Stellar Mergers with HPX-Kokkos and SYCL: Methods of using an Asynchronous Many-Task Runtime System with SYCL

Gregor Daiß, University of Stuttgart
Patrick Diehl, Hartmut Kaiser and Dirk Pflüger
HPX and SYCL

What is HPX?

- Asynchronous, Distributed Many-Task Runtime System
What is HPX?

- **Asynchronous, Distributed Many-Task Runtime System**
- **Asynchronous**: Build task graph using futures and continuations (`then`, `when_all`)

```cpp
hpx::future<void> fut1 = hpx::async([](){...});
hpx::future<void> fut2 = fut1.then([](){...});
hpx::future<void> futX = when_all(fut1, fut2);
```
HPX and SYCL

What is HPX?

- **Asynchronous, Distributed Many-Task Runtime System**
- **Asynchronous**: Build task graph using futures and continuations (then, when_all)
- **Distributed**: Task graph across compute nodes (remote function calls, HPX channels, multiple backends available)
What is HPX?

- **Asynchronous, Distributed Many-Task Runtime System**
- **Asynchronous**: Build task graph using futures and continuations (then, when_all)
- **Distributed**: Task graph across compute nodes (remote function calls, HPX channels, multiple backends available)
- **Many Tasks**: Few HPX worker threads (one per core) working on millions of lightweight (suspendable) HPX tasks
Why combine HPX with SYCL?

Compute Node 1 (HPX locality 1)
- Task 1
- Task 2
- Task 3

Compute Node 2 (HPX locality 2)
- Task 4
  - SYCL 1
  - SYCL 2

?
Why combine HPX with SYCL?

- More choices: SYCL for HPX applications, HPX for distributed SYCL applications (instead of MPI)
- Better integrations: Better integration of HPX with other libraries that use SYCL (Kokkos)
- More efficiency: Complement strengths
HPX and SYCL

Why combine HPX with SYCL?

- **More choices**: SYCL for HPX applications, HPX for distributed SYCL applications (instead of MPI)
- **Better integrations**: Better integration of HPX with other libraries that use SYCL (Kokkos)
- **More efficiency**: Complement strengths

How to combine HPX with SYCL?

- The problem: Integrate task-graphs asynchronously and efficiently
  - **No active waiting** *(no `event.wait()*) Avoid barriers / blocking of worker threads
  - **Overhead?**
How to combine HPX with SYCL?

- **We have**: SYCL events to check if asynchronous SYCL actions are done
- **We need**: HPX futures to check if asynchronous SYCL actions are done
- Get an HPX future from a SYCL event without actively waiting or blocking the thread

```
sycl::event
submitted ➔ running ➔ completed

hpx::future<T>
not ready ➔ ready
```
HPX-SYCL Integration Basics

How to combine HPX with SYCL?

- **We have**: SYCL events to check if asynchronous SYCL actions are done
- **We need**: HPX futures to check if asynchronous SYCL actions are done
- Get an HPX future from a SYCL event without actively waiting or blocking the thread
- HPX scheduler takes care of the rest (triggering continuations)

```
sycl::event
  submitted \rightarrow running \rightarrow completed
hpx::future<T>
  not ready \rightarrow ready
```
How to combine HPX with SYCL?

- **We have**: SYCL events to check if asynchronous SYCL actions are done
- **We need**: HPX futures to check if asynchronous SYCL actions are done
- Get an HPX future from a SYCL event without actively waiting or blocking the thread
- HPX scheduler takes care of the rest (triggering continuations)

**TODOs for integration**

- Add specialization for HPX future_data
- Add callback mechanism that is called when the SYCL event is completed
- Use it to set the future to ready
HPX-SYCL Integration Variant 1: Using SYCL `host_tasks`

Use SYCL `host_tasks` as callback mechanism
Use SYCL host_tasks as callback mechanism

- Advantages:
  - Easiest way to implement the HPX-SYCL integration

- Disadvantages:
  - host_tasks not executed by HPX workers
    → Overhead/contention problem?
Use SYCL `host_tasks` as callback mechanism

- **Advantages:**
  - Easiest way to implement the HPX-SYCL integration
- **Disadvantages:**
  - `host_tasks` not executed by HPX workers
    → Overhead/contention problem?
Use SYCL host_tasks as callback mechanism

- Advantages:
  - Easiest way to implement the HPX-SYCL integration
- Disadvantages:
  - host_tasks not executed by HPX workers
    → Overhead/contention problem?

