OpenMP API Version 6.0
What to Expect

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

Booth 307
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

- 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

PROGRAMMING YOUR GPU WITH OPENMP
Performance Portability for GPUs
Tom Deakin and Timothy G. Mattson

THE OPENMP COMMON CORE
Making OpenMP Simple Again
Timothy G. Mattson, Yuxi (Helen) He, and Alice E. Kontigét

USING OPENMP – THE NEXT STEP
Affinity, Accelerators, Tasking, and SPM
Rudolf van der Pol, Eric Stofz, and Christian Tietzen

HIGH PERFORMANCE PARALLEL RUNTIMES
Design and Implementation
Michael Klemm, Jim Cowtan

OpenMP Application Programming Interface
Specification Version 5.2
OpenMP Architecture Review Board
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

OpenMP ARB Releases Technical Report 12

- This is a preview of OpenMP 6.0, that will be released in 2024
- TR12 has improved support for tasking, devices, and C/C++
- TR12 is downloadable [here](#)

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

Xinmin Tian
Intel Corporation
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.

© Intel Corporation. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Other names and brands may be claimed as the property of others.
LLVM Powering the New Generation of Compilers

intel

+ 

= 

[Diagram showing layers and overlap with Intel and LLVM logos]
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

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

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

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

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

Gaining Momentum! oneAPI 2022.0 will be available soon!
Thank you!
it starts with Intel.
Programming the NVIDIA Platform

Unmatched Developer Flexibility

Languages & Programming Models
- Accelerated Standard Languages
  - C++
  - Fortran
  - Python
- Incremental Optimization
  - OpenACC
  - OpenMP
- Platform Specialization
  - CUDA
  - C++ | Fortran | Python

Libraries & Frameworks
- 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

STREAM triad: \( Z += a \times X + Y \)

<table>
<thead>
<tr>
<th>GPU (Year)</th>
<th>CUDA</th>
<th>CUDA MM</th>
<th>OpenMP C++</th>
<th>OpenMP C</th>
<th>OpenMP Fortran</th>
</tr>
</thead>
<tbody>
<tr>
<td>A100-40G (2020)</td>
<td>1.38</td>
<td>1.35</td>
<td>1.33</td>
<td>1.33</td>
<td>1.32</td>
</tr>
<tr>
<td>A100-80G (2021)</td>
<td>1.81</td>
<td>1.79</td>
<td>1.70</td>
<td>1.70</td>
<td>1.70</td>
</tr>
<tr>
<td>H100-PCIe (2022)</td>
<td>1.91</td>
<td>1.91</td>
<td>1.84</td>
<td>1.83</td>
<td>1.85</td>
</tr>
<tr>
<td>GH200 (2023)</td>
<td>3.77</td>
<td>3.75</td>
<td>3.74</td>
<td>3.74</td>
<td>3.62</td>
</tr>
</tbody>
</table>
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: 2023-02-08R3.22

SPECaccel2023 (17):

