OpenMP
Enabling HPC since 1997

OpenMP API Version 6.0 What to Expect

Michael Klemm

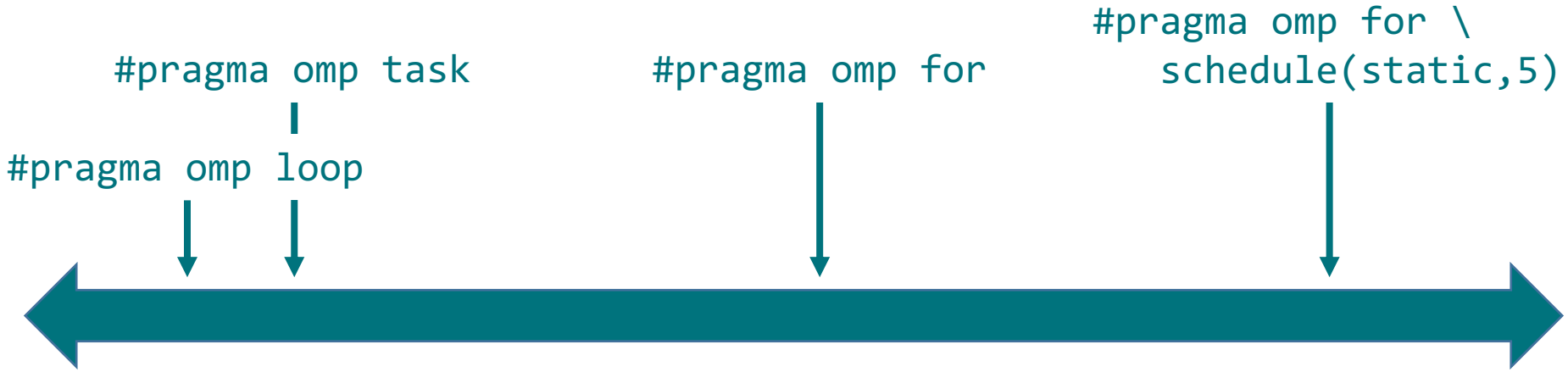
Chief Executive Officer
OpenMP Architecture Review Board
michael.klemm@openmp.org

OpenMP Architecture Review Board

The mission of the OpenMP ARB (Architecture Review Board) is to standardize directive-based multi-language high-level parallelism that is performant, productive and portable.

The OpenMP API moves common approaches into an industry standard to simplify a developers' life.

Continuum of Control



Descriptive

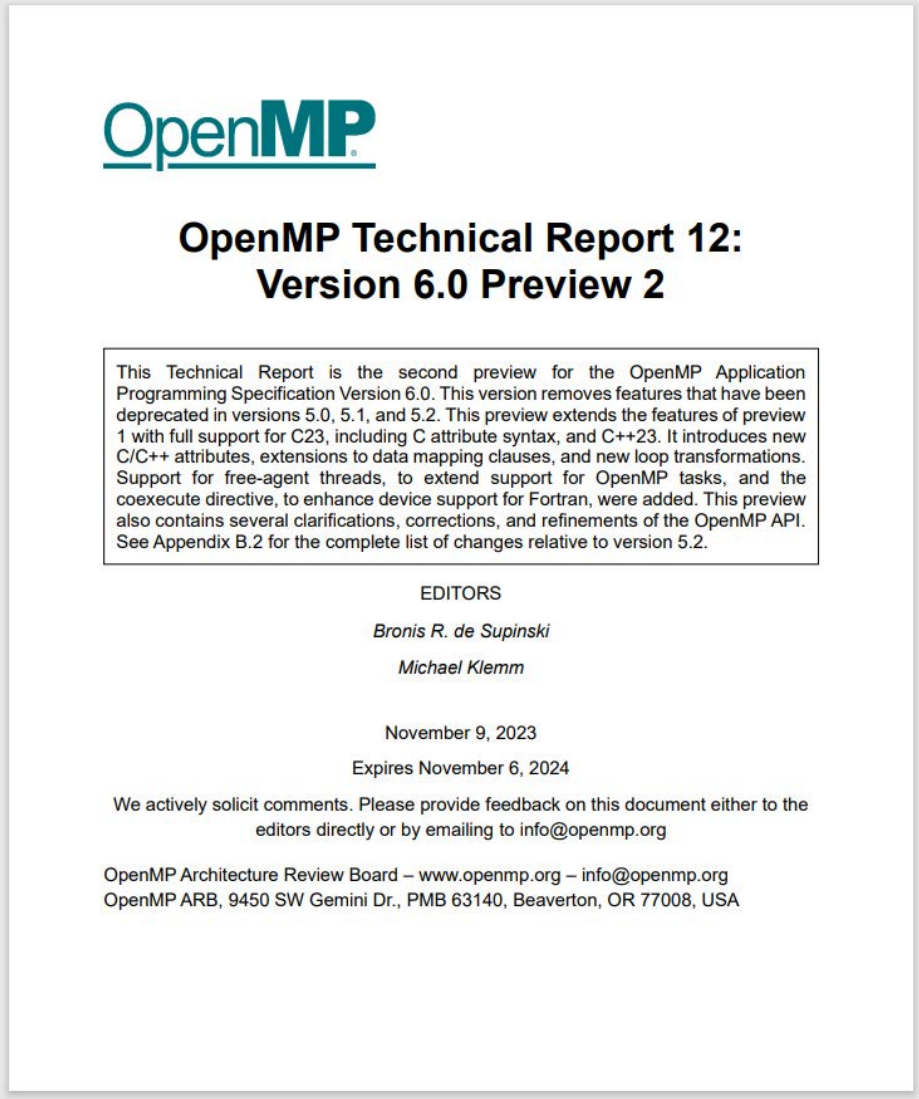
- Express “what”
- Ignore implementation
- Rely on quality of implementation

Prescriptive

- Express “how”
- Focus on implementation
- Expose control over execution

- OpenMP strives to
  - Support a useful subset of this spectrum
  - Provide a structured path from descriptive to prescriptive where needed

Technical Report 12



OpenMP

OpenMP Technical Report 12: Version 6.0 Preview 2

This Technical Report is the second preview for the OpenMP Application Programming Specification Version 6.0. This version removes features that have been deprecated in versions 5.0, 5.1, and 5.2. This preview extends the features of preview 1 with full support for C23, including C attribute syntax, and C++23. It introduces new C/C++ attributes, extensions to data mapping clauses, and new loop transformations. Support for free-agent threads, to extend support for OpenMP tasks, and the coexecute directive, to enhance device support for Fortran, were added. This preview also contains several clarifications, corrections, and refinements of the OpenMP API. See Appendix B.2 for the complete list of changes relative to version 5.2.

EDITORS

Bronis R. de Supinski

Michael Klemm

November 9, 2023

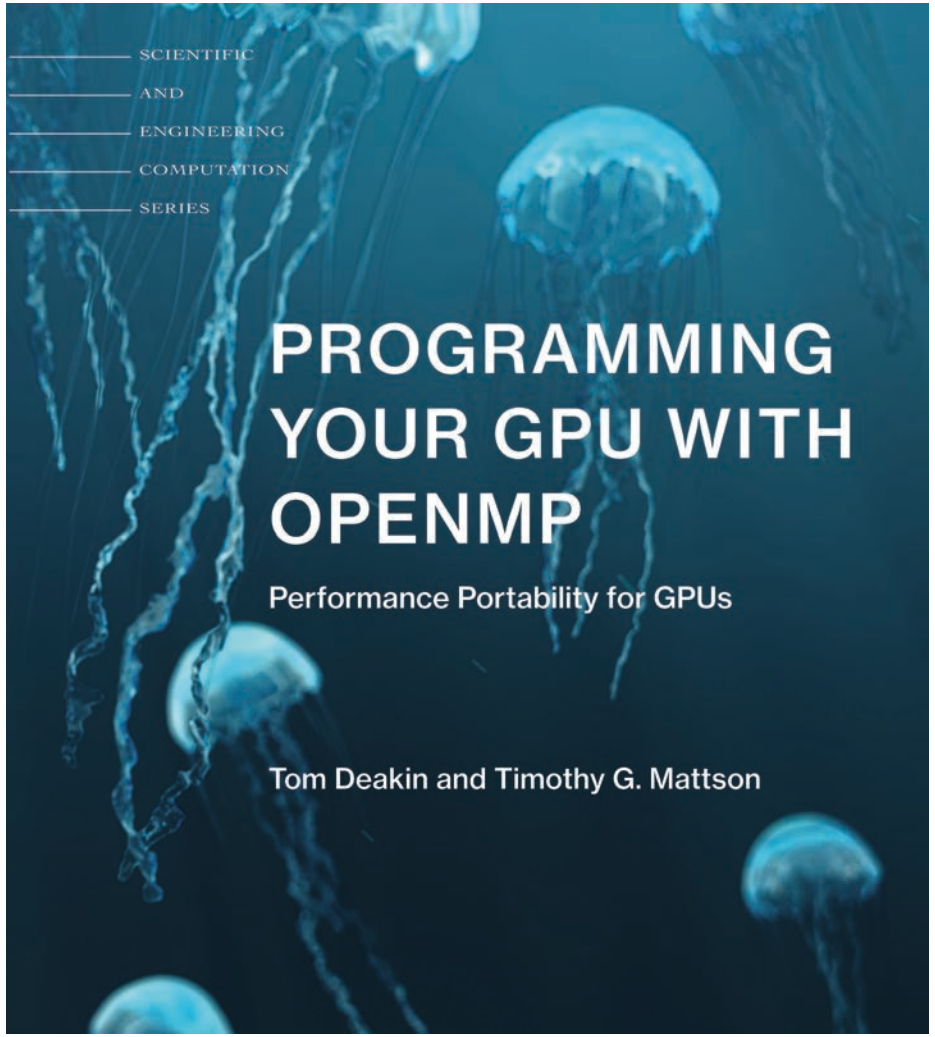
Expires November 6, 2024

We actively solicit comments. Please provide feedback on this document either to the editors directly or by emailing to info@openmp.org

OpenMP Architecture Review Board – www.openmp.org – info@openmp.org
OpenMP ARB, 9450 SW Gemini Dr., PMB 63140, Beaverton, OR 77008, USA

