COMPILER-ASSISTED ND-RANGE PARALLEL-FOR IMPLEMENTATIONS ON CPU IN HIPSYCL

POSTER BY JOACHIM MEYER (SAARLAND UNIVERSITY) AND
AKSEL ALPAY, HOLGER FRÖNING, VINCENT HEUVELINE (HEIDELBERG UNIVERSITY)

jmeyer@cs.uni-saarland.de

TARGETING PERFORMANCE PORTABILITY WITH `nd_range` `parallel_for`

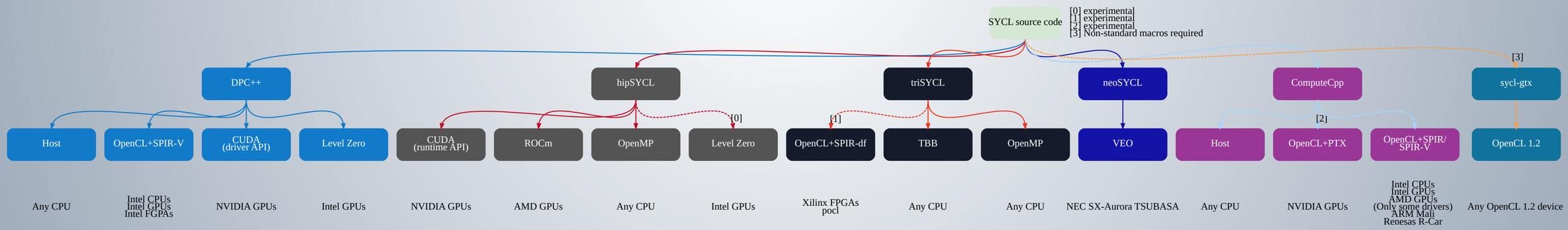
- CUDA, OpenCL, SYCL, .. offer hierarchical execution models
- The hierarchy is tailored to GPUs' architecture
  - Work-item (Thread)
  - Sub-group (Warp)
  - Work-group (Block ⇒ SM)
  - Global (Grid)
- High costs for work-group synchronization on CPUs!

TARGETING PERFORMANCE PORTABILITY WITH `nd_range` `parallel_for`

- CUDA, OpenCL, SYCL, .. offer hierarchical execution models
- The hierarchy is tailored to GPUs' architecture
  - Work-item (Thread)
  - Sub-group (Warp)
  - Work-group (Block ⇒ SM)
  - Global (Grid)
- **High costs for work-group synchronization on CPUs!**

HIPSYCL

- Multi-target SYCL Implementation by Uni Heidelberg
- Uses Clang HIP, CUDA and DPC++ augmented by plugin for GPU
- OpenMP CPU backend (traditionally library-only)



WORK-GROUP SYNCHRONIZATION

```
cgh.parallel_for(sycl::nd_range<1>{global_size, group_size},
  [=](sycl::nd_item<1> item) noexcept {
    const auto lid = item.get_local_id(0);
    scratch[lid] = acc[item.get_global_id()];
    for(size_t i = group_size / 2; i > 0; i /= 2) {
      item.barrier();
      if(lid < i) scratch[lid] += scratch[lid + i];
    }

    if(lid == 0) acc[item.get_global_id()] = scratch[lid];
  });
```

WORK-GROUP SYNCHRONIZATION

```
1  cgh.parallel_for(sycl::nd_range<1>{global_size, group_size},
2    [=](sycl::nd_item<1> item) noexcept {
3      const auto lid = item.get_local_id(0);
4      scratch[lid] = acc[item.get_global_id()];
5      for(size_t i = group_size / 2; i > 0; i /= 2) {
6        item.barrier();
7        if(lid < i) scratch[lid] += scratch[lid + i];
8      }
9
10     if(lid == 0) acc[item.get_global_id()] = scratch[lid];
11   });
```

WORK-GROUP SYNCHRONIZATION – GPU

- Execution of many (mostly) independent threads
  → Forward-Progress guarantees
- `__synchthreads()`

