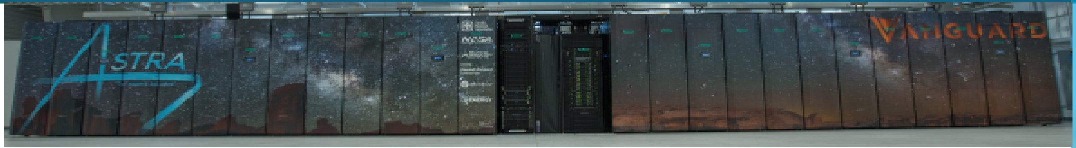
SAND2020-12772C

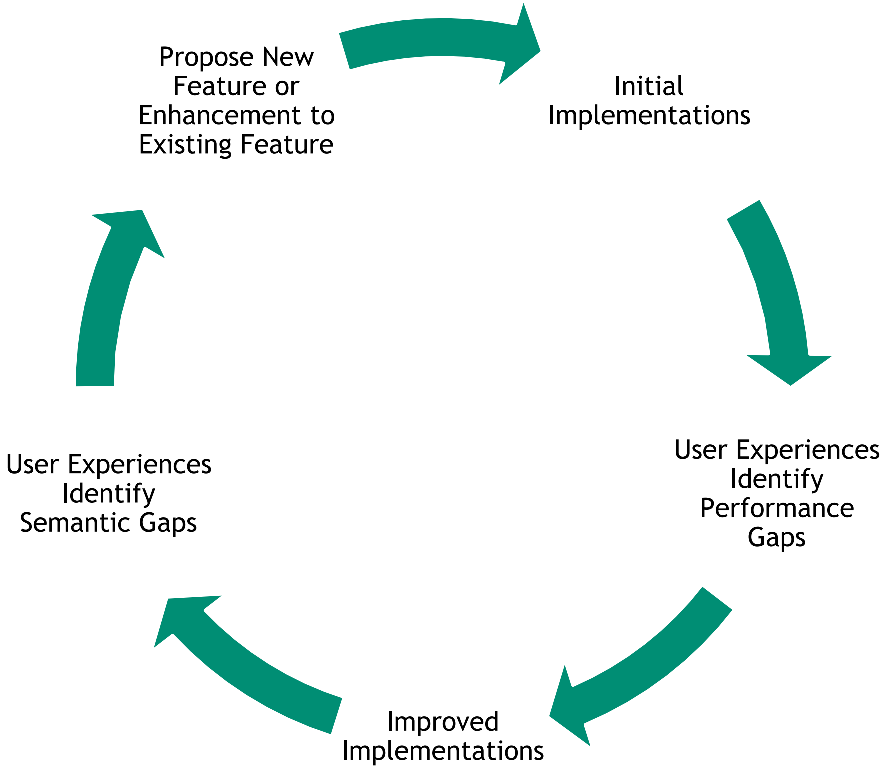
Programming Model Developments Present Opportunities for Runtime and Operating Systems



