The 11th International workshop on OpenCL and SYCL

IWOCL & SYCLcon 2023

Implementation Techniques for SPMD Kernels on CPUs

Joachim Meyer (Saarland University)

Aksel Alpay, Holger Fröning, Vincent Heuveline (Heidelberg University)
Sebastian Hack (Saarland University)

April 18-20, 2023 | University of Cambridge, UK iwocl.org

CONTENT

- Recap: what are SPMD kernels?

CONTENT

- Recap: what are SPMD kernels?
- 4 approaches to implementing SPMD kernels on CPUs:

CONTENT

- Recap: what are SPMD kernels?
- 4 approaches to implementing SPMD kernels on CPUs:
  - library-only:
    - 1 thread : 1 work-item
    - 1 fiber : 1 work-item

CONTENT

- Recap: what are SPMD kernels?
- 4 approaches to implementing SPMD kernels on CPUs:
  - library-only:
    - 1 thread : 1 work-item
    - 1 fiber : 1 work-item
  - compiler supported:
    - deep loop fission (DLF)
    - continuation-based synchronization (CBS)

CONTENT

- Recap: what are SPMD kernels?
- 4 approaches to implementing SPMD kernels on CPUs:
  - library-only:
    - 1 thread : 1 work-item
    - 1 fiber : 1 work-item
  - compiler supported:
    - deep loop fission (DLF)
    - continuation-based synchronization (CBS)
- Semantic flaw in DLF

CONTENT

- Recap: what are SPMD kernels?
- 4 approaches to implementing SPMD kernels on CPUs:
  - library-only:
    - 1 thread : 1 work-item
    - 1 fiber : 1 work-item
  - compiler supported:
    - deep loop fission (DLF)
    - continuation-based synchronization (CBS)
- Semantic flaw in DLF
- An important optimization: propagating contiguity

CONTENT

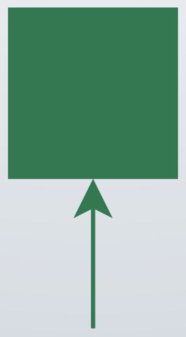
- Recap: what are SPMD kernels?
- 4 approaches to implementing SPMD kernels on CPUs:
  - library-only:
    - 1 thread : 1 work-item
    - 1 fiber : 1 work-item
  - compiler supported:
    - deep loop fission (DLF)
    - continuation-based synchronization (CBS)
- Semantic flaw in DLF
- An important optimization: propagating contiguity
- Performance results: library-only < DLF ≈ CBS

SINGLE-PROGRAM MULTIPLE-DATA KERNEL

```
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];
  });
```

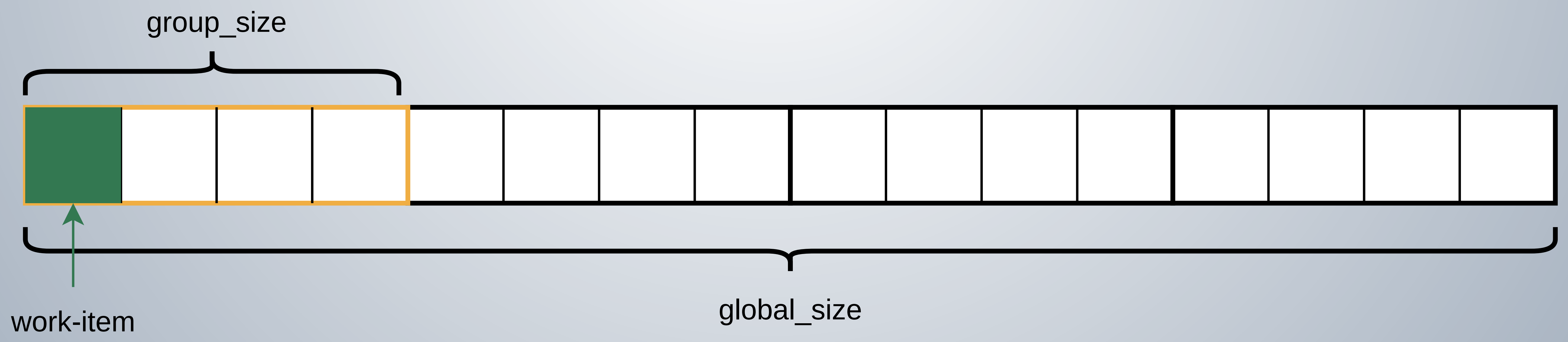
SINGLE-PROGRAM MULTIPLE-DATA KERNEL

```
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];
  });
```



SINGLE-PROGRAM MULTIPLE-DATA KERNEL

```
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];
  });
```

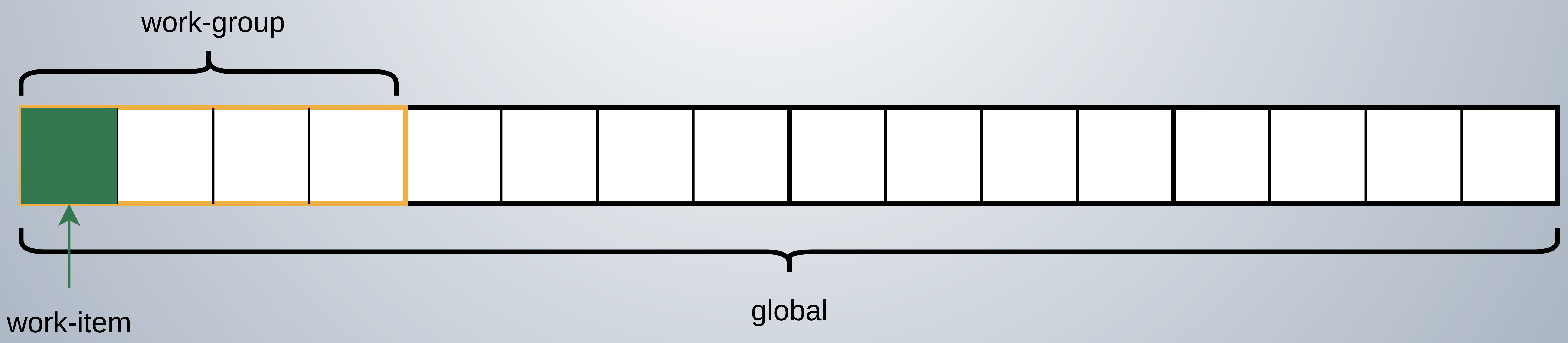


UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386 hipSYCL

UNIVERSITÄT DES SAARLANDES SIC Saarland Informatics Campus

THE BIG BLOCKER ON CPUS

```
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];
  });
```



THIS IS "SIMPLE" ON GPUS

- Execution of many (mostly) independent threads
  → Forward-Progress guarantees
- Hardware support for work-group barriers

HOW TO MAP THIS TO CPUS?

