universität innsbruck

FZ HPC

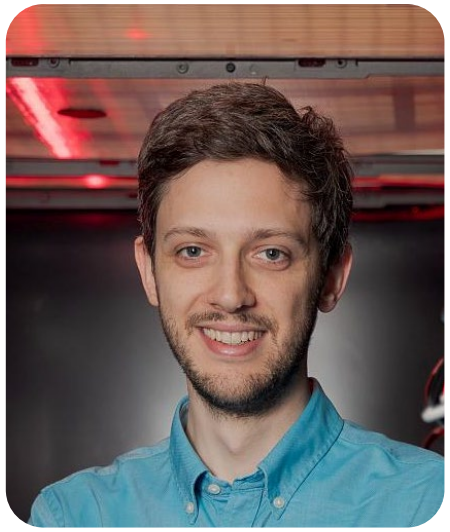
Celerity

Distributed-memory Accelerator Programming Made ~~Easy~~ Easier

Philipp Gschwandtner, Peter Thoman
University of Innsbruck

About Myself

- PhD in Computer Science, currently head of the Research Center HPC @ University of Innsbruck
- Research in and around HPC
  - Measurement/optimization/modeling of performance, energy, efficiency, …
  - APIs, programming models, runtime systems, compilers, ...
- Aid researchers at UIBK in developing and optimizing parallel applications
  - Project proposals (research + infrastructure access)
  - Performance optimizations
  - Training courses



Some Background

- Currently, distributed memory clusters with accelerators provide some of the **best cost- and energy-efficiency** in HPC

  → **39 out of the top 50** entries in the June 2024 Top500 list are accelerator clusters

  → **10 out of the top 10** entries in the June 2024 Green500 list are accelerator clusters

- However, from a user perspective, these systems **combine two very challenging development aspects**
  - *Distributed memory programming*, and
  - *Accelerator computing*

Motivation: Celerity

- Traditionally, optimized HPC software is known to generally be
  - Difficult and time-consuming to construct
  - Highly *manually or semi-manually* tuned – often in a system-specific fashion
  - Implemented with *explicit* message passing and low-level GPU compute (i.e. MPI + CUDA/HIP)