PRESENTED BY

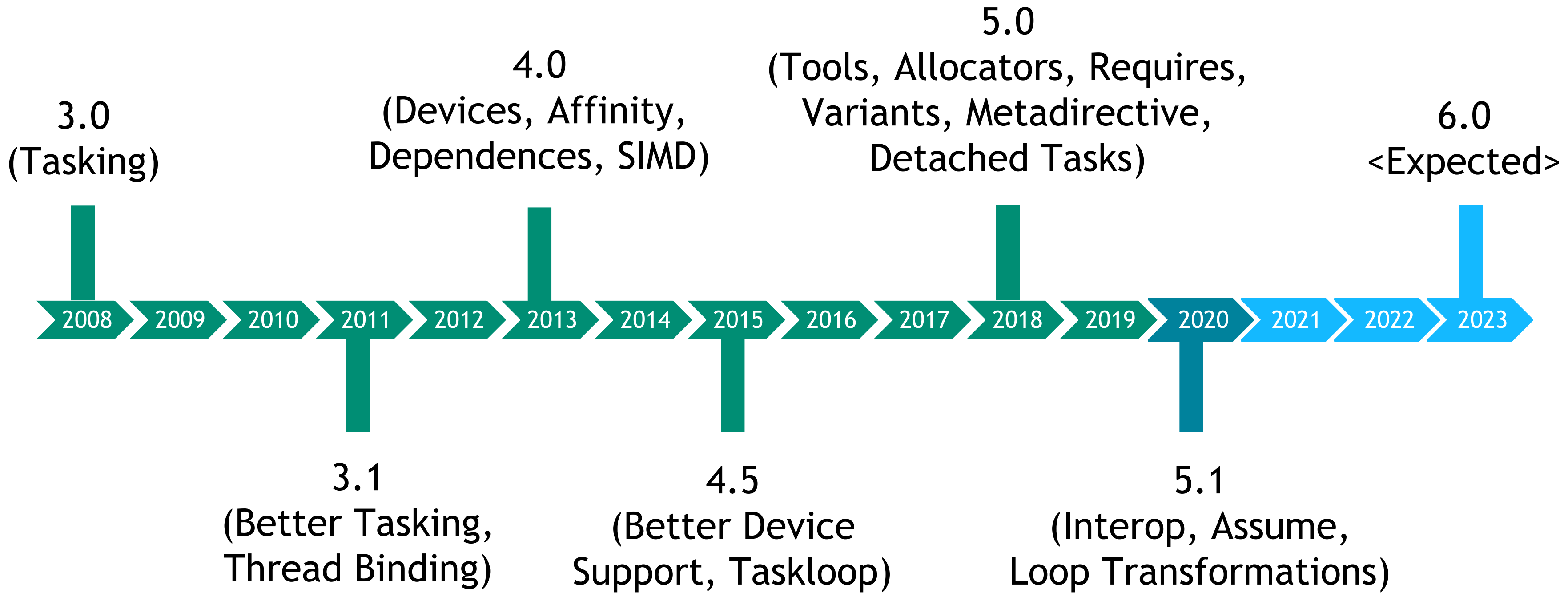
Stephen Olivier

Feature Development Cycle


Propose New Feature or Enhancement to Existing Feature
Initial Implementations
User Experiences Identify Performance Gaps
Improved Implementations
User Experiences Identify Semantic Gaps

Some Recent Developments in OpenMP

OpenMP Timeline


3.0
(Tasking)
4.0
(Devices, Affinity, Dependences, SIMD)
5.0
(Tools, Allocators, Requires, Variants, Metadirective, Detached Tasks)
6.0
<Expected>
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
3.1
(Better Tasking, Thread Binding)
4.5
(Better Device Support, Taskloop)
5.1
(Interop, Assume, Loop Transformations)

Recommended Reading: “The Ongoing Evolution of OpenMP” (*Proc. of the IEEE*)

OpenMP 5.0: *Requires* directive (Program Only Valid if…)

Sets system conditions for program execution:

- Unified address space across host and devices
- Unified shared memory across host and devices
- Default memory order for atomics, e.g., release/acquire
- Supports "reverse offload" from accelerator back to host
- Supports OpenMP allocators in offloaded code

Can also define implementation-defined clauses

- OpenMP-approved way to allow for extensions

Can make the program less portable

- If the conditions aren't met by the implementation, compilation and/or execution can fail

OpenMP 5.0: Allocators (User Chooses Memory Space + Traits)

Where to put it
[Choose 1]

Memory space name	Storage selection intent
omp_default_mem_space	Represents the system default storage.
omp_large_cap_mem_space	Represents storage with large capacity.
omp_const_mem_space	Represents storage optimized for variables with constant values. The result of writing to this storage is unspecified.
omp_high_bw_mem_space	Represents storage with high bandwidth.
omp_low_lat_mem_space	Represents storage with low latency.

