Karlsruher Institut für Technologie

13th HPC Café

03.02.2026

Agenda HPC Café – 03.02.2026

Topics for today:

- **Performance lessons learned from porting a stencil code to GPUs (Dr. Holger Obermaier, SCC)**
- **Updates from the HPC operations side**
- Discussion and open floor

As usual this format is meant to be interactive.
Don’t hesitate to ask questions!

Karlsruher Institut für Technologie

Performance lessons learned from porting a stencil code to GPUs

Performance Lessons Learned from Porting a Stencil Code to GPUs

Begatim Bytyqi René Caspart Holger Obermaier | 3. Feb. 2026

Scientific Computing Center

Introduction

Starting Point

- *Voucher project* of the Software Sustainability and Performance Engineering (SSPE) Team
- Target application: IFOS3D (*Inversion of Full Observed Seismograms*)
- Goal: Enable rapid implementation and testing of new scientific ideas
- Challenge: Current runtime and scalability limits practical experimentation

Code Optimizations

- Identify bottlenecks via profiling and performance analysis ⇒ Code is *memory-bound*
- Compute kernels: improve cache reuse and SIMD efficiency (loop order, tiling, vectorization)
- MPI: reduce communication overhead and overlap communication with computation (non-blocking MPI)
- I/O: optimize temporary-data handling for local, job-local, and global storage
  ⇒ particularly relevant for back-propagation workflow
- Optimize program setup (best balance between number of tasks and communication and IO overhead)
- Modernization: Porting code from C to C++:
  - Leverage modern language features for maintainability
  - Replaced pointer arithmetic in 3D-tensor types with direct mapping of 3D-index to 1D-array

Introduction

Pre-Considerations for GPU Offloading

- One code base
  $\Rightarrow$ Avoid divergence between CPU and GPU code bases
- Software requirements
  - Compilers
  - Libraries
- Hardware portability
  - Intel, AMD, NVIDIA GPUs
  - x86, ARM CPUs
- Decision criteria
  - How future-proof is the offload solution?
  - How readable is the implementation?
  - How easy can the code be extended?
  - How good does the current code base fit the offload solution?
  - Expected performance vs. expected implementation effort
    Does the gain in performance justify the necessary effort?

Proxy App

Why Use a Proxy App?

Finding

- Majority of the time spent in stencil code

Implications

- Avoid complex setup: No MPI, no storage, no build system
- Focus on performance-critical parts
- Clean, reproducible hotspot measurement
- Compare multiple implementations
- Simpler verification of code correctness

Proxy App

3D-Stencil Code

```
constexpr void stencil(const int &i, const int &j, const int &k) noexcept {
    float sp_x =
        dx * (b1 * (mat[i, j + 1, k] - mat[i, j,     k]) +
              b2 * (mat[i, j + 2, k] - mat[i, j - 1, k]));  // 6 FLOP
    float sp_y =
        dy * (b1 * (mat[i + 1, j, k] - mat[i,     j, k]) +
              b2 * (mat[i + 2, j, k] - mat[i - 1, j, k]));  // 6 FLOP
    float sp_z =
        dz * (b1 * (mat[i, j, k + 1] - mat[i, j, k]) +
              b2 * (mat[i, j, k + 2] - mat[i, j, k - 1]));  // 6 FLOP
    result[i, j, k] += sp_x + sp_y + sp_z;                  // 3 FLOP
}
```

Proxy App

3 Loops, Serial, Vectorized

```
for (int i = lb0; i <= ub0; i++) {
    for (int j = lb1; j <= ub1; j++) {
        #pragma omp simd
        for (int k = lb2; k <= ub2; k++) {
            stencil(i, j, k);
        }
    }
}
```

OpenMP (Open Multi-Processing)

Overview

- Directive-based parallel programming model for C, C++ and Fortran
- Originally only targeted shared-memory multiprocessing
- GPU offload support added more recently
- Managed by nonprofit corporation *OpenMP Architecture Review Board*

Supported Compilers

- AMD ROCm Compiler
- GCC
- Intel oneAPI Compiler
- LLVM
- NVIDIA HPC SDK Compiler

Hardware portability

- CPUs
- AMD GPUs
- Intel GPUs
- NVIDIA GPUs

OpenMP - Implementations

3 Loops, Collapsed to one Loop

```
#pragma omp target teams distribute parallel for \
    defaultmap(none) map(present, to : mat, result) collapse(3)
for (int i = lb0; i <= ub0; i++) {
    for (int j = lb1; j <= ub1; j++) {
        for (int k = lb2; k <= ub2; k++) {
            stencil(i, j, k);
        }
    }
}
```