→ **Rigid** implementation

→ Exploring new algorithms at scale requires significant up-front investment

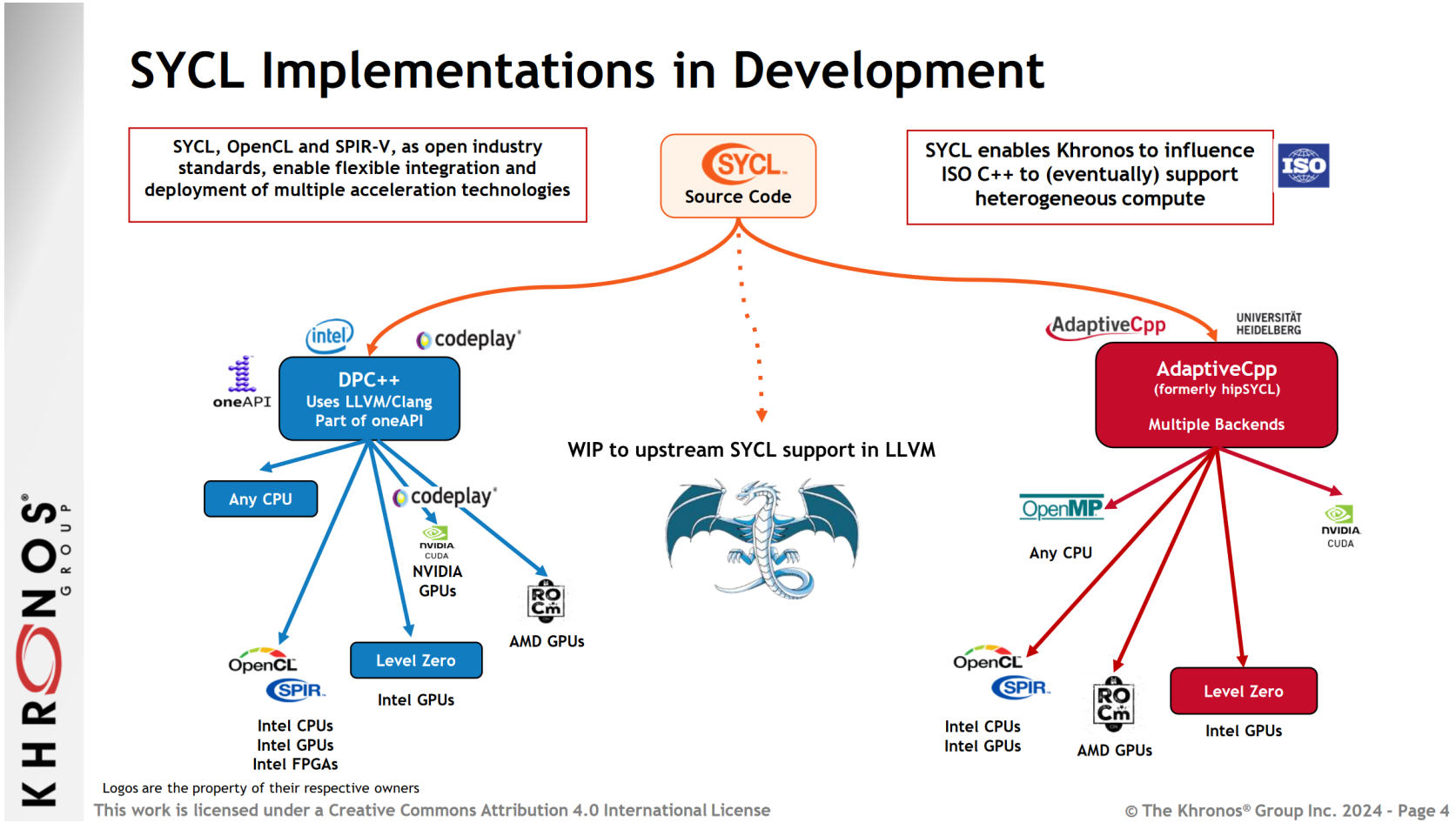
Celerity: Towards a Solution

- Goal: provide a **high-level API** which allows for exploring new implementation ideas on GPU clusters with manageable overhead
  - Only valuable in HPC if this system can offer **competitive performance**
- Lots of things to do behind the scenes to enable this:
  1. A completely **distributed**, asynchronous scheduling execution model for GPU clusters [CCGrid 23]
  2. Automatic **discovery** and optimization of **collective** communication patterns in a high-level representation [HLPP 23]
  3. Effective management of **overhead** and asynchronicity [SNCS 24]

