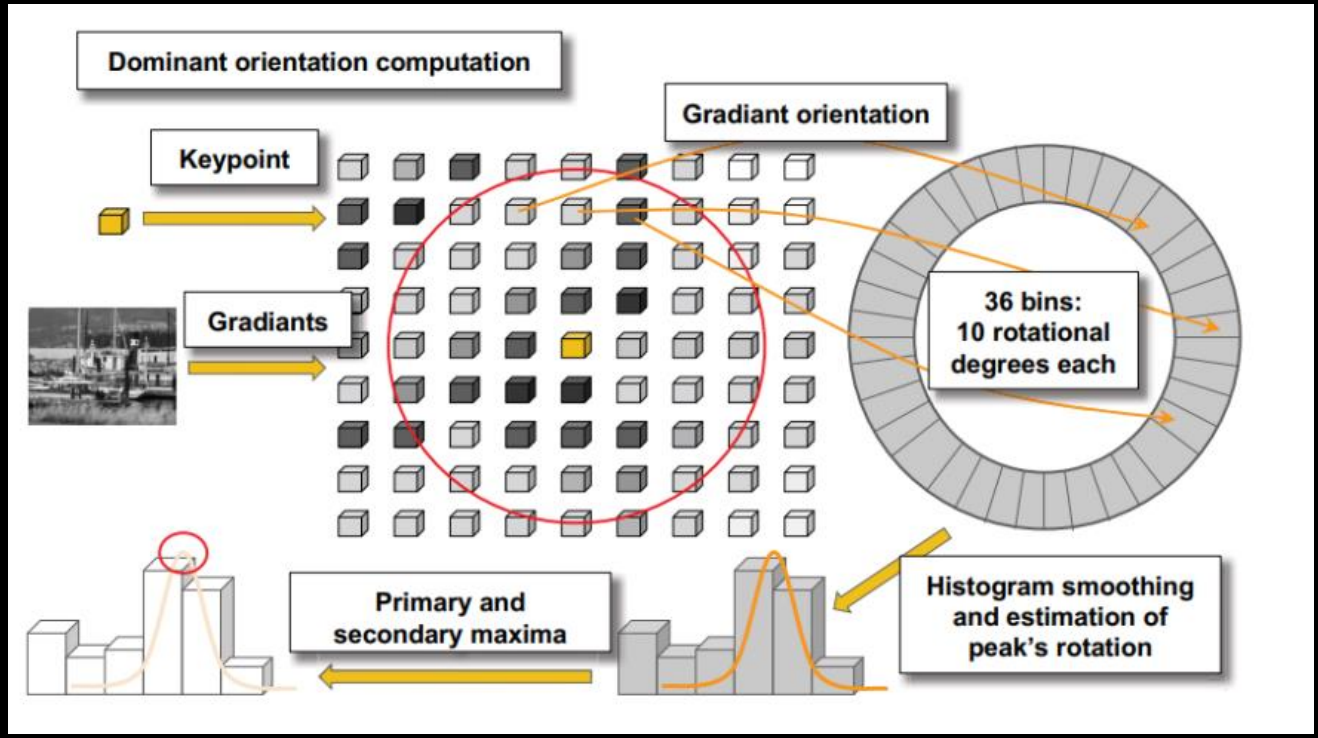
SIFT Algorithm

- Difference-of-Gaussian (DoG)
- Keypoint detection



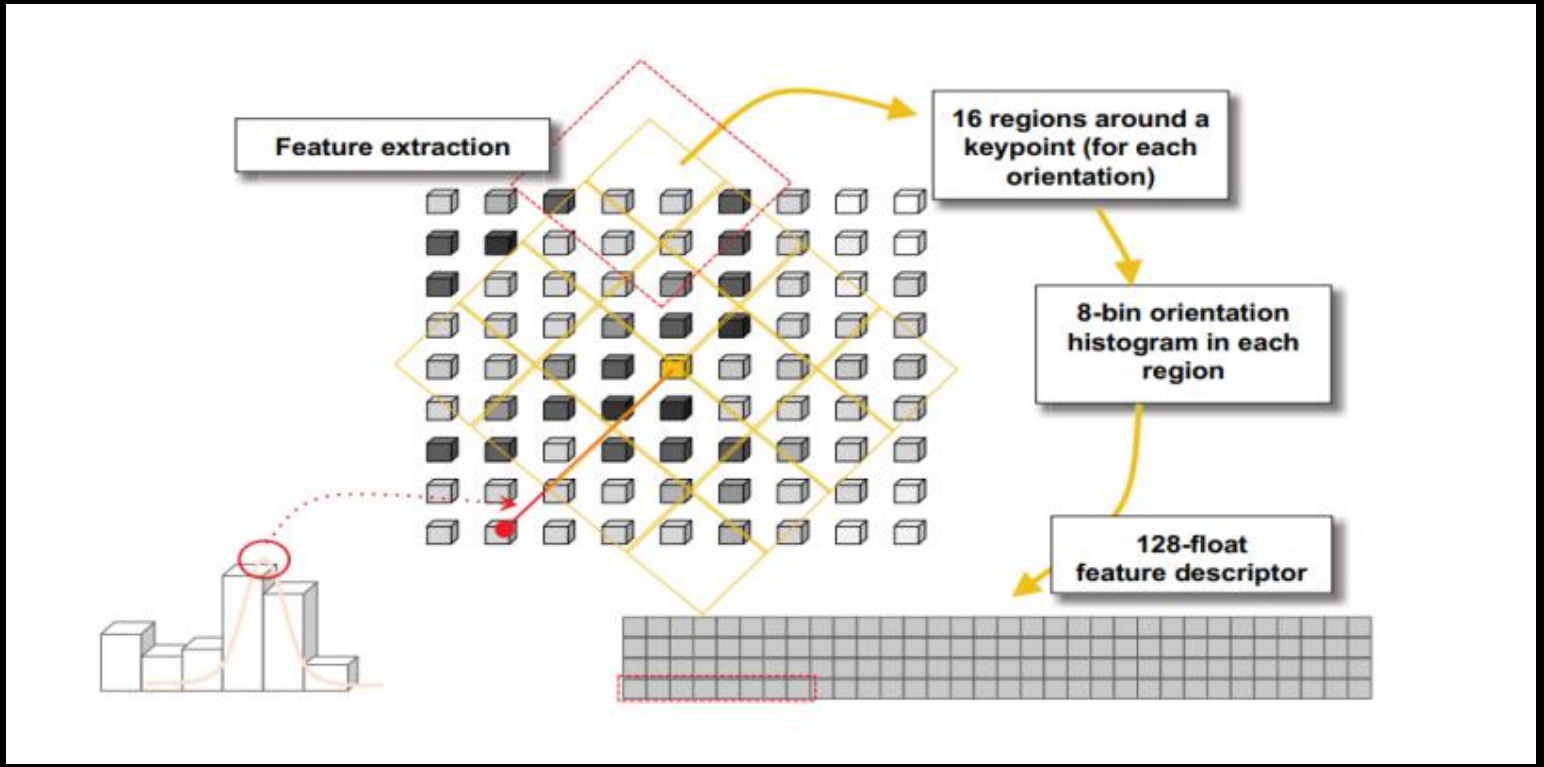
SIFT Algorithm

- Orientation assignment



SIFT Algorithm

- Descriptor Extraction