CONCURRENCY!

- 1 work-item : 1 thread
  - E.g. OpenMP parallel for + #pragma omp barrier

⚡ Many threads → scheduling overhead

⚡ No vectorization across work-items

CONCURRENCY!

- 1 work-item : 1 thread
  - E.g. OpenMP parallel for + #pragma omp barrier

⚡ Many threads → scheduling overhead

⚡ No vectorization across work-items

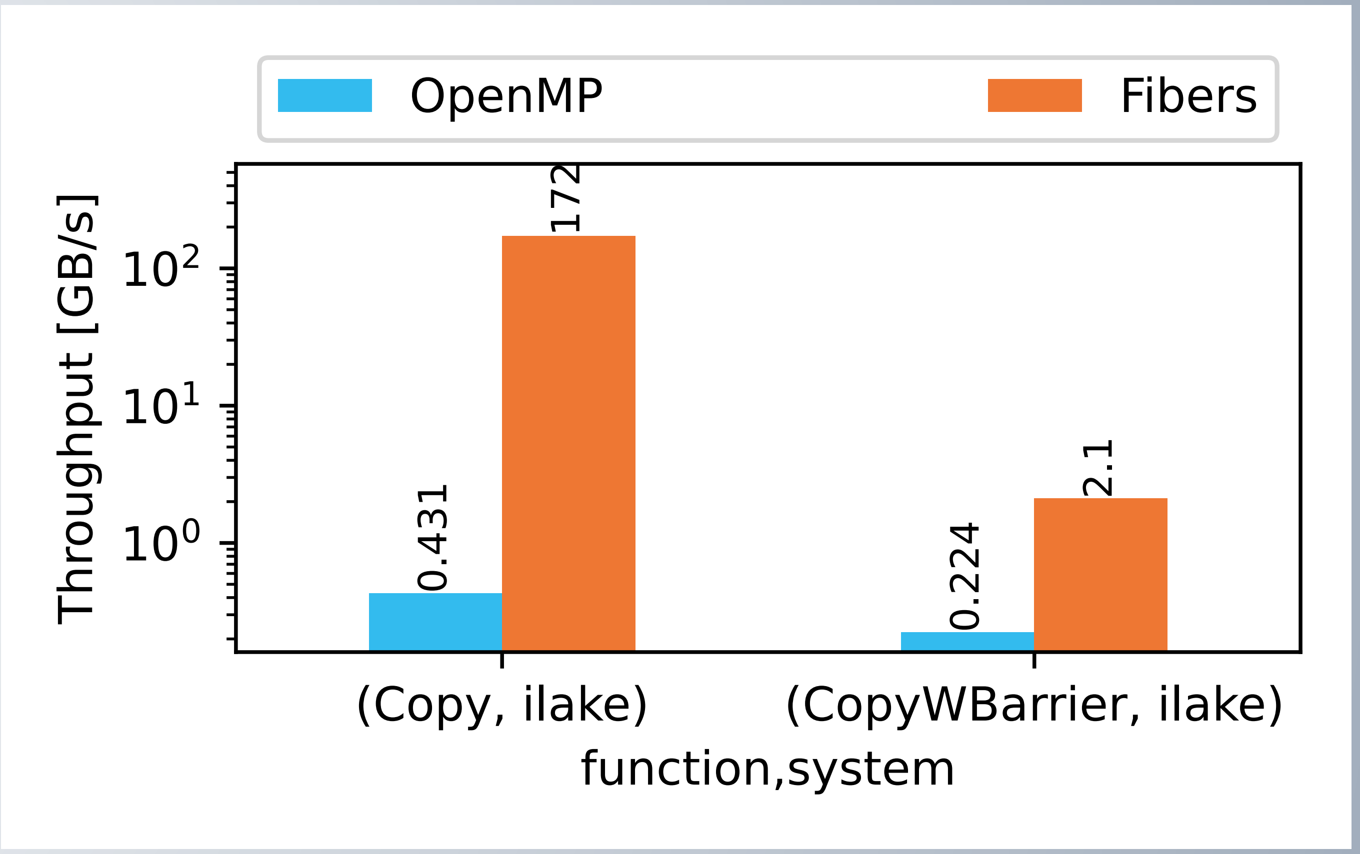
- 1 work-item : 1 fiber
  - Lightweight threads + synchronization
  - Can optimize barrier-free kernels!

⚡ Context-switch overhead

⚡ Limited vectorization across work-items

CONCURRENCY!

