Panel Session:

Prioritizing OpenMP Features to Provide for Performance, Portability and Productivity

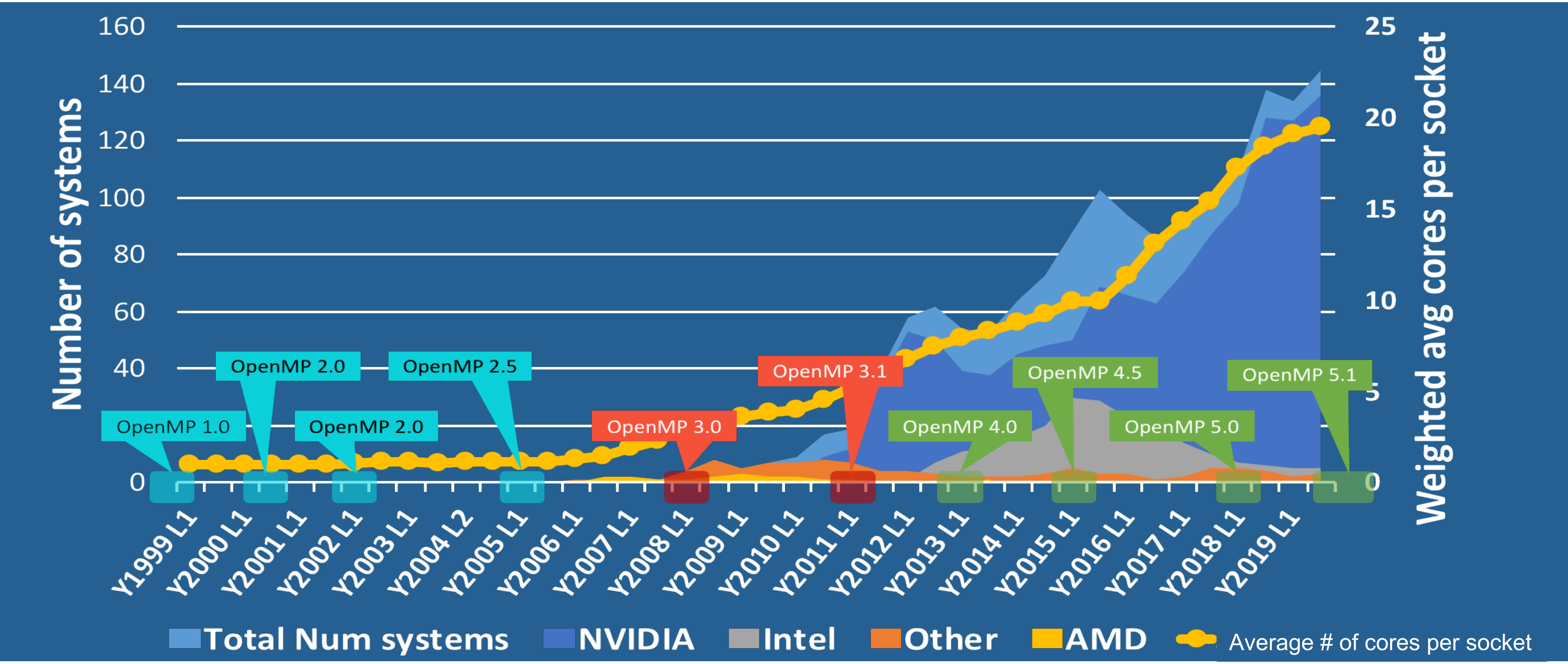
Oscar Hernandez (ORNL)

Vivek Kale (BNL)

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.

How OpenMP evolves compared with HPC trends (www.top500.org)