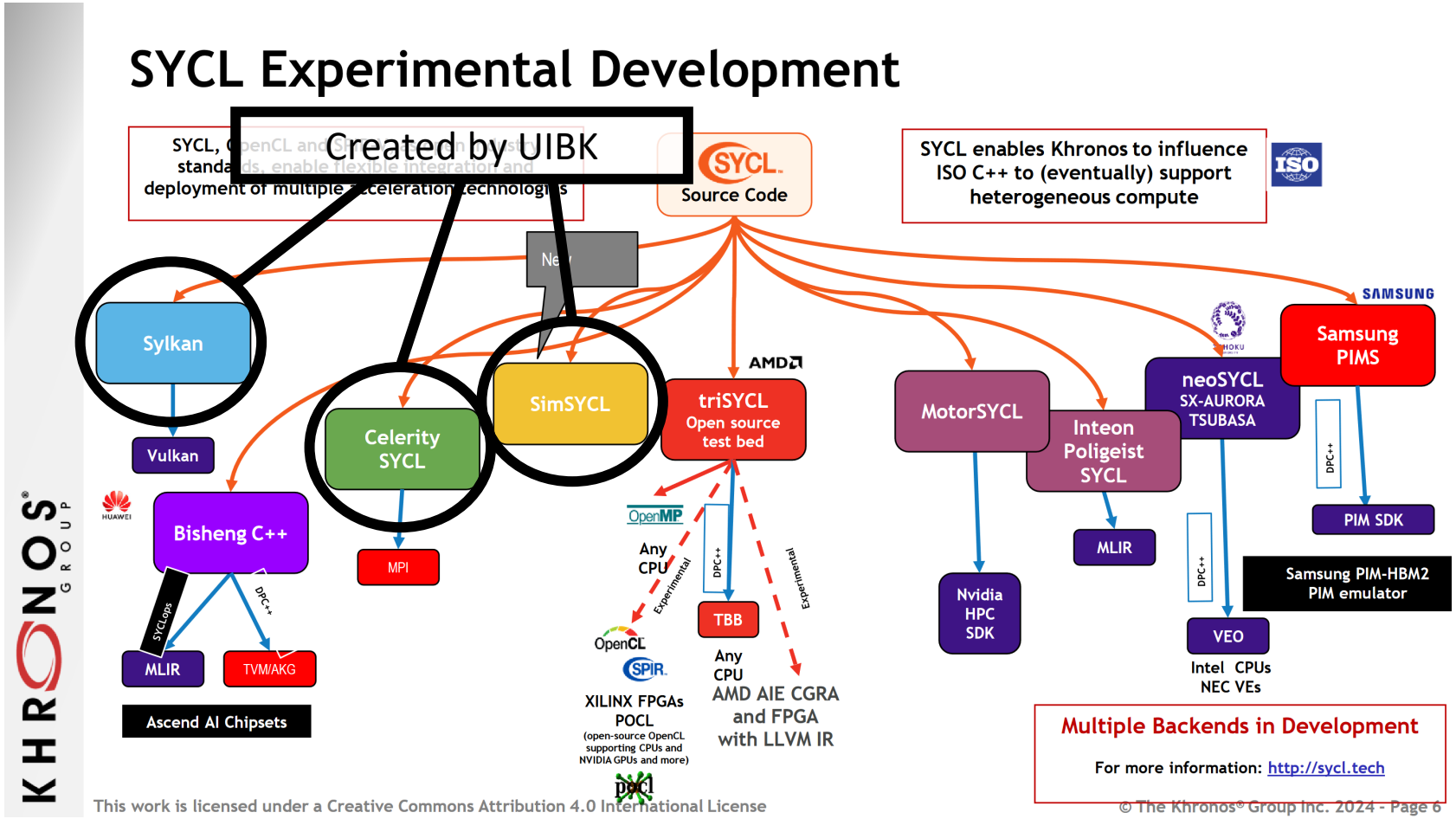
The Celerity Idea

- A **high-level API** designed from the ground up for **accelerator** clusters
  - Allows to constrain data structures and processing patterns to ones efficient on accelerators → less complex than fully general distributed memory programming, does not require a compiler – see AllScale (H2020, 2015-2018)
- Based on the SYCL Khronos industry standard
  - Single-source, modern C++ for accelerators (“embedded DSL”)
  - Designed to run on most hardware supported by OpenCL
  - Several implementations and plethora of platforms
- No explicit distribution, synchronization or communication
  - Derived entirely from data flow

Main SYCL Implementations



Research/Experimental SYCL Implementations



Jacobi Example (1/2)

```
using namespace sycl;
for(int i = 0; i < num_iterations; ++i) {
    queue.submit([=](celerity::handler& cgh) {
        auto nbr = celerity::access::neighborhood<2>{1, 1};
        auto o2o = celerity::access::one_to_one{};
        celerity::accessor in(in_buf, cgh, nbr, read_only);
        celerity::accessor out(out_buf, cgh, o2o, write_only, no_init);
        cgh.parallel_for<class Jacobi>(range<2>{N - 2, N - 2}, {1, 1},
          [=](item<2> itm) {
            const auto i = itm[0];
            const auto j = itm[1];
            out[{i, j}] = (in[{i, j - 1}] + in[{i, j + 1}] +
                           in[{i - 1, j}] + in[{i + 1, j}]) / 4.f;
          });
    });
    std::swap(in_buf, out_buf);
}
```

- **Buffers** encapsulate 1D-3D dense, typed data
  - Accesses are declared explicitly
- **Command groups** are submitted to the **distributed queue**
  - Tying kernels to the buffers they operate on
- **Kernels** execute over N-dimensional range of (virtual) threads