Init. using compile-time constants or firstprivate

Configure as desired
[or use defaults]

Allocator trait	Allowed values	Default value
sync_hint	contended, uncontended, serialized, private	contended
alignment	A positive integer value that is a power of 2	1 byte
access	all, cgroup, pteam, thread	all
pool_size	Positive integer value	Implementation defined
fallback	default_mem_fb, null_fb, abort_fb, allocator_fb	default_mem_fb
fb_data	an allocator handle	(none)
pinned	true, false	false
partition	environment, nearest, blocked, interleaved	environment

What happens if not available

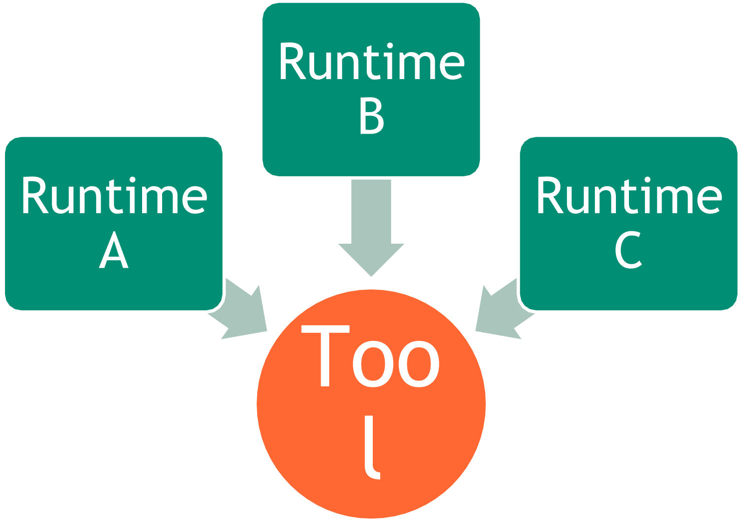
NUMA

OpenMP 5.0: Tools and Debugging Interfaces (OMPT and OMPD)

Huge untapped potential in the area of tools and debuggers for multithreaded programming

Need for common interfaces between tools/debuggers and OpenMP implementations
- Previously untenable *m : n* mapping of tools to runtimes
- Focus of tool development shifts to developing more capabilities rather than maintaining multiple interfaces

Now can understand OpenMP runtime behavior on its own terms
- Measure overheads of parallel regions threads, tasks
- Reveal thread idleness, scheduling decisions
- Opens doors for runtime introspection



[Video: “How to Write an OMPT-Based Tool”]

OpenMP 5.0: *Loop* Directive (Descriptive Instead of Prescriptive)

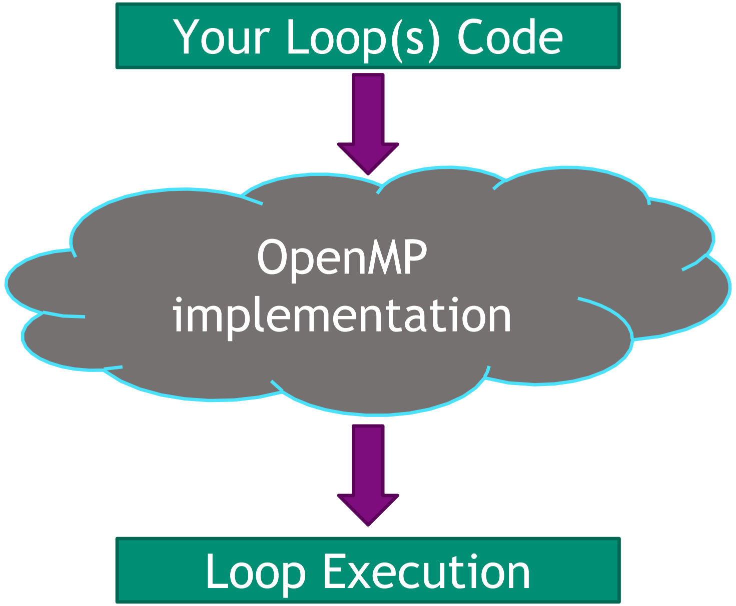
Wait, hasn't OpenMP supported parallel loops since 1997?