- Normalization



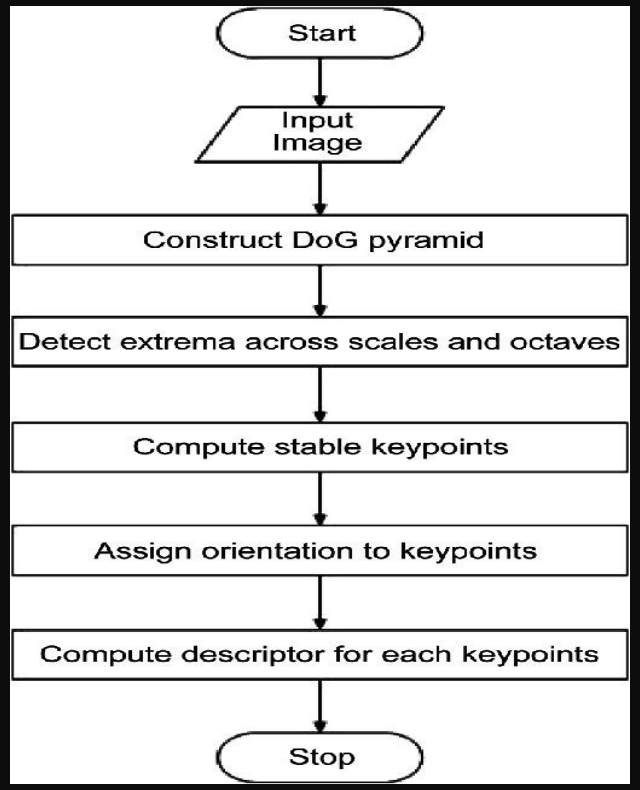
PopSift

Mixed Computational Charactersistics

- Memmory bound
- Senstive to cache
- Interpolation support

- Portability framwoks impact

- Non-uniform behaviuor



Texture support