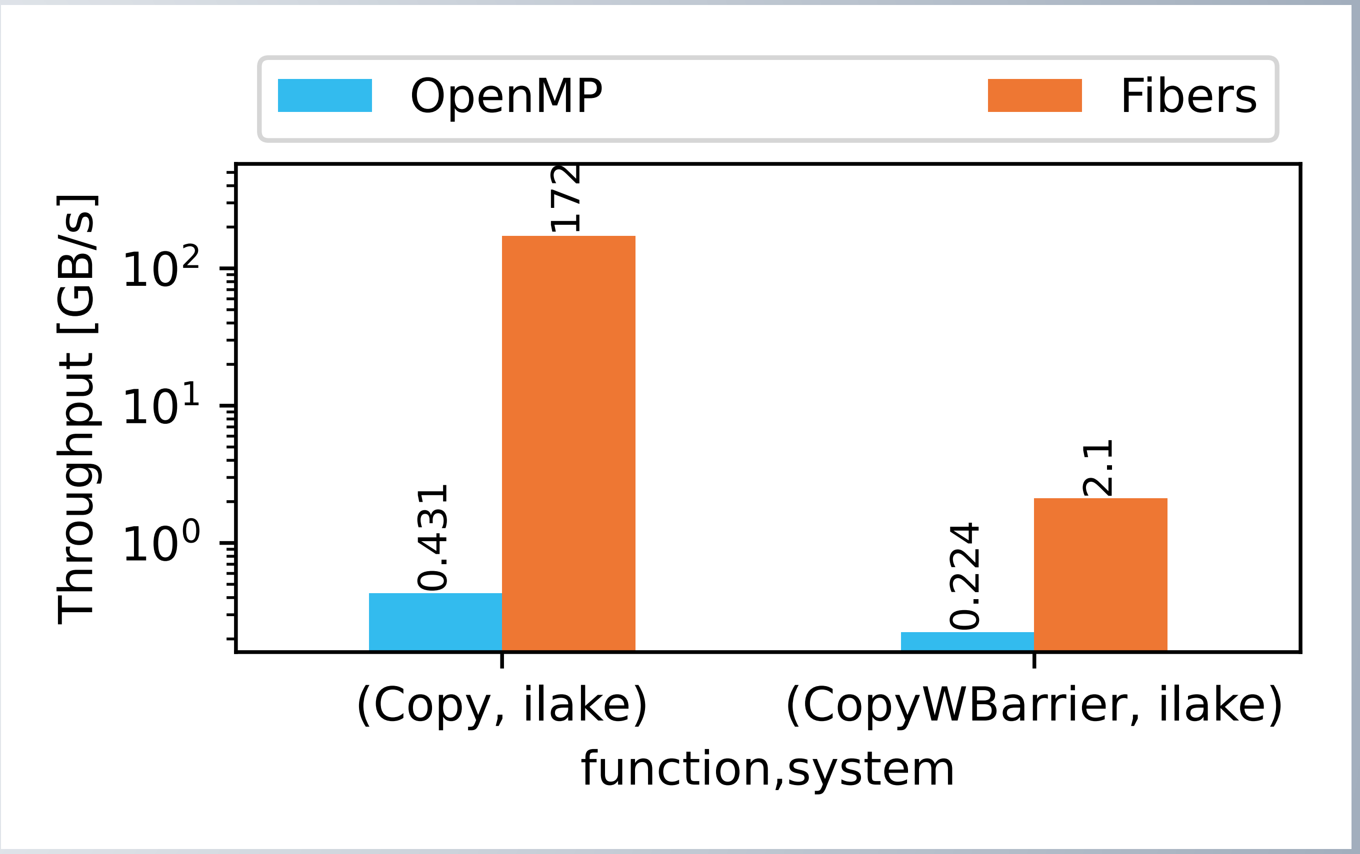
- 1 work-item : 1 thread
  - E.g. OpenMP parallel for + #pragma omp barrier
  - ⚡ Many threads → scheduling overhead
  - ⚡ No vectorization across work-items
- 1 work-item : 1 fiber
  - Lightweight threads + synchronization
  - Can optimize barrier-free kernels!
  - ⚡ Context-switch overhead
  - ⚡ Limited vectorization across work-items



github.com/UoB-HPC/BabelStream

CONCURRENCY!

- 1 work-item : 1 thread
  - E.g. OpenMP parallel for + #pragma omp barrier
  - ⚡ Many threads → scheduling overhead
  - ⚡ No vectorization across work-items
- 1 work-item : 1 fiber
  - Lightweight threads + synchronization
  - Can optimize barrier-free kernels!
  - ⚡ Context-switch overhead
  - ⚡ Limited vectorization across work-items



github.com/UoB-HPC/BabelStream

✅ **Can be implemented without dedicated compiler support!**

WANT PERFORMANCE? USE THE COMPILER!

- Threading on the **work-group level**
- Loop over work-items in a single thread
- Compiler extension to **split kernel** at barriers

✅ Vectorization across work-items possible
✅ Improves performance over library-only by up to **several orders of magnitude**

```
#pragma omp parallel for
for(group : groups)
  #pragma omp simd
  for(item : itemsInGroup)
    kernel_before_barrier(nd_item{group, item})
  // implicit synchronization
  #pragma omp simd
  for(item : itemsInGroup)
    kernel_after_barrier(nd_item{group, item})
```

DEEP LOOP FISSION (POCL)

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

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

```
for(lid : items[0:])
  // A
// barrier
for(i = group_size / 2; i > 0; i /= 2)
  // B (lid = 0)
  for(lid : items[1:])
    // B
  // barrier
for(lid : items[0:])
  // C
```

DEEP LOOP FISSION (POCL)

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

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

```
for(lid : items[0:])
  // A
// barrier
for(i = group_size / 2; i > 0; i /= 2)
  // B (lid = 0)
  for(lid : items[1:])
    // B
  // barrier
for(lid : items[0:])
  // C
```

DEEP LOOP FISSION (POCL)

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

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

```
for(lid : items[0:])
  // A
// barrier
for(i = group_size / 2; i > 0; i /= 2)
  // B (lid = 0)
  for(lid : items[1:])
    // B
  // barrier
for(lid : items[0:])
  // C
```

DEEP LOOP FISSION (POCL)

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

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

```
for(lid : items[0:])
  // A
// barrier
for(i = group_size / 2; i > 0; i /= 2)
  // B (lid = 0)
  for(lid : items[1:])
    // B
  // barrier
for(lid : items[0:])
  // C
```

DEEP LOOP FISSION (POCL)

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

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

```
for(lid : items[0:])
  // A
// barrier
for(i = group_size / 2; i > 0; i /= 2)
  // B (lid = 0)
  for(lid : items[1:])
    // B
  // barrier
for(lid : items[0:])
  // C
```

DEEP LOOP FISSION – SEMANTIC PROBLEM

```
[=](sycl::nd_item<1> item) noexcept {
  const auto lid = item.get_local_id(0);

  scratch[lid] = acc[item.get_global_id()]; // A
  item.barrier();

  for(size_t i = 0; i < 2 + lid; i++) {
    scratch[lid] += i; // B
    // only call the barrier if all work-items still run the loop.
    if(i < 2) item.barrier();
  }
  acc[item.get_global_id()] = scratch[lid]; // C
}
```

```
for(lid : items[0:])
  // A
// barrier
for(i : [0,1])
  // B (lid = 0)
  for(lid : items[1:])
    // B
  if(i < 2)
    // barrier
for(lid : items[0:])
  // C
```