- Yes, but in a prescriptive way: the worksharing loop (*omp for/do*)
- Many constraints on execution, mapping iterations to threads

New loop directive allows implementation freedom

- Inspired by OpenACC loop
- "…associated loops may execute concurrently…"
- In the absence of clauses constraining behavior, just make it run fast using the execution resources available

For completeness, note also availability of *taskloop* and *distribute* directives introduced in previous OpenMP versions



OpenMP 5.0: *Declare Variant* and *Metadirective* (Pick Code A in Situation X)

Context selector defines properties of the code and the system used to determine which code to run

- Vendor, ISA, architecture
- On host or device
- Enclosing OpenMP constructs (e.g., parallel, teams)

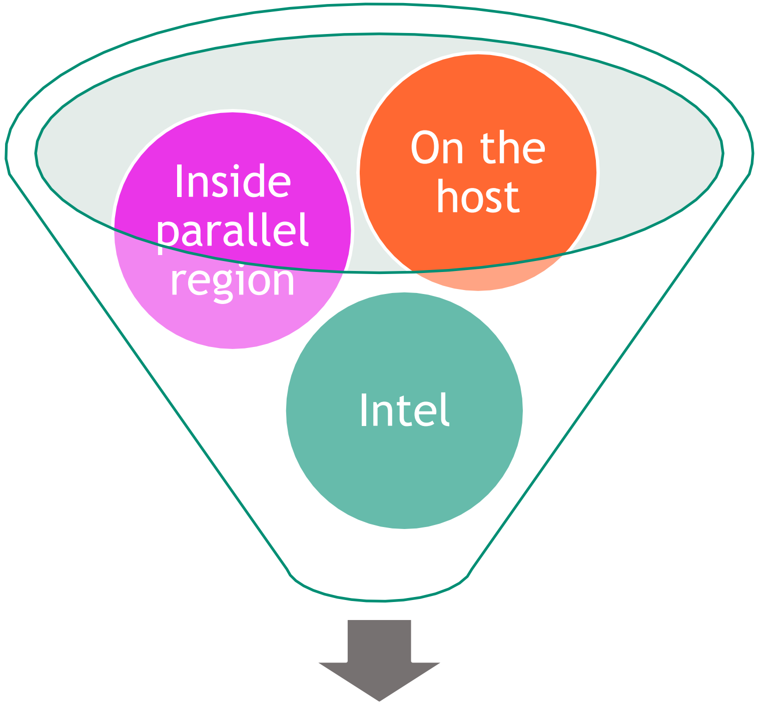
Specify code to run based on context selector

- *Declare variant* designates different versions of a function
- *Metadirective* chooses directive to use based on context
- Extended in OpenMP 5.1 with the *dispatch* directive

Performance portability requires some effort

- Added maintenance requirement for code specialization
- Burden for consistency across variants is on the programmer

YouTube video (includes 5.1 extensions)



Choose variant f42()

OpenMP 5.0: Detached Tasks (Event Handles for Task Completion)

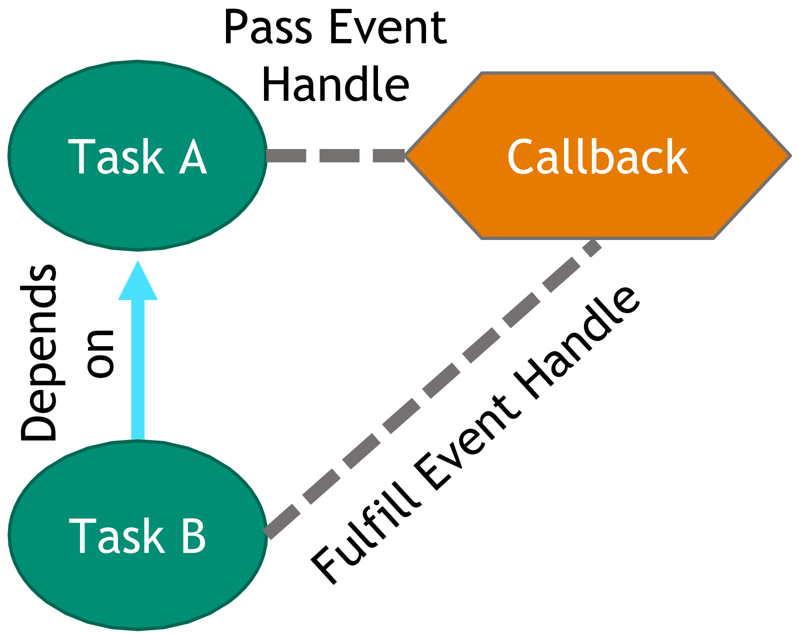
Ordinarily, reaching the end of an OpenMP task's body of code results in task completion, but…