- Normalized cordinates
- Hardware bilinear interpolation & sampling

Keypoint detection:
- Coalesced memory
- Branch-minimzed extrema testing

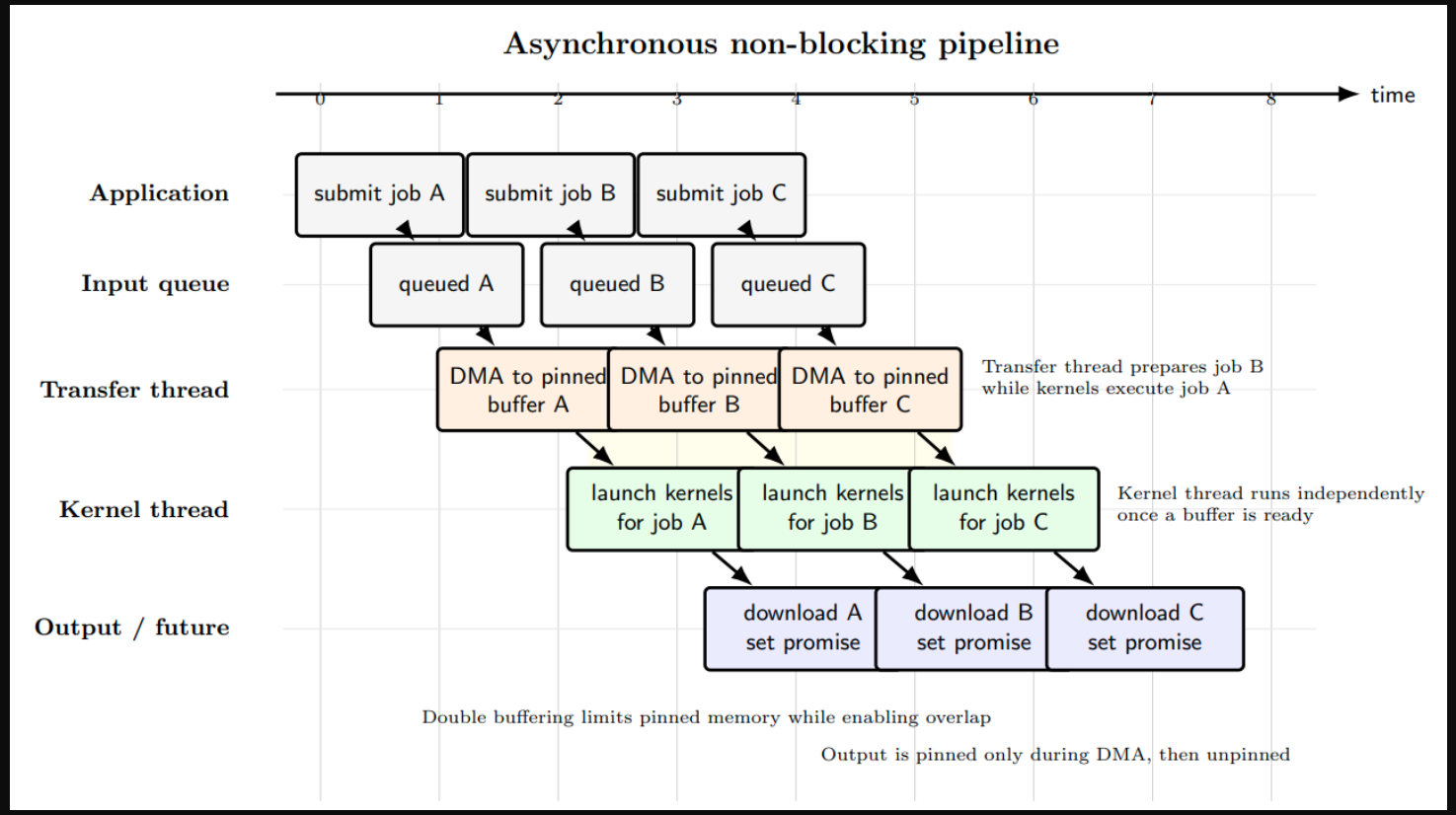
Orientation & Descriptor extraction:
- CUDA primitives