DEEP LOOP FISSION – SEMANTIC PROBLEM

```
[=](sycl::nd_item<1> item) noexcept {
  const auto lid = item.get_local_id(0);

  scratch[lid] = acc[item.get_global_id()]; // A
  item.barrier();

  for(size_t i = 0; i < 2 + lid; i++) {
    scratch[lid] += i; // B
    // only call the barrier if all work-items still run the loop.
    if(i < 2) item.barrier();
  }
  acc[item.get_global_id()] = scratch[lid]; // C
}
```

```
for(lid : items[0:])
  // A
// barrier
for(i : [0,1])
  // B (lid = 0)
  for(lid : items[1:])
    // B
  if(i < 2)
    // barrier
for(lid : items[0:])
  // C
```

CONTINUATION-BASED SYNCHRONIZATION TO THE RESCUE

```
[=](sycl::nd_item<1> item) noexcept { // 0
  const auto lid = item.get_local_id(0);

  scratch[lid] = acc[item.get_global_id()]; // A
  item.barrier(); // 1

  for(size_t i = 0; i < 2 + lid; i++) {
    scratch[lid] += i; // B
    // only call the barrier if all work-items still run the loop.
    if(i < 2) item.barrier(); // 2
  }
  acc[item.get_global_id()] = scratch[lid]; // C
} // -1
```

Continuation-based Synchronization

```




    case 0:

        // A
        // barrier

    case 1:

        i = 0
        while(i < 2 + lid)
          // B
          if(i < 2) // barrier
          i++;
        // C


    case 2:

        i++;
        while(i < 2 + lid)
          // B
          if(i < 2)  // barrier
          i++;
        // C
```

CONTINUATION-BASED SYNCHRONIZATION TO THE RESCUE

UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386 · hipSYCL · UNIVERSITÄT DES SAARLANDES · SIC Saarland Informatics Campus

```
[=](sycl::nd_item<1> item) noexcept { // 0
  const auto lid = item.get_local_id(0);

  scratch[lid] = acc[item.get_global_id()]; // A
  item.barrier(); // 1

  for(size_t i = 0; i < 2 + lid; i++) {
    scratch[lid] += i; // B
    // only call the barrier if all work-items still run the loop.
    if(i < 2) item.barrier(); // 2
  }
  acc[item.get_global_id()] = scratch[lid]; // C
} // -1
```

Continuation-based Synchronization

```




    case 0:

        // A
        // barrier

    case 1:

        i = 0
        while(i < 2 + lid)
          // B
          if(i < 2) // barrier
          i++;
        // C


    case 2:

        i++;
        while(i < 2 + lid)
          // B
          if(i < 2)  // barrier
          i++;
        // C
```

CONTINUATION-BASED SYNCHRONIZATION TO THE RESCUE

```
[=](sycl::nd_item<1> item) noexcept { // 0
  const auto lid = item.get_local_id(0);

  scratch[lid] = acc[item.get_global_id()]; // A
  item.barrier(); // 1

  for(size_t i = 0; i < 2 + lid; i++) {
    scratch[lid] += i; // B
    // only call the barrier if all work-items still run the loop.
    if(i < 2) item.barrier(); // 2
  }
  acc[item.get_global_id()] = scratch[lid]; // C
} // -1
```

Continuation-based Synchronization

```




    case 0:

      // A
      // barrier

    case 1:

      i = 0
      while(i < 2 + lid)
        // B
        if(i < 2) // barrier
        i++;
      // C


    case 2:

      i++;
      while(i < 2 + lid)
        // B
        if(i < 2)  // barrier
        i++;
      // C
```

UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386 | hipSYCL | UNIVERSITÄT DES SAARLANDES | SIC Saarland Informatics Campus

CONTINUATION-BASED SYNCHRONIZATION TO THE RESCUE

```cpp
[=](sycl::nd_item<1> item) noexcept { // 0
  const auto lid = item.get_local_id(0);

  scratch[lid] = acc[item.get_global_id()]; // A
  item.barrier(); // 1

  for(size_t i = 0; i < 2 + lid; i++) {
    scratch[lid] += i; // B
    // only call the barrier if all work-items still run the loop.
    if(i < 2) item.barrier(); // 2
  }
  acc[item.get_global_id()] = scratch[lid]; // C
} // -1
```

Continuation-based Synchronization

```
    case 0:

        // A
        // barrier

    case 1:

        i = 0
        while(i < 2 + lid)
          // B
          if(i < 2) // barrier
          i++;
        // C


    case 2:

        i++;
        while(i < 2 + lid)
          // B
          if(i < 2)  // barrier
          i++;
        // C
```

CONTINUATION-BASED SYNCHRONIZATION TO THE RESCUE

UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386
hipSYCL
UNIVERSITÄT DES SAARLANDES
SIC Saarland Informatics Campus

```
[=](sycl::nd_item<1> item) noexcept { // 0
  const auto lid = item.get_local_id(0);

  scratch[lid] = acc[item.get_global_id()]; // A
  item.barrier(); // 1

  for(size_t i = 0; i < 2 + lid; i++) {
    scratch[lid] += i; // B
    // only call the barrier if all work-items still run the loop.
    if(i < 2) item.barrier(); // 2
  }
  acc[item.get_global_id()] = scratch[lid]; // C
} // -1
```

Continuation-based Synchronization

```
i[items] = alloca ..;



    case 0:
      for(lid : items[0:])
        // A
        // barrier

    case 1:
      for(lid : items[0:])
        i[lid] = 0
        while(i[lid] < 2 + lid)
          // B
          if(i[lid] < 2) // barrier
          i[lid]++;
        // C


    case 2:
      for(lid : items[0:])
        i[lid]++;
        while(i[lid] < 2 + lid)
          // B
          if(i[lid] < 2) // barrier
          i[lid]++;
        // C
```

CONTINUATION-BASED SYNCHRONIZATION TO THE RESCUE

```
[=](sycl::nd_item<1> item) noexcept { // 0
  const auto lid = item.get_local_id(0);

  scratch[lid] = acc[item.get_global_id()]; // A
  item.barrier(); // 1

  for(size_t i = 0; i < 2 + lid; i++) {
    scratch[lid] += i; // B
    // only call the barrier if all work-items still run the loop.
    if(i < 2) item.barrier(); // 2
  }
  acc[item.get_global_id()] = scratch[lid]; // C
} // -1
```