OpenMP - Implementations

3 Loops Divided into Tiles, Tile Loops Collapsed to one Loop

```
#pragma omp target teams distribute parallel for \
    defaultmap(none) map(present, to : mat, result) collapse(3)
#pragma omp tile sizes(32, 2, 2)
for (int i = lb0; i <= ub0; i++) {
    for (int j = lb1; j <= ub1; j++) {
        for (int k = lb2; k <= ub2; k++) {
            stencil(i, j, k);
        }
    }
}
```

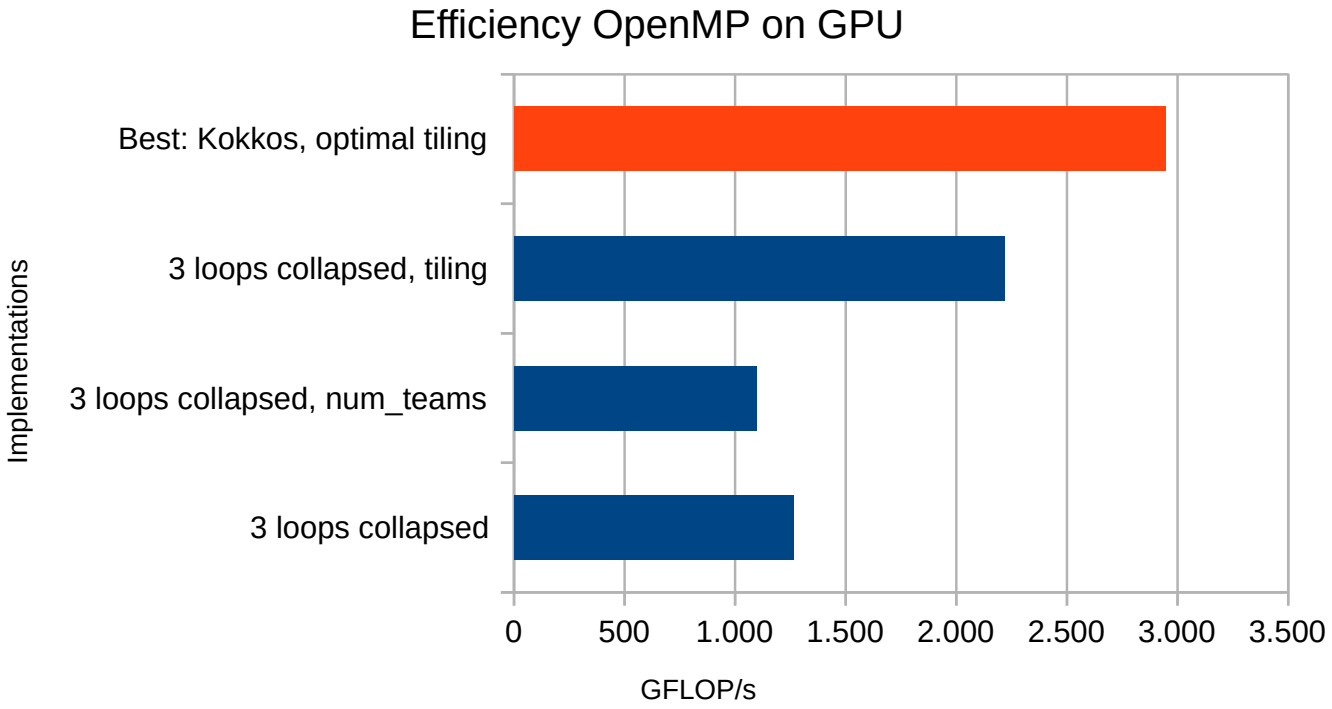
OpenMP - Implementations

3 Loops Collapsed to one Loop, Explicit Configured Number of Teams

```
const size_t num_stencil_0 = lb0 - ub0 + 1;
const size_t num_stencil_1 = lb1 - ub1 + 1;
const size_t num_stencil_2 = lb2 - ub2 + 1;
const size_t num_stencil = num_stencil_0 * num_stencil_1 * num_stencil_2;
const size_t threadLimit = 256;

#pragma omp target teams distribute parallel for \
    num_teams(num_stencil / threadLimit) thread_limit(threadLimit) \
    defaultmap(none) map(present, to : mat, result) collapse(3)
for (int i = lb0; i <= ub0; i++) {
    for (int j = lb1; j <= ub1; j++) {
        for (int k = lb2; k <= ub2; k++) {
            stencil(i, j, k);
        }
    }
}
```