PopSift

- Asynchronous Pipeline Design
- Non-Blocking Dataflow Architecture
- Input images placed in a queue
- Two background threads:
- Host → GPU data transfer
- GPU kernel execution

- Double buffering enables overlap of:
- Data transfer
- Computation

- Uses std::future for asynchronous results



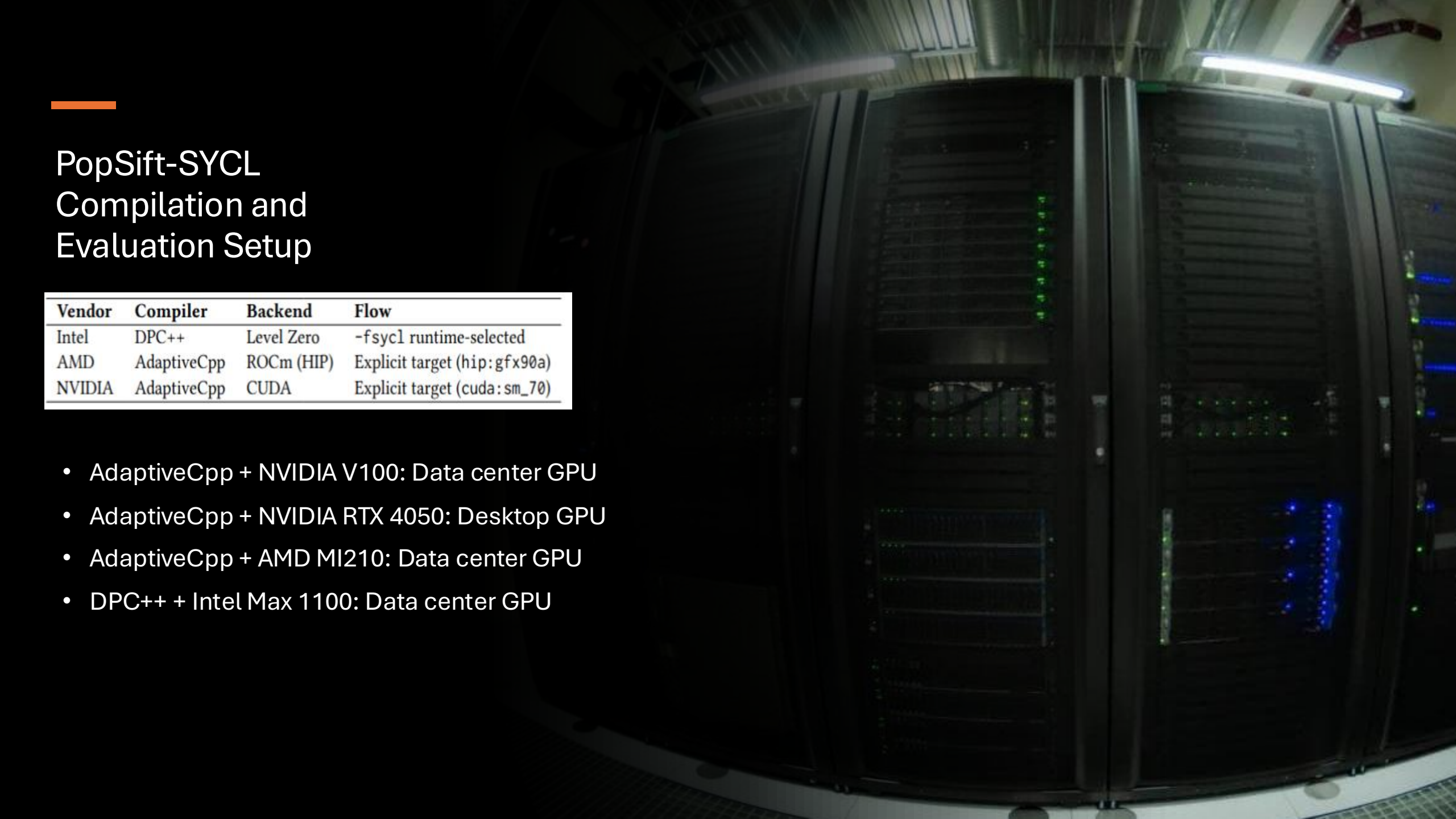
SSMC-First PopSift-SYCL: Portable SIFT Across Intel DPC++ and AdaptiveCpp