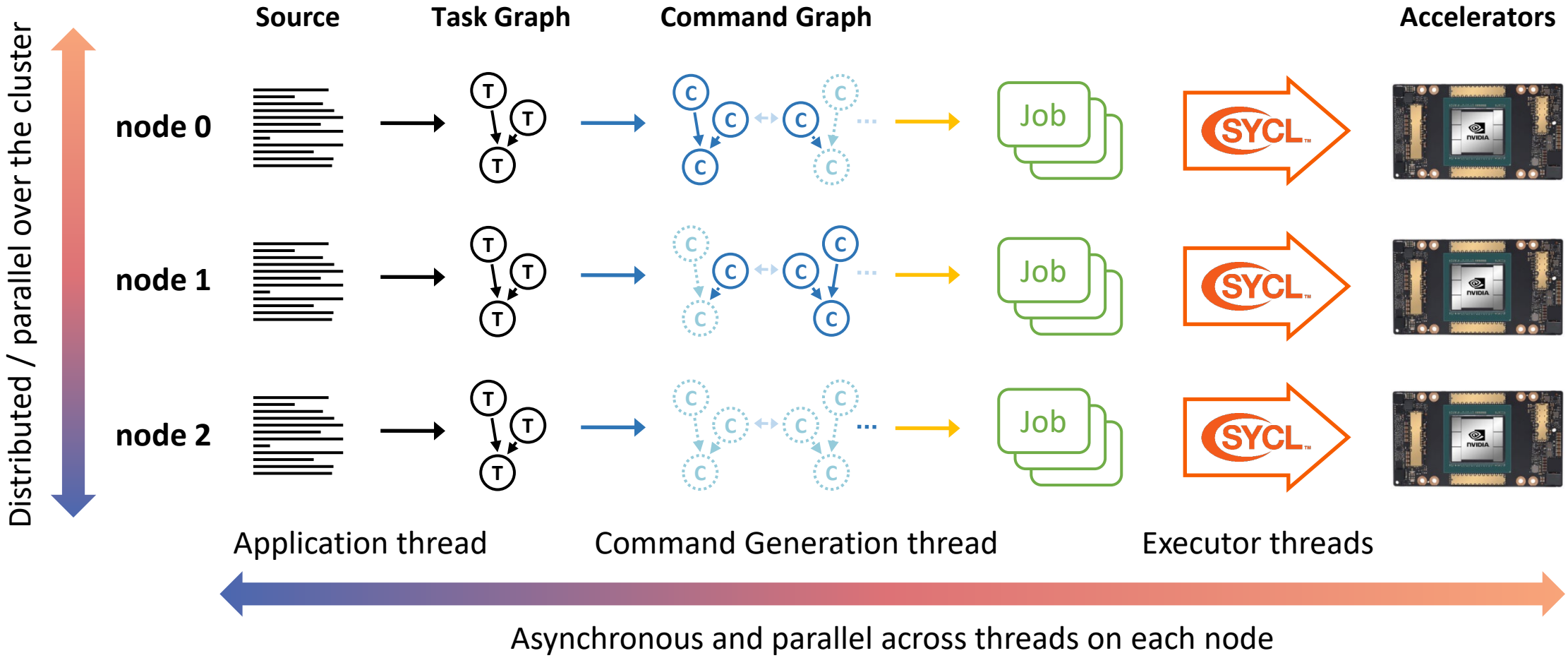
→ All of this is equal to single-GPU SYCL!

Jacobi Example (2/2)

```
using namespace sycl;
for(int i = 0; i < num_iterations; ++i) {
    queue.submit([=](celerity::handler& cgh) {
        auto nbr = celerity::access::neighborhood<2>{1, 1};
        auto o2o = celerity::access::one_to_one{};
        celerity::accessor in(in_buf, cgh, nbr, read_only);
        celerity::accessor out(out_buf, cgh, o2o, write_only, no_init);
        cgh.parallel_for<class Jacobi>(range<2>{N - 2, N - 2}, {1, 1},
          [=](item<2> itm) {
            const auto i = itm[0];
            const auto j = itm[1];
            out[{i, j}] = (in[{i, j - 1}] + in[{i, j + 1}] +
                           in[{i - 1, j}] + in[{i + 1, j}]) / 4.f;
          });
    });
    std::swap(in_buf, out_buf);
}
```