- 2nd preview of OpenMP version 6.0
- Available at https://www.openmp.org

Latest Book on OpenMP

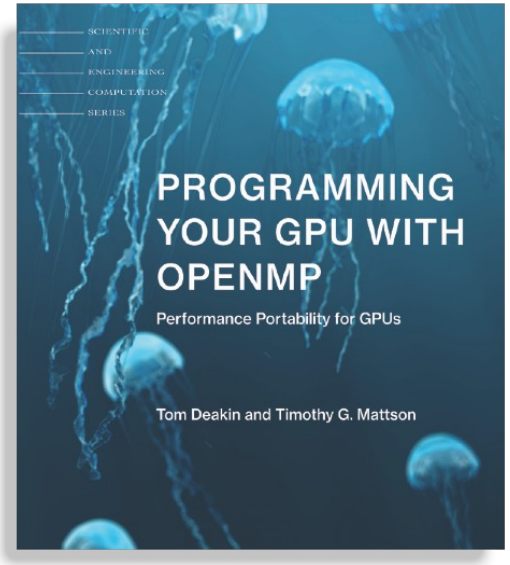


Programming your GPU with OpenMP Performance Portability for GPUs

By Tom Deakin and Tim Mattson

- Released November 7, 2023
- Available from MIT Press and Amazon

OpenMP Books



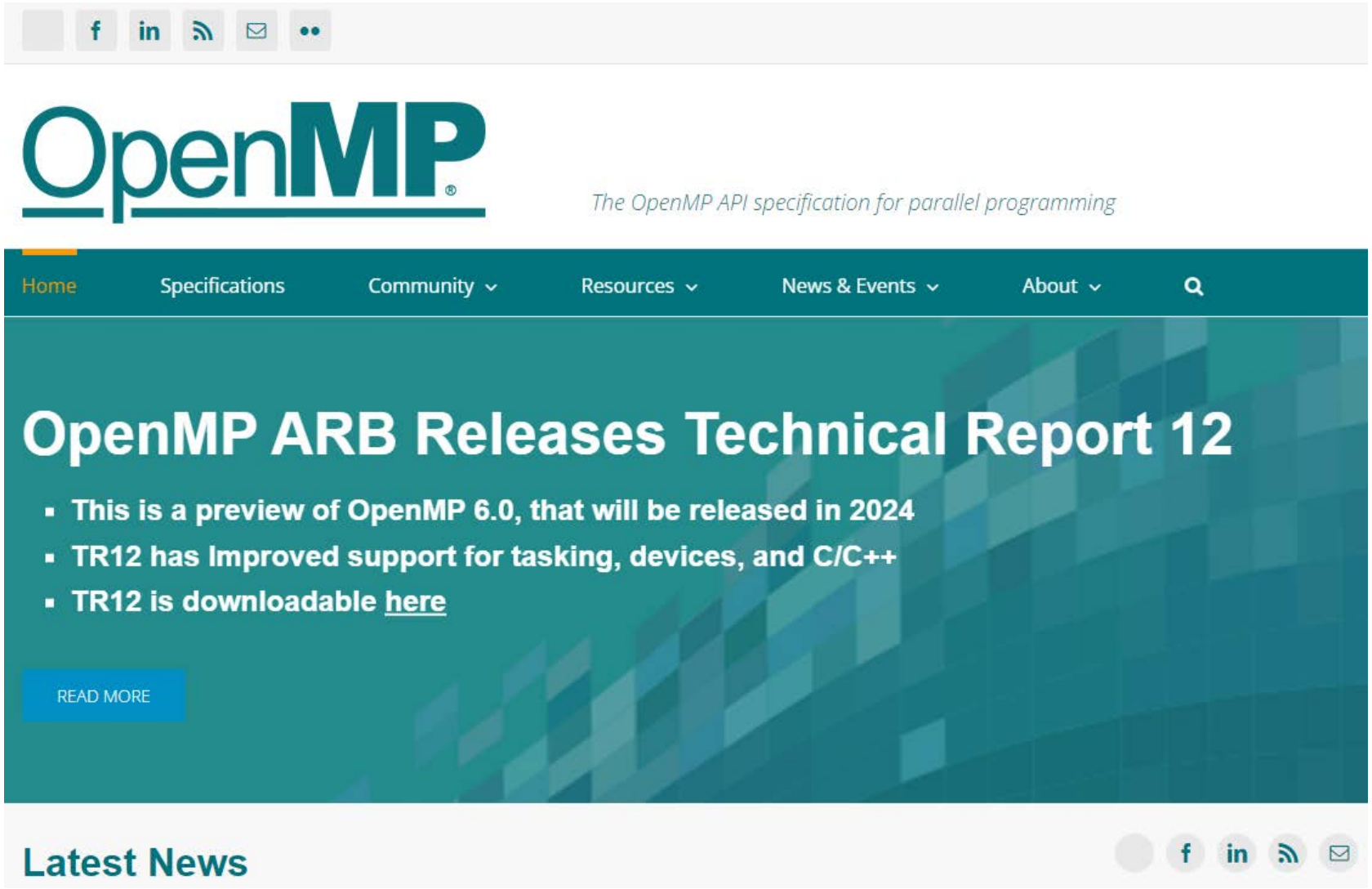
Community Interaction

Check out https://www.openmp.org/news/events-calendar/

Help Us Shape the Future of OpenMP

- OpenMP ARB continues to grow
  - 34 members currently
  - Likely 37 in 2024
- You can contribute to our annual releases
  - Participation in the discussions of the language committee
  - Vote on the inclusion of OpenMP features into the specification
- OpenMP membership types now include less expensive memberships
- Please let us know if you would be interested

Visit www.openmp.org for Information



Agenda

- Vendor Talks and Panel (3m per talk)
  - Xinmin Tian, Intel
  - Graham Lopez, NVIDIA
  - Zach Tschirhart, HPE
  - Johannes Doerfert, LLVM
  - Seungwon Lee, Samsung
  - Dhruva Chakrabarti, AMD
- OpenMP LC Talks and Panel (3m per talk, except Bronis ☺)
  - Bronis de Supinski, Language Committee
  - Stephen Olivier, Tasking
  - Xinmin Tian, C/C++/Fortran
  - Tom Scogland, Accelerator
  - Christian Terboven, Affinity

SC'2023 OpenMP BoF, Nov. 12-17 @ Denver, CO

Intel® oneAPI C/C++ and Fortran Compilers for OpenMP* on CPUs and GPUs

Xinmin Tian
Intel Corporation

Supercomputing 2023

it starts with intel

Legal Notices and Disclaimers

Performance varies by use, configuration and other factors. Learn more on the Performance Index site.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See backup for configuration details. No product or component can be absolutely secure.

Your costs and results may vary.

Intel technologies may require enabled hardware, software or service activation.

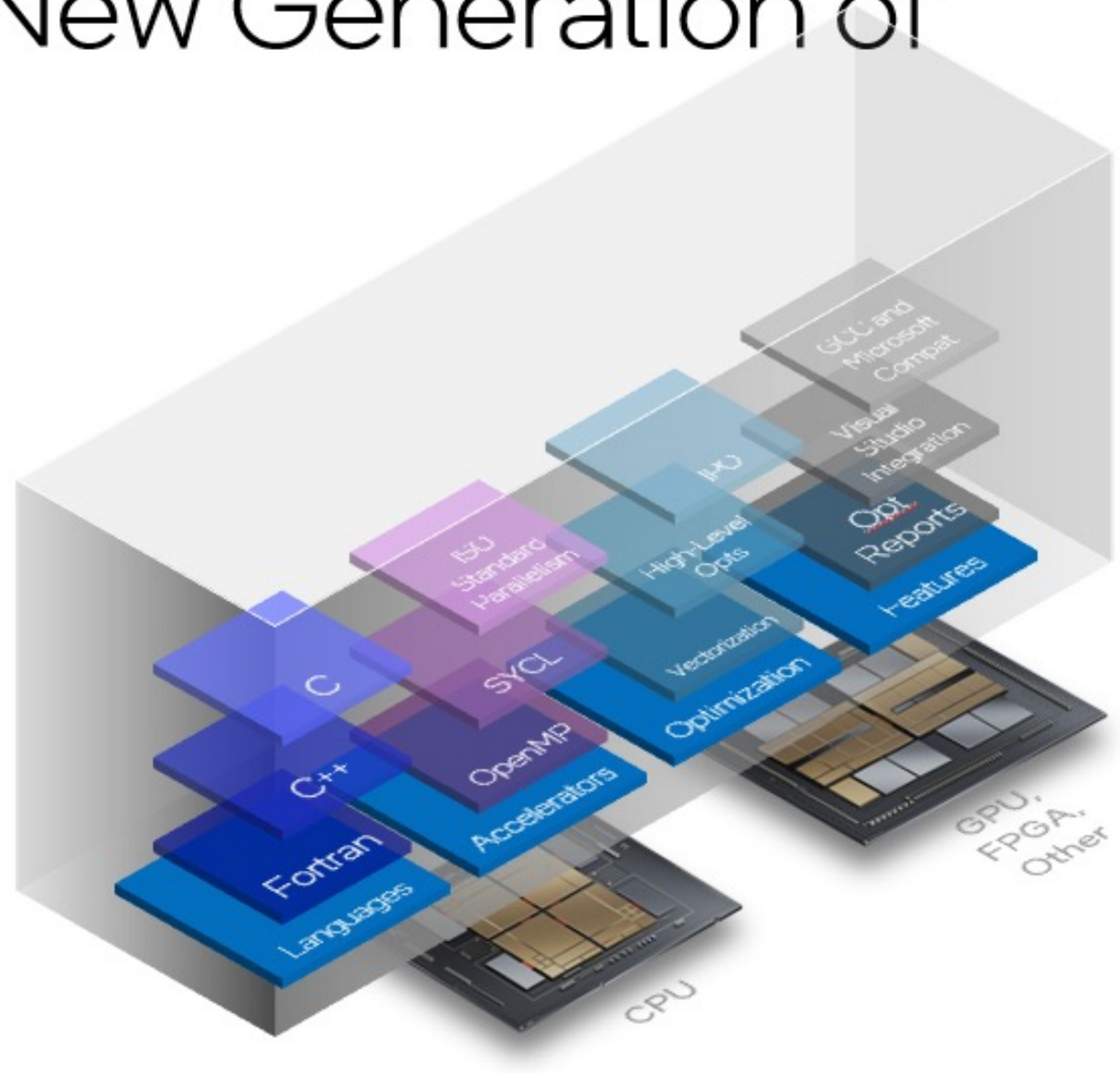
Intel does not control or audit third-party data. You should consult other sources to evaluate accuracy.

LLVM Powering the New Generation of Compilers

intel

+

=



What's new in 2023 Intel® Compiler Releases?

Enhanced Just-In-Time (JIT) and Ahead –Of-Time (AOT) Compilations for Intel® Xeon™ CPUs and Xe GPUs hardware enabling

OpenMP 5.1/5.2/TR12 features

- *reproducible* and *unconstrained* modifiers for *order* clause, *doacross* clause for ordered, *omp_cur_iterations*, *firstprivate* for *scope*, **allocate** clause for *scope*, *task* modifier for *reduction* clause, *step* modifier for *linear* clause, *nothing* directive, *error* directive, *begin declare variant*/*end declare variant* directive, *thread_limit* for target, *affinity* clause for **task**, *groupprivate,* declare target *local, …* etc.
- *ompx_assert* clause for SIMD, *need_device_ptr* clause for *dispatch* to support variadic functions.

C++23, Fortran 2008, Fortran 2018, Fortran 2023 OpenMP Offloading

Improved Unified Shared Memory (USM) Support

Enhanced OpenMP and SYCL/DPC++ Composability

Multi-GPU and Multi-Tile Support Improvements

In-order Kernel Execution with Asynchronous Offloading

Optimization Report Improvements

Performance Optimizations (auto-vectorization for GPUs, do-concurrent auto-offloading, NG-range for explicit SIMD execution, loop optimizations, … etc.)

Get Started with oneAPI Today!

Resources

Intel® DevCloud

Start in the Cloud - No Download, No Installation, No Setup – Sign up here - software.intel.com/devcloud/oneAPI

oneAPI Toolkits

Develop On-Prem – Download & Develop - Get them here - software.intel.com/oneAPI

Community Collaboration

Community – Working with community closely for OpenMP 5.1/5.2/6.0-TR12 support.

OpenMP Specification

Stay with Standard Body - Cross-industry, open, standards-based unified programming model across architectures – Learn more here – openmp.org

Learn about Intel® oneAPI Toolkits

Build Heterogeneous Applications

Build XPU Applications
Intel® Xeon, FPGA, Integrated Graphics GPUs & Xᵉ GPU, DG1**

Evaluate Workloads

Prototype Your Project
Evaluate Workloads

Learn Data Parallel C++

Interoperable with MPI, OpenMP, Fortran, C/++

Gaining Momentum!
oneAPI 2022.0 will be available soon!