- With the *detach* clause, completion is postponed until an event handle is set to fulfilled by an *omp_fulfill_event* runtime call

Can pass the event handle object to other functions

- E.g., callbacks for asynchronous native device operations
- Since detached tasks are part of the OpenMP dependence graph, such operations can trigger further OpenMP tasks

Step in the direction of more interoperability with other programming models



OpenMP 5.1: *Assume* Directive (Invariant Information About the Code)

Notifies implementation of invariants in code

- No OpenMP code
- No OpenMP runtime calls
- No OpenMP parallelism
- Certain constructs absent OR code contains them
- Given expression evaluates to true throughout

Rich trove of information for runtime decisions

- Choose optimized "fast paths" for some cases
- Estimate overhead costs
- Influence scheduling of tasks

YouTube presentation on this feature

OpenMP 5.1: *Interop* Directive (Cooperating with Other Runtimes)

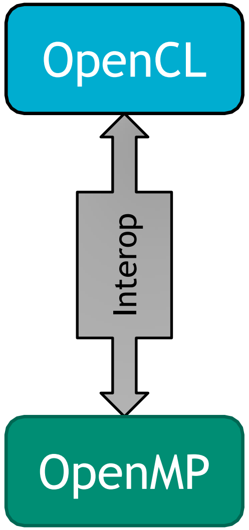
Longstanding need to enable cooperation between directive-based programming and "native" device code

- E.g., CUDA, OpenCL, SYCL

First in what will likely be many steps toward such interoperability, but for now:

- Get device context / platform properties
- Get CUDA/OpenCL stream
- Avoid blocking where not required

(Note that OpenMP 5.0 already provided simple runtime calls to pause the OpenMP runtime)



YouTube presentation on this feature

OpenMP 6.0: What's on the Horizon (Your Contributions Welcome!)

Many new possibilities for tasks

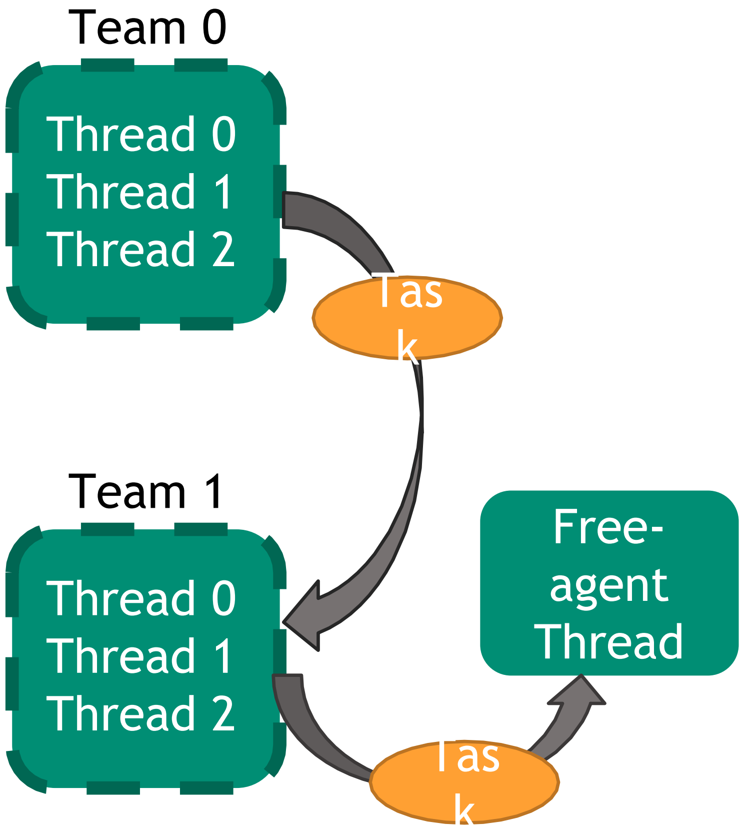
- Tasks transferrable between teams of threads
- Allow the runtime to manage "free-agent threads" that don't belong to any team but can execute tasks from any teams
- Tasks with progress guarantees to enable event-loop parallelism, polling, and real-time systems use cases