Continuation-based Synchronization

```
i[items] = alloca ..;
next = 0;


    case 0:
      for(lid : items[0:])
        // A
        next = 1;

    case 1:
      cont1: for(lid : items[0:])
        i[lid] = 0
        while(i[lid] < 2 + lid)
          // B
          if(i[lid] < 2) next = 2; goto cont1;
          i[lid]++;
        // C
        next = -1;

    case 2:
      cont2: for(lid : items[0:])
        i[lid]++;
        while(i[lid] < 2 + lid)
          // B
          if(i[lid] < 2) next = 2; goto cont2;
          i[lid]++;
        // C
        next = -1;
```

CONTINUATION-BASED SYNCHRONIZATION TO THE RESCUE

```
[=](sycl::nd_item<1> item) noexcept { // 0
  const auto lid = item.get_local_id(0);

  scratch[lid] = acc[item.get_global_id()]; // A
  item.barrier(); // 1

  for(size_t i = 0; i < 2 + lid; i++) {
    scratch[lid] += i; // B
    // only call the barrier if all work-items still run the loop.
    if(i < 2) item.barrier(); // 2
  }
  acc[item.get_global_id()] = scratch[lid]; // C
} // -1
```

Continuation-based Synchronization

```
i[items] = alloca ..;
next = 0;
while(next != -1) {
  switch(next) {
    case 0:
      for(lid : items[0:])
        // A
        next = 1;
      break;
    case 1:
      cont1: for(lid : items[0:])
        i[lid] = 0
        while(i[lid] < 2 + lid)
          // B
          if(i[lid] < 2) next = 2; goto cont1;
          i[lid]++;
        // C
        next = -1;
      break;
    case 2:
      cont2: for(lid : items[0:])
        i[lid]++;
        while(i[lid] < 2 + lid)
          // B
          if(i[lid] < 2) next = 2; goto cont2;
          i[lid]++;
        // C
        next = -1;
      break;
  }
}
```

CONTINUATION-BASED SYNCHRONIZATION TO THE RESCUE

```
[=](sycl::nd_item<1> item) noexcept { // 0
  const auto lid = item.get_local_id(0);

  scratch[lid] = acc[item.get_global_id()]; // A
  item.barrier(); // 1

  for(size_t i = 0; i < 2 + lid; i++) {
    scratch[lid] += i; // B
    // only call the barrier if all work-items still run the loop.
    if(i < 2) item.barrier(); // 2
  }
  acc[item.get_global_id()] = scratch[lid]; // C
} // -1
```

Deep Loop Fission

```
for(lid : items[0:])
  // A
// barrier
for(i : [0,1])
  // B (lid = 0)
  for(lid : items[1:])
    // B
  if(i < 2)
    // barrier
for(lid : items[0:])
  // C
```

Continuation-based Synchronization

```
i[items] = alloca ..;
next = 0;
while(next != -1) {
  switch(next) {
    case 0:
      for(lid : items[0:])
        // A
        next = 1;
      break;
    case 1:
      cont1: for(lid : items[0:])
        i[lid] = 0
        while(i[lid] < 2 + lid)
          // B
          if(i[lid] < 2) next = 2; goto cont1;
          i[lid]++;
        // C
        next = -1;
      break;
    case 2:
      cont2: for(lid : items[0:])
        i[lid]++;
        while(i[lid] < 2 + lid)
          // B
          if(i[lid] < 2) next = 2; goto cont2;
          i[lid]++;
        // C
        next = -1;
      break;
  }
}
```

HOW ARE WORK-ITEM PRIVATE VALUES STORED?

✅ Dynamically-sized stack arrays with large alignment (64)

```
value = global[offset + lid];
item.barrier();
use(value);
```

⇒

```
value[items] = alloca ..;
case 1:
  for(lid : items[0:])
    value1 = global[offset + lid];
    value[lid] = value1;
case 2:
  for(lid : items[0:])
    value2 = value[lid];
    use(value2);
```

AVOID STORING UNIFORM VALUES TO THOSE ARRAYS

✅ Value shape analysis

```
offset[items] = alloca ..;
case 1:
  for(lid : items[0:])
    offset1 = 0; // uniform
    offset[lid] = offset1;
case 2:

  for(lid : items[0:])
    offset2 = offset[lid];
```

⇒

```
offset = alloca ..;
case 1:
  for(lid : items[0:])
    offset1 = 0; // uniform
    offset = offset1;
case 2:
  offset2 = offset;
  for(lid : items[0:])
    // ..
```

PROPAGATE VALUE CONTIGUITY TO THE OPTIMIZER

✅ Value shape analysis + trace cont values to uniform values & wi-index + replicate trace after barrier

```
idx[items] = alloca ..;
case 1:
  for(lid : items[0:])
    idx1 = offset1 + lid; // contiguous
    idx[lid] = idx1
case 2:

  for(lid : items[0:])
    idx2 = idx[lid];
    // is this a contiguous access?
    ptr[idx2] = ..;
```

⇒

```
offset = alloca  ..; // uniform
case 1:
  for(lid : items[0:])
    idx1 = offset1 + lid; // contiguous
    offset = offset1
case 2:
  offset2 = offset
  for(lid : items[0:])
    idx2 = offset2 + lid;
    // this is a contiguous access!
    ptr[idx2] = ..;
```

PROPAGATE VALUE CONTIGUITY TO THE OPTIMIZER

✅ Value shape analysis + trace cont values to uniform values & wi-index + replicate trace after barrier

```
idx[items] = alloca ..;
case 1:
  for(lid : items[0:])
    idx1 = offset1 + lid; // contiguous
    idx[lid] = idx1
case 2:

  for(lid : items[0:])
    idx2 = idx[lid];
    // is this a contiguous access?
    ptr[idx2] = ..;
```

GEOMEAN SPEEDUP ⇒ POCL: 7%, HIPSYCL: 17%

```
offset = alloca  ..; // uniform
case 1:
  for(lid : items[0:])
    idx1 = offset1 + lid; // contiguous
    offset = offset1
case 2:
  offset2 = offset
  for(lid : items[0:])
    idx2 = offset2 + lid;
    // this is a contiguous access!
    ptr[idx2] = ..;
```