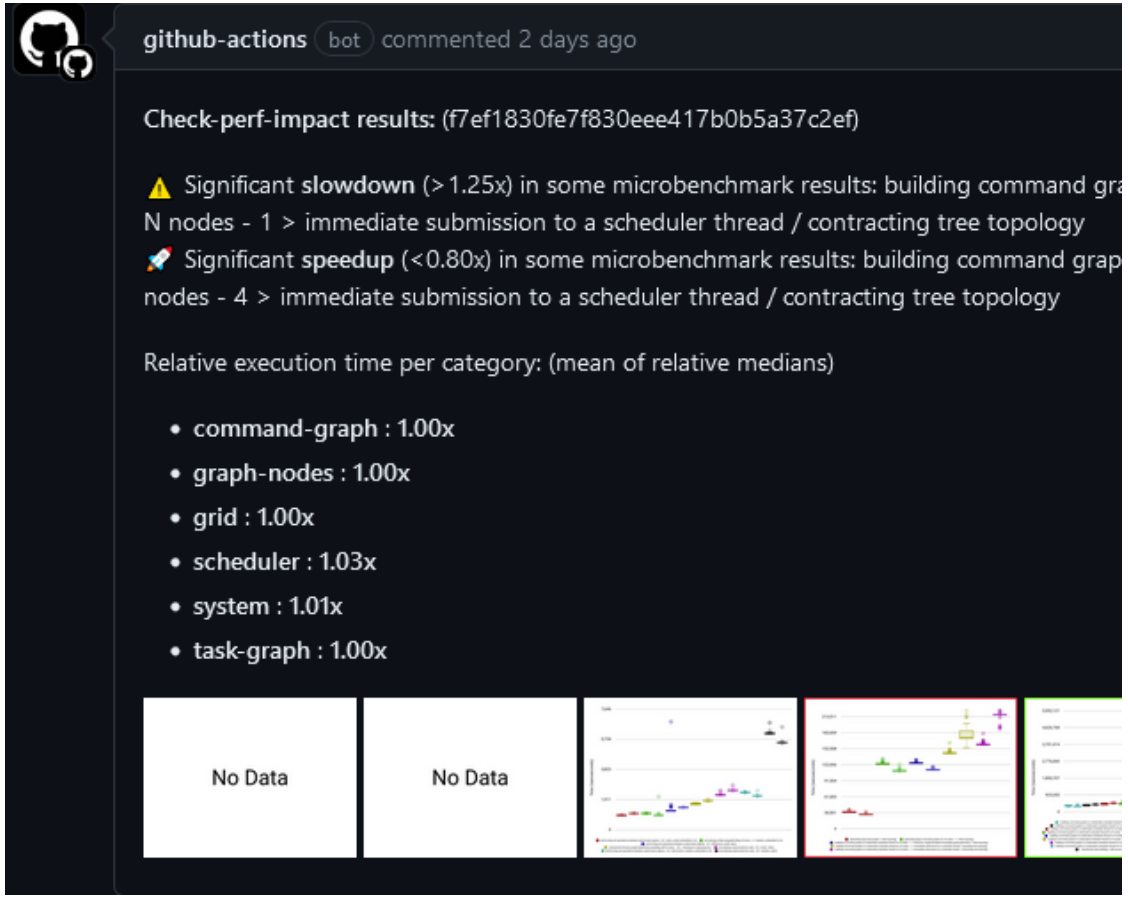
- **Range mappers** declare mapping of kernel subranges to buffer subranges
  - Which data is required to compute *part* of the kernel
  - This allows **splitting** of tasks
- This program **can run on an arbitrary number of nodes**
  - Distributed memory portion is almost completely hidden from the user

Celerity – Overview



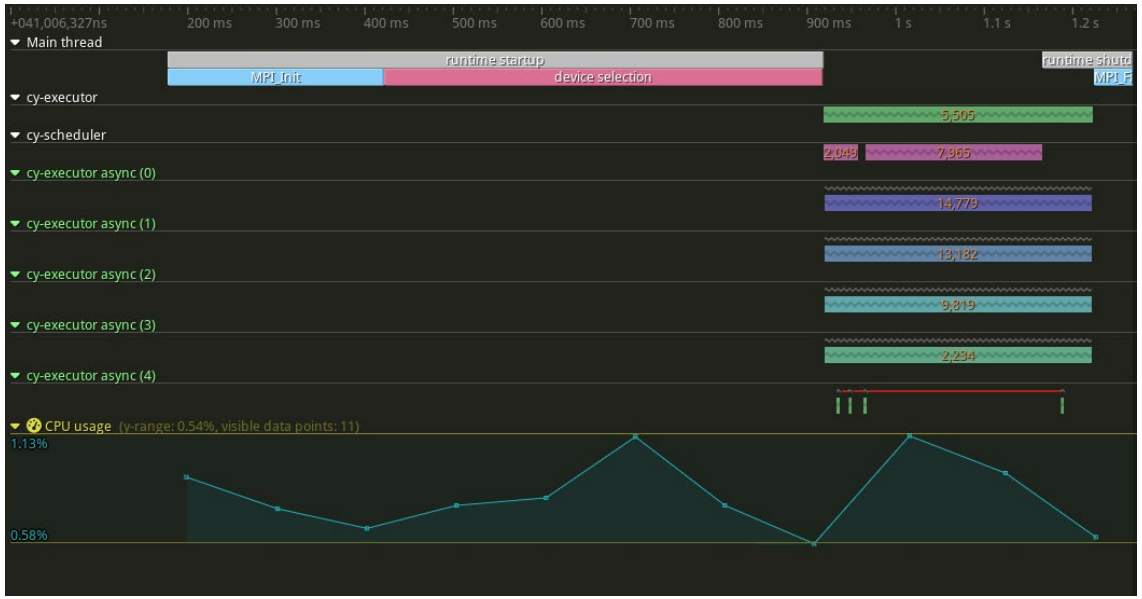
Software Engineering Around Celerity

- Automatic tests
  - CI/CD: automatically triggered distributed-memory tests with SLURM
  - Code Coverage: 95.087%
  - Performance regressions
- Performance Trace Visualization via tracy
- Clang plugin for statically-detectable errors