WORK-GROUP SYNCHRONIZATION – CPU

- 1 thread : 1 work-item (< v0.9)
  - #pragma omp barrier

⚡Many threads → OS / scheduling overhead

⚡No vectorization across work-items

WORK-GROUP SYNCHRONIZATION – CPU

- 1 thread : 1 work-item (< v0.9)
  - #pragma omp barrier

  ⚡Many threads → OS / scheduling overhead

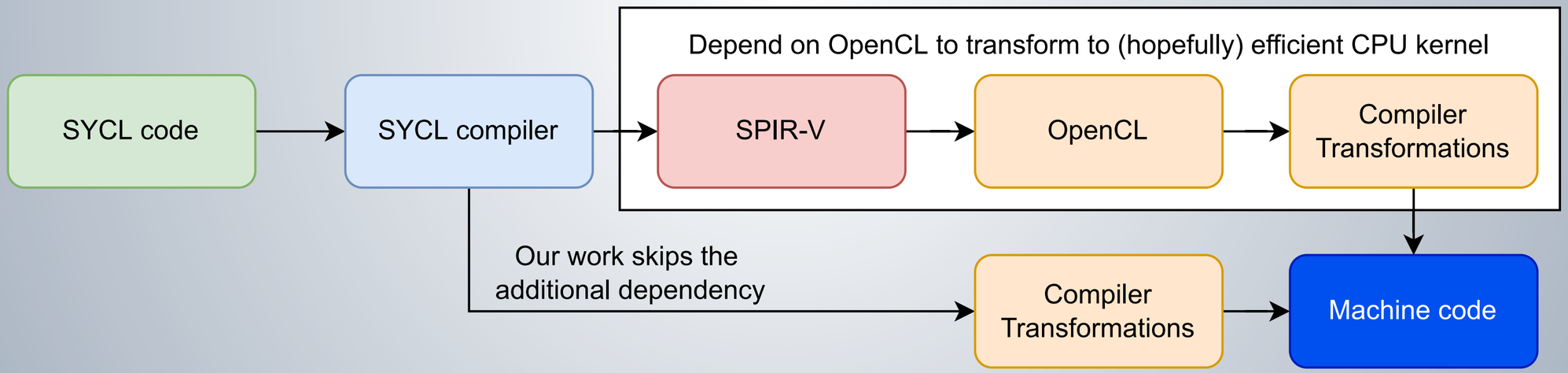
  ⚡No vectorization across work-items
- Boost.Fiber? (>= 0.9)
  - Lightweight threads + synchronization
  - Can optimize barrier-free kernels!

  ⚡Scheduling overhead

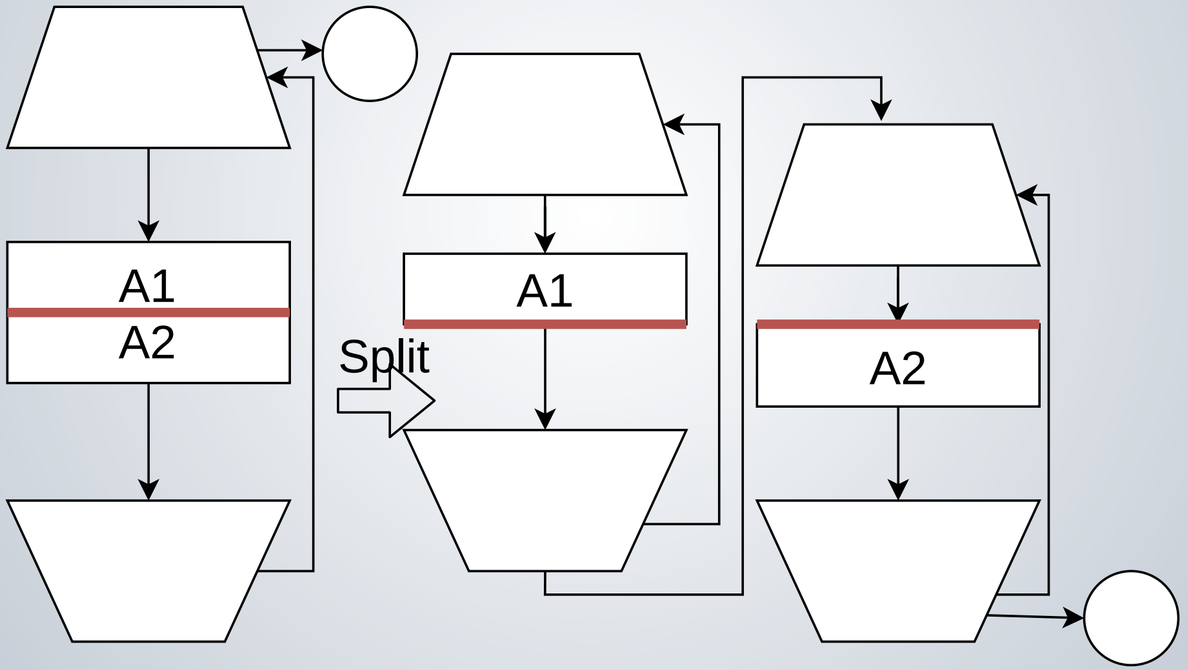
  ⚡Limited vectorization across work-items