<table>
<thead>
<tr>
<th>Test Sponsor</th>
<th>System Name</th>
<th>Accelerator</th>
<th>Base Threads</th>
<th>Processor</th>
<th>Processor</th>
<th>Base Model</th>
<th>Peak Model</th>
<th>Results</th>
<th>Energy</th>
</tr>
</thead>
<tbody>
<tr>
<td>NVIDIA Corporation</td>
<td>SuperServer SYS-1823QG-TMT</td>
<td>Tesla V100-PCE-16GB</td>
<td>1</td>
<td>48</td>
<td>2</td>
<td>ACC</td>
<td>ACC</td>
<td>1.23</td>
<td>1.23</td>
</tr>
<tr>
<td>NVIDIA Corporation</td>
<td>SuperServer SYS-1823QG-TMT</td>
<td>Tesla V100-PCE-16GB</td>
<td>1</td>
<td>48</td>
<td>2</td>
<td>2LPU</td>
<td>2LPU</td>
<td>1.23</td>
<td>1.23</td>
</tr>
<tr>
<td>NVIDIA Corporation</td>
<td>SuperServer SYS-1823QG-TMT</td>
<td>Tesla V100-PCE-16GB</td>
<td>1</td>
<td>48</td>
<td>2</td>
<td>TGT</td>
<td>TGT</td>
<td>6.999</td>
<td>6.999</td>
</tr>
<tr>
<td>NVIDIA Corporation</td>
<td>DGX-A100</td>
<td>Tesla A100-SM-8GB</td>
<td>1</td>
<td>120</td>
<td>2</td>
<td>2LPU</td>
<td>2LPU</td>
<td>2.76</td>
<td>2.76</td>
</tr>
<tr>
<td>NVIDIA Corporation</td>
<td>DGX-A100</td>
<td>Tesla A100-SM-8GB</td>
<td>1</td>
<td>120</td>
<td>2</td>
<td>LOP</td>
<td>LOP</td>
<td>2.63</td>
<td>2.63</td>
</tr>
<tr>
<td>NVIDIA Corporation</td>
<td>DGX-A100</td>
<td>Tesla A100-SM-8GB</td>
<td>1</td>
<td>120</td>
<td>2</td>
<td>TGT</td>
<td>TGT</td>
<td>4.47</td>
<td>4.47</td>
</tr>
<tr>
<td>NVIDIA Corporation</td>
<td>DGX-H100</td>
<td>Tesla H100-SM-8GB</td>
<td>1</td>
<td>112</td>
<td>2</td>
<td>LOP</td>
<td>LOP</td>
<td>4.11</td>
<td>4.11</td>
</tr>
<tr>
<td>NVIDIA Corporation</td>
<td>DGX-H100</td>
<td>Tesla H100-SM-8GB</td>
<td>1</td>
<td>112</td>
<td>2</td>
<td>2LPU</td>
<td>2LPU</td>
<td>3.38</td>
<td>3.38</td>
</tr>
<tr>
<td>NVIDIA Corporation</td>
<td>RCA-Q200</td>
<td>Tesla H100-64GB</td>
<td>1</td>
<td>72</td>
<td>1</td>
<td>2LPU</td>
<td>2LPU</td>
<td>6.32</td>
<td>6.32</td>
</tr>
<tr>
<td>NVIDIA Corporation</td>
<td>RCA-Q200</td>
<td>Tesla H100-64GB</td>
<td>1</td>
<td>72</td>
<td>1</td>
<td>LOP</td>
<td>LOP</td>
<td>1.74</td>
<td>1.74</td>
</tr>
<tr>
<td>NVIDIA Corporation</td>
<td>RCA-Q200</td>
<td>Tesla H100-64GB</td>
<td>1</td>
<td>72</td>
<td>1</td>
<td>TGT</td>
<td>TGT</td>
<td>4.49</td>
<td>4.49</td>
</tr>
<tr>
<td>NVIDIA Corporation</td>
<td>128Q-TRNT</td>
<td>Tesla H300-PCE-8GB</td>
<td>1</td>
<td>64</td>
<td>2</td>
<td>2LPU</td>
<td>2LPU</td>
<td>2.08</td>
<td>2.08</td>
</tr>
<tr>
<td>NVIDIA Corporation</td>
<td>128Q-TRNT</td>
<td>Tesla H300-PCE-8GB</td>
<td>1</td>
<td>64</td>
<td>2</td>
<td>TGT</td>
<td>TGT</td>
<td>2.52</td>
<td>2.52</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Test Sponsor</th>
<th>System Name</th>
<th>Accelerator</th>
<th>Base Threads</th>
<th>Processor</th>
<th>Processor</th>
<th>Base Model</th>
<th>Peak Model</th>
<th>Results</th>
<th>Energy</th>
</tr>
</thead>
<tbody>
<tr>
<td>NVIDIA Corporation</td>
<td>128Q-TRNT</td>
<td>Tesla H300-PCE-8GB</td>
<td>1</td>
<td>64</td>
<td>2</td>
<td>TGT</td>
<td>TGT</td>
<td>2.52</td>
<td>2.52</td>
</tr>
<tr>
<td>NVIDIA Corporation</td>
<td>128Q-TRNT</td>
<td>Tesla A100-PCE-8GB</td>
<td>1</td>
<td>64</td>
<td>2</td>
<td>2LPU</td>
<td>2LPU</td>
<td>2.57</td>
<td>2.57</td>
</tr>
<tr>
<td>NVIDIA Corporation</td>
<td>128Q-TRNT</td>
<td>Tesla A100-PCE-8GB</td>
<td>1</td>
<td>64</td>
<td>2</td>
<td>LOP</td>
<td>LOP</td>
<td>2.57</td>
<td>2.57</td>
</tr>
<tr>
<td>NVIDIA Corporation</td>
<td>128Q-TRNT</td>
<td>Tesla A100-PCE-8GB</td>
<td>1</td>
<td>64</td>
<td>2</td>
<td>TGT</td>
<td>TGT</td>
<td>2.08</td>
<td>2.08</td>
</tr>
</tbody>
</table>