- SSMC goal: one SYCL codebase for Intel DPC++ and AdaptiveCpp.
- Fair comparison: kept core PopSift/SIFT logic, simplified host-side orchestration.
- Portability choices: synchronous host flow, no pinned-memory/double-buffer pipeline, no *sycl::image* dependency.
- Implementation path: SYCL buffers + manual interpolation + event-driven synchronization.
- Result: full SIFT chain preserved; bulk queue/event scheduling was competitive with CUDA in some stages.

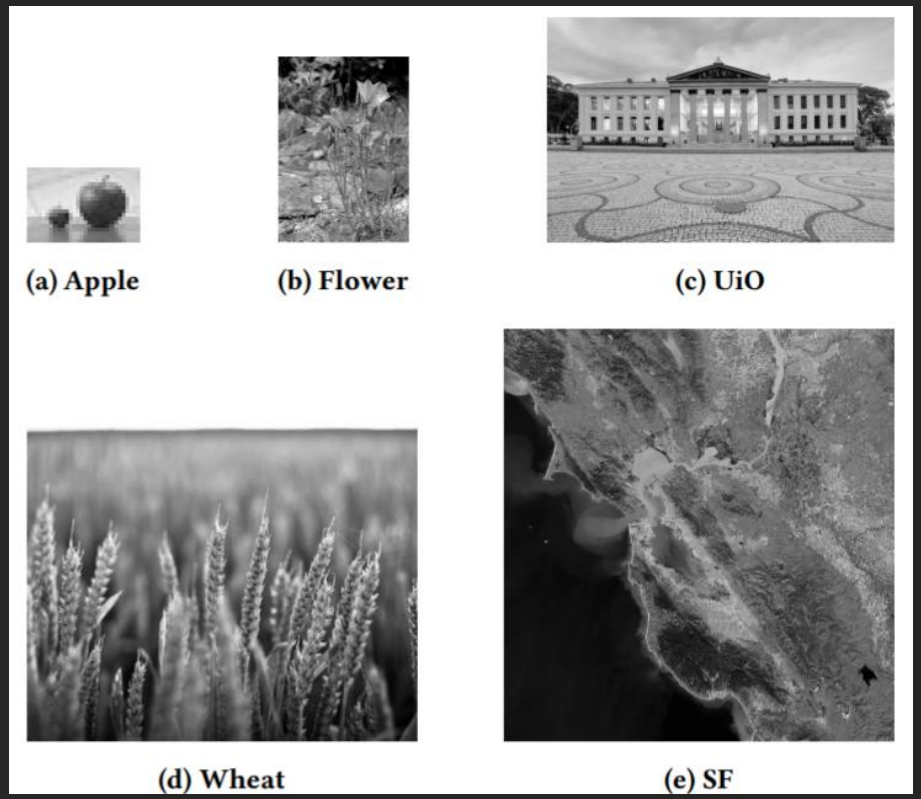
PopSift-SYCL Compilation and Evaluation Setup

Vendor	Compiler	Backend	Flow
Intel	DPC++	Level Zero	`-fsycl` runtime-selected
AMD	AdaptiveCpp	ROCm (HIP)	Explicit target (`hip:gfx90a`)
NVIDIA	AdaptiveCpp	CUDA	Explicit target (`cuda:sm_70`)