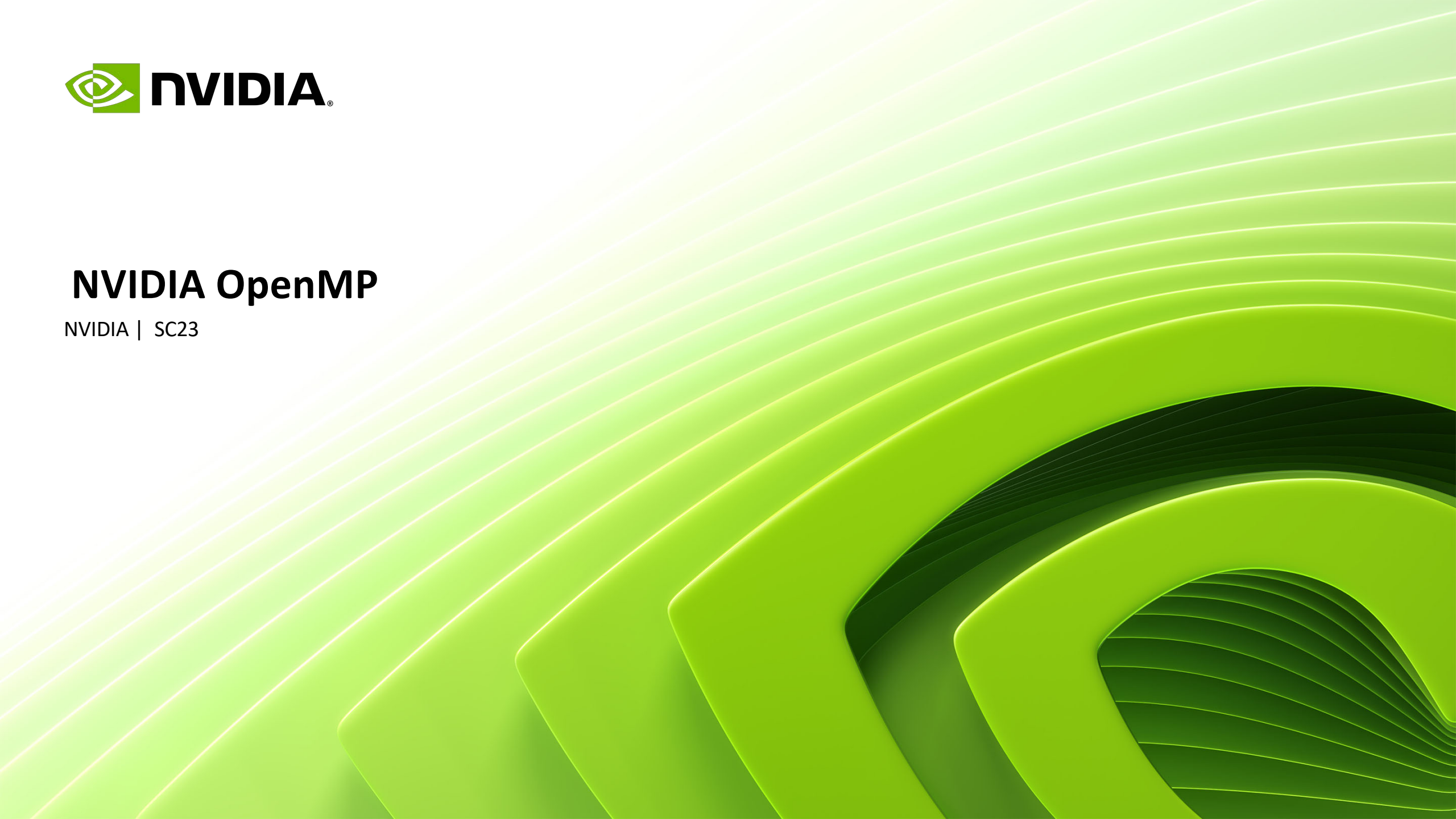
SC23 Denver, CO | i am hpc.

intel

it starts with

Thank you!


it
starts
with
intel



NVIDIA OpenMP

NVIDIA | SC23

Programming the NVIDIA Platform

Unmatched Developer Flexibility

	Accelerated Standard Languages	Incremental Optimization	Platform Specialization
Languages & Programming Models	C++, python™, Fortran	OpenACC More Science, Less Programming; OpenMP	NVIDIA CUDA; C++ | Fortran | Python

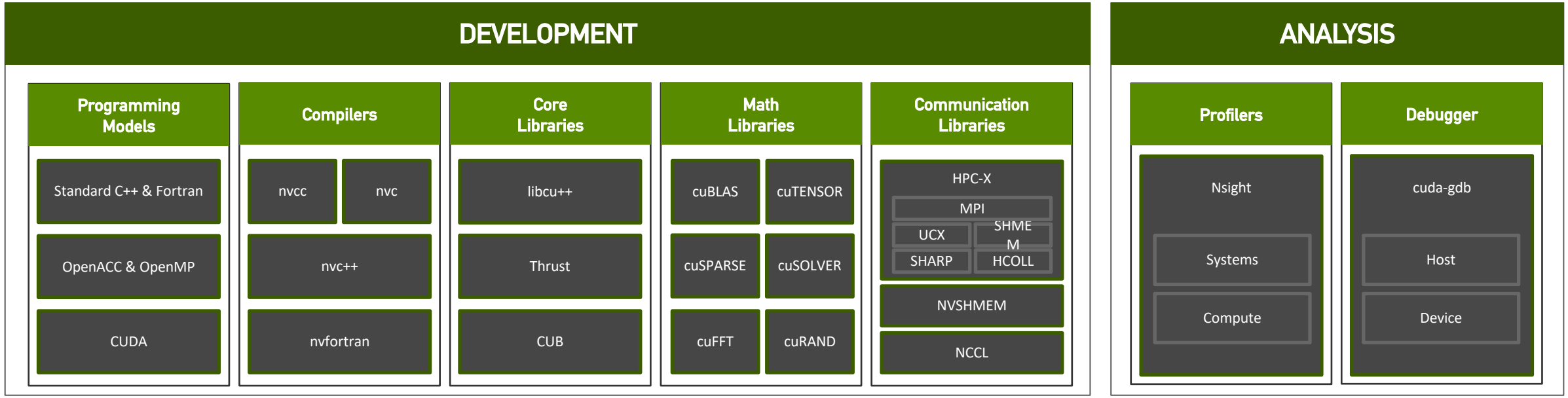
Libraries & Frameworks	Acceleration Libraries					
	Core	Math	Communication	Data Analytics	AI	DSLs

Hardware	GPU	CPU	Interconnect

Wherever You Run				
PCs	Workstations	On Prem	Cloud	At the Edge

NVIDIA HPC SDK

Available at developer.nvidia.com/hpc-sdk, on NGC, via Spack, and in the Cloud



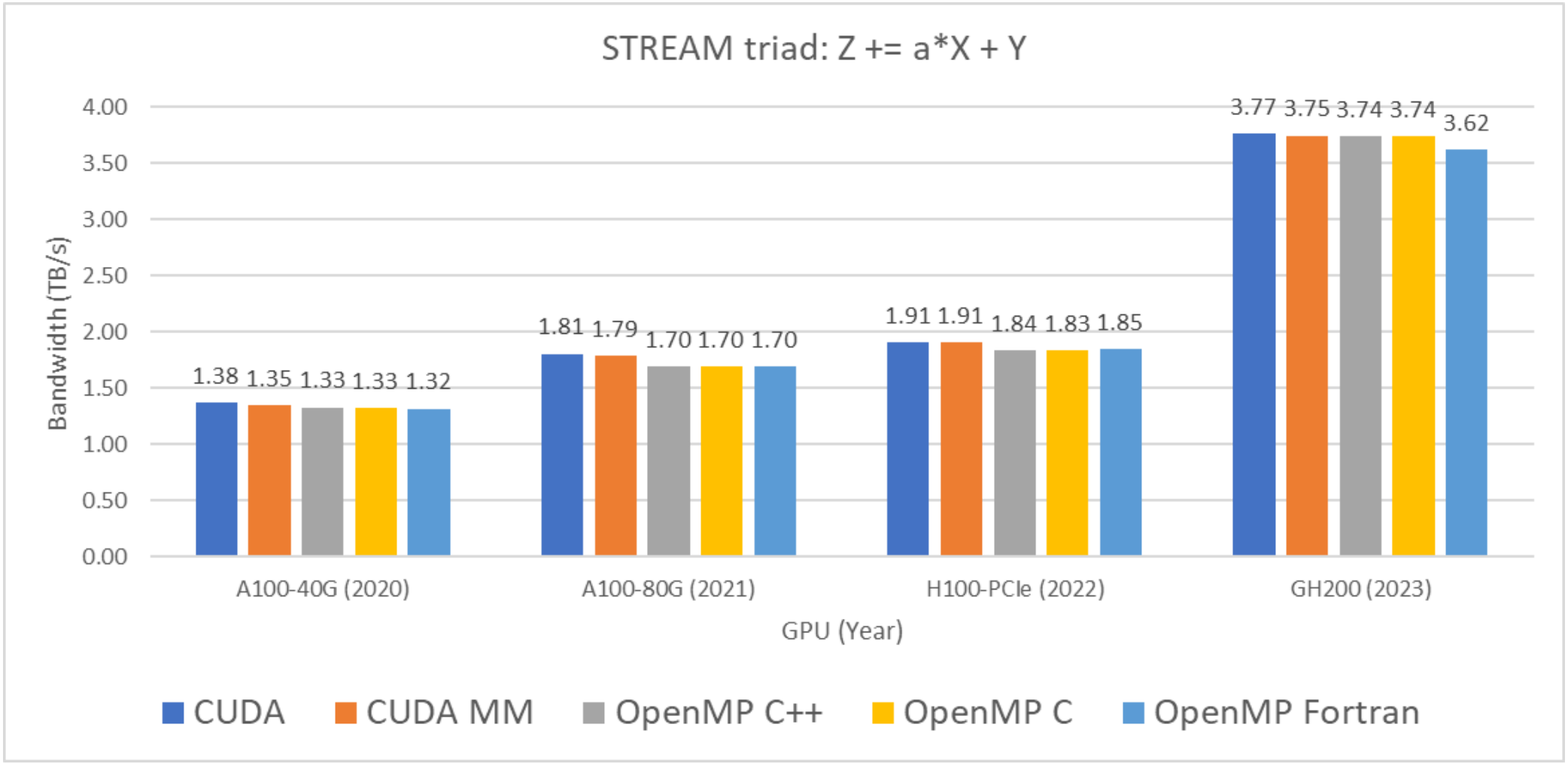
Develop for the NVIDIA Platform: GPU, CPU and Interconnect

Libraries | Accelerated C++ and Fortran | Directives | CUDA

x86_64 | Arm | OpenPOWER

7-8 Releases Per Year | Freely Available

Continuous OpenMP performance improvements



Providing performance and choice for developers

All SPECaccel2023 Results Published by SPEC

These results have been submitted to SPEC; see the disclaimer before studying any results.

Last update: *2023-11-08t13:12*

SPECaccel2023 (17):