OUR WORK: COMPILER-AIDED LOOP-FISSION

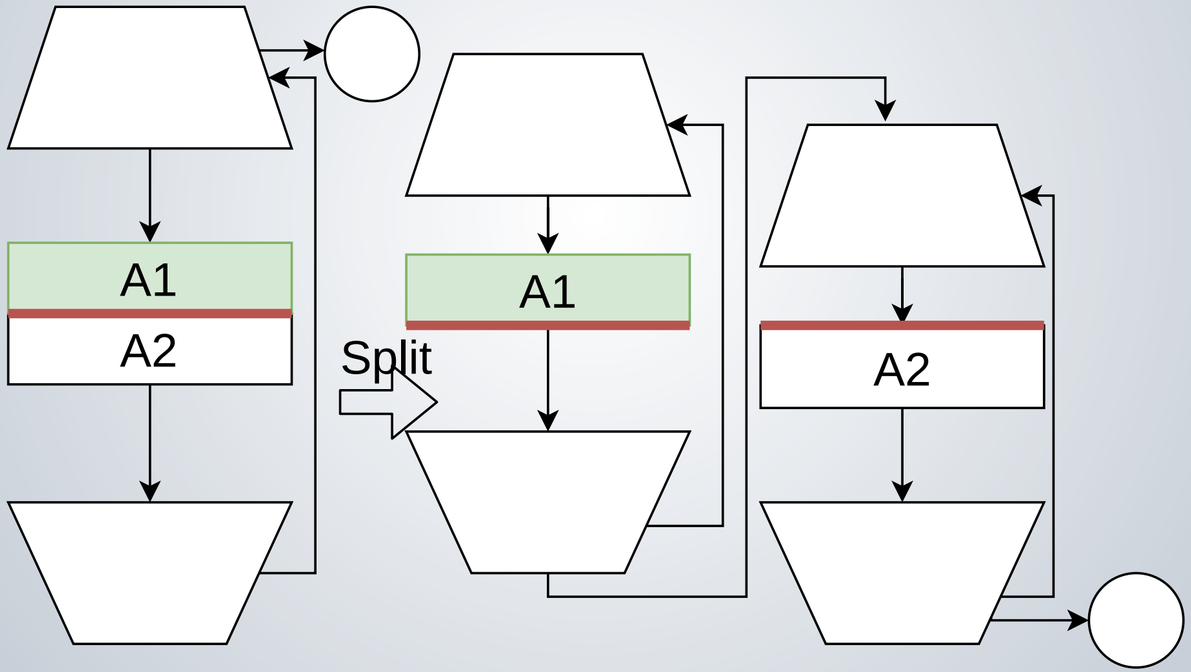
- Threading on the **work-group level**
- Loop over work-items in a single thread
- Compiler extension to **split work-item loop** at barriers
- Support targets that Clang supports OpenMP on, **no OpenCL runtime** required