- AdaptiveCpp + NVIDIA V100: Data center GPU
- AdaptiveCpp + NVIDIA RTX 4050: Desktop GPU
- AdaptiveCpp + AMD MI210: Data center GPU
- DPC++ + Intel Max 1100: Data center GPU



Workload Coverage and Cross-Platform Correctness



apple.pgm: 30×20 pixels (6 features) — minimal workload

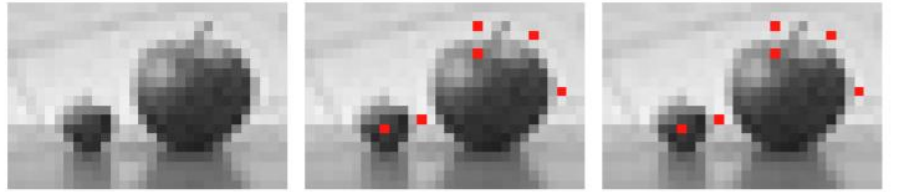
blomst.pgm: 360×515 pixels (~2,200 features) — low resolu- tion

uio.pgm: 1250×812 pixels (~7,200 features) — medium reso- lution

Wheat.pgm: 1939×2048 pixels (~7,700 features) — high reso- lution

SF.pgm: 1939×2048 pixels (~21,000 features) — high resolu- tion & feature-rich

Image	Feature Points		Descriptors	
	Absolute	Relative	Absolute	Relative
apple.pgm	0	0.00%	0	0.00%
blomst.pgm	−3	−0.13%	−33	−1.23%
uio.pgm	+1	+0.01%	−31	−0.36%
Wheat.pgm	−6	−0.08%	−53	−0.60%
SF.pgm	−30	−0.14%	−398	−1.49%