Create Callback during future_data creation

```cpp
sycl_queue.submit([fdp = hpx::intrusive_ptr<future_data>(this),
sycl_event](cl::sycl::handler& h) {
  h.depends_on(sycl_event);
  h.host_task([fdp]() {
    fdp->set_data(hpx::util::unused); });
});
```

Stellar Mergers with HPX-Kokkos and SYCL: Methods of using an Asynchronous Many-Task Runtime System with SYCL 5 / 17
HPX-SYCL Integration Variant 2: Using Event Polling

Event polling within the HPX scheduler

- Store event-callback pairs in HPX scheduler
- Worker threads poll events in-between tasks and invoke callbacks
- Only one thread polls (others skip if mutex is already locked)
- Use concurrent queue for adding and mutex-protected vector for later checking
HPX-SYCL Integration Variant 2: Using Event Polling

Event polling within the HPX scheduler

- Store event-callback pairs in HPX scheduler
- Worker threads poll events in-between tasks and invoke callbacks
- Only one thread polls (others skip if mutex is already locked)
- Use concurrent queue for adding and mutex-protected vector for later checking

Advantages

- HPX worker run callbacks themselves → One threadpool
- Works with SYCL implementations that do not yet support host_tasks
Integrating HPX and SYCL

Scientific Application as a Benchmark: Octo-Tiger

Results with Octo-Tiger

Conclusion

---

**HPX-SYCL Integration Variant 2: Using Event Polling**

**Event polling within the HPX scheduler**

- Store event-callback pairs in HPX scheduler
- Worker threads poll events in-between tasks and invoke callbacks
- Only one thread polls (others skip if mutex is already locked)
- Use concurrent queue for adding and mutex-protected vector for later checking

**Advantages**

- HPX worker run callbacks themselves → One threadpool
- Works with SYCL implementations that do not yet support `host_tasks`

**Disadvantages**

- Requires additions to the HPX scheduler
- Event creations, deletions and polling can cause overheads
Dummy SYCL kernel/task

```cpp
sycl::event my_event = queue.submit([&](sycl::handler &h) {
    /* insert SYCL dependencies */
    h.parallel_for(num_items, [=](auto i) {
        /* insert numeric code here */
    });
});
```
**Dummy SYCL kernel/task**

```cpp
sycl::event my_event = queue.submit([&](sycl::handler &h) {
    /* insert SYCL dependencies */
    h.parallel_for(num_items, [=](auto i) {
        /* insert numeric code here */
    });
});
```

**Call HPX-SYCL integration**

```cpp
hpx::future<void> my_future =
    hpx::sycl::experimental::detail::get_future(my_event);
```
Dummy SYCL kernel/task

```cpp
sycl::event my_event = queue.submit([&](sycl::handler &h) {
    /* insert SYCL dependencies */
    h.parallel_for(num_items, [=](auto i) {
        /* insert numeric code here */
    });
});
```

Call HPX-SYCL integration

```cpp
hpx::future<void> my_future =
    hpx::sycl::experimental::detail::get_future(my_event);
```

Add HPX continuation asynchronously

```cpp
hpx::future<void> continuation_future =
    my_future.then([&](auto&& fut) {
        /* insert CPU work, communication,... */
    });
```
**Dummy SYCL kernel/task**

```cpp
sycl::event my_event = queue.submit([&](sycl::handler &h) {
    /* insert SYCL dependencies */
    h.parallel_for(num_items, [=](auto i) {
        /* insert numeric code here */
    });
});
```

**Call HPX-SYCL integration**

```cpp
hpx::future<void> my_future =
    hpx::sycl::experimental::detail::get_future(my_event);
```

**Add HPX continuation asynchronously**

```cpp
hpx::future<void> continuation_future =
    my_future.then([&](auto &&fut) { /* insert CPU work, communication, ... */ });
```

**Suspend calling HPX task until everything is done**

```cpp
continuation_future.get()
```
Use HPX-SYCL Executor for convenience

- Wrapper for in-order SYCL queues
- Allows passing SYCL queue functions directly to hpx::async

```cpp
hpx::sycl::experimental::sycl_executor
  exec(sycl::default_selector{});
  auto fut = hpx::async(exec,
    &sycl::queue::submit, [&](sycl::handler& h) {
      /* insert buffer accessors */
      h.parallel_for(num_items, [=](auto i) {
        /* insert numeric code here */ });
    });
```
Scientific Application as a Benchmark: Octo-Tiger