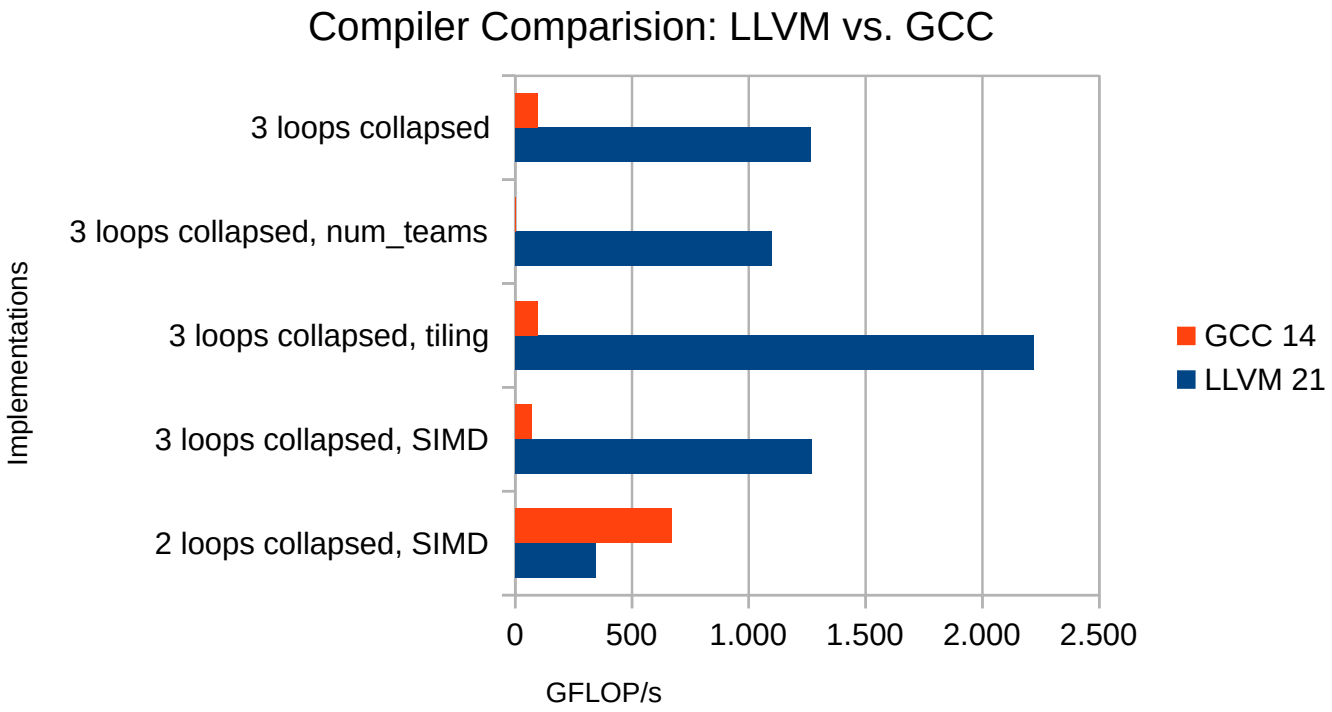
OpenMP - Performance Results



Measurement Environment

- HoreKa Partition: `accelerated-h100`
- GPU: NVIDIA H100
  - Architecture: Hopper (`sm_90`)
  - Memory: 80GB, 3.35TB/s
- Compiler: NVIDIA HPC SDK 25.11