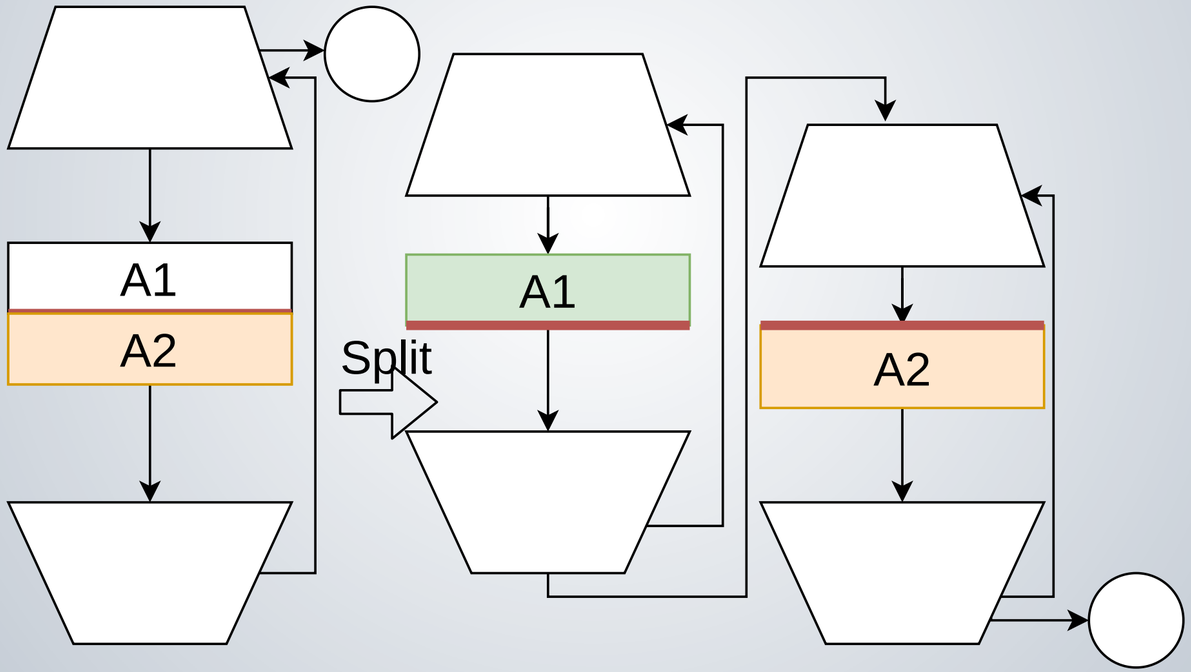
POCL'S LOOP-FISSION APPROACH



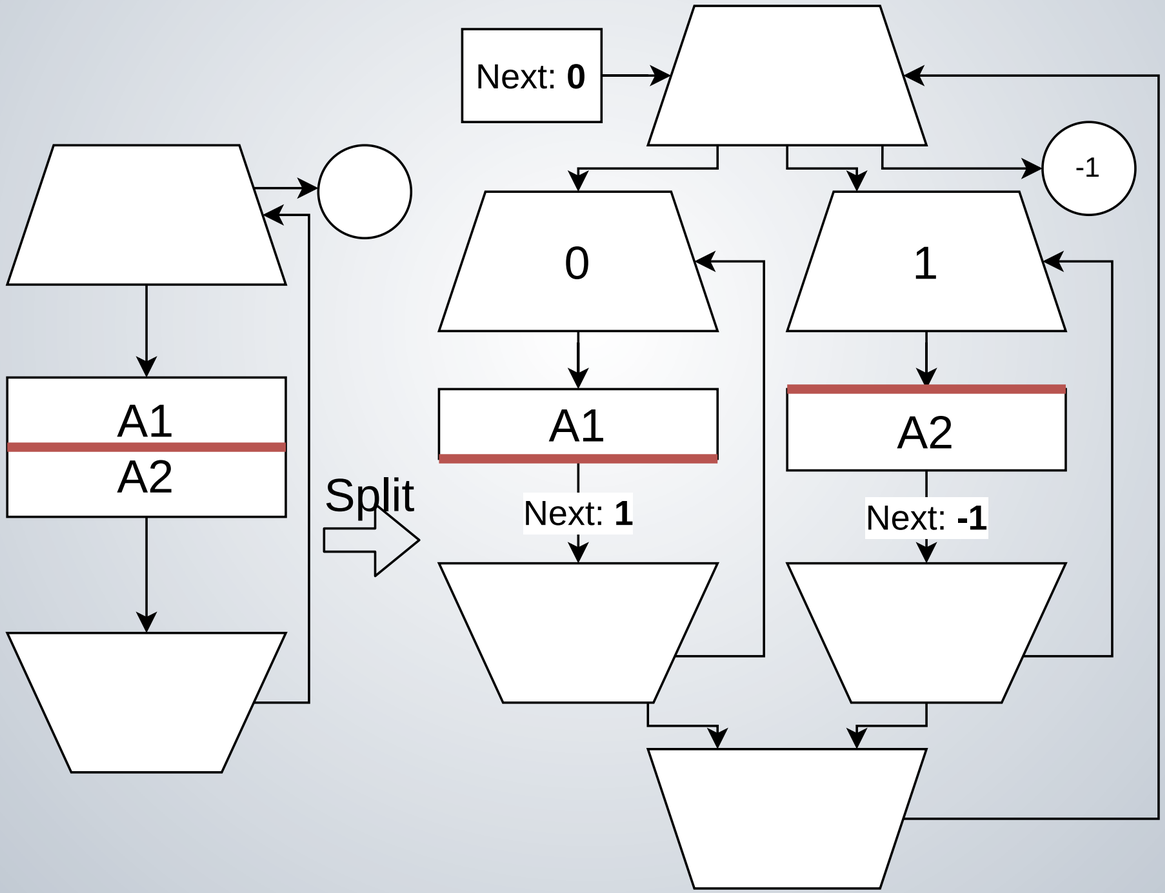
POCL'S LOOP-FISSION APPROACH



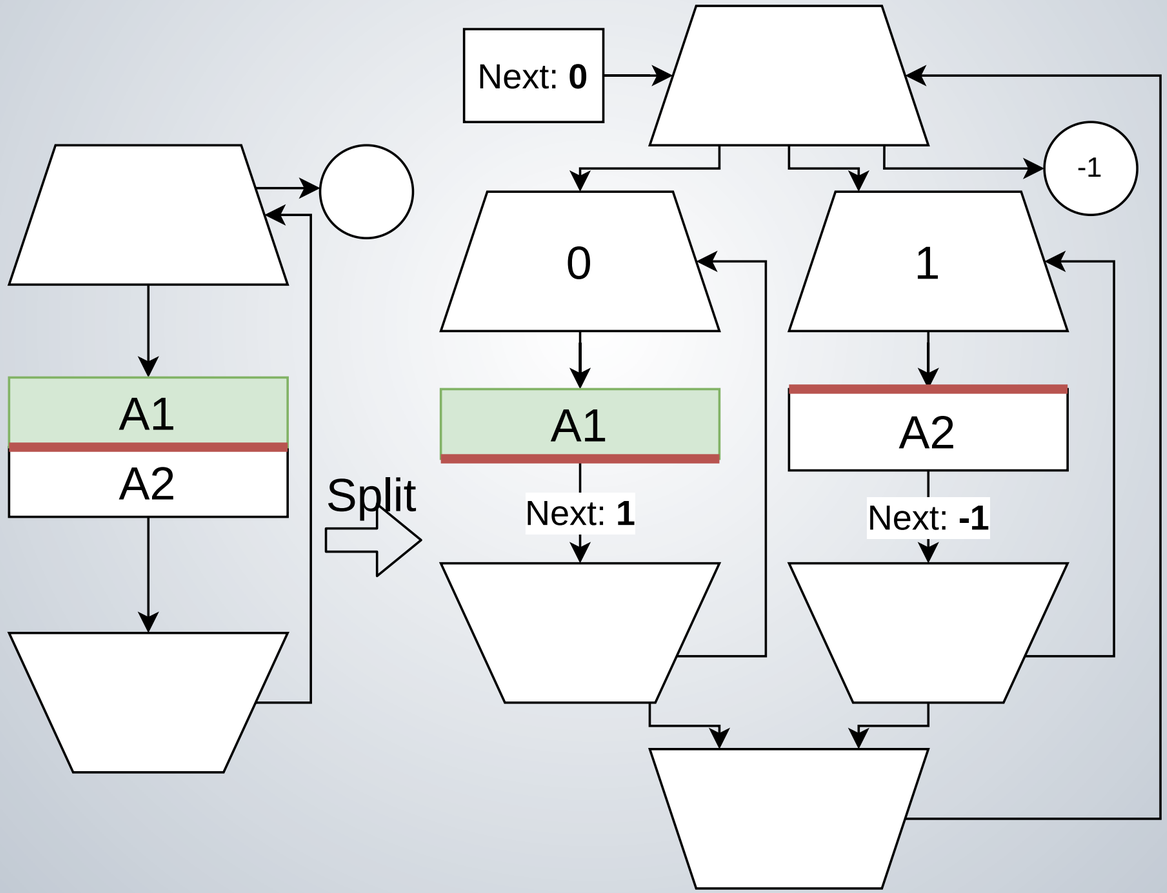
POCL'S LOOP-FISSION APPROACH



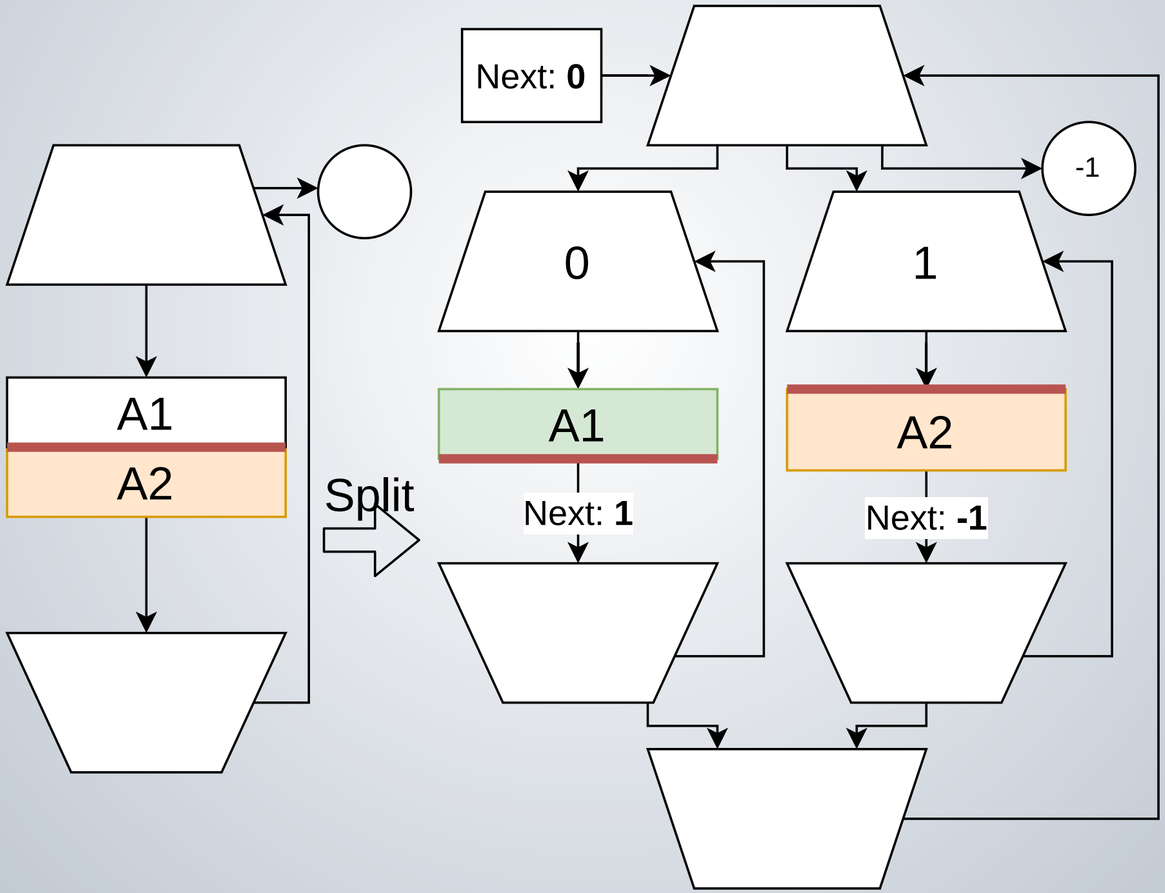
CBS LOOP-FISSION APPROACH



CBS LOOP-FISSION APPROACH



CBS LOOP-FISSION APPROACH



POCL – SEMANTIC PROBLEM

```
1  [=](sycl::nd_item<1> item) noexcept {
2    const auto lid = item.get_local_id(0);
3
4    scratch[lid] = acc[item.get_global_id()];
5    item.barrier();
6
7    for(size_t i = 0; i < 2 + lid; i++) {
8      scratch[lid] += i;
9      // only call the barrier if all work-items still run the loop.
10     if(i < 2) item.barrier();
11   }
12   acc[item.get_global_id()] = scratch[lid];
13 }
```