Software Engineering Around Celerity

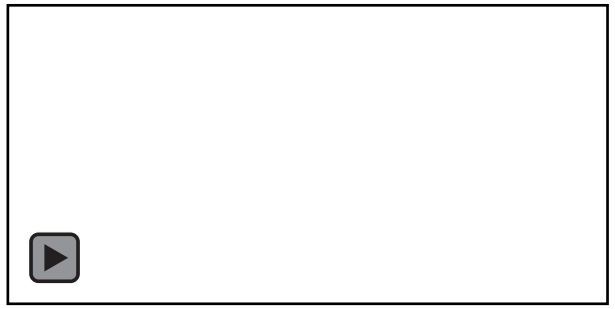
- Automatic tests
  - CI/CD: automatically triggered distributed-memory tests with SLURM
  - Code Coverage: 95.087%
  - Performance regressions
- Performance Trace Visualization via tracy
- Clang plugin for statically-detectable errors

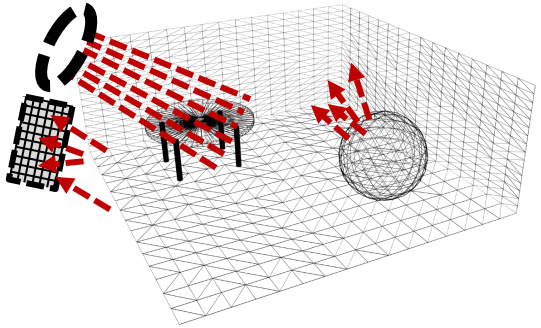


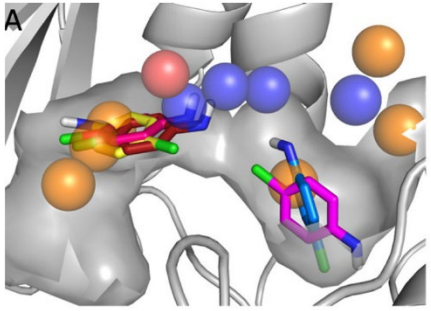
Evaluation through LIGATE (H2020, 2021-2024)

- https://www.ligateproject.eu/

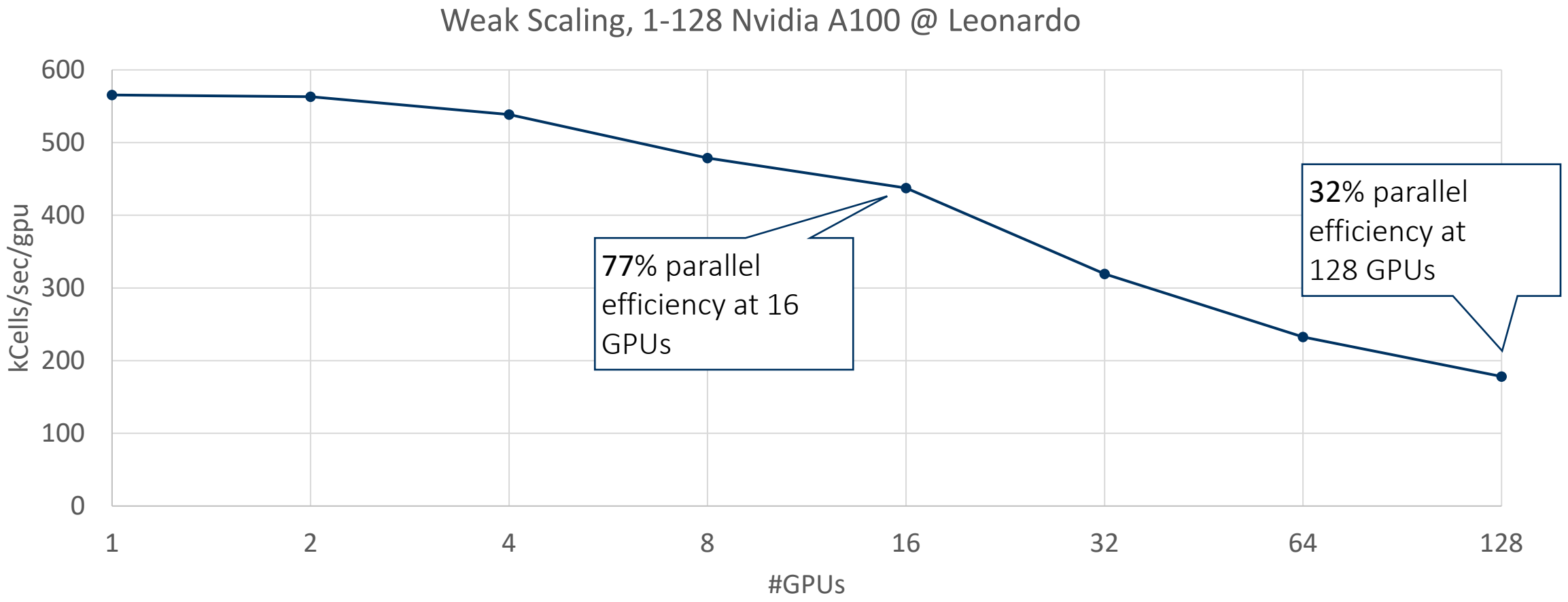
- Cronos
  - structured grid finite volume problem (astrophysics)
  - latency-bound due to dynamic Δt
- RSim
  - radiosity time series simulation (light propagation)
  - heavy on dependency-tracking
- Ligen
  - molecular docking simulation (computer-aided drug design)
  - strong load inbalance dependent on input data