Test Sponsor	System Name	Accelerator	Base Threads	Processor			Base Model	Peak Model	Results		Energy	
				Enabled Cores	Enabled Chips	Threads/ Core			Base	Peak	Base	Peak
NVIDIA Corporation	SuperServer SYS-1029GQ-TRT HTML | CSV | Text | PDF | PS | Config	Tesla V100-PCIE-16GB	1	40	2	2	ACC	ACC	1.23	1.23	--	--
NVIDIA Corporation	SuperServer SYS-1029GQ-TRT HTML | CSV | Text | PDF | PS | Config	Tesla V100-PCIE-16GB	1	40	2	2	LOP	LOP	1.21	1.21	--	--
NVIDIA Corporation	SuperServer SYS-1029GQ-TRT HTML | CSV | Text | PDF | PS | Config	Tesla V100-PCIE-16GB	1	40	2	2	TGT	TGT	0.999	0.999	--	--
NVIDIA Corporation	DGX-A100 HTML | CSV | Text | PDF | PS | Config	Tesla A100-SXM-80GB	1	128	2	2	ACC	ACC	2.76	2.76	--	--
NVIDIA Corporation	DGX-A100 HTML | CSV | Text | PDF | PS | Config	Tesla A100-SXM-80GB	1	128	2	2	LOP	LOP	2.63	2.63	--	--
NVIDIA Corporation	DGX-H100 HTML | CSV | Text | PDF | PS | Config	Tesla H100 SXM 80GB	1	112	2	2	ACC	ACC	4.47	4.47	--	--
NVIDIA Corporation	DGX-H100 HTML | CSV | Text | PDF | PS | Config	Tesla H100 SXM 80GB	1	112	2	2	LOP	LOP	4.11	4.11	--	--
NVIDIA Corporation	DGX-H100 HTML | CSV | Text | PDF | PS | Config	Tesla H100 SXM 80GB	1	112	2	2	TGT	TGT	3.38	3.38	--	--
NVIDIA Corporation	MGX-GH200 HTML | CSV | Text | PDF | PS | Config	Tesla H100 96GB	1	72	1	1	ACC	ACC	6.32	6.32	--	--
NVIDIA Corporation	MGX-GH200 HTML | CSV | Text | PDF | PS | Config	Tesla H100 96GB	1	72	1	1	LOP	LOP	5.74	5.74	--	--
NVIDIA Corporation	MGX-GH200 HTML | CSV | Text | PDF | PS | Config	Tesla H100 96GB	1	72	1	1	TGT	TGT	4.49	4.49	--	--
NVIDIA Corporation	120GQ-TNRT HTML | CSV | Text | PDF | PS | Config	Tesla H100 PCIe 80GB	1	64	2	2	ACC	ACC	3.21	3.21	--	--
NVIDIA Corporation	120GQ-TNRT HTML | CSV | Text | PDF | PS | Config	Tesla H100 PCIe 80GB	1	64	2	2	LOP	LOP	2.98	2.98	--	--
Test Sponsor	**System Name**	**Accelerator**	**Base Threads**	**Processor**			**Base Model**	**Peak Model**	**Results**		**Energy**	
				Enabled Cores	**Enabled Chips**	**Threads/ Core**			**Base**	**Peak**	**Base**	**Peak**
NVIDIA Corporation	120GQ-TNRT HTML | CSV | Text | PDF | PS | Config	Tesla H100 PCIe 80GB	1	64	2	2	TGT	TGT	2.52	2.52	--	--
NVIDIA Corporation	120GQ-TNRT HTML | CSV | Text | PDF | PS | Config	Tesla A100 PCIe 80GB	1	64	2	2	ACC	ACC	2.69	2.69	--	--
NVIDIA Corporation	120GQ-TNRT HTML | CSV | Text | PDF | PS | Config	Tesla A100 PCIe 80GB	1	64	2	2	LOP	LOP	2.57	2.57	--	--
NVIDIA Corporation	120GQ-TNRT HTML | CSV | Text | PDF | PS | Config	Tesla A100 PCIe 80GB	1	64	2	2	TGT	TGT	2.08	2.08	--	--

Last update: *2023-11-08t13:12*

- https://www.spec.org/accel2023/results/accel2023.html

HPC SDK 23.11

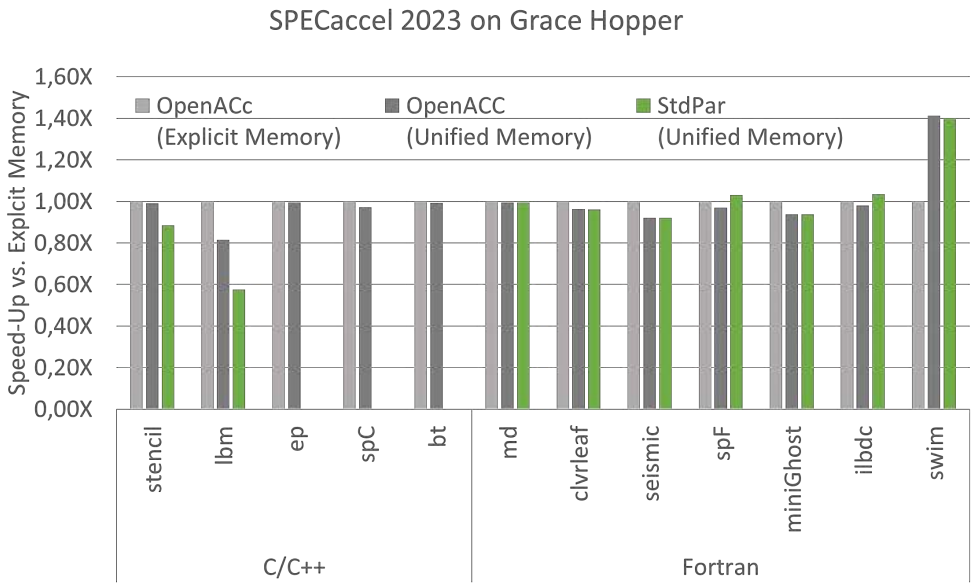
Grace Hopper, unified memory, and more

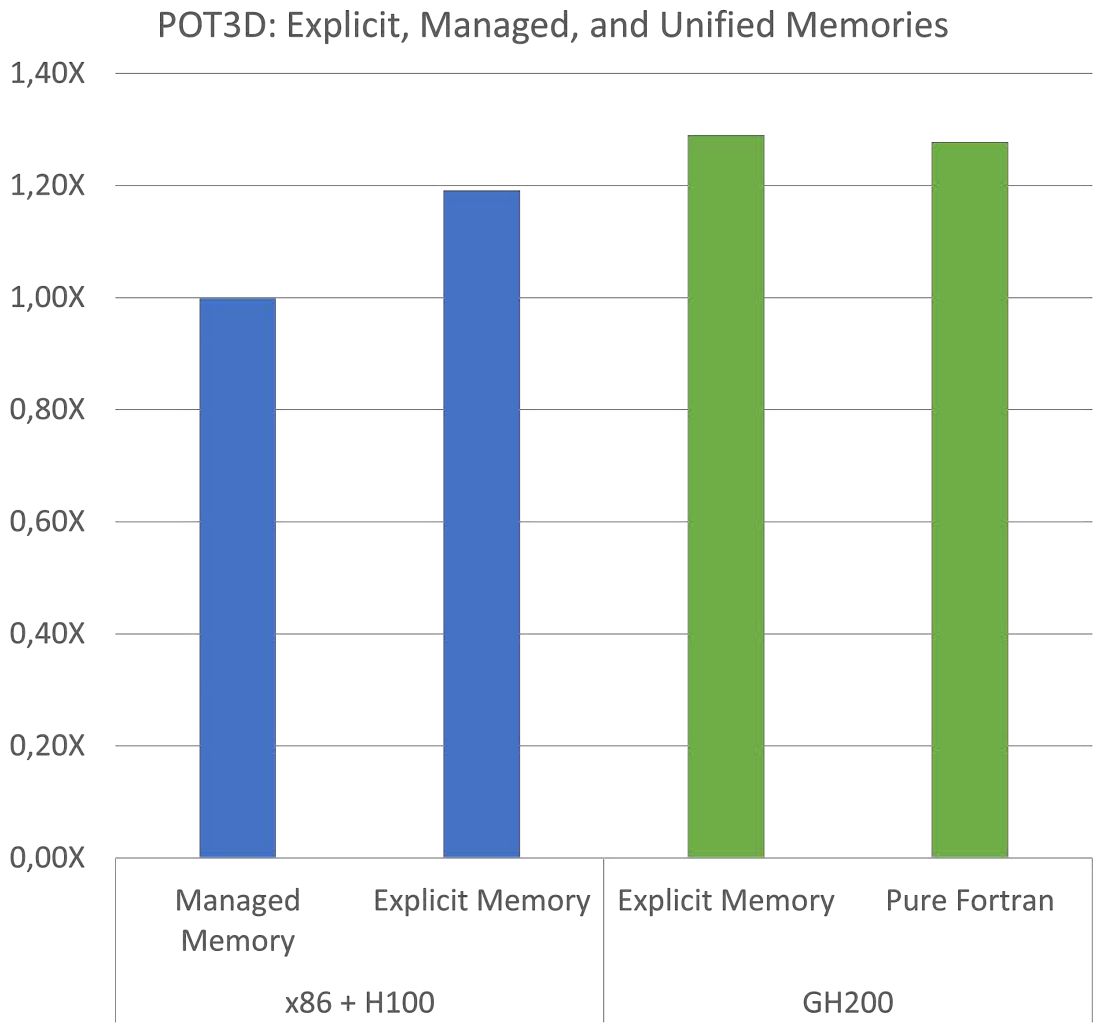
- **HPC SDK 23.11:**
  - NVTX improvements for stdpar codes
    - Now you can see your stdpar in NSight: improved tools support, developer experience, performance optimizations
  - C-Fortran Interface
    - Better multi-paradigm interoperability for mixed C, C++, and Fortran codes
    - F2008 MPI bindings for nvfortran
  - C++20 Coroutines for CPU
    - Future GPU support will enable alternative async models for stdpar
  - Support for Grace Hopper in all bundled components
    - Compilers, Math Libraries, Networking, Tools.
  - HPC-X is the default MPI implementation optimized for NV platform
  - Grace(/Arm) performance (-tp=neoverse-v2)
    - Re-engineered vectorizer, intrinsics, system math library functions
  - Unified memory support for stdpar, OpenACC, and CUDA C++/Fortran

- **Unified memory:**
  - C++ stdpar improvements
  - Fortran stdpar improvements
  - OpenACC improvements
  - CUDA Fortran
  - Unified Functions
  - Launch Blog Post

HPC Compilers Support Unified Memory

- The NVIDIA HPC Compilers now support unified memory systems.
  - Grace Hopper
  - Linux HMM on x86
- No need for explicit data management
- Standard Parallelism (StdPar) works for *all* memories.
- Composable with CUDA for memory optimization





Hewlett Packard
Enterprise

HPE CRAY OPENMP BOF UPDATE

Zachary Tschirhart, Sr Manager CCE

November 15, 2023

OPENMP STATUS AS OF CCE 16

CCE 10.0 (May 2020)

- OMP_TARGET_OFFLOAD
- reverse offload
- implicit declare target
- omp_get_device_num
- OMP_DISPLAY_AFFINITY
- OMP_AFFINITY_FORMAT
- set/get affinity display
- display/capture affinity
- requires
- unified_address
- unified_shared_memory
- atomic_default_mem_order
- dynamic_allocators
- reverse_offload
- combined master constructs
- acq/rel memory ordering (Fortran)
- deprecate nested-var
- taskwait depend
- simd nontemporal (Fortran)
- lvalue map/motion list items
- allow != in canonical loop
- close modifier (C/C++)
- extend defaultmap (C/C++)

CCE 11.0 (Nov 2020)

- noncontig update
- map Fortran DVs
- host teams
- use_device_addr
- nested declare target
- allocator routines
- OMP_ALLOCATOR
- allocate directive
- allocate clause
- order(concurrent)
- atomic hints
- default nonmonotonic
- imperfect loop collapse
- pause resources
- atomics in simd
- simd in simd
- detachable tasks
- omp_control_tool
- OMPT
- OMPD
- declare variant (Fortran)
- loop construct
- metadirectives (Fortran)
- pointer attach
- array shaping
- acq/rel memory ordering (C/C++)
- device_type (C/C++)
- non-rectangular loop collapse (C/C++)

CCE 12.0 (Jun 2021)

- device_type (Fortran)
- affinity clause
- conditional lastprivate (C/C++)
- simd if (C/C++)
- iterator in depend (C/C++)
- depobj for depend (C/C++)
- task reduction (C/C++)
- task modifier (C/C++)
- simd nontemporal (C/C++)
- scan (C/C++)
- lvalue list items for depend
- mutexinoutset (C/C++)
- taskloop cancellation (C/C++)

CCE 13.0 (Nov 2021)

- declare variant (C/C++)
- metadirectives (C/C++)
- mapper (C/C++)
- extend defaultmap (Fortran)
- close modifier (Fortran)
- mutexinoutset (Fortran)
- inoutset dependence type
- primary policy for the proc_bind clause
- present behavior for defaultmap (C/C++ only)
- masked construct without filter clause (Fortran only)
- metadirective dynamic user condition and target_device selectors (Fortran only)

CCE 14.0 (May 2022)

- task reduction (Fortran)
- task modifier (Fortran)
- target task reduction (Fortran)
- simd if (Fortran)

CCE 15.0 (Nov 2022)

- target task reduction (C/C++)
- uses_allocators
- error directive (Fortran only)
- compare clause on atomic construct (C/C++ only)
- assume and assumes directives (Fortran only)
- nothing directive (Fortran only)
- otherwise clause for metadirective (Fortran only)

CCE 16.0 (May 2023)

- non-rectangular loop collapse (Fortran)
- depobj for depend (Fortran)
- iterator in depend (Fortran)
- scan (Fortran)
- conditional lastprivate (Fortran)
- mapper (Fortran)
- OMPT external monitoring interface

Refer to CCE release notes or intro_openmp man page for current implementation status. OpenMP 5.1/5.2 features are green.