Further improvements for heterogeneous systems

- Coordinated work across multiple devices
- Affinity across host and device threads, tasks, and data
- Support for persistent memory

Additional power and flexible parallelism for *loop* directive

User-defined loop schedules



OpenMP: How to Contribute

LLVM open-source OpenMP runtime library

- Repository: https://github.com/llvm
- 12.0 status: https://clang.llvm.org/docs/OpenMPSupport.html
- Most vendor OpenMP implementations are LLVM-based
- Implementing proposed features in LLVM (or GCC) greatly increases chances of eventual inclusion in the specification

Related runtime efforts, e.g., Argobots / BOLT, StarPU, Nanos++ / OmpSs

Socialize your work and ideas with the OpenMP community

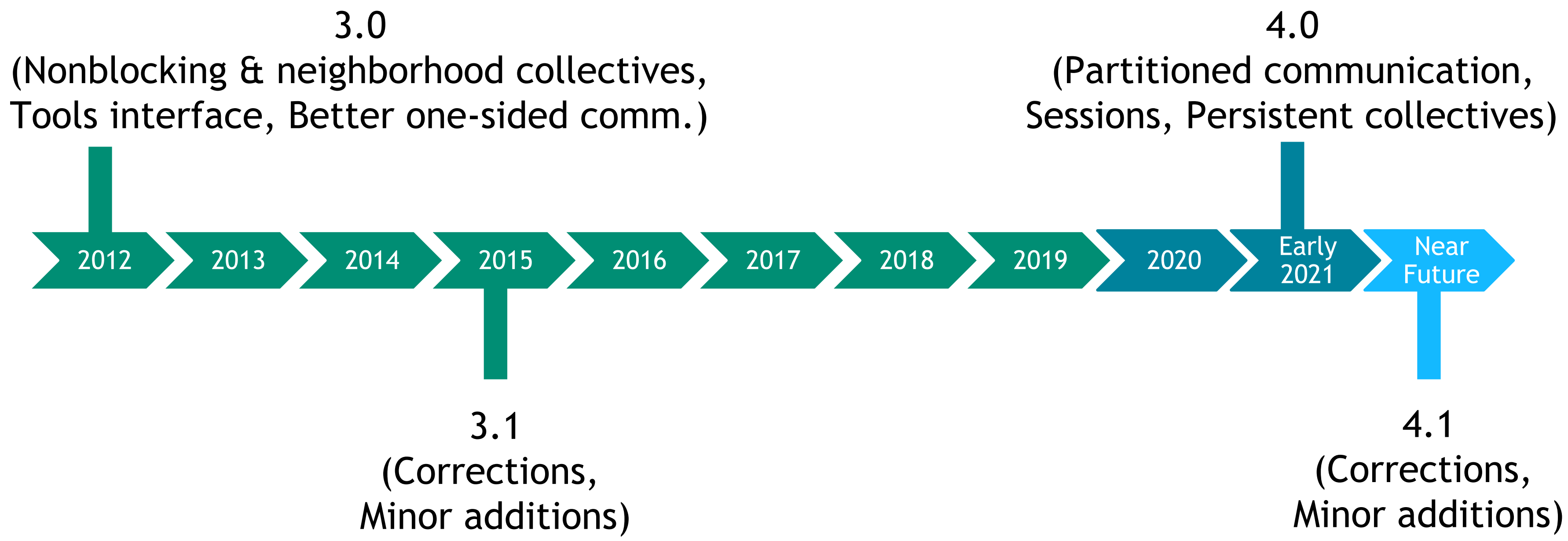
- Intl. Workshop on OpenMP (IWOMP): http://www.iwomp.org
- Presenting at IWOMP bestows membership in cOMPunity, the OpenMP users group that is part of the Language Committee
- Language Committee meets 3X/year and has weekly telecons
- Other opportunities posted at https://www.openmp.org
- SC20 BoF (link)

LLVM dragon logo copyright Apple Inc.