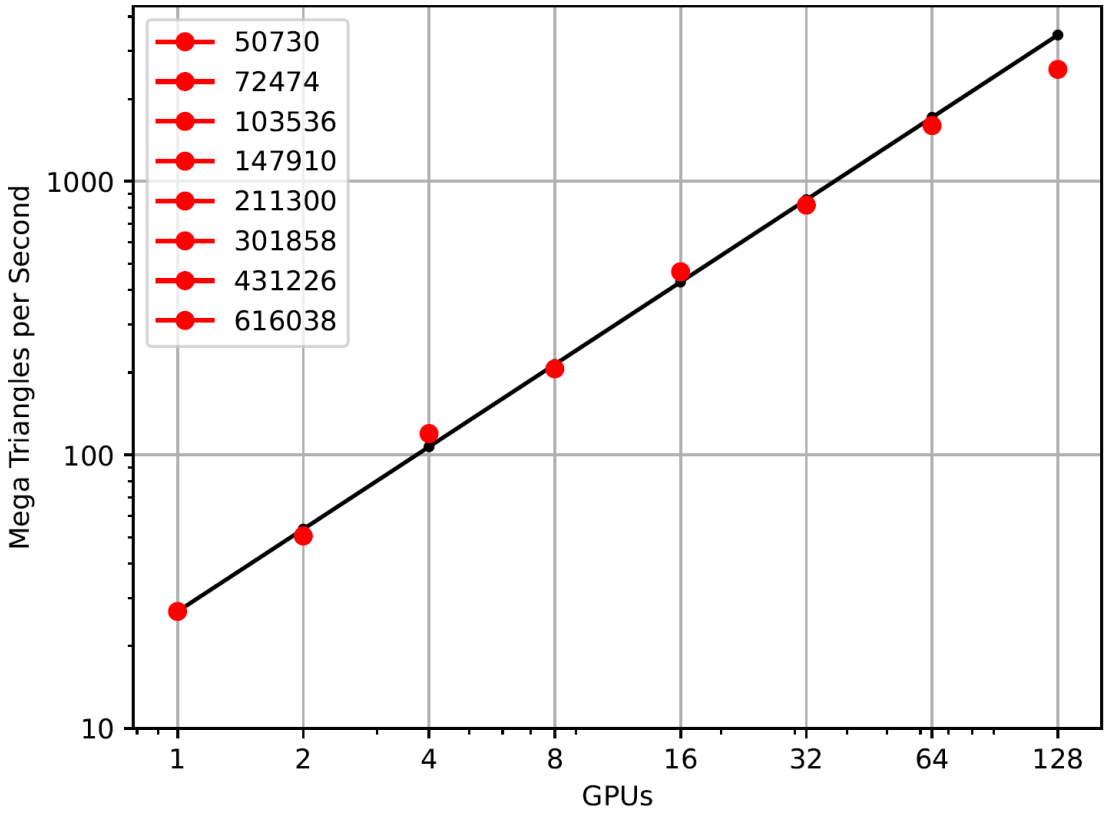


Cronos Weak Scaling on Leonardo (CINECA)