FUTURE FOCUS AND FEATURES

- Fully OpenMP 5.0 compliant and will continue to add even more 5.x features over the next year
  - Our users drive prioritization of OpenMP features we implement

- Our focus has been on improving OpenMP performance across supported platforms (e.g. MI250/300, Grace-Hopper, Sapphire Rapids, etc.)
  - Better support for user-defined mappers, improved unified memory support, general performance uplift, etc.

- Upcoming features expected in CCE 18 and beyond:
  - **interop** construct and supporting API routines (C/C++/Fortran)
  - complete **compare** and **fail** support for **atomic** construct (C/C++/Fortran)
  - **present** modifier for **map** and **to**/**from** clauses, and support for **present** in **defaultmap** clause (C/C++/Fortran)
  - **nowait** clause for **taskwait** (C/C++/Fortran)
  - **omp_target_memcpy_async** and **omp_target_memcpy_rect_async** (C/C++/Fortran)
  - support **private** and **firstprivate** in **default** clause (C/C++)
  - (Targeting CCE 19) **scan** directive (C/C++/Fortran)
  - (Targeting CCE 19) **tile** and **unroll** loop transformations (C/C++/Fortran)

THANK YOU

Zachary.Tschirhart@hpe.com

Documentation:
https://cpe.ext.hpe.com/docs

LLVM/OpenMP

SC 23 Update

OpenMP for std::par in LLVM/libcxx

```
template<typename T1, typename T2, typename T3>
void axpy(const T1 a,std::vector<T2>& x, std::vector<T3>& y)
{
        std::transform(std::execution::par_unseq,x.begin(),x.end(), y.begin(), y.begin(),
                [=](T2 xi, T3 yi){ return a*xi + yi; });
}
```

Available in 2-4 weeks!

OpenMP Support for LibC functions

```
int main() {
  FILE *f = fopen("log.txt", "w");
#pragma omp target teams num_teams(8)
#pragma omp parallel num_threads(8)
  {
    const char *str = "A simple string\n";
    char *buf = malloc(strlen(str) + 1);
    strcpy(buf, str);
    for (int i = 0; i < strlen(str); ++i)
      buf[i] = toupper(buf[i]);
    fputs(buf, f);
    free(buf);
  }
  fclose(f);
}
```

https://libc.llvm.org/gpu

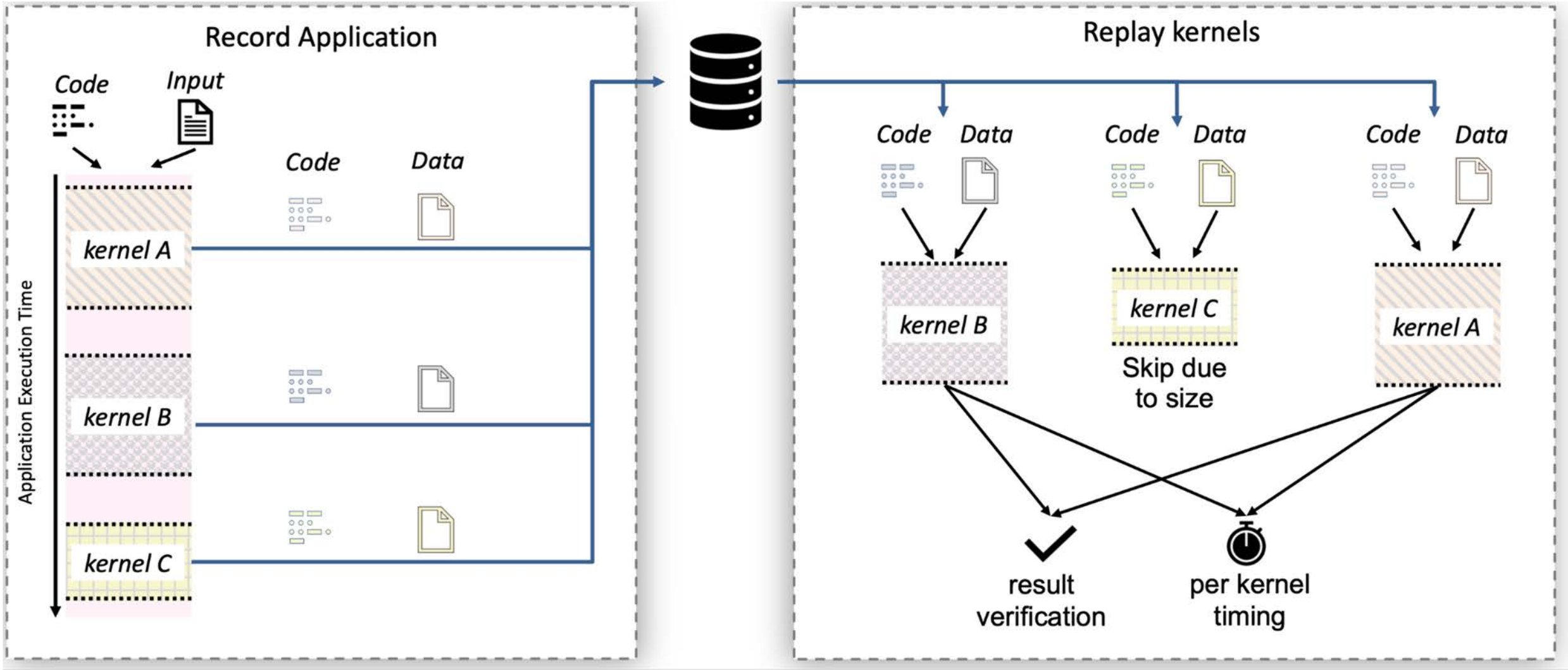
OpenMP (CUDA-like) Kernel Language Extensions

```
__device__ int use(int &a, int &b) { … }

__global__ void kernel(int *a, int *b, int n) {
  __shared__ int shared[128];

  int tid = threadIdx.x;
  if (tid == 0) {
    /* initialize shared */
  }
  __syncthreads();
  int idx = blockIdx.x *
            blockDim.x + tid;
  if (idx < n)
    b[idx] = use(a[idx], shared[tid]);
}
```

```
int use(int &a, int &b) { … }

void kernel(int *a, int *b, int n) {
  int shared[128];
#pragma omp groupprivate(team: shared)
  int tid = ompx_thread_idx_x();
  if (tid == 0) {
    /* initialize shared */
  }
  ompx_sync_block_acq_rel();
  int idx = ompx_block_idx_x() *
                        ompx_block_dim_x() + t
  if (idx < n)
    b[idx] = use(a[idx], shared[tid]);
}
```

```
#pragma omp target teams ompx_bare num_teams(NB) thread_limit(NT) \
        ompx_dyn_cgroup_mem(DynMem) depend(...) nowait
kernel(h_a, h_b, n);
```

OpenMP Kernel Record and Replay



SAMSUNG

Processing In Memory (PIM)

Accelerating AI with Memory

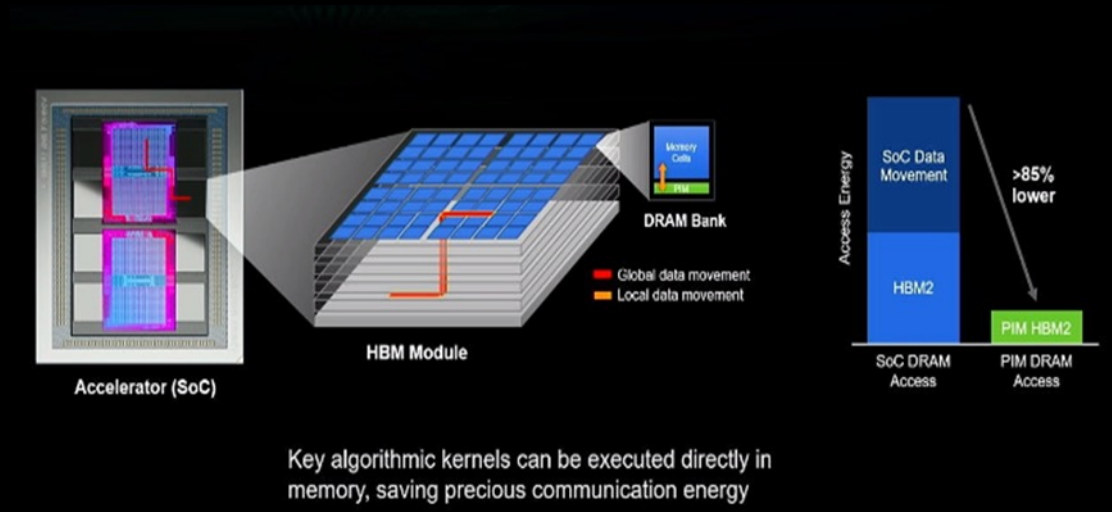
OpenMP BoF

Nov 15 | 5:00 PM

Energy Advantage of PIM on Generative AI

- Since OpenAI focuses on developing new AI technologies and pushing the boundaries of what can be done with AI, it is likely that they will explore the use of PIM technology in the future.
- In ISSCC 2023, AMD mentioned
  - Key algorithmic kernels can be executed directly in memory, saving precious communication energy
  - PIM can reduce energy by 85% compared with conventional HBMs