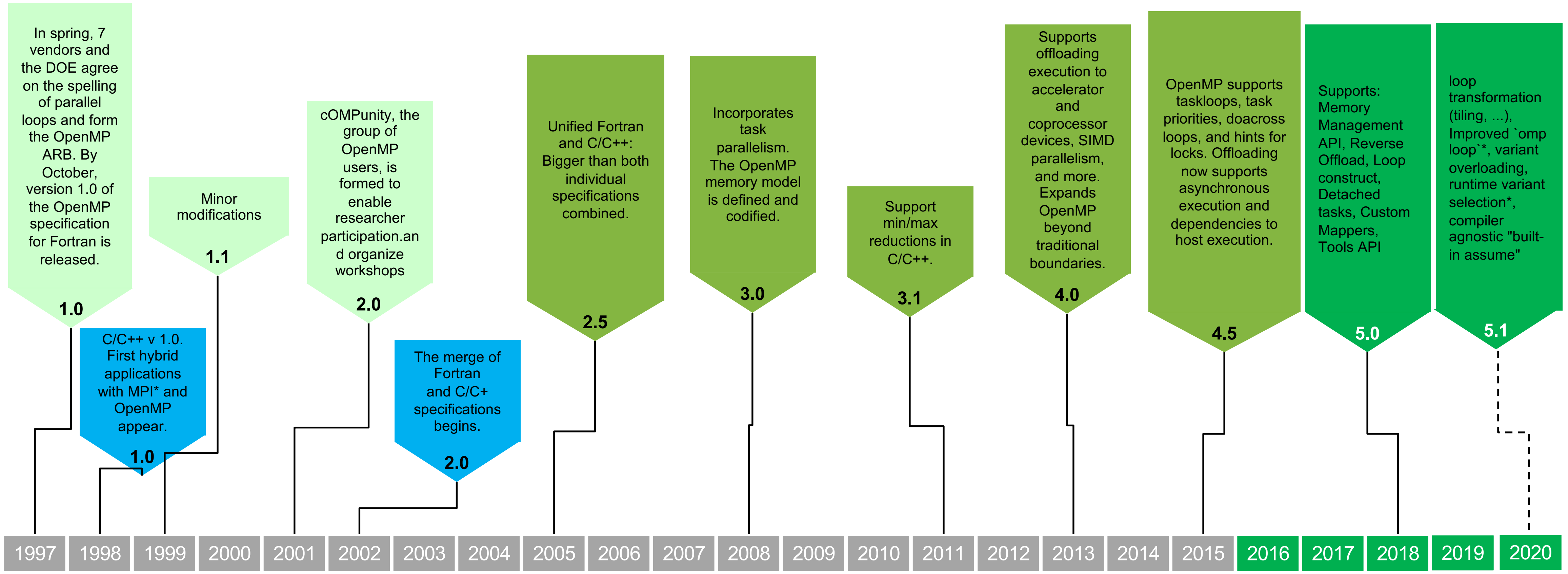


Parallel regions | Tasking | Device offloading / SIMD

Credit: Jose Monsalve Diaz, at University of Delaware

EXASCALE COMPUTING PROJECT

History of OpenMP: 1997 - 2020



1.0 — In spring, 7 vendors and the DOE agree on the spelling of parallel loops and form the OpenMP ARB. By October, version 1.0 of the OpenMP specification for Fortran is released.

1.0 — C/C++ v 1.0. First hybrid applications with MPI* and OpenMP appear.

1.1 — Minor modifications

2.0 — cOMPunity, the group of OpenMP users, is formed to enable researcher participation.an d organize workshops

2.0 — The merge of Fortran and C/C+ specifications begins.

2.5 — Unified Fortran and C/C++: Bigger than both individual specifications combined.

3.0 — Incorporates task parallelism. The OpenMP memory model is defined and codified.

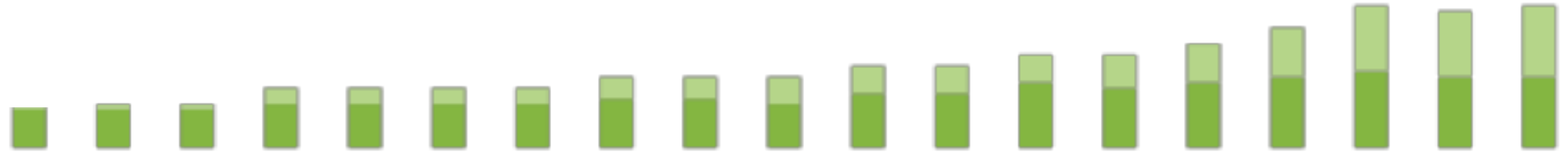
3.1 — Support min/max reductions in C/C++.

4.0 — Supports offloading execution to accelerator and coprocessor devices, SIMD parallelism, and more. Expands OpenMP beyond traditional boundaries.

4.5 — OpenMP supports taskloops, task priorities, doacross loops, and hints for locks. Offloading now supports asynchronous execution and dependencies to host execution.

5.0 — Supports: Memory Management API, Reverse Offload, Loop construct, Detached tasks, Custom Mappers, Tools API

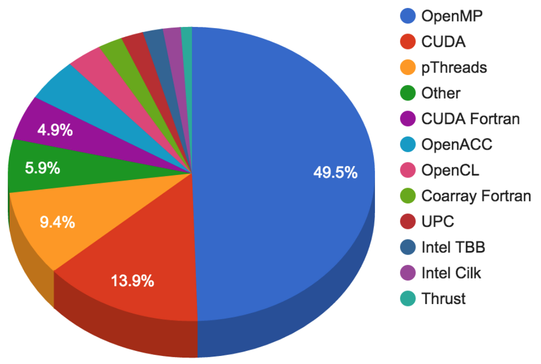
5.1 — loop transformation (tiling, ...), Improved `omp loop`*, variant overloading, runtime variant selection*, compiler agnostic "built-in assume"



🟩 Permanent ARB ▨ Auxiliary ARB

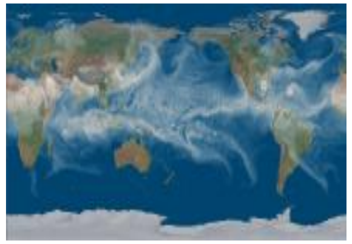
Relevance of OpenMP

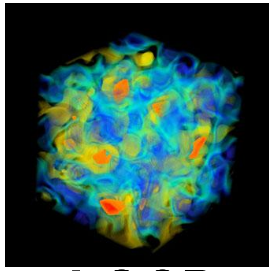
OpenMP is about 50%, out of all choices of X

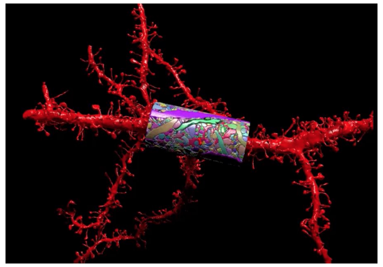


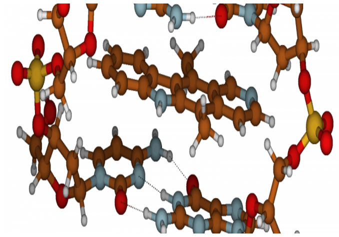
Update late 2016: 75% of codes use OpenMP

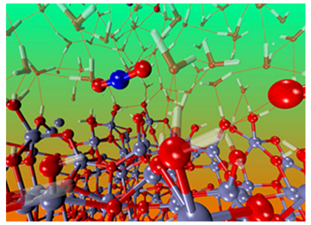
- Programming Accelerators