OpenMP logo copyright OpenMP ARB.

Some Recent Developments in MPI

MPI Timeline

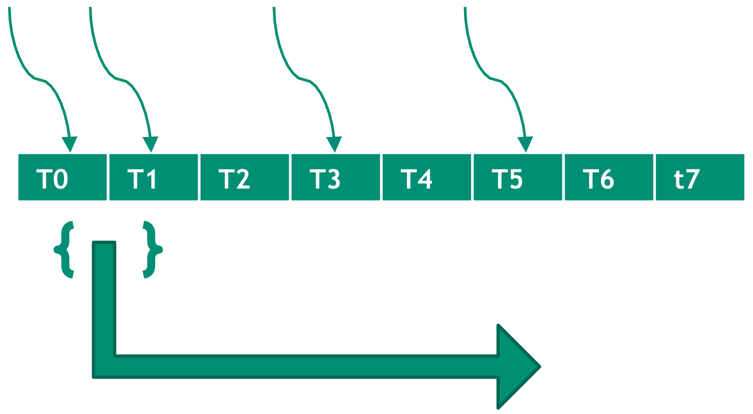

3.0
(Nonblocking & neighborhood collectives,
Tools interface, Better one-sided comm.)
4.0
(Partitioned communication,
Sessions, Persistent collectives)
2012
2013
2014
2015
2016
2017
2018
2019
2020
Early 2021
Near Future
3.1
(Corrections,
Minor additions)
4.1
(Corrections,
Minor additions)

Recommended Presentation: “Final Steps to MPI 4.0… and What’s Next” (EuroMPI 2020)

MPI 4.0: Partitioned Communication (A Way Forward for Threading and MPI)

Low-overhead multithreaded operations

- Each thread in sender process contributes its portion of data into its own partition of the buffer
- Can start transferring filled partitions to partitioned buffer at receiver while other send partitions not yet filled
- Does not inflate the rank space



Expected follow-on for MPI 4.x/5.0

- Enhanced GPU support when paired with NIC offering triggered operations
- Apply to other MPI features
- Partitioned collectives?

Detailed presentation on Partitioned Communication

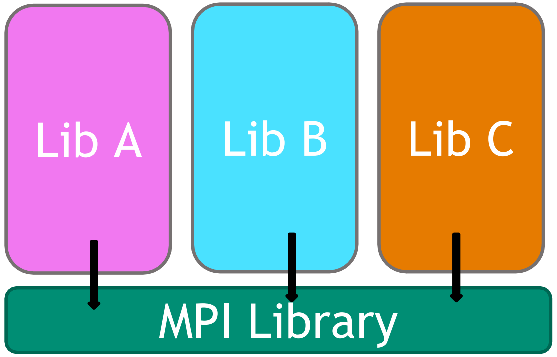
MPI 4.0: Sessions (Enabling Scalability and Composition)

Current limitations of MPI stand in the way of scalability and composition

- Applications increasingly composed of multiple district libraries or constituent components
- MPI initialization, MPI_COMM_WORLD set up, and communicator splitting not well suited to this reality

New approach: MPI sessions

- MPI session is a handle to the MPI library
- Through the session handle, can query available process list and construct a group of processes
- Communicator can be formed based on the group



IEEE CLUSTER paper on sessions implementation in OpenMPI

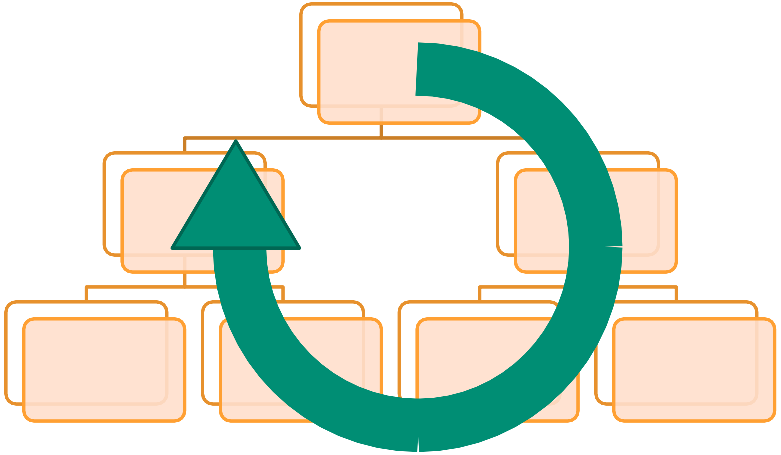
MPI 4.0: Persistent Collectives (Because Optimization Loves Repetition)