Octo-Tiger: Overview

- Simulation of interacting binary star systems and stellar mergers
  - Double white dwarf mergers
  - Contact binary v1309 and its merger
  - R Coronae Borealis stars
- Intended for large scale, distributed runs
  - Previous runs: Cori, Piz Daint, Summit
  - Current target: Perlmutter
- Based on the HPX runtime
Scientific Application as a Benchmark: Octo-Tiger

Octo-Tiger: Overview

- Simulation of interacting binary star systems and stellar mergers
  - Double white dwarf mergers
  - Contact binary v1309 and its merger
  - R Coronae Borealis stars
- Intended for large scale, distributed runs
  - Previous runs: Cori, Piz Daint, Summit
  - Current target: Perlmutter
- Based on the HPX runtime

Octo-Tiger as an HPX-SYCL Benchmark

- All major solvers are implemented with Kokkos
Octo-Tiger: Overview

- Simulation of interacting binary star systems and stellar mergers
  - Double white dwarf mergers
  - Contact binary v1309 and its merger
  - R Coronae Borealis stars
- Intended for large scale, distributed runs
  - Previous runs: Cori, Piz Daint, Summit
  - Current target: Perlmutter
- Based on the HPX runtime

Octo-Tiger as an HPX-SYCL Benchmark

- All major solvers are implemented with Kokkos
- Kokkos supports various CPU/GPU execution and memory spaces (CUDA, HIP, HPX and SYCL spaces available)
Scientific Application as a Benchmark: Octo-Tiger

Octo-Tiger: Overview

- Simulations of interacting binary star systems and stellar mergers
  - Double white dwarf mergers
  - Contact binary v1309 and its merger
  - R Coronae Borealis stars
- Intended for large scale, distributed runs
  - Previous runs: Cori, Piz Daint, Summit
  - Current target: Perlmutter
- Based on the HPX runtime

Octo-Tiger as an HPX-SYCL Benchmark

- All major solvers are implemented with Kokkos
- Kokkos supports various CPU/GPU execution and memory spaces (CUDA, HIP, HPX and SYCL spaces available)
- Kokkos kernels can run a SYCL execution space
Octo-Tiger: Overview

- Simulation of interacting binary star systems and stellar mergers
  - Double white dwarf mergers
  - Contact binary v1309 and its merger
  - R Coronae Borealis stars
- Intended for large scale, distributed runs
  - Previous runs: Cori, Piz Daint, Summit
  - Current target: Perlmutter
- Based on the HPX runtime

Octo-Tiger as an HPX-SYCL Benchmark

- All major solvers are implemented with Kokkos
- Kokkos supports various CPU/GPU execution and memory spaces (CUDA, HIP, HPX and SYCL spaces available)
- Kokkos kernels can run a SYCL execution space
- HPX-SYCL integration $\rightarrow$ non-blocking HPX futures for Kokkos kernels running on the SYCL space
Self-gravitating astrophysical fluids

- Inviscid Euler equations (Hydro) → Finite Volumes
- Newtonian Gravity (Gravity) → Fast Multipole Method
Self-gravitating astrophysical fluids

- Inviscid Euler equations (Hydro) → Finite Volumes
- Newtonian Gravity (Gravity) → Fast Multipole Method
- Adaptive Mesh Refinement (AMR)
- Octree refined to maximize resolution for the atmosphere between the stars
- Entire sub-grid in each tree-node
Octo-Tiger: Datastructure and Solvers

Self-gravitating astrophysical fluids

- Inviscid Euler equations (Hydro) $\rightarrow$ Finite Volumes
- Newtonian Gravity (Gravity) $\rightarrow$ Fast Multipole Method
- Adaptive Mesh Refinement (AMR)
- Octree refined to maximize resolution for the atmosphere between the stars
- Entire sub-grid in each tree-node

Kokkos Compute Kernels:

- Solvers traverse the tree, calling compute kernels on each sub-grid individually
- Each Kokkos kernel works on one sub-grid with many concurrent kernels being launched

From [1]
Octo-Tiger: Datastructure and Solvers

Self-gravitating astrophysical fluids

- Inviscid Euler equations (Hydro) $\rightarrow$ Finite Volumes
- Newtonian Gravity (Gravity) $\rightarrow$ Fast Multipole Method
- Adaptive Mesh Refinement (AMR)
- Octree refined to maximize resolution for the atmosphere between the stars
- Entire sub-grid in each tree-node