POCL – SEMANTIC PROBLEM

```
1  [=](sycl::nd_item<1> item) noexcept {
2    const auto lid = item.get_local_id(0);
3
4    scratch[lid] = acc[item.get_global_id()];
5    item.barrier();
6
7    for(size_t i = 0; i < 2 + lid; i++) {
8      scratch[lid] += i;
9      // only call the barrier if all work-items still run the loop.
10     if(i < 2) item.barrier();
11   }
12   acc[item.get_global_id()] = scratch[lid];
13 }
```

BENCHMARK – SYSTEMS

- Fujitsu A64FX "a64fx"
  - 1x 1.8GHz 48-core CPU
  - 512bit SVE
  - 32GB HBM2 RAM
- Marvell ThunderX2 "xci"
  - 2x 2.1GHz 32-core, 128-threads CPU
  - NEON
  - 256GB DDR4-2666MHz

Provided by

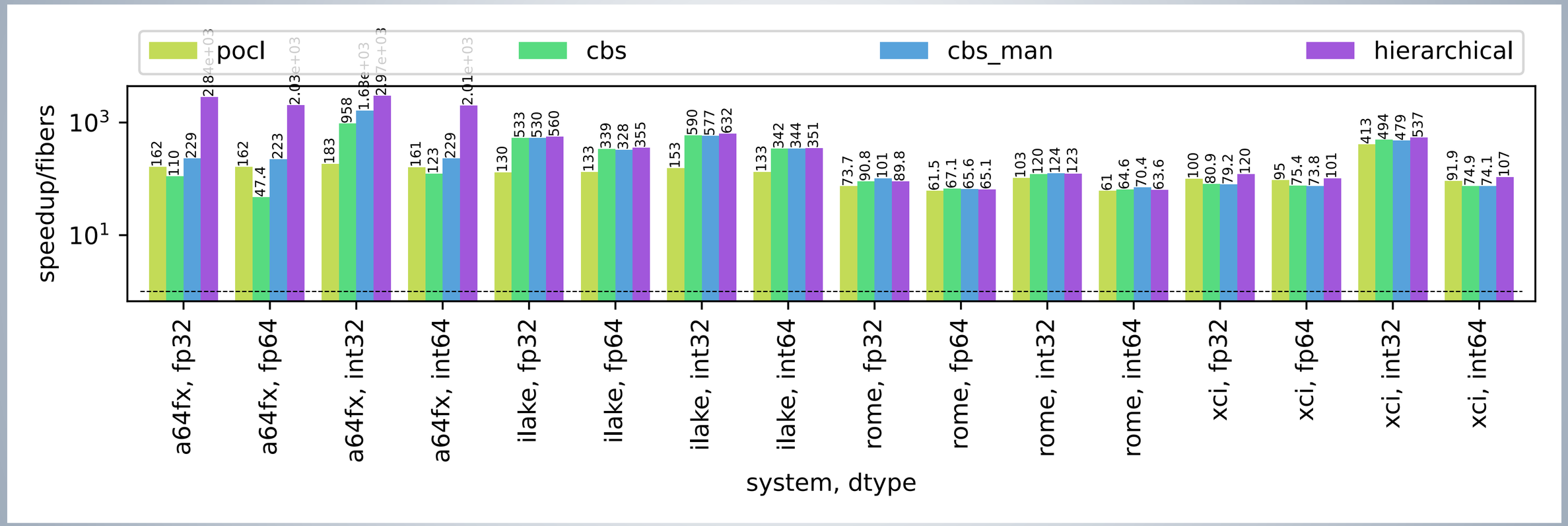
Isambard 2 UK

National Tier-2

HPC Service

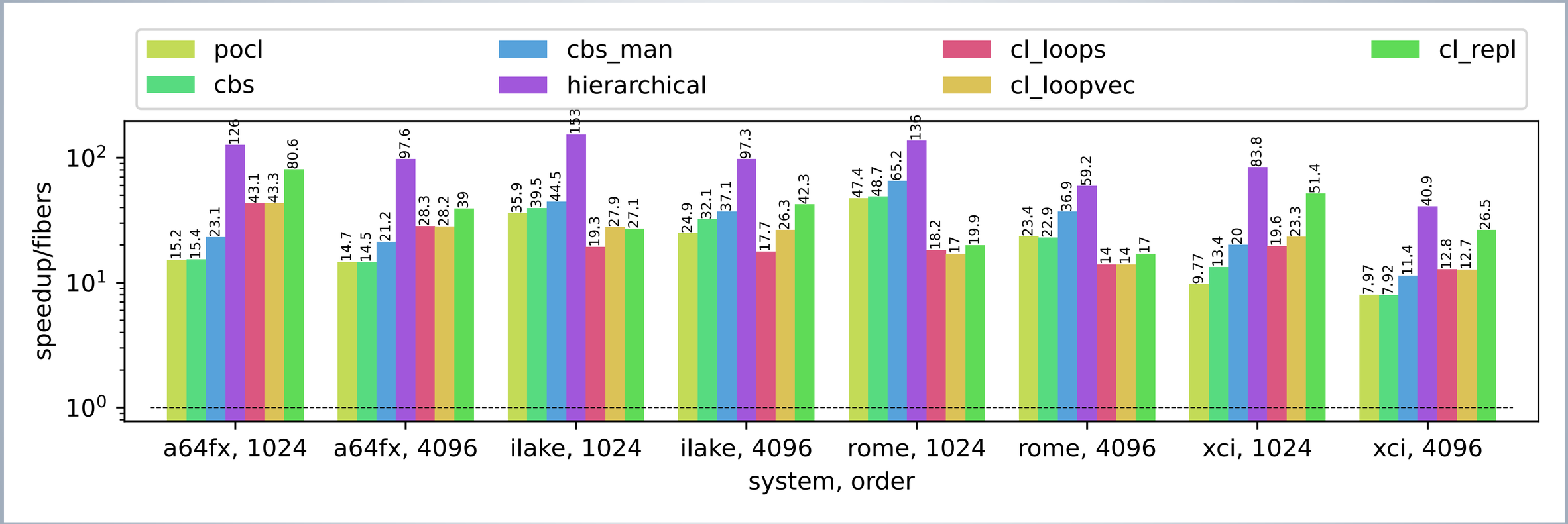
- Intel Xeon Gold 6338 "ilake"
  - 2x 2.00GHz 32-core, 64-threads Icelake CPU
  - AVX512
  - 256GB DDR4 RAM
- AMD Epyc 7442 "rome"
  - 2x 2.25GHz 64-core, 128-threads Rome CPU
  - AVX2
  - 256GB DDR4-3200Mhz

SYCL-BENCH REDUCTION



LLVM 14-git, Boost 1.77, Speedup of respective max. throughput

DGEMM



LLVM 14-git, Boost 1.77, Speedup of respective max. throughput