Persistent operations useful for iterative patterns commonly encountered in applications

- Available for point-to-point communication since early days of MPI standard

Now extended to collectives

- Initial setup costs, but subsequent calls optimized
- Runtime can develop and reuse a plan
- Possible to further optimize as more calls made



Parallel Computing article on persistent collectives

MPI: How to Contribute

Major open-source implementations with online repositories you can fork

- OpenMPI: https://www.open-mpi.org/ (repo)
- MPICH: https://www.mpich.org (repo)
- Most vendor implementations are based on one of the above implementations
- Also MPICH-based MVAPICH (can download source at OSU-hosted site)

Socialize your work and ideas with the MPI community

- EuroMPI conference: https://eurompi.github.io
- ExaMPI conference (today): https://sites.google.com/site/workshopexampi/
- MPI Forum meets (typically) 4X per year and has weekly telecons
- SC20 BoF (link)

Other Programming Models and Closing Thoughts

C++ Parallelism and SYCL (Gaining Momentum)

C++ Performance portability libraries popular with Exascale app developers

- Kokkos (Sandia Labs) and RAJA (Lawrence Livermore National Lab)

ISO C++ embracing parallelism/concurrency in recent/upcoming versions

- Parallel STL in C++17 (sequential, parallel, vectorized options)
- C++20 coroutines, atomic_ref, synchronization (latches, barriers, etc.)
- Executors in C++23 (proposed based on best practices of Kokkos and RAJA)
- OpenMP language committee closely monitoring advances in C++ to anticipate and define interactions between base language features and OpenMP

SYCL (standard from Kronos – the OpenCL & OpenGL people)

- Based on standard C++11
- oneAPI's Data Parallel C++ geared toward Intel devices: CPUs, GPUs, and FPGAs
- hipSYCL for some AMD and NVIDIA GPUs
- triSYCL reference implementation
- Codeplay's ComputeCpp

RAJA / LLNL

Kokkos / Sandia National Labs

SYCL and the SYCL logo are trademarks of the Khronos Group Inc.

oneAPI logo trademark Intel Corp.

Chapel (Reimagining Parallel Programming)

Unlike other programming models discussed in this talk
- Distinct programming language (not C/C++/Fortran extension)
- Effort headed by a dedicated team at Cray (now HPE)
- Initially part of DARPA’s High Productivity Computing Systems program

Global view parallelism approach
- Parallel loops and data structures distributed across nodes
- Task parallelism with flexible synchronizations

Runtime incorporates several distinct layers
- Berkeley GASNet for communication, Partitioned Global Address Space
- Sandia Qthreads for on-node task parallelism

Chapel Community resources
- CHUIW workshop usually in conjunction with IPDPS
- Cray Chapel web site

Chapel logo by Jim Cissell, Kristina Davis, Oli Laurelle, and Timothy Stitt.

Qthreads

Closing Thoughts

Staying power of MPI and OpenMP not to be underestimated, but they are looking over their shoulders…

- Applications thrive on sustainable programming models
- Early success of OpenACC pushed OpenMP to prioritize accelerator support
- MPI Forum considering separation of semantics and bindings in the standard to accommodate new base languages like Python
- C++ and Fortran upping their parallelism support in each new version

More ways than ever to make an impact

- Ample room for already implemented features to optimized
- Some features even specified before implementation
- New capabilities rely on efficient runtime, and some cases, OS foundations
- Open-source runtimes enable realistic prototyping and smooth feature uptake
- Many ideas and techniques transferable across programming models

It's an exciting time to be working in these communities!

The Python logo is a trademark of the Python Software Foundation