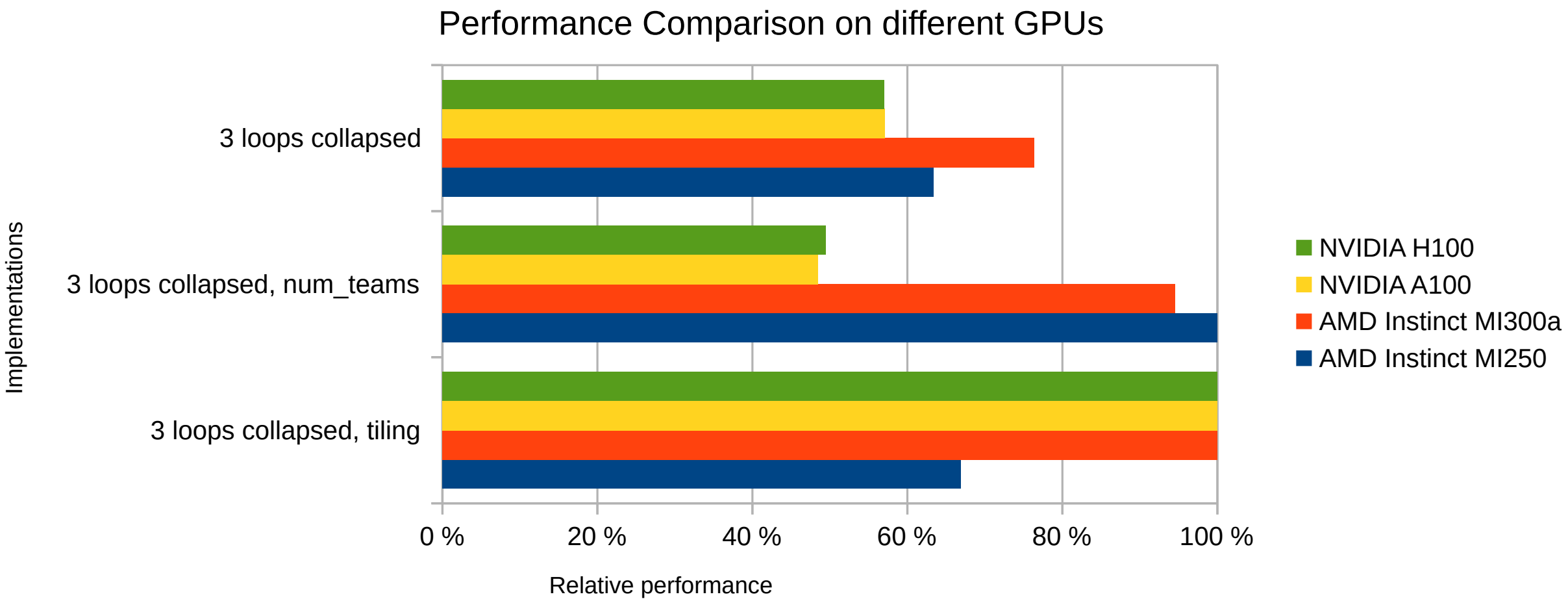
OpenMP - Performance Results



Measurement Environment

- HoreKa Partition: `accelerated-h100`
- GPU: NVIDIA H100
  - Architecture: Hopper (`sm_90`)
  - Memory: 80GB, 3.35TB/s
- Compiler: GCC 14, LLVM 21

OpenMP - Performance Results



Kokkos

Overview

- Programming model in C++ for performance portable applications
- Abstractions for both parallel code execution and data management
- Open Source, Linux Foundation project

Supported Compilers

- All C++ compilers

Hardware Portability

- CPUs (OpenMP backend)
- AMD GPUs (HIP backend)
- Intel GPUs (SYCL backend)
- NVIDIA GPUs (CUDA backend)

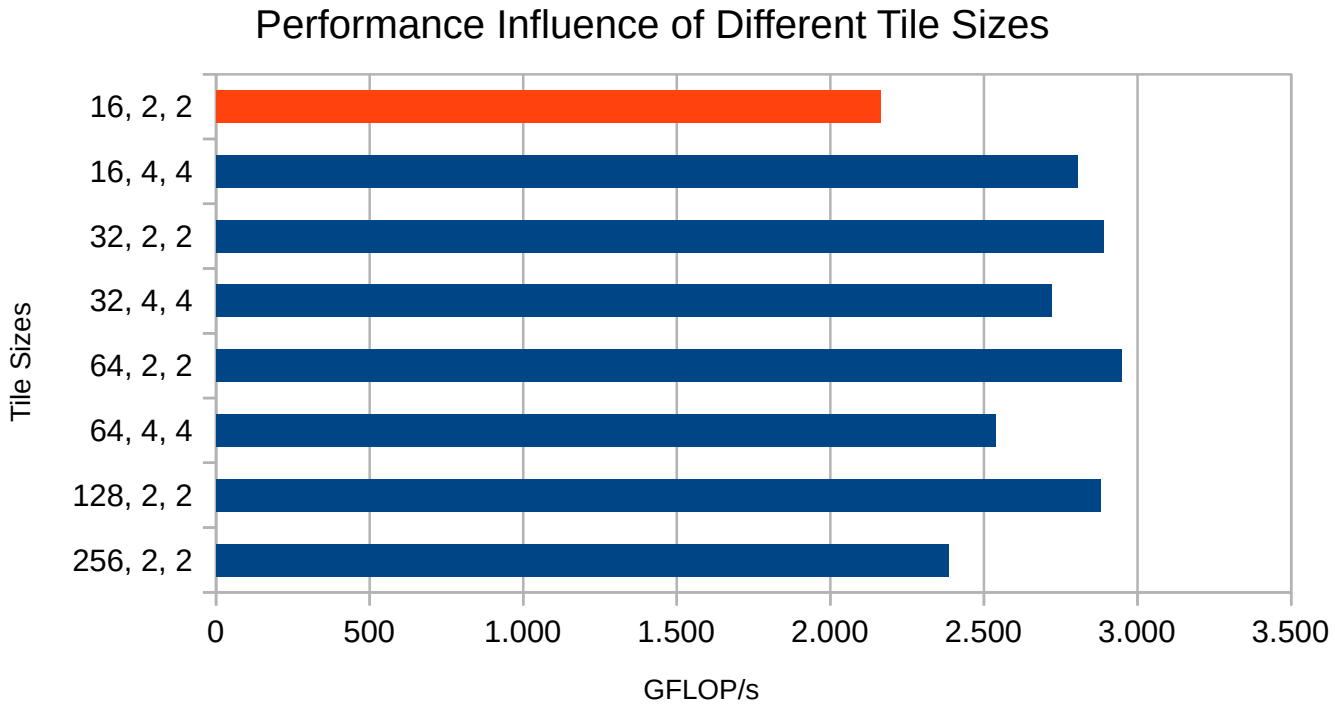
Kokkos - Implementation

3 Loops Divided into Tiles

```
long int tiles[3] = {16, 2, 2};
const Kokkos::MDRangePolicy loop_range{
    {lb0, lb1, lb2},
    {ub0, ub1, ub2},
    tiles
};

Kokkos::parallel_for(
    "stencil iteration on gpu",
    loop_range,
    KOKKOS_LAMBDA(const int &i, const int &j, const int &k) {
        stencil(i, j, k);
    }
);
```