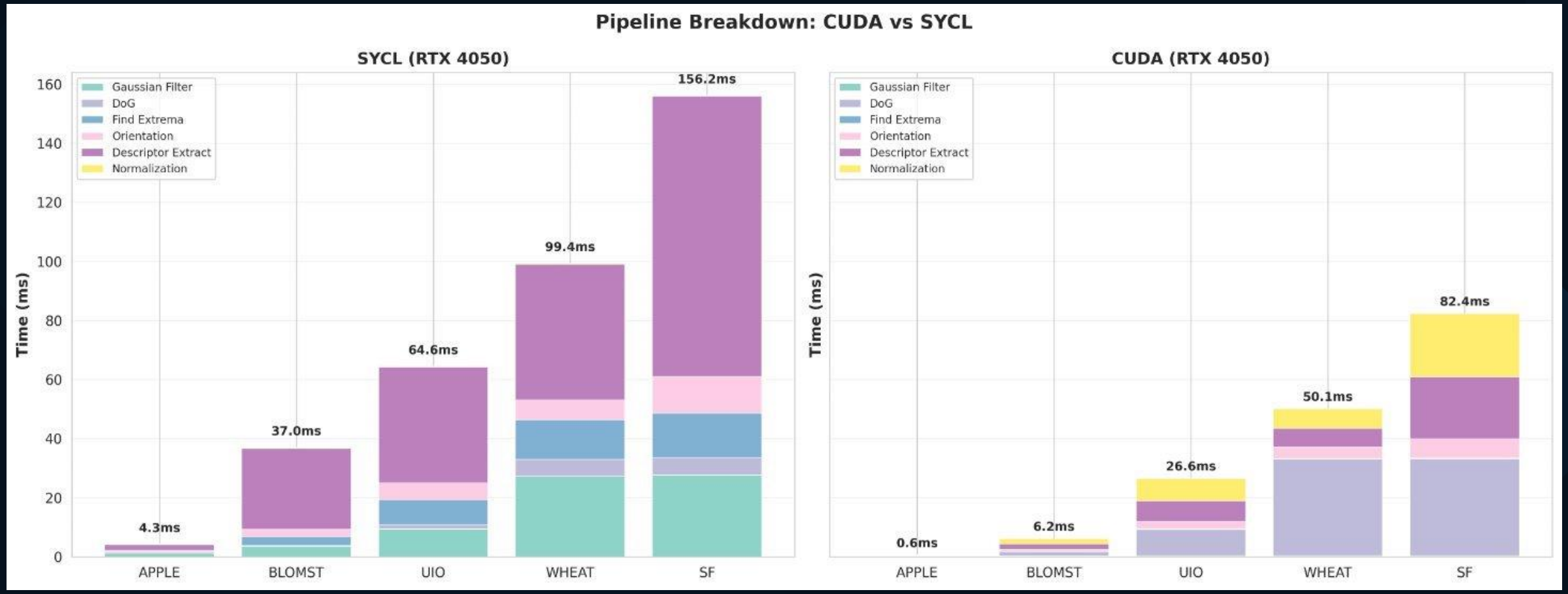


Performance Gap Analysis: CUDA vs Portable SYCL

Graph Explanation (Pipeline Breakdown: CUDA vs SYCL)

This stacked bar chart shows execution time across six pipeline stages for five test images. Key takeaways:

- CUDA outperforms SYCL across all images—from 4.3ms to 156.2ms total time
- Descriptor Extract and Find Extrema dominate SYCL's execution time, indicating memory bottlenecks
- Performance gap grows with image size, showing SYCL scales poorly on larger workloads



SYCL Slowdown Factor (SYCL time / CUDA time)

Stage	APPLE	BLOMST	UIO	WHEAT	SF
Gaussian Filter	6.62×	14.02×	29.75×	95.05×	89.28×
DoG	1.99×	0.20×	0.16×	0.18×	0.18×
Find Extrema	6.09×	41.20×	45.93×	92.63×	97.79×
Orientation	4.49×	3.90×	2.25×	1.72×	1.89×
Descriptor Extract	20.69×	14.89×	5.66×	7.24×	4.50×
Normalization	1.39×	0.17×	0.04×	0.06×	0.01×
Total Pipeline	**6.87×**	**6.00×**	**2.43×**	**1.98×**	**1.90×**

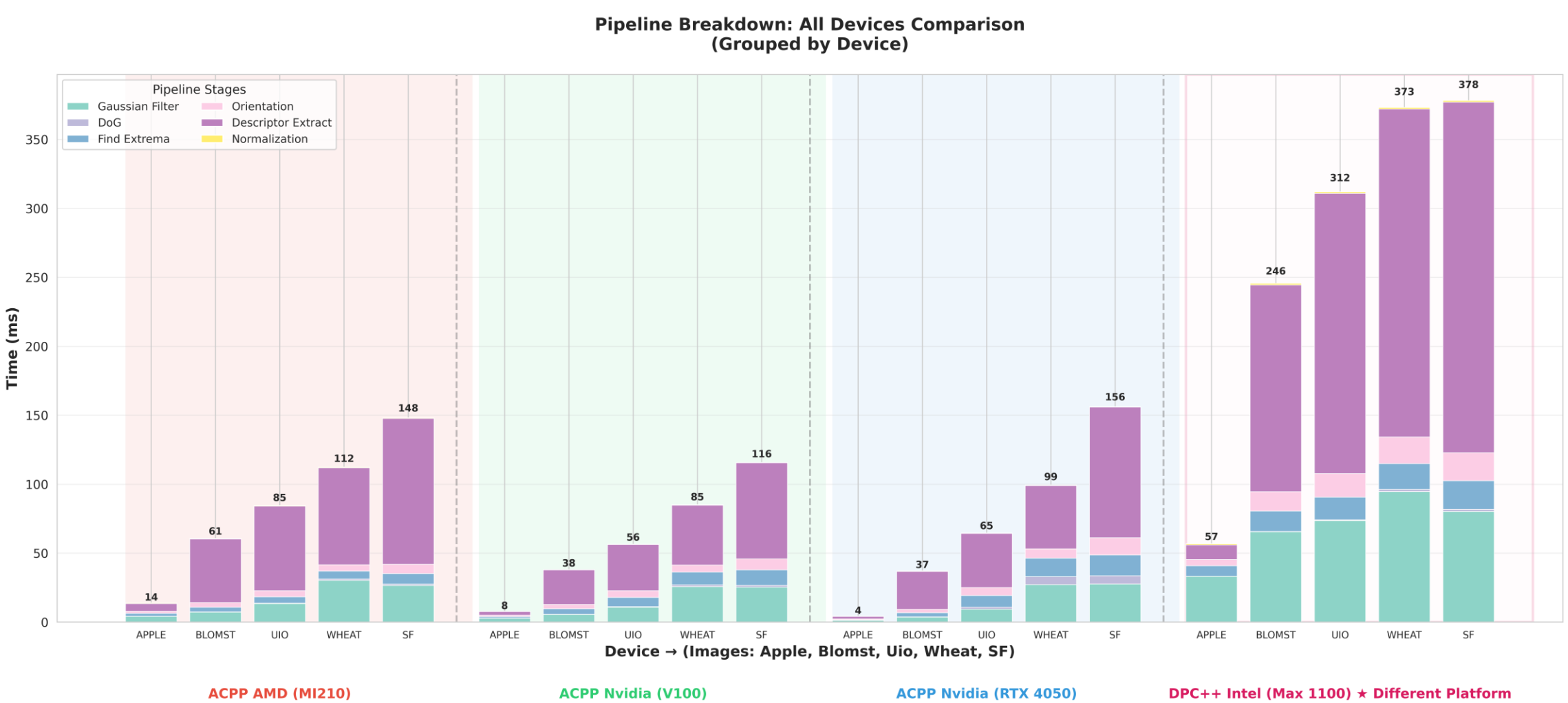
Color code: Green (≤1.0×) | Yellow (1.0-2.5×) | Red (>2.5×)