Processing-in-Memory



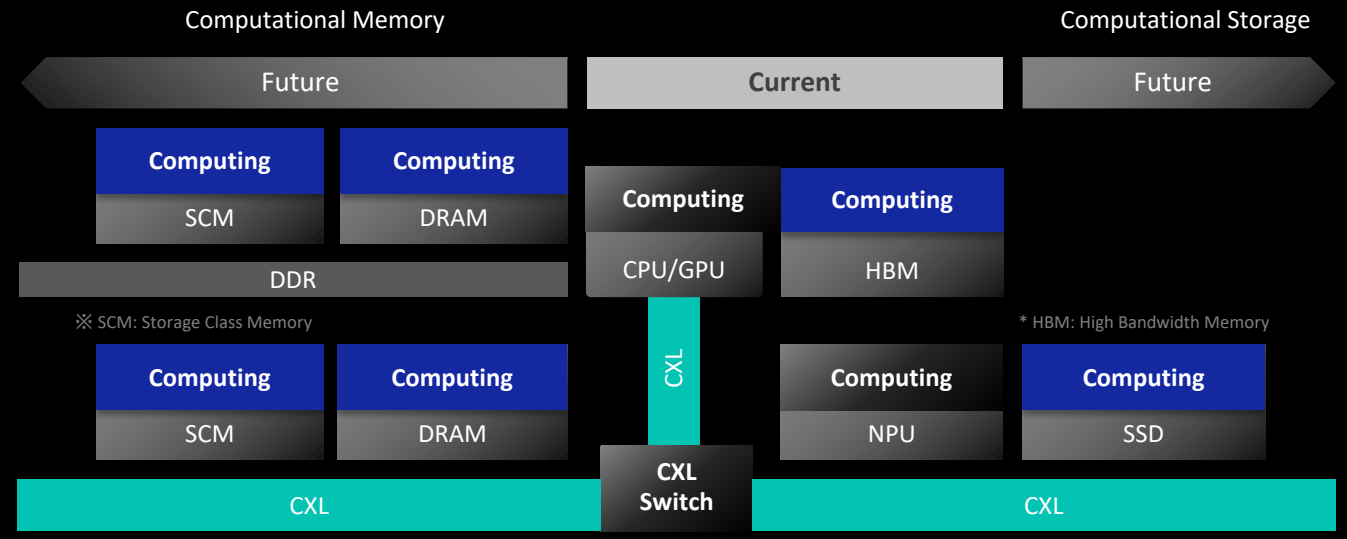
[source] AMD ISSCC

Future HBM-PIM concept

[source] Samsung, MemCon

Future: Computing in Everywhere

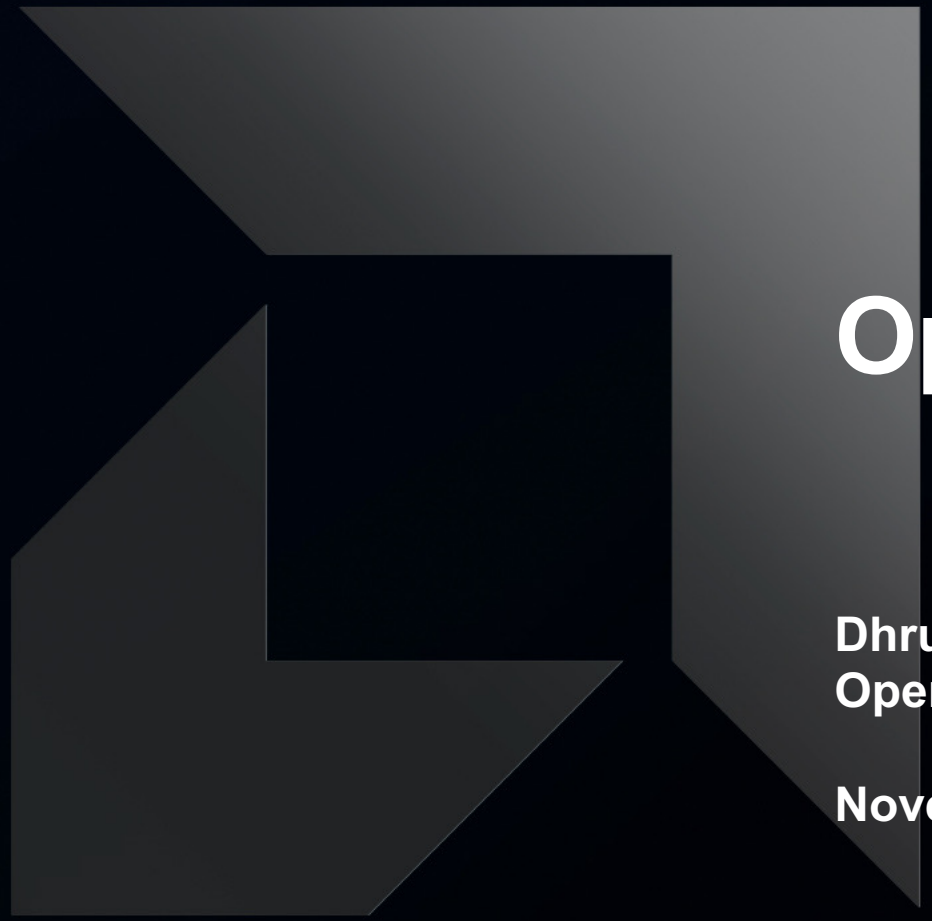
- AI and Simulation have been major single large jobs and "Memory Wall" remains a critical issue
- Computation-enabled memory and storage are good approaches for breaking the "Memory Wall", but they increase the system complexity
- Need to choose the optimal combination of data movement and job migration
  → Job and Data Co-location problem
- We believe that new programming model and system SW such as OS will solve the problem



Programming Models for PIM

- OpenMP
  - Directive-based, C/C++/Fortran
  - De facto standard for multi-core CPU parallel programming
  - Open-source implementations for CPUs (GCC, LLVM)
- OpenACC
  - Directive-based, C/C++/Fortran
  - Designed for many-core accelerators
- SYCL
  - Template-based, C++
  - Programming model for heterogeneous architecture
- Candidate Programming Model for PIM
  - OpenMP and OpenACC: Directive programming model for legacy code
  - SYCL: Future application code development

Thank You



OpenMP® at AMD

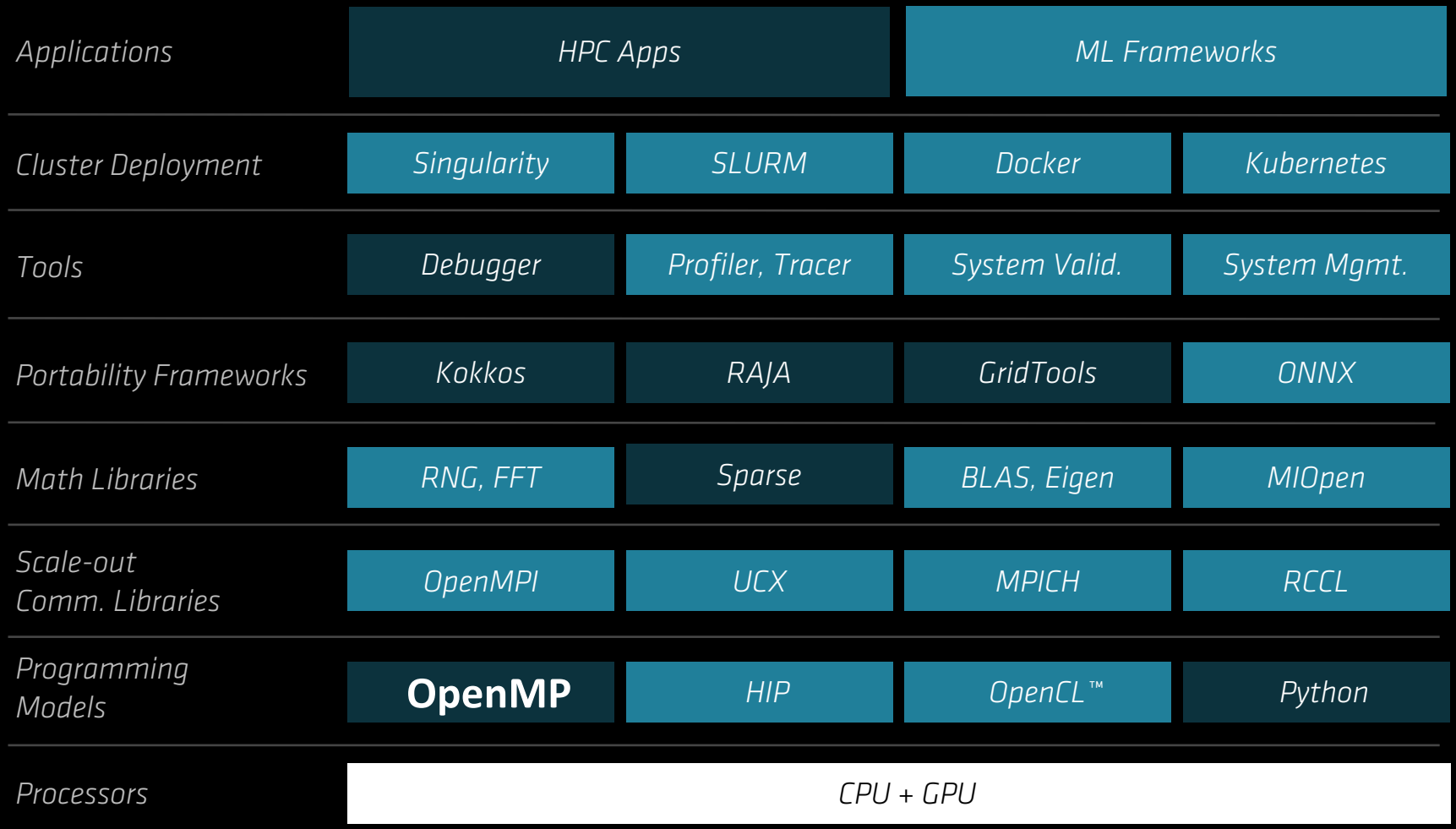
Dhruva Chakrabarti, AMD
OpenMP BoF, Supercomputing

November 15, 2023

AMD
together we advance_

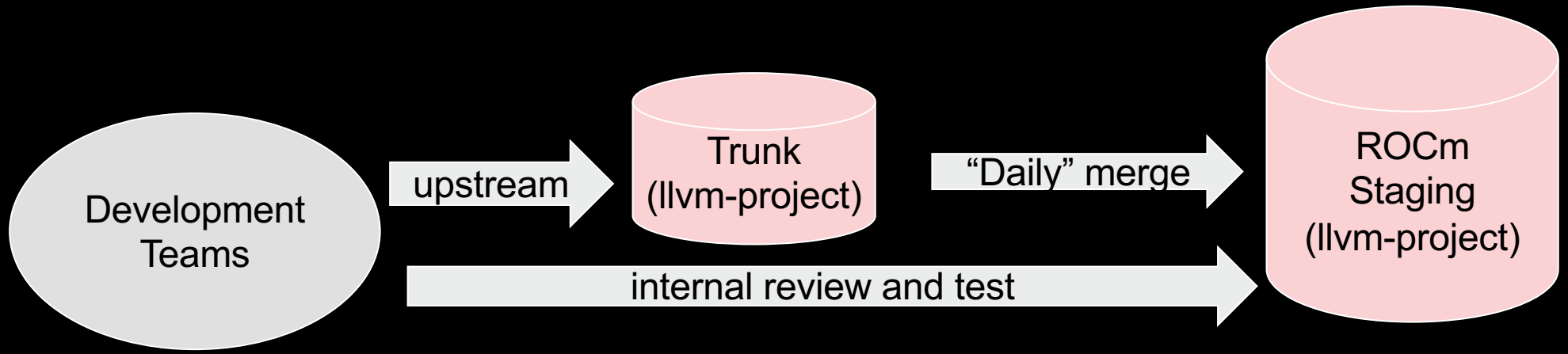
OPENMP® in the ROCm™ Stack *

https://rocm.docs.amd.com/en/latest/



* 2022 Stack, not updated for TF2 and AI stacks

AMD ROCm™ OpenMP® Compiler Ecosystem



- OpenMP offload support is in clang/flang distributed as part of ROCm.
- Open-sourced at https://github.com/RadeonOpenCompute
- Developer versions available at https://github.com/ROCm-Developer-Tools/aomp
- Upstream trunk + additional features, enhancements, testing, and fixes.
- Specialized kernels for OpenMP target regions.
- Additional support for OMPT and ASAN.
- ROCprofiler for OpenMP offload programs.
- Unified shared memory support.
- Host-exec support for printf, fprintf. Libm support.
- Some remaining multi-arch support for target-id and new code-object.

Disclaimer

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions, and typographical errors. The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. Any computer system has risks of security vulnerabilities that cannot be completely prevented or mitigated. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

THIS INFORMATION IS PROVIDED ‘AS IS.” AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS, OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION. AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY RELIANCE, DIRECT, INDIRECT, SPECIAL, OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

42


AMD

Panel

- Xinmin Tian, Intel
- Graham Lopez, NVIDIA
- Zach Tschirhart, HPE
- Johannes Doerfert, LLVM
- Seungwon Lee, Samsung
- Dhruva Chakrabarti, AMD

OpenMP 6.0 Outlook: TR12 and Beyond

Bronis R. de Supinski

Chair

OpenMP Language Committee

November 15, 2023

LLNL-PRES-857029

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

Lawrence Livermore National Laboratory

OpenMP 6.0 will be released in November 2024

- TR12 demonstrates appropriate progress for second TR of a major version
- Major new feature targets have been clearly identified and are on track for 2024
  - → Free-agent threads significantly change execution model, implementations
  - → User-defined induction and `induction` clause expand parallelism support
  - → Many significant device support improvements (e.g., `memscope(all)`) added or planned
  - → Several other additions and improvements planned, including:
    - → Rationalization of definition of combined constructs
    - → Task dependences between concurrently generated tasks
  - → Significant improvements to usability and correctness of specification
  - → TR12 includes 153 completed issues, considering over 300 others (2 more already passed)
  - → TR13 (final comment draft) will be released in summer 2024

Major new features will characterize OpenMP 6.0

- Free-agent threads (Stephen will detail)
  - → Support for top-level task parallelism (i.e., explicit `parallel` directive not needed)
  - → "Any" thread can execute explicit tasks for which `threadset` clause evaluates is `omp_pool`
  - → Adds associated runtime routines, environment variables and ICVs
- Major improvements for use of a single device (Expect Tom to cover most if not all)
  - → Explicit progress guarantee adopted in TR11
  - → Default device and visible devices to simplify control of device use and availability
  - → Mechanisms to simplify use of device memory (by providing greater certainty or clarity)
    - → New `groupprivate` directive in TR11 is an initial mechanism in this direction
    - → Added `selfmap` modifier to ensure no copy is created when possible
    - → Unified host and device allocators and added significant cross-device improvements
  - → TR12 added `coexecute` directive (i.e., descriptive array language offload support)