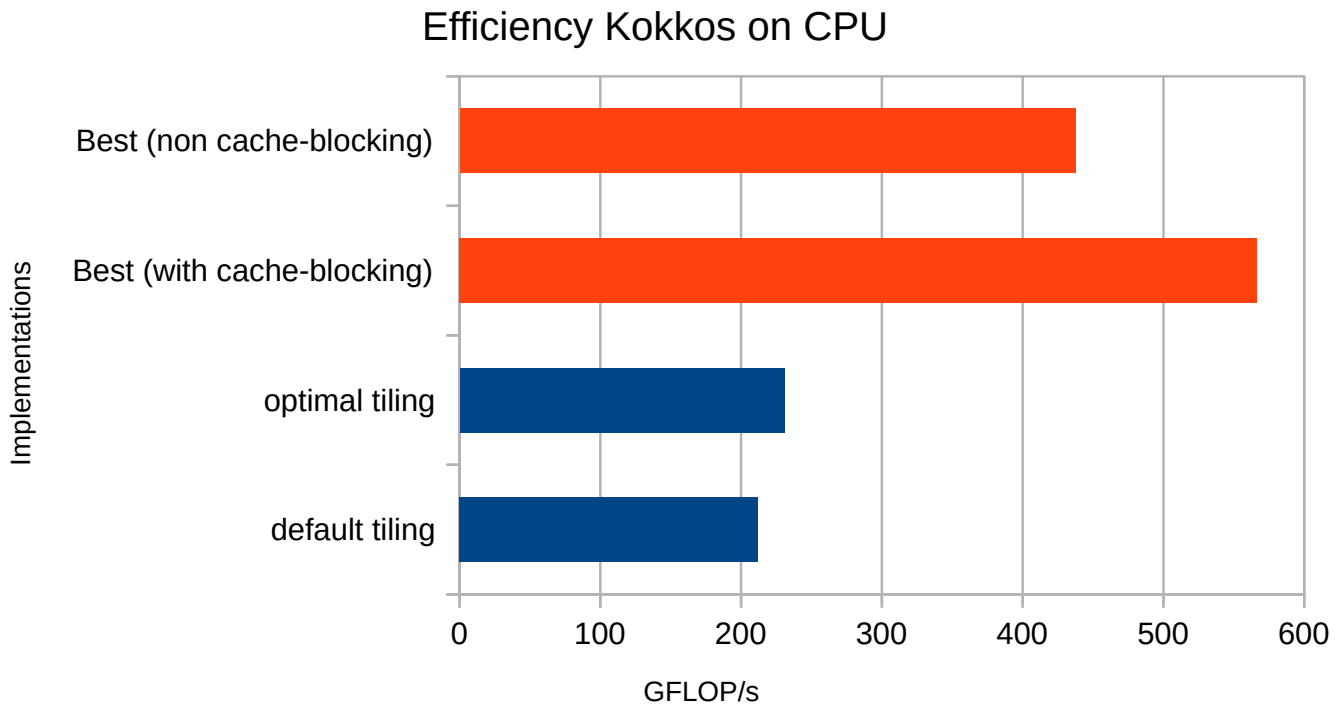
Kokkos - Performance Results



Measurement Environment

- HoreKa Partition: `accelerated-h100`
- GPU: NVIDIA H100
  - Architecture: Hopper (`sm_90`)
  - Memory: 80GB, 3.35TB/s
- Compiler: NVIDIA HPC SDK 25.11
- Default tile size: 16, 2, 2

Kokkos - Performance Results



Measurement Environment

- HoreKa Partition: `accelerated-h100`
- CPU: AMD EPYC 9354
  - Cores: 32, HT: on
  - Memory: 12 x DDR5-4800, 460,8 GB/s
- Compiler: GCC 15
- Best Result (non cache-blocking): OpenMP
- Best Result (with cache-blocking): OpenMP

C++ Standard Parallelism

Overview

- C++17 introduced parallel algorithms, extended in C++20
  - Includes parallel loops operations e.g. `for_each` and `transform_reduce`
  - Execution policies (`seq`, `par`) give compiler hints
  - Single source code for CPU and accelerator
- No explicit data placement / device selection
- Execution can be serial! Parallel execution on CPUs or GPUs needs compiler support