Kokkos Compute Kernels:

- Solvers traverse the tree, calling compute kernels on each sub-grid individually
- Each Kokkos kernel works on one sub-grid with many concurrent kernels being launched
- Even small scenarios contains thousands of kernel launches within $< 250\,\text{ms} \rightarrow$ good stress test
- Not launching enough kernels in parallel can cause starvation (smallish kernels)
Octo-Tiger: Execution Model

DAG of Compute Kernels

- HPX and Kokkos integrations exist
- Get futures for Kokkos kernels using **HPX-Kokkos compatibility library** (by calling `get_future` specializations within HPX)
- HPX-Kokkos only works for supported execution spaces (previously the CUDA, HIP and HPX spaces)
- Run individual Kokkos kernels either on a CPU (HPX) or GPU execution space
Octo-Tiger: Execution Model

**DAG of Compute Kernels**

- HPX and Kokkos integrations exist
- Get futures for Kokkos kernels using **HPX-Kokkos compatibility library** (by calling `get_future` specializations within HPX)
- HPX-Kokkos only works for supported execution spaces (previously the CUDA, HIP and HPX spaces)
- Run individual Kokkos kernels either on a CPU (HPX) or GPU execution space

**Kernel CPU Execution:**

- Kernel gets split into HPX tasks
- Kernel gets instantiated with appropriate SIMD types

**Kernel GPU Execution:**

- SIMD template types get instantiated with scalar types
- Run on GPU execution space (CUDA, HIP, SYCL?)

---

Adapted from [2]

---

Stellar Mergers with HPX-Kokkos and SYCL: Methods of using an Asynchronous Many-Task Runtime System with SYCL

Gregor Daiß
Octo-Tiger: Execution Model

**DAG of Compute Kernels**

- HPX and Kokkos integrations exist
- Get futures for Kokkos kernels using HPX-Kokkos compatibility library (by calling get_future specializations within HPX)
- HPX-Kokkos only works for supported execution spaces (previously the CUDA, HIP and HPX spaces)
- Run individual Kokkos kernels either on a CPU (HPX) or GPU execution space

**Kernel CPU Execution:**

- Kernel gets split into HPX tasks
- Kernel gets instantiated with appropriate SIMD types

**Kernel GPU Execution:**

- SIMD template types get instantiated with scalar types
- Run on GPU execution space (CUDA, HIP, SYCL?)

Adapted from [2]

---

Adapted from [3]
Required (SYCL-related) Software Additions for Octo-Tiger and its Dependencies

- **HPX:**
  - **Changes:** Implemented both presented HPX-SYCL integration variants
  - **PR:** [https://github.com/STEllAR-GROUP/hpx/pull/6085](https://github.com/STEllAR-GROUP/hpx/pull/6085)
**Required (SYCL-related) Software Additions for Octo-Tiger and its Dependencies**

- **HPX:**
  - **Changes:** Implemented both presented HPX-SYCL integration variants
  - **PR:** [https://github.com/STEllAR-GROUP/hpx/pull/6085](https://github.com/STEllAR-GROUP/hpx/pull/6085)

- **HPX-Kokkos:**
  - **Purpose:** Compatibility layer for HPX and Kokkos. Allows treating Kokkos kernels as HPX tasks IF the \texttt{get\_future} functionality exists for the underlying execution space.
  - **Changes:** Plug in the HPX-SYCL \texttt{get\_future} call. Add \texttt{deep\_copy\_async} overload using the SYCL event directly
  - **PR:** [https://github.com/STEllAR-GROUP/hpx-kokkos/pull/13](https://github.com/STEllAR-GROUP/hpx-kokkos/pull/13)
Required (SYCL-related) Software Additions for Octo-Tiger and its Dependencies

- **HPX:**
  - **Changes:** Implemented both presented HPX-SYCL integration variants
  - **PR:** https://github.com/STEllAR-GROUP/hpx/pull/6085

- **HPX-Kokkos:**
  - **Purpose:** Compatibility layer for HPX and Kokkos. Allows treating Kokkos kernels as HPX tasks IF the `get_future` functionality exists for the underlying execution space.
  - **Changes:** Plug in the HPX-SYCL `get_future` call. Add `deep_copy_async` overload using the SYCL event directly
  - **PR:** https://github.com/STEllAR-GROUP/hpx-kokkos/pull/13