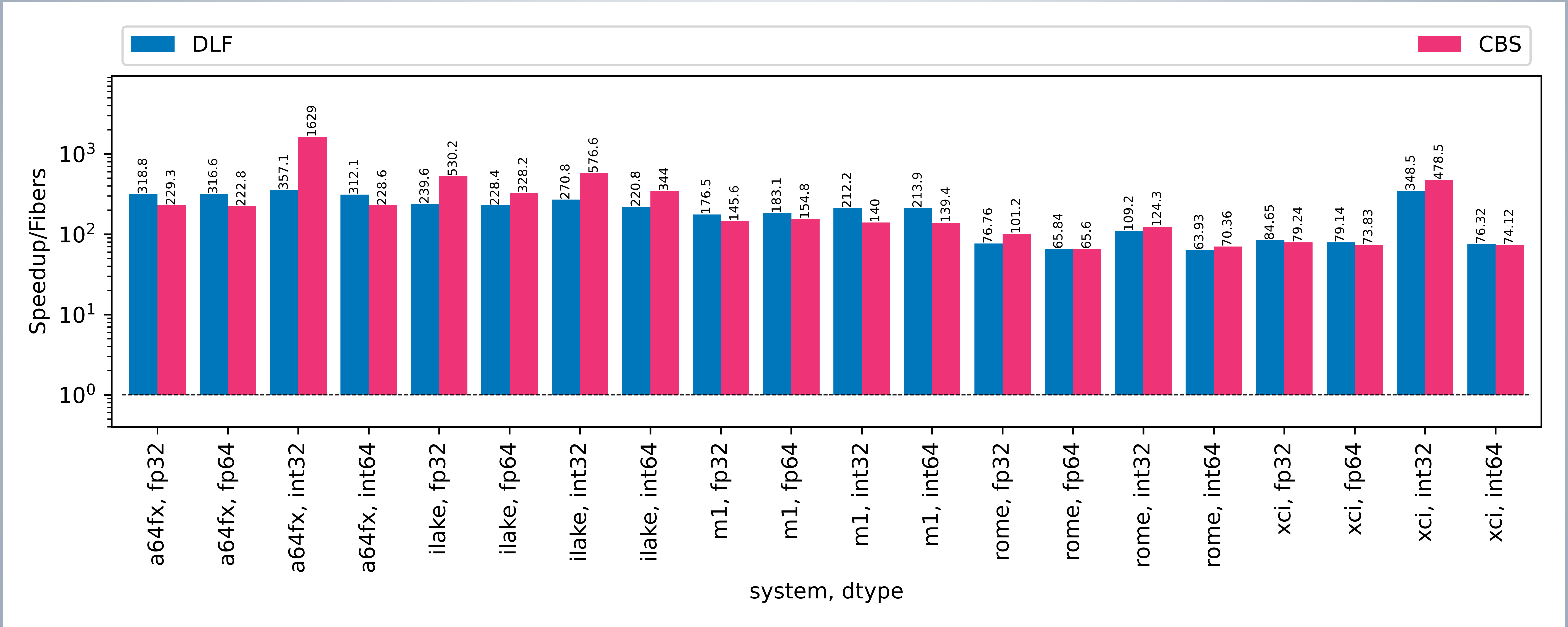
HOW ABOUT PERFORMANCE? EVERYWHERE?

BENCHMARK – SYSTEMS

- Fujitsu A64FX **"a64fx"**
  - 1x 1.8GHz 48-core CPU
  - 512bit SVE
- Marvell ThunderX2 **"xci"**
  - 2x 2.1GHz 32-core, 128-threads CPU
  - NEON
- Mac Mini M1 **"m1"**
  - 1x 4e+4p-core CPU
  - NEON AdvSIMD
- Intel Xeon Gold 6338 **"ilake"**
  - 2x 2.00GHz 32-core, 64-threads Icelake CPU
  - AVX512
- AMD Epyc 7442 **"rome"**
  - 2x 2.25GHz 64-core, 128-threads Rome CPU
  - AVX2