Supported Compilers

- AMD ROCm Compiler
- GCC (CPU only)
- Intel oneAPI Compiler (CPU only)
- LLVM (CPU only)
- NVIDIA HPC SDK Compiler

Hardware Portability

- CPUs
- AMD GPUs
- NVIDIA GPUs

C++ Standard Parallelism - Implementations

3 Loops, 3D-Iterator Created as Cartesian Product of 1D-Iterators

```
auto collapse_3 = std::views::cartesian_product(
    std::ranges::iota_view<int, int>{lb0, ub0 + 1},
    std::ranges::iota_view<int, int>{lb1, ub1 + 1},
    std::ranges::iota_view<int, int>{lb2, ub2 + 1});

std::for_each(
    std::execution::par_unseq, // parallel, unsequenced order
    collapse_3.begin(), collapse_3.end(),
    [&result, &mat](const std::tuple<int, int, int> &index) constexpr noexcept -> void {
        const auto &[i, j, k] = index;
        stencil(i, j, k);
    });
```

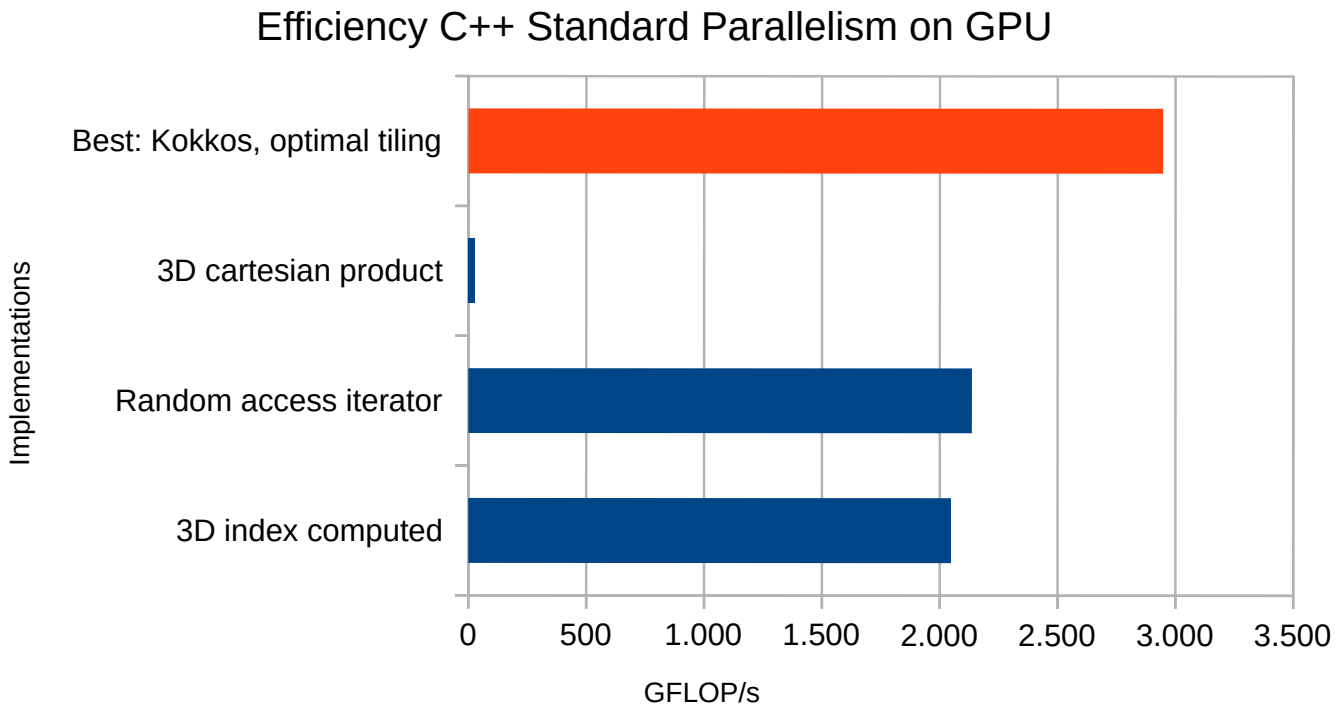
C++ Standard Parallelism - Implementations

3 Loops Collapsed to one Loop, Index i, j, k Computed from 1D-Iterator Index

```
const size_t extent0 = ub0 - lb0 + 1, extent1 = ub1 - lb1 + 1, extent2 = ub2 - lb2 + 1;
const size_t size = extent0 * extent1 * extent2;
const size_t stride0 = extent1 * extent2, stride1 = extent2;
auto iota_loop = std::ranges::iota_view<size_t, size_t>{0, size};
std::for_each(
    std::execution::par_unseq, // parallel, unsequenced order
    iota_loop.begin(), iota_loop.end(),
    [&result, &mat, &stride0, &stride1, &lb0, &lb1, &lb2](size_t idx) constexpr noexcept -> void {
        size_t i = idx / stride0;  idx -= i * stride0;  i += lb0;
        size_t j = idx / stride1;  idx -= j * stride1;  j += lb1;
        size_t k = idx;                                 k += lb2;

        stencil(i, j, k);
    });
```