- **CPPuddle:**
  - **Purpose:** Memory and executor utility library for task-based programming. Provides memory recycling allocators
  - **Changes:** Add allocators for SYCL memory pools on the device
  - **PR:** https://github.com/SC-SGS/CPPuddle/pull/15
Required (SYCL-related) Software Additions for Octo-Tiger and its Dependencies

- **HPX:**
  - **Changes:** Implemented both presented HPX-SYCL integration variants
  - **PR:** https://github.com/STEllAR-GROUP/hpx/pull/6085

- **HPX-Kokkos:**
  - **Purpose:** Compatibility layer for HPX and Kokkos. Allows treating Kokkos kernels as HPX tasks IF the `get_future` functionality exists for the underlying execution space.
  - **Changes:** Plug in the HPX-SYCL `get_future` call. Add `deep_copy_async` overload using the SYCL event directly
  - **PR:** https://github.com/STEllAR-GROUP/hpx-kokkos/pull/13

- **CPPuddle:**
  - **Purpose:** Memory and executor utility library for task-based programming. Provides memory recycling allocators
  - **Changes:** Add allocators for SYCL memory pools on the device
  - **PR:** https://github.com/SC-SGS/CPPuddle/pull/15

- **Octo-Tiger:**
  - **Changes:** Use correct SYCL execution space and memory allocators
  - **PR:** https://github.com/STEllAR-GROUP/octotiger/pull/432
Required (SYCL-related) Software Additions for Octo-Tiger and its Dependencies

- **HPX**:
  - **Changes**: Implemented both presented HPX-SYCL integration variants
  - **PR**: https://github.com/STEllAR-GROUP/hpx/pull/6085

- **HPX-Kokkos**:
  - **Purpose**: Compatibility layer for HPX and Kokkos. Allows treating Kokkos kernels as HPX tasks IF the `get_future` functionality exists for the underlying execution space.
  - **Changes**: Plug in the HPX-SYCL `get_future` call. Add `deep_copy_async` overload using the SYCL event directly
  - **PR**: https://github.com/STEllAR-GROUP/hpx-kokkos/pull/13

- **CPPuddle**:
  - **Purpose**: Memory and executor utility library for task-based programming. Provides memory recycling allocators
  - **Changes**: Add allocators for SYCL memory pools on the device
  - **PR**: https://github.com/SC-SGS/CPPuddle/pull/15

- **Octo-Tiger**:
  - **Changes**: Use correct SYCL execution space and memory allocators
  - **PR**: https://github.com/STEllAR-GROUP/octotiger/pull/432

- **Kokkos**:
  - **Already contained SYCL execution and memory space**
  - **Required changes**: Some CMake additions to allow using the SYCL execution space on AMD GPUs
  - **Optional optimization**: Removing internal execution space barriers for in-order queues
  - **PR**: Not yet upstreamed
### Experiment Setup

#### Scenario Size, Number of Kernel Calls per Time-Step

<table>
<thead>
<tr>
<th>Grid parameters</th>
<th>GPU metrics per time-step</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sub-grid size</td>
<td>Overall number of cells</td>
</tr>
<tr>
<td>$8^3$ (512)</td>
<td>262144</td>
</tr>
</tbody>
</table>

#### Scenario

- **Goal:** Evaluate performance with and without the HPX-SYCL integration turned on
- Use patch to turn off the integration by inserting event wait commands and returning ready futures
- Vary number of HPX worker threads (steers contention)
- **Simple Node-Level Hydro-Only Scenario:** Sedov-Taylor Blast Wave
- Using Intel DPC++/OneAPI

#### Hardware

- **NVIDIA® GPU node**
  - CPU: Intel® Xeon® Platinum 8358 CPU
  - NVIDIA A100 GPU
- **AMD® GPU node**
  - CPU: AMD EPYC™ 7H12 CPU.
  - GPU: AMD MI100 GPU
- **Use best combination of performance parameters** for each node (number of concurrent GPU executors, dynamic work aggregation limit)
**Results: Host Task Integration**

**Sedov Blast Wave Scenario on a NVIDIA A100:**
Time-per-timestep with and without the (host task) HPX-SYCL Integration
(Using 32 GPU executors, with up to 8 kernels aggregated)