MASSIVE SPEEDUPS OF COMPILER APPROACHES

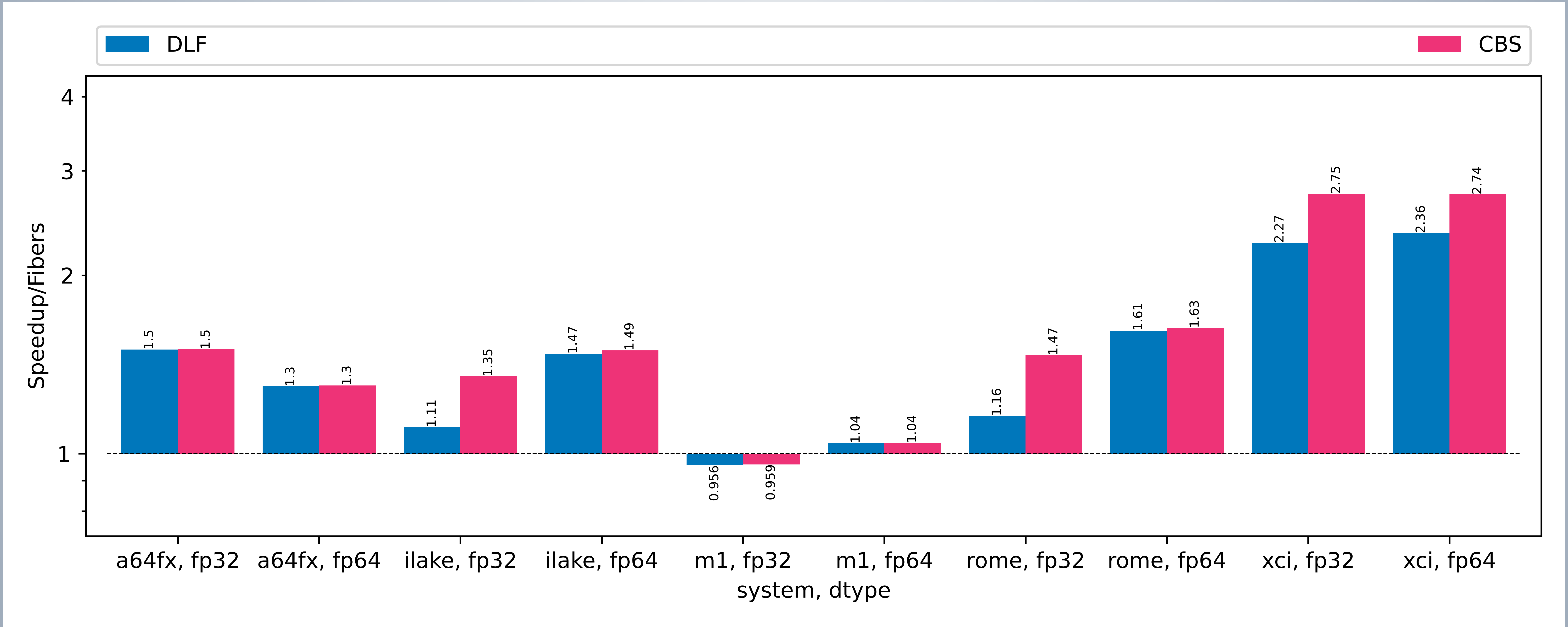
SYCL-Bench Reduction in hipSYCL



github.com/bcosenza/sycl-bench

FIBERS OK FOR HIGH COMPUTE/BARRIER RATIO

SYCL-Bench N-body in hipSYCL



github.com/bcosenza/sycl-bench

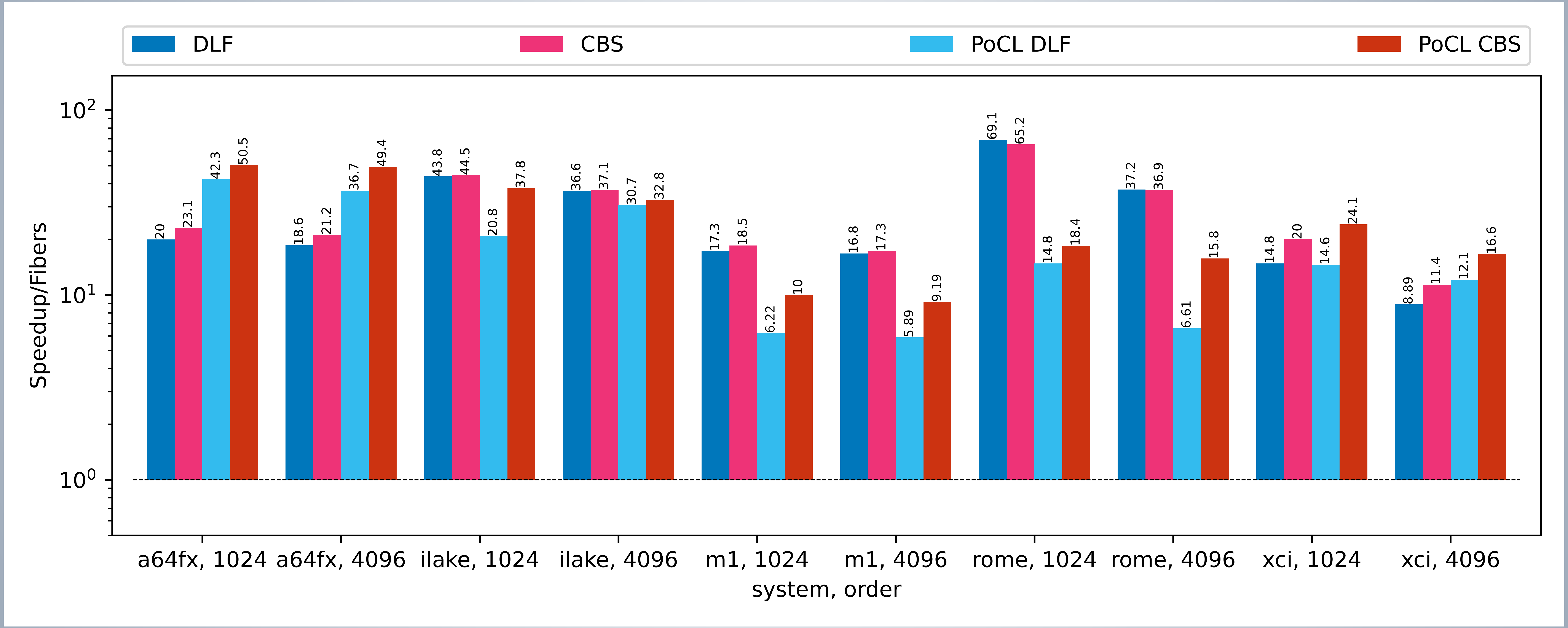
CBS IS COMPETITIVE

hipSYCL summary

	DLF	CBS
Geomean speedup/fibers	29.2	37.7
Number of Best	13	39

PROGRAMMING MODEL INDEPENDENT

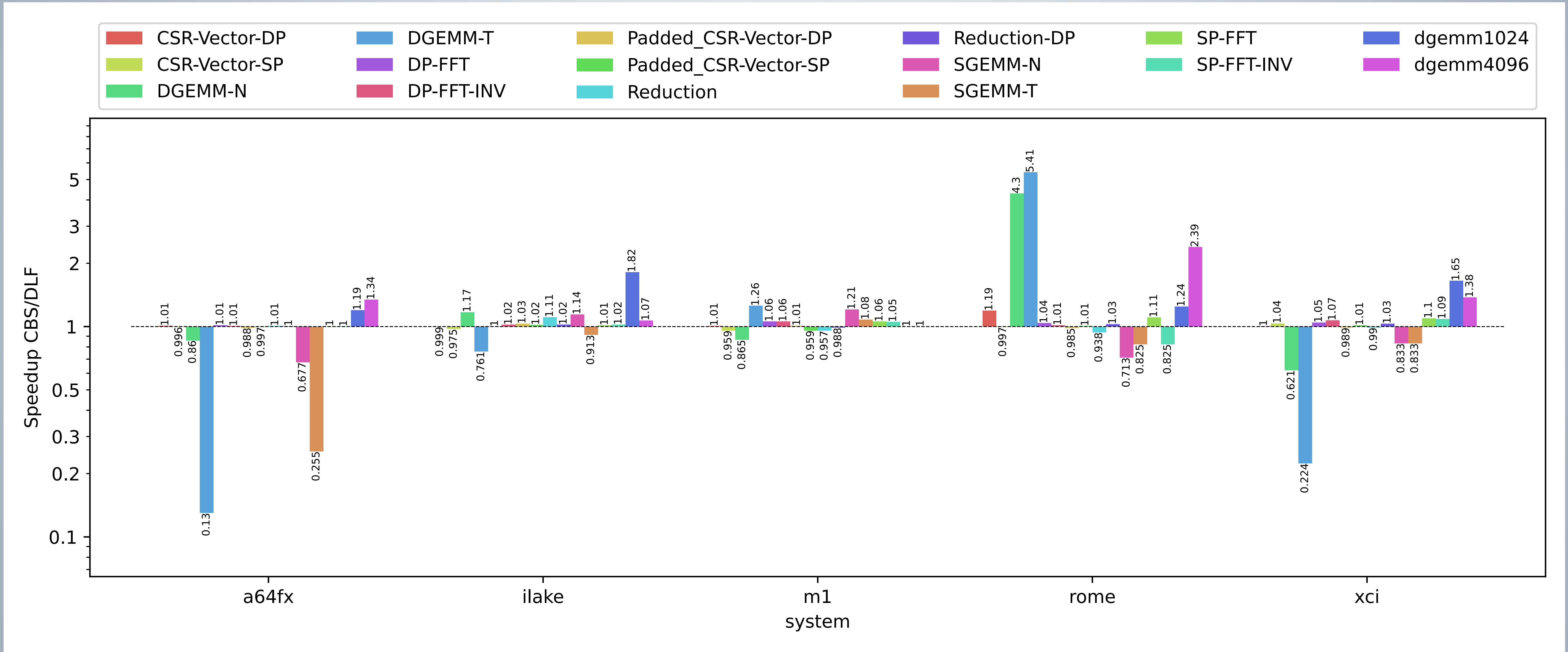
SYCL DGEMM in hipSYCL & PoCL



github.com/UoB-HPC/sycl_dgemm

ARM HPC NOT AS HAPPY WITH CBS

Performance in PoCL



github.com/vetter/shoc
github.com/UoB-HPC/sycl_dgemm

DLF AND CBS COMPARABLE IN OPENCL

PoCL summary

	a64fx	ilake	m1	rome	xci	overall
Geomean CBS/DLF	0.81	1.05	1.03	1.27	0.93	1.00
CBS #of Best in 16	9	12	9	10	10	41/64

CONCLUSION

	OpenMP	Fibers	DLF	CBS
Library-only	✔	✔	❌	❌
Performance barrier-free	❌	✔	✔	✔
Performance with barrier	❌	❌	✔	✔
Performance on HPC ARM	❌	❌	✔	🟨
Covering full barrier semantic	✔	✔	❌	✔

THANK YOU FOR LISTENING!

LOOKING FORWARD TO QUESTIONS AND DISCUSSIONS

Contact: Joachim Meyer

jmeyer@cs.uni-saarland.de

github.com/OpenSYCL/OpenSYCL

github.com/pocl/pocl

BACKUP

COMPILER APPROACHES SIMILAR TO HIERARCHICAL PARALLEL_FOR

Hierarchical parallel_for

```
h.parallel_for_work_group(
  range<1>(groups), [=](group<1> grp) {
    grp.parallel_for_work_item([=](h_item<1> item) {
      before_barrier(..);
    });
    // implicit work-group barrier
    grp.parallel_for_work_item([=](h_item<1> item) {
      after_barrier(..);
    });
  });
```

nd_range parallel_for after kernel splitting

```
#pragma omp parallel for
for(group : groups)
  #pragma omp simd
  for(item : itemsInGroup)
    kernel_before_barrier(nd_item{group, item})
  // implicit synchronization
  #pragma omp simd
  for(item : itemsInGroup)
    kernel_after_barrier(nd_item{group, item})
```

COMPILER APPROACHES SIMILAR TO HIERARCHICAL PARALLEL_FOR

Hierarchical parallel_for

```