OpenMP 6.0 will include other significant new features

- A more complete set of loop transforming directives
  - → TR12 includes `fuse`, `reverse` and `interchange` directives
  - → Considering other transformations that include `fission` and `nestify`
  - → Can now transform generated loops using the `apply` clause
- Clauses and directives to support generalized induction
  - → Capture computation that follows a well-defined sequence across loop iterations
  - → Generalizes behavior of `linear` clause and of loop iteration variables
  - → Related to reductions, including addition of `declare induction` directive

Extending `parallel` directive to support complete user control of number of threads

- The `parallel` directive will accept a new modifier and two "new" clauses

```
#pragma omp parallel [num_threads(prescriptiveness: nthreads)] \
                     [severity(fatal|warning)][message(msg-string)]
  structured-block
```

- Using `strict` *prescriptiveness* requires `nthreads` to be provided
- Clauses, previously available on `error` directive, effective with `strict` if cannot provide `nthreads`
  - → Display `msg-string` as part of implementation-defined message
  - → If `severity` is `fatal` execution is terminated
  - → If `severity` is `warning` message is displayed but execution continues
- Also now allowed to provide a list for `nthreads` to support nested parallelism

Some other improvements expected in OpenMP 6.0

- May further extend descriptive and prescriptive control
- Removal of features that were deprecated in 5.0, 5.1 or 5.2
- Dependences and affinity for the `taskloop` construct
- Wider use of C++ attribute syntax: Make C++ support "more C++-like"
  - → Likely to include improvements for `threadprivate` and `declare target`
  - → Also clarified conditions for implicitly declared reduction operators for class types
  - → Will also be supported in C
- Adding latest versions of base languages (C23, C++23, Fortran 2023)
- Continuing to extend support for tool interfaces

Some other improvements expected in OpenMP 6.0

- Expect to define combined constructs based on properties (a priority)
- Strengthening task-oriented execution changes begun in OpenMP 3.0
- Extending specification improvements begun in OpenMP 5.2
  - → Includes making all clauses accept arguments
  - → All clauses will take a directive name modifier for better control of combined constructs
- Immediate focus is several high priority, nearly completed issues
  - → More single device and tasking improvements
  - → Better control of number of threads
  - → Simpler, broader user control of defaults

Things likely to be deferred beyond 6.0

- True support for using multiple devices
  - → Device-to-device scoping support for atomic and other memory operations
  - → Support for bulk launch
  - → Support to update data on multiple devices (broadcast/multicast, other collectives)
  - → Support for work distribution across devices
  - → Considering relaxing restrictions on nested `target` regions
- Efficient use of multiple compilation units (i.e., support for efficient IPO)
- Characterizing loop-based work distribution constructs as transformations
- Support for pipelining, data-flow, other parallelization models
- Support for event-based parallelism

Lawrence Livermore
National Laboratory

Sandia National Laboratories

Exceptional service in the national interest

OPENMP 6.0 TASKING: RISE OF THE FREE-AGENT THREADS

Stephen Olivier, Tasking Subcommittee Chair

OpenMP Language Committee

SC23 OpenMP BoF

November 2023

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc. for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

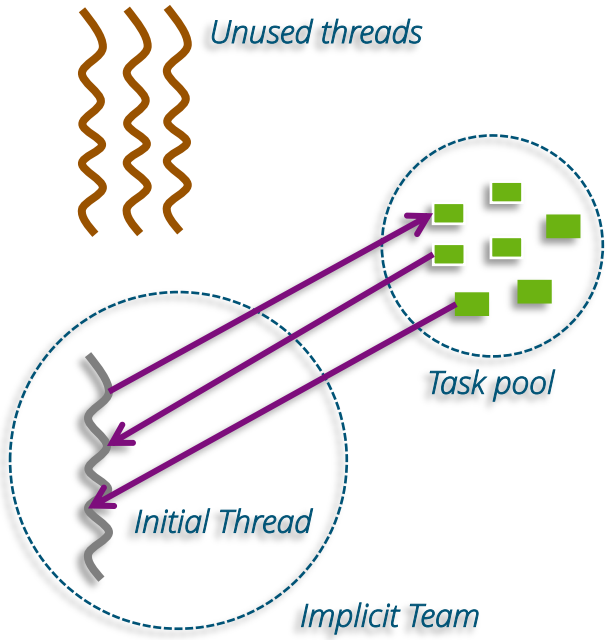
SAND2023-13437C

TASK PARALLELISM AS OF OPENMP 5.X: TEAMING REQUIRED

A key limitation is that only threads in the team may execute tasks generated in that team.

```
while (elem != NULL) {
   #pragma omp task
      compute(elem);
   elem = elem->next;
}
```

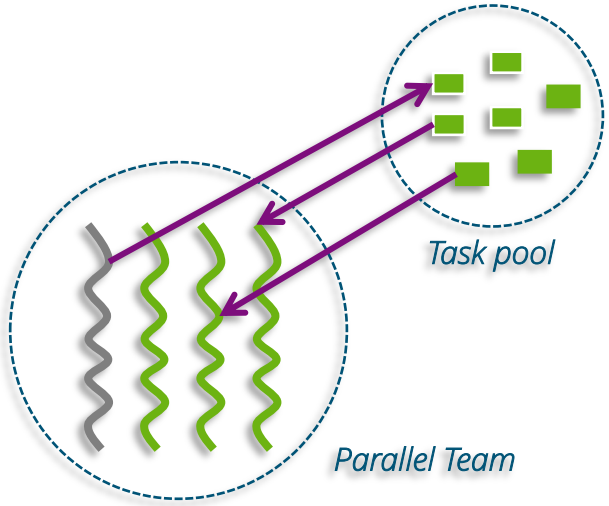
Potentially multiple tasks, but only one thread in the implicit parallel region to execute the tasks…



Implementations may have many more threads available, but current semantics prevent their use in this scenario.

```
#pragma omp parallel
#pragma omp single
while (elem != NULL) {
   #pragma omp task
      compute(elem);
   elem = elem->next;
}
```

Here the parallel construct creates a team of threads to execute tasks and the single construct avoids duplicate task creation.



An otherwise task-only program also currently needs to worry about a team of threads to execute the tasks.

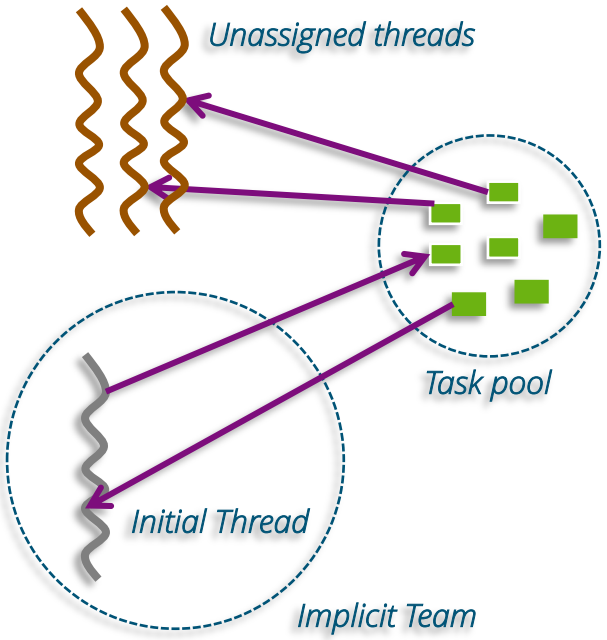
OPENMP 6.0: "FREE-AGENT THREADS" CAN ALSO EXECUTE TASKS

Tasks may be schedulable onto these threads in the contention group that are unassigned.

```
while (elem != NULL) {
   #pragma omp task threadset(omp_pool)
      compute(elem);
   elem = elem->next;
}
```

New threadset clause indicates which threads may execute the task:

- omp_team: Only threads in the team (default)
- omp_pool: Threads in the team AND unassigned threads in the contention group



Balance of structured parallelism and free-agent threads governed by ICVs that can be controlled through OMP_THREADS_RESERVE:

```
setenv OMP_THREADS_RESERVE "structured(4),free_agent(2)"
```

In the example above, four threads are reserved for structured parallelism (assignment to teams) and two threads to act as free-agents.

- Minimum for structured parallelism is one (the initial thread)
- Sum of reservations should not exceed *thread-limit-var* ICV

OPENMP 6.0 AND BEYOND: OTHER ACTIVE TASKING TOPICS

The following topics are under discussion but not yet adopted features/changes:

- “Taskgraph”: Capability to capture and replay task dependency graphs
- Support for affinity and dependences on taskloops
- Mechanism for dependences of child tasks to be treated as dependences of parent task
- Additional dependence types and recategorization of dependence types
- Allow dependences based on integer values rather than data locations

OpenMP C/C++/Fortran Subcommittee Update

- What's new in TR12 for C/C++?
  - Rebasing OpenMP 6.0 Preview (TR12) with C23 and C++23
    - C23 attribute
    - C++23 Attribute [[assume]]
    - C++23 Multidimensional subscript operator
    - C++23 Extending the lifetime of temporaries in range-based for loop initializer
    - C++23 Extend init-statement (of for loop) to allow alias-declaration
    - … …
  - Add new rules on virtual function offloading support
- Future Work
  - Pro-actively looking into proposed ISO C++26 features for OpenMP
  - C/C++ Optimization hints for OpenMP

OpenMP C/C++/Fortran Subcommittee Update

- What's New in TR 12 for Fortran
  - **coexecute** construct for Fortran array language
  - allow BLOCK constructs in atomic structured blocks; more forms for atomic *conditional-update-statement*
  - the **loop** construct can now be used on DO CONCURRENT loops
  - support more Fortran 2018 features
    - assumed-rank dummy argument and SELECT RANK construct
    - interoperable procedure enhancements
    - declared type of a polymorphic allocatable component in structure constructure
  - various clarifications
- Future work
  - Fortran pointer and allocatable mapping clarification
  - complete rebasing Fortran 2018
    - the remaining item is to support assumed-type dummy argument
  - initial support for Fortran 2023 features
  - look into enhancing the usability of the omp routines (e.g. omp_target_* routines require using type(c_ptr) variables etc)
  - further enhancement of using DO CONCURRENT with OMP constructs

New accelerator features: coexecute
Parallelize Fortran array syntax across teams

```
subroutine axpy_array_coexecute(a, x, y, n)
    use iso_fortran_env
    implicit none
    integer ::n
    real(kind = real32)::a
    real(kind = real32), dimension(n)::x
    real(kind = real32), dimension(n)::y
    !$omp target teams coexecute
    y = a * x + y
    !$omp end target teams coexecute
 end subroutine axpy_array
```

New Accelerator Features: OMP_AVAILABLE_DEVICES

- Select, order, reorder and hide devices with a flexible interface
- OMP_VISIBLE_DEVICES=
  - "(kind(gpu)&&vendor(nvidia))[:4],kind(gpu)&&vendor(amd)”
  - The first four NVIDIA GPUs followed by all AMD GPUs will be visible
- OMP_DEFAULT_DEVICE=
  - "kind(gpu)&&vendor(amd),(kind(gpu)&&vendor(nvidia))[1]”
  - Prefer any AMD GPU, if unavailable use the second NVIDIA GPU

New Accelerator Features

- groupprivate directive – Team-private memory for static-lifetime variables
- Safesync clause and omp_get_max_progress_width – Allow explicitly controlling thread independence in SIMT/SIMD contexts
- Atomics over 64bit are allowed in offload
- Better handling of unsized objects in Fortran
- omp_target_associate_ptr actually associates pointers!