- **Time-per-timestep HPX-SYCL OFF**
- **Time-per-timestep HPX-SYCL ON**

**A100: Best combinations**
- HPX-SYCL ON Speedup (w.r.t to OFF)
  - 1.17x
  - 1.10x
  - 0.93x
  - 0.73x
  - 0.48x
  - 0.33x

**Sedov Blast Wave Scenario on an AMD MI100:**
Time-per-timestep with and without the (host task) HPX-SYCL Integration
(Using 8 GPU executors, with up to 32 kernels aggregated)

**MI100: Best combinations**
- HPX-SYCL ON Speedup (w.r.t to OFF)
  - 1.12x
  - 1.10x
  - 1.11x
  - 1.00x
  - 0.85x
  - 0.87x

---

Stellar Mergers with HPX-Kokkos and SYCL: Methods of using an Asynchronous Many-Task Runtime System with SYCL

Gregor Daiß

14 / 17
Results: Host Task Integration

Sedov Blast Wave Scenario on a NVIDIA A100:
Time-per-timestep with and without the (host task) HPX-SYCL Integration
(Using 32 GPU executors, with up to 8 kernels aggregated)

Sedov Blast Wave Scenario on a AMD MI100:
Time-per-timestep with and without the (host task) HPX-SYCL Integration
(Using 8 GPU executors, with up to 32 kernels aggregated)

→ Runtime degrades when using the host_task-based HPX-SYCL integration (at least when using all CPU cores)
**Results: Event Polling Integration**

Sedov Blast Wave Scenario on a NVIDIA A100:
Time-per-timestep with and without the HPX-SYCL Integration
(Using 32 GPU executors and aggregate up to 8 kernels per launch)

<table>
<thead>
<tr>
<th>Number HPX Worker Threads [log]</th>
<th>Time-per-timestep HPX-SYCL OFF</th>
<th>Time-per-timestep HPX-SYCL ON</th>
</tr>
</thead>
<tbody>
<tr>
<td>1047</td>
<td>1.19x</td>
<td>1.16x</td>
</tr>
<tr>
<td>562</td>
<td>1.21x</td>
<td>1.16x</td>
</tr>
<tr>
<td>308</td>
<td>1.16x</td>
<td>1.09x</td>
</tr>
<tr>
<td>183</td>
<td>1.16x</td>
<td>1.11x</td>
</tr>
<tr>
<td>125</td>
<td>1.09x</td>
<td>1.11x</td>
</tr>
<tr>
<td>107</td>
<td>1.11x</td>
<td>1.09x</td>
</tr>
</tbody>
</table>

HPX-SYCL ON Speedup (w.r.t to OFF)

1.19x, 1.21x, 1.16x, 1.16x, 1.09x, 1.11x

Sedov Blast Wave Scenario on a AMD MI100:
Time-per-timestep with and without the HPX-SYCL Integration
(Using 8 GPU executors and aggregate up to 32 kernels per launch)

<table>
<thead>
<tr>
<th>Number HPX Worker Threads [log]</th>
<th>Time-per-timestep HPX-SYCL OFF</th>
<th>Time-per-timestep HPX-SYCL ON</th>
</tr>
</thead>
<tbody>
<tr>
<td>1314</td>
<td>1.10x</td>
<td>1.11x</td>
</tr>
<tr>
<td>733</td>
<td>1.12x</td>
<td>1.18x</td>
</tr>
<tr>
<td>421</td>
<td>1.18x</td>
<td>1.11x</td>
</tr>
<tr>
<td>308</td>
<td>1.11x</td>
<td>1.09x</td>
</tr>
<tr>
<td>213</td>
<td>1.09x</td>
<td>1.15x</td>
</tr>
<tr>
<td>182</td>
<td>1.15x</td>
<td>1.11x</td>
</tr>
<tr>
<td>100</td>
<td>1.11x</td>
<td>1.15x</td>
</tr>
</tbody>
</table>

HPX-SYCL ON Speedup (w.r.t to OFF)

1.10x, 1.12x, 1.18x, 1.11x, 1.09x, 1.15x

(a) A100: Best combinations

(b) MI100: Best combinations
Results: Event Polling Integration

Sedov Blast Wave Scenario on a NVIDIA A100:
Time-per-timestep with and without the HPX-SYCL Integration
(Using 32 GPU executors and aggregate up to 8 kernels per launch)

(a) A100: Best combinations

→ Runtime consistently improves when using the event polling HPX-SYCL integration (even for this small scenario)

Sedov Blast Wave Scenario on a AMD MI100:
Time-per-timestep with and without the HPX-SYCL Integration
(Using 8 GPU executors and aggregate up to 32 kernels per launch)

(b) MI100: Best combinations
Conclusion

- Developed HPX-SYCL integration allowing us to treat SYCL events as HPX tasks
- Adapted entire Octo-Tiger software stack for SYCL to benchmark the integration(s)
- Event polling integration performs better than (DPC++) host tasks integration
- Integration is beneficial (over synchronous execution without it), even when just running simple, single-node scenarios
- Software stack is still experimental, lots of potential for optimizations

Outlook

- Intel GPUs?
- Integration speedup with distributed runs?