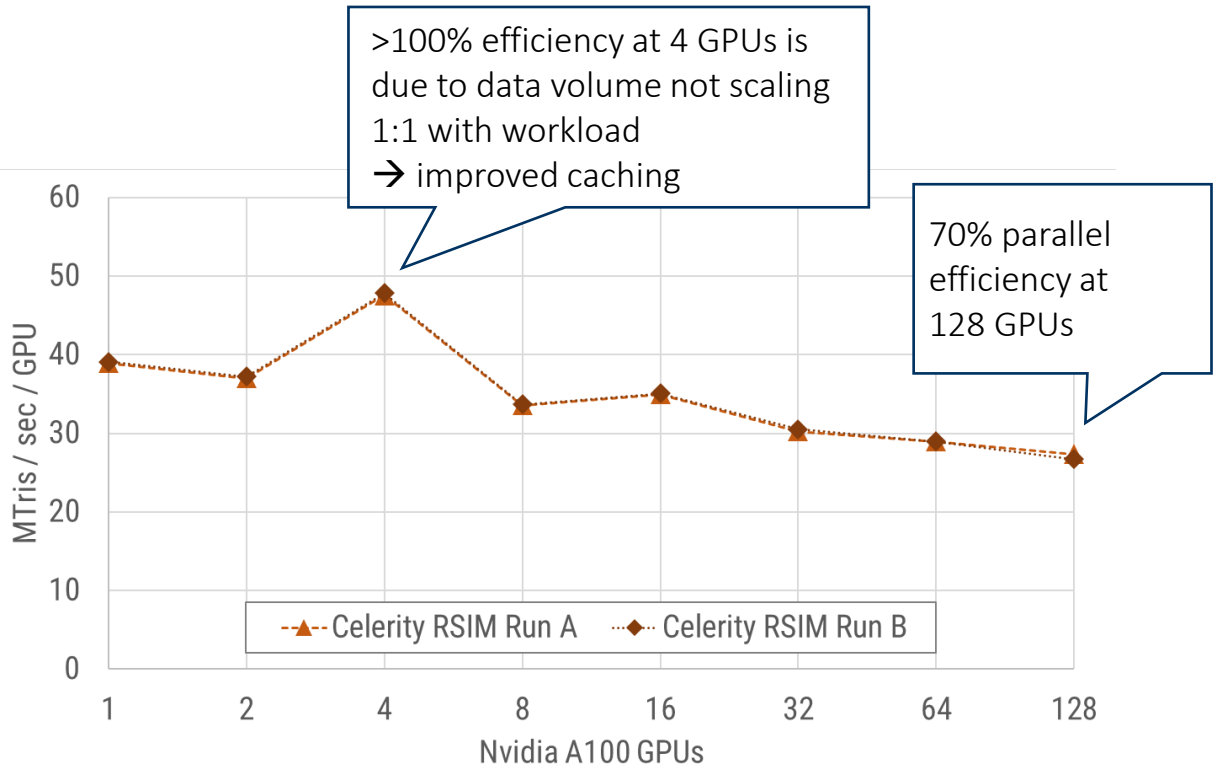


RSim Weak Scaling

On M100 (CINECA)



On Leonardo (CINECA)



Comparison to Related Work

- C/C++
  - StarPU
    - low-level API, explicit distributed memory only
  - Legion
    - performance-first approach, hard to use for non-experts
  - Kokkos
    - *“Note: Kokkos Remote Spaces is in an experimental development stage.”*
- (e)DSLs
  - OpmSs, Chapel, Regent, Julia, …
  - limited tooling support
- MPI+X

Ongoing Work and Future Ideas

- improve performance
  - dynamic load balancing, complex data types
- extend tools and tool support
- auto-generate (some) range mappers
  - involves compiler work
- provide high-level wrappers and (skeleton) libraries to lift the user requirement of mastering modern C++
  - Python (numpy), Julia, Matlab, etc.

Open Issues and Lessons Learned

- Most users *really* don't want to write C++
  - AllScale (H2020, 2015-2018) paper review (paraphrased):
    "I'd rather have to manually match my MPI send and receive calls in C than write C++"
- Based on SYCL, which can't compete with CUDA (yet)
  - SYCL is a specification
  - there are multiple implementations, most somewhat unstable
- Academic project, dependent on funding and recruitment options

Thank you for your attention! **Questions?**

EuroHPC Joint Undertaking

Fabian Knorr
Philip Salzmann
Gabriel Mitterrutzner
Facundo Molina
Peter Thoman
Markus Wippler

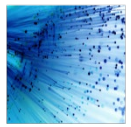
https://celerity.github.io

https://discord.gg/k8vWTPB

LIGATE

https://www.ligateproject.eu

Celerity
High-level C++ for Accelerator Clusters



DPS
Distributed and Parallel Systems

universität innsbruck