Last update: 2023-02-08R3.22

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

---

SPECaccel 2023 on Grace Hopper

- OpenACC
- OpenACC (Unified Memory)
- StdPar (Unified Memory)

---

POT3D: Explicit, Managed, and Unified Memories

- Managed Memory
- Explicit Memory
- Explicit Memory
- Pure Fortran

x86 + H100
GH200
# OPENMP STATUS AS OF CCE 16

<table>
<thead>
<tr>
<th>CCE 10.0 (May 2020)</th>
<th>CCE 11.0 (Nov 2020)</th>
<th>CCE 12.0 (Jun 2021)</th>
<th>CCE 13.0 (Nov 2021)</th>
<th>CCE 15.0 (Nov 2022)</th>
</tr>
</thead>
<tbody>
<tr>
<td>• OMP_TARGET_OFFLOAD</td>
<td>• noncontig update</td>
<td>• device_type (Fortran)</td>
<td>• declare variant (C/C++)</td>
<td>• target task reduction (C/C++)</td>
</tr>
<tr>
<td>• reverse offload</td>
<td>• map Fortran DVs</td>
<td>• affinity clause</td>
<td>• metadirectives (C/C++)</td>
<td>• uses_allocators</td>
</tr>
<tr>
<td>• implicit declare target</td>
<td>• host teams</td>
<td>• conditional lastprivate (C/C++)</td>
<td>• mapper (C/C++)</td>
<td>• error directive (Fortran only)</td>
</tr>
<tr>
<td>• omp_get_device_num</td>
<td>• use_device_addr</td>
<td>• simd if (C/C++)</td>
<td>• extend defaultmap (Fortran)</td>
<td>• compare clause on atomic construct (C/C++ only)</td>
</tr>
<tr>
<td>• OMP_DISPLAY_AFFINITY</td>
<td>• nested declare target</td>
<td>• iterator in depend (C/C++)</td>
<td>• close modifier (Fortran)</td>
<td>• assume and assumes directives (Fortran only)</td>
</tr>
<tr>
<td>• OMP_AFFINITY_FORMAT</td>
<td>• allocator routines</td>
<td>• depobj for depend (C/C++)</td>
<td>• mutexinoutset (Fortran)</td>
<td>• nothing directive (Fortran only)</td>
</tr>
<tr>
<td>• set/get affinity display</td>
<td>• OMP_ALLOCATOR</td>
<td>• task reduction (C/C++)</td>
<td>• inoutset dependence type</td>
<td>• otherwise clause for metadirective (Fortran only)</td>
</tr>
<tr>
<td>• display/capture affinity</td>
<td>• allocate directive</td>
<td>• task modifier (C/C++)</td>
<td>• present behavior for defaultmap (C/C++ only)</td>
<td></td>
</tr>
<tr>
<td>• requires</td>
<td>• allocate clause</td>
<td>• simd nontemporal (C/C++)</td>
<td>• masked task construct without filter clause (Fortran only)</td>
<td></td>
</tr>
<tr>
<td>• unified_address</td>
<td>• order(concurrent)</td>
<td>• scan (C/C++)</td>
<td>• metadirective dynamic user condition and target_device selectors (Fortran only)</td>
<td></td>
</tr>
<tr>
<td>• unified_shared_memory</td>
<td>• atomic hints</td>
<td>• Ivalue list items for depend</td>
<td>• taskloop cancellation (C/C++)</td>
<td></td>
</tr>
<tr>
<td>• atomic_default_mem_order</td>
<td>• default nonmonotonic</td>
<td>• mutexinoutset (C/C++)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• dynamic_allocators</td>
<td>• imperfect loop collapse</td>
<td>• simd nontemporal (C/C++)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• reverse_offload</td>
<td>• pause resources</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• combined master constructs</td>
<td>• atomics in simd</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• acq/rel memory ordering (Fortran)</td>
<td>• simd in simd</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• deprecate nested-var</td>
<td>• detachable tasks</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• taskwait depend</td>
<td>• omp_control_tool</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• simd nontemporal (Fortran)</td>
<td>• OMPT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• deprecate nested-var</td>
<td>• OMPD</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• taskwait depend</td>
<td>• declare variant (Fortran)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• simd nontemporal (Fortran)</td>
<td>• loop construct</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• lvalue map/motion list items</td>
<td>• metadirectives (Fortran)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• allow != in canonical loop</td>
<td>• pointer attach</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• close modifier (C/C++)</td>
<td>• array shaping</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• extend defaultmap (C/C++)</td>
<td>• acq/rel memory ordering (C/C++)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• device_type (C/C++)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• non-rectangular loop collapse (Fortran)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• device_type (C/C++)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• non-rectangular loop collapse (C/C++)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