h.parallel_for_work_group(
  range<1>(groups), [=](group<1> grp) {
    grp.parallel_for_work_item([=](h_item<1> item) {
      before_barrier(..);
    });
    // implicit work-group barrier
    grp.parallel_for_work_item([=](h_item<1> item) {
      after_barrier(..);
    });
  });
```

nd_range parallel_for after kernel splitting

```
#pragma omp parallel for
for(group : groups)
  #pragma omp simd
  for(item : itemsInGroup)
    kernel_before_barrier(nd_item{group, item})
  // implicit synchronization
  #pragma omp simd
  for(item : itemsInGroup)
    kernel_after_barrier(nd_item{group, item})
```

COMPILER APPROACHES SIMILAR TO HIERARCHICAL PARALLEL_FOR

Hierarchical parallel_for

```
h.parallel_for_work_group(
  range<1>(groups), [=](group<1> grp) {
    grp.parallel_for_work_item([=](h_item<1> item) {
      before_barrier(..);
    });
    // implicit work-group barrier
    grp.parallel_for_work_item([=](h_item<1> item) {
      after_barrier(..);
    });
  });
```

nd_range parallel_for after kernel splitting

```
#pragma omp parallel for
for(group : groups)
  #pragma omp simd
  for(item : itemsInGroup)
    kernel_before_barrier(nd_item{group, item})
  // implicit synchronization
  #pragma omp simd
  for(item : itemsInGroup)
    kernel_after_barrier(nd_item{group, item})
```

COMPILER APPROACHES SIMILAR TO HIERARCHICAL PARALLEL_FOR

Hierarchical parallel_for

```
h.parallel_for_work_group(
  range<1>(groups), [=](group<1> grp) {
    grp.parallel_for_work_item([=](h_item<1> item) {
      before_barrier(..);
    });
    // implicit work-group barrier
    grp.parallel_for_work_item([=](h_item<1> item) {
      after_barrier(..);
    });
  });
```

nd_range parallel_for after kernel splitting

```
#pragma omp parallel for
for(group : groups)
  #pragma omp simd
  for(item : itemsInGroup)
    kernel_before_barrier(nd_item{group, item})
  // implicit synchronization
  #pragma omp simd
  for(item : itemsInGroup)
    kernel_after_barrier(nd_item{group, item})
```