POCL VS. CBS

	POCL	CBS
SYCL conforming semantic	❌	✔️
Linear control-flow	✔️	❌
Code size before vectorization	+14-45%	
Independence of flags	❌	✔️
Expected maintenance cost	❌	✔️
Geomean speedup	23	34

IMPACT

IMPACT

Loop fission improves nd-range performance significantly

⇒ first SYCL implementation with competitive nd-range parallel-for without OpenCL runtime

IMPACT

Loop fission improves nd-range performance significantly

⇒ first SYCL implementation with competitive nd-range parallel-for without OpenCL runtime

CBS available in hipSYCL since v0.9.2

IMPACT

Loop fission improves nd-range performance significantly
⇒ first SYCL implementation with competitive nd-range parallel-for without OpenCL runtime

CBS available in hipSYCL since v0.9.2

Interest in having CBS upstreamed in LLVM?
Usable for OpenCL, SYCL, (CUDA, HIP,..?) CPU runtimes
(Interface: if kernel & barrier identified by annotation found, apply)

hipSYCL

SIC Saarland Informatics Campus

THANKS FOR WATCHING!

LOOKING FORWARD TO YOUR QUESTIONS AND FEEDBACK

Contact: Joachim Meyer
jmeyer@cs.uni-saarland.de
github.com/illuhad/hipSYCL