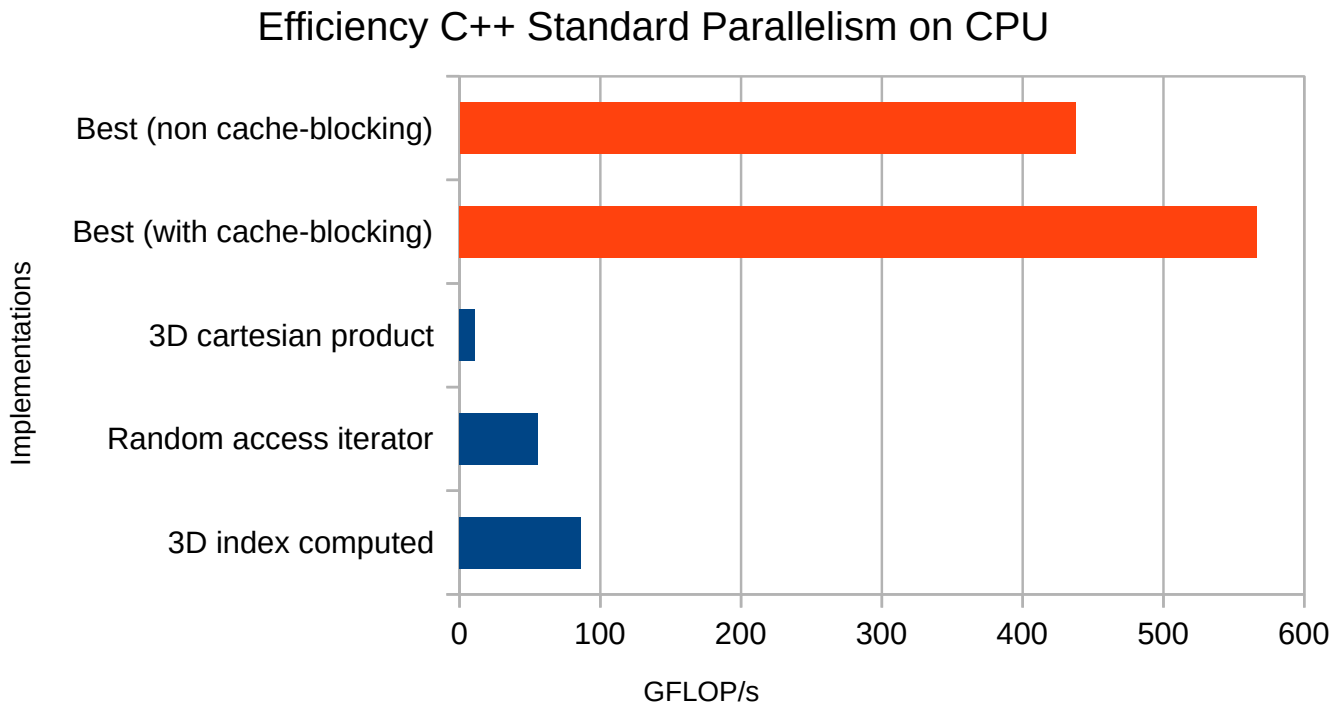
C++ Standard Parallelism - Performance Results



Measurement Environment

- HoreKa Partition: `accelerated-h100`
- GPU: NVIDIA H100
  - Architecture: Hopper (`sm_90`)
  - Memory: 80GB, 3.35TB/s
- Compiler: NVIDIA HPC SDK 25.11