CCE 14.0 (May 2022)

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

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

Available in 2-4 weeks!

```cpp
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; });
}
```
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]);
    }
}

#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);

int use(int &a, int &b) { … }

void kernel(int *a, int *b, int n) {
    int shared[128];
    #pragma omp group private(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() + tid;
    if (idx < n) {
        b[idx] = use(a[idx], shared[tid]);
    }
}
OpenMP Kernel Record and Replay
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
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® in the ROCm™ Stack**

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

<table>
<thead>
<tr>
<th>Applications</th>
<th>HPC Apps</th>
<th>ML Frameworks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster Deployment</td>
<td>Singularity</td>
<td>SLURM</td>
</tr>
<tr>
<td>Tools</td>
<td>Debugger</td>
<td>Profiler, Tracer</td>
</tr>
<tr>
<td>Portability Frameworks</td>
<td>Kokkos</td>
<td>RAJA</td>
</tr>
<tr>
<td>Math Libraries</td>
<td>RNG, FFT</td>
<td>Sparse</td>
</tr>
<tr>
<td>Scale-out Comm. Libraries</td>
<td>OpenMPI</td>
<td>UCX</td>
</tr>
<tr>
<td>Programming Models</td>
<td>OpenMP</td>
<td>HIP</td>
</tr>
<tr>
<td>Processors</td>
<td>CPU + GPU</td>
<td></td>
</tr>
</tbody>
</table>

*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](https://github.com/RadeonOpenCompute)
- Developer versions available at [https://github.com/ROCm-Developer-Tools/aomp](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.

© 2023 Advanced Micro Devices, Inc. All rights reserved.
AMD, the AMD Arrow logo, EPYC, Instinct and combinations thereof are trademarks of Advanced Micro Devices, Inc. PCIe is a registered trademark of PCI-SIG Corporation. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.
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
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

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

- 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
OPENMP 6.0 TASKING: RISE OF THE FREE-AGENT THREADS

Stephen Olivier, Tasking Subcommittee Chair
OpenMP Language Committee

SC23 OpenMP BoF
November 2023
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...

```
#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.

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

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.

```c
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`:

```bash
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

```fortran
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:

**OMPAVAILABLE_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
Did you know that you can ... allocate in high-bandwidth memory?

```c
#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

   ```c
   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

   ```c
   #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
**Question:** Does it matter?

Here: measurement on AMD MI210

---

<table>
<thead>
<tr>
<th>Core</th>
<th>Relative Latency</th>
<th>Relative Bandwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.9 5</td>
<td>0.9 7</td>
</tr>
<tr>
<td></td>
<td>1.0 1</td>
<td>1.0 3</td>
</tr>
<tr>
<td></td>
<td>1.0 5</td>
<td>1.0 7</td>
</tr>
<tr>
<td></td>
<td>1.1 1</td>
<td>1.1 3</td>
</tr>
</tbody>
</table>

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)`

```c
// 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