Performance Gap Analysis: CUDA vs Portable SYCL

Table Explanation (SYCL Slowdown Factor)

This table shows the performance ratio (SYCL / CUDA) per pipeline stage:

- Extreme Slowdowns (Red): Gaussian Filter and Find Extrema are 6–98× slower due to CUDA's texture memory advantage
- Competitive Performance (Green): DoG and Normalization are nearly equivalent (0.01–1.99×), showing good portability for compute-heavy stages
- Overall Impact: SYCL is 1.9–6.9× slower depending on image size and memory access patterns


Pipeline Breakdown: All Devices Comparison
(Grouped by Device)
Pipeline Stages
Gaussian Filter
DoG
Find Extrema
Orientation
Descriptor Extract
Normalization
Time (ms)
0
50
100
150
200
250
300
350
14
61
85
112
148
8
38
56
85
116
4
37
65
99
156
57
246
312
373
378
APPLE
BLOMST
UIO
WHEAT
SF
Device → (Images: Apple, Blomst, Uio, Wheat, SF)
ACPP AMD (MI210)
ACPP Nvidia (V100)
ACPP Nvidia (RTX 4050)
DPC++ Intel (Max 1100) ★ Different Platform

Conclusion: Portability Proven, Performance Trade-Offs Remain

- One SYCL codebase works across vendors.
- Correctness is consistent and deterministic.
- CUDA is still faster overall.
- Slowdown is stage-specific, not uniform.
- SYCL for portability; CUDA for peak NVIDIA speed.

Research Trade-Offs and Future Directions

- Prioritized clean SSMC evaluation over peak performance.
- Production builds could add async pipeline and backend interop, but this diverges from pure single-source portability.
- Top priority: portable texture-memory support (highest impact on slowdown).
- Second: restore async host-device pipeline engineering while preserving cross-vendor semantics.
- Third: broader benchmarking across architectures and toolchain versions.
- Fourth: analyze numerical sensitivity and floating-point behavior across backends.

Practical Guidance for Cross-Platform Porting

- SYCL default; CUDA fast path only where needed.
- Validate on features, not bitwise identity.
- Benchmark cold-start and steady-state separately.
- Stage-level timing breakdown to find bottlenecks.
- Fixed workloads, reproducible stacks, full transparency.

Key Takeaway: Portable SIFT, Transparent Trade-Offs

- One SYCL codebase runs correctly across NVIDIA, AMD, and Intel.
- Performance penalty is real but localized to memory/sampling stages.
- Signals where SYCL works and where the ecosystem needs to mature.
- Practical path forward for hardware-independent deployments.
- Portability and performance are not mutually exclusive—they require explicit trade-off decisions.

UiO : Universitetet i Oslo

simula