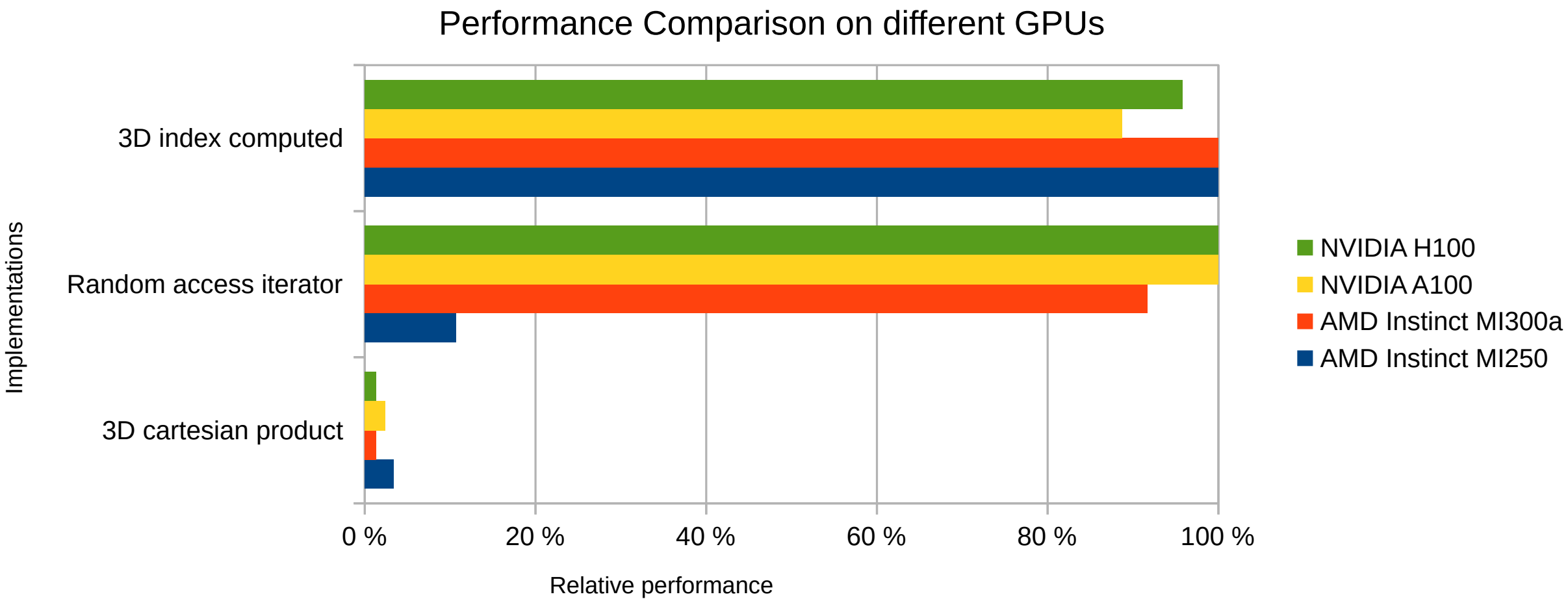
C++ Standard Parallelism - Performance Results



Measurement Environment

- HoreKa Partition: `accelerated-h100`
- CPU: AMD EPYC 9354
  - Cores: 32, HT: on
  - Memory: 12 x DDR5-4800, 460,8 GB/s
- Compiler: NVIDIA HPC SDK 25.11
- Best Result (non cache-blocking): OpenMP
- Best Result (with cache-blocking): OpenMP

C++ Standard Parallelism - Performance Results



Conclusions

Lessons Learned

- Performance varies widely across implementations
- Some GPU implementations can even underperform the CPU version
- GPU focused implementations typically do not deliver optimal performance on CPUs
- Best parameters and best implementation are both compiler- and hardware-dependent

$\Rightarrow$ No one-size-fits-all solution

Proposed Solution

- Provide multiple implementations tailored to:
  - CPU vs GPU
  - different compilers
  - different hardware targets
- Select optimal implementation at:
  - Compile time $\Rightarrow$ smaller binary, less code included
  - Run time $\Rightarrow$ higher flexibility, better portability across systems

Proposed Solution

Implementation Selectable by Template Parameter

```
enum loop3D_impl_T {
    serial,
    serial_tiled,
    omp_tiled
};
constexpr loop3D_impl_T default_impl = serial_tiled;

template <loop3D_impl_T impl = default_impl>
    requires(impl == omp_tiled)
void loop3D(const loop3d_range_T range, auto f) noexcept {
    // OpenMP implementation of 3D-loop with tiling
    // For each indes i, j, k, execute f(i, j, k)
    // ...
}
```

Karlsruher Institut für Technologie

Updates from the HPC operations side

Maintenance and Transition to HoreKa 2

- Maintenance work in the HPC building at Campus North
  - Extensive electrical work in preparation for HoreKa 2
  - Schedule: **Downtime in CW 7 and CW 8**
    - Starting Monday, 9 February 2026 (next Monday)
    - Ending Wednesday, 18 February 2026

- Transition to HoreKa 2 / **no news at this time**
  - Most importantly: you will **be able to use your granted compute time**
  - There will **always be CPU and accelerated resources available**
  - Transition to the new system will happen at a specific date (tbd)

  → We will keep you informed about any steps you might need to do in advance

Karlsruher Institut für Technologie

Time for Discussion

Next HPC Café

- Next HPC Café planned for **5 March 2026, 10 am**
- New regular timeslot for the HPC Café
  - Every **first Thursday of the month 10:00 am**

- As always, open to hear ideas for topics and talks, from …
  - Yourself
  - Colleagues and collaborators
  - Also just expressions for “topics of interest”