Affinity Subcommittee Report

- Memory Management
- Thread-to-Device Affinity
- Data-to-Device Affinity
- Taskloop Dependencies and Affinity


Christian Terboven

Co-lead: Jannis Klinkenberg

Memory Management (since OpenMP 5.0)

- Did you know that you can … allocate in high-bandwidth memory?

```
#include <omp.h>
double *x = omp_alloc(N * sizeof(double), omp_high_bw_mem_alloc);
```

- Recent work:
  - → New allocator traits for finer placement control
    - → `partition`: partitioning of allocated memory over storage resources:

      `environment`, `nearest`, `blocked`, `interleaved`, `user` (allows writing and specifying custom partitioner)
    - → `part_size`: specifies the size of parts allocated over storage resources
  - → Allow upper bound and stride for `OMP_PLACES` together with abstract names
    - → Examples: `OMP_PLACES=cores(4)` or `OMP_PLACES=ll_caches(1:2)`
  - → Unify allocator and target memory runtime routines
    - → Capability to allocate device memory with OpenMP allocators: new routines returning target memory spaces
    - → Memory space containing storage resources accessible by all devices as requested

Data/Thread-to-Device Affinity / 1

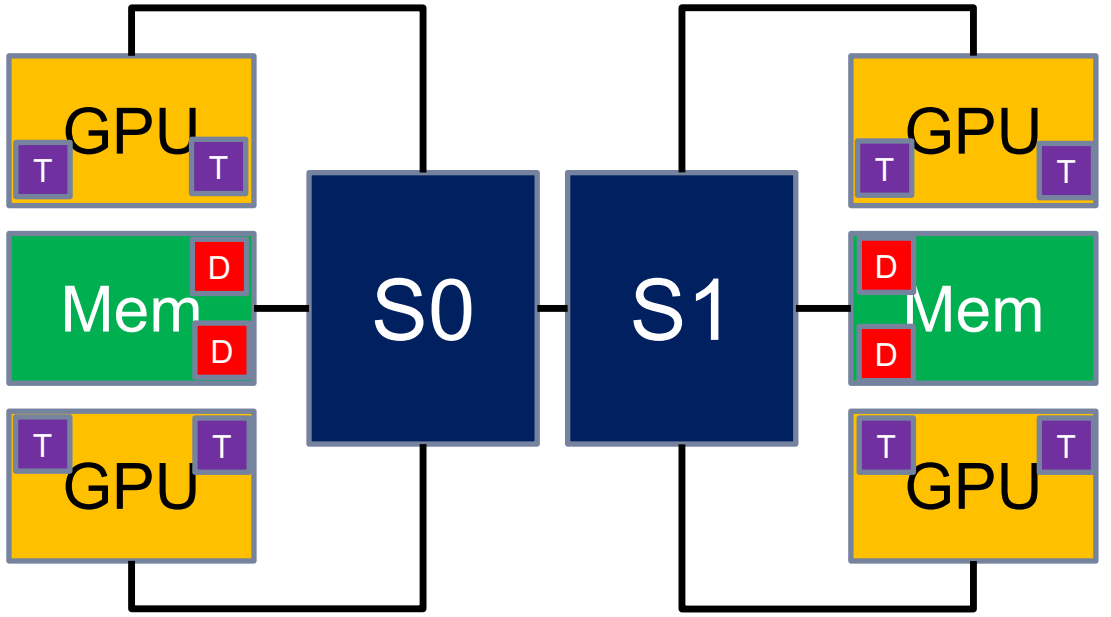
- **Idea:** Find devices that are close to the current thread

1. **Find devices that are close to the current thread**

```
int n=20;           // desired number of devices
int n_dev_found;    // actual number of devices
int dev_ids[n];
n_dev_found = omp_get_devices_in_order
                  (n, dev_ids, <trait_lowest_distance>);

#pragma omp target device(dev_ids[0]) // closest device
...
#pragma omp target device(dev_ids[n_dev_found-1])
```

2. **Use devices that are close to data used in target**

```
#pragma omp task affinity(data[start:len])
{
    #pragma omp target map(tofrom: data[start:len]) \
        device_affinity(data[start:len])
    {
        // content of the target task
    }
}
```

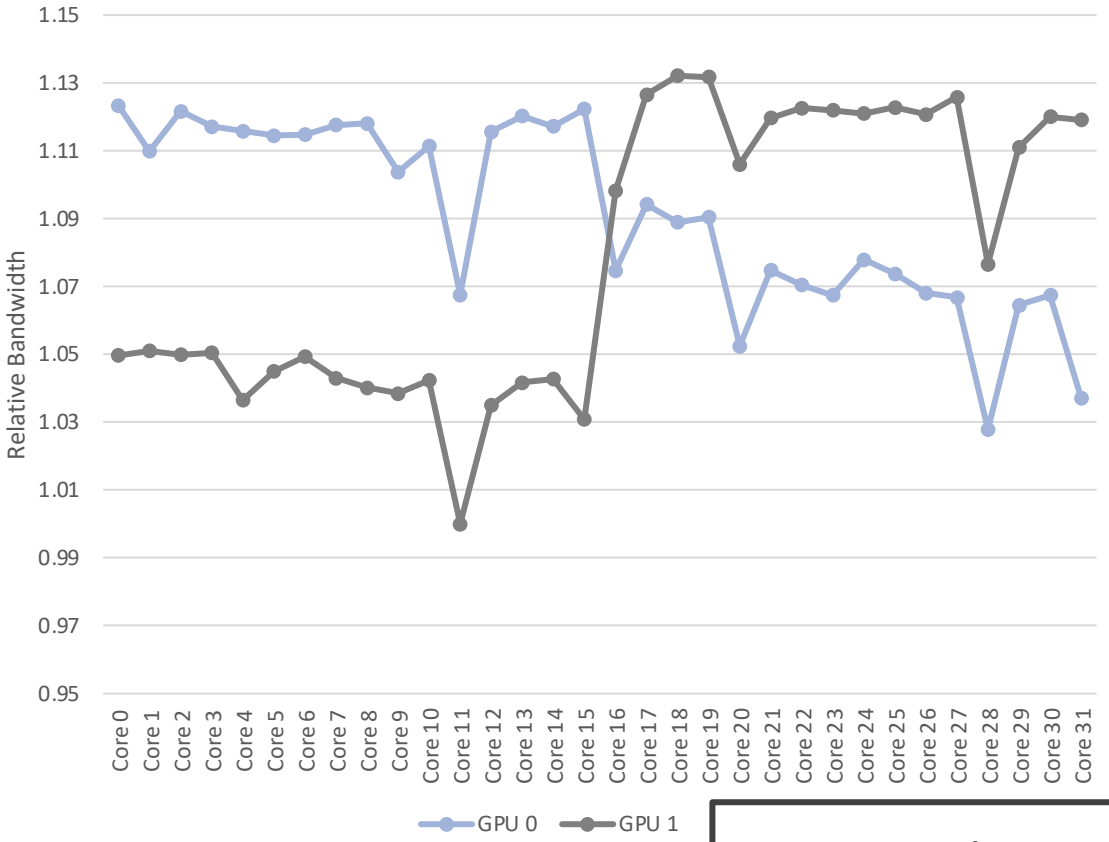


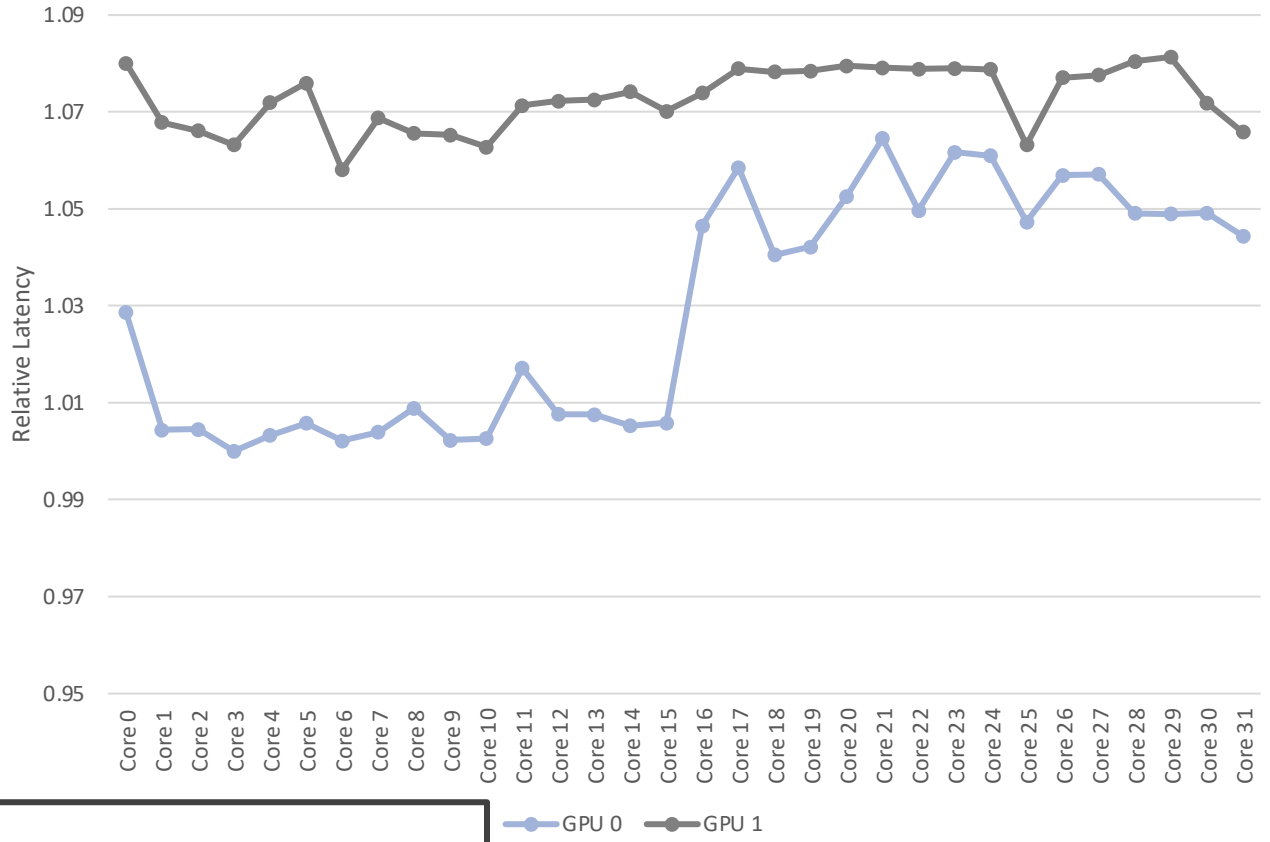
- **Scenario 1: Data not mapped to any device**
  - → Use device that is close to thread or data in host memory
- **Scenario 2: Offload to device that already holds part of required data**
  - → Minimize data movement
  - → Reuse existing data

Data/Thread-to-Device Affinity / 2

- **Question:** Does it matter?

Here: measurement on AMD MI210





- 2 socket / 8 NUMA domains: 32x AMD EPYC 7F52
- 2x AMD MI210
  - GPU 0 connected to NUMA domain 2
  - GPU 1 connected to NUMA domain 7

Taskloop Dependencies and Affinity

- **Motivation**
  - → Support the `depend` and `affinity` clause in combination with the `taskloop` construct (#2072, #2142)
  - → Exploit strengths and performance improvements from both sides
- **Sketch of the idea**
  - → Add option to specify dependencies and affinity at the start of the loop body with the `iteration_attribute` directive
  - → Add option to specify dependency flow in and out of `taskloop` with `gate(enter)` or `gate(exit)`

```
// Example: Dependencies between tasks within a taskloop as well as between taskloop and standalone task
#pragma omp taskloop collapse(2) grainsize(strict: 1) depend(gate(exit): A[5][10]) nogroup
#pragma omp tile sizes(32,32)
for (int i = 1; i < N; ++i) {
    for (int j = 1; j < N; ++j) {
        #pragma omp iteration_attribute depend(inout: A[i][j]) depend(in: A[i-1][j-1]) if(<cond>)
        A[i][j] += A[i-1][j-1];
    }
}

#pragma omp task depend(A[5][10])
{
    do_work(A[5][10]);
}
```

Panel

- Bronis de Supinski, Language Committee
- Stephen Olivier, Tasking
- Xinmin Tian, C/C++/Fortran
- Tom Scogland, Accelerator
- Christian Terboven, Affinity