Universität Stuttgart
Thank you for your attention!


Performance using various execution spaces:

(a) Best runs on the NVIDIA A100
(b) Best runs on the AMD MI100
Speedup when removing barriers within Kokkos for in-order queues:

(Sedov Blast Wave Scenario on a NVIDIA A100: Time-per-timestep with and without the Kokkos patch
(Using 32 GPU executors and aggregate up to 8 kernels per launch)

Time-per-timestep without Kokkos fence patch
Time-per-timestep with Kokkos fence patch

Kokkos Patch Speedup

(a) A100: Best combinations

(Sedov Blast Wave Scenario on a AMD MI100: Time-per-timestep with and without the Kokkos patch
(Using 8 GPU executors and aggregate up to 32 kernels per launch)

Time-per-timestep without Kokkos fence patch
Time-per-timestep with Kokkos fence patch

Kokkos Patch Speedup

(b) MI100: Best combinations
Event polling integration: Runtime with varying number of executors

Sedov Blast Wave Scenario on a NVIDIA A100:
Time-per-timestep with and without the HPX-SYCL Integration
(Using 32 HPX worker threads, without dynamic work aggregation)

Time-per-timestep HPX-SYCL OFF
Time-per-timestep HPX-SYCL ON

HPX-SYCL ON Speedup (w.r.t to OFF)

(a) A100: Increasing Number of GPU executors

Sedov Blast Wave Scenario on a AMD MI100:
Time-per-timestep with and without the HPX-SYCL Integration
(Using 32 HPX worker threads, without dynamic work aggregation)

Time-per-timestep HPX-SYCL OFF
Time-per-timestep HPX-SYCL ON

HPX-SYCL ON Speedup (w.r.t to OFF)

(b) MI100: Increasing Number of GPU executors
Event polling integration: Runtime with varying number of aggregated kernels

Sedov Blast Wave Scenario on a NVIDIA A100:
Time-per-timestep with and without the HPX-SYCL Integration
(Using 32 HPX worker threads and 1 GPU executor)

<table>
<thead>
<tr>
<th>Number Max Aggregation [log]</th>
<th>Time-per-timestep HPX-SYCL OFF</th>
<th>Time-per-timestep HPX-SYCL ON</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>773</td>
<td>773</td>
</tr>
<tr>
<td>2</td>
<td>434</td>
<td>434</td>
</tr>
<tr>
<td>4</td>
<td>262</td>
<td>262</td>
</tr>
<tr>
<td>8</td>
<td>203</td>
<td>203</td>
</tr>
<tr>
<td>16</td>
<td>181</td>
<td>181</td>
</tr>
<tr>
<td>32</td>
<td>161</td>
<td>161</td>
</tr>
<tr>
<td>64</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>

Sedov Blast Wave Scenario on a AMD MI100:
Time-per-timestep with and without the HPX-SYCL Integration
(Using 32 HPX worker threads and 1 GPU executor)

<table>
<thead>
<tr>
<th>Number Max Aggregation [log]</th>
<th>Time-per-timestep HPX-SYCL OFF</th>
<th>Time-per-timestep HPX-SYCL ON</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1589</td>
<td>1589</td>
</tr>
<tr>
<td>2</td>
<td>851</td>
<td>851</td>
</tr>
<tr>
<td>4</td>
<td>499</td>
<td>499</td>
</tr>
<tr>
<td>8</td>
<td>322</td>
<td>322</td>
</tr>
<tr>
<td>16</td>
<td>243</td>
<td>243</td>
</tr>
<tr>
<td>32</td>
<td>216</td>
<td>216</td>
</tr>
<tr>
<td>64</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>

HPX-SYCL ON Speedup (w.r.t to OFF)

(c) A100: Increasing number of kernels aggregated
(d) MI100: Increasing number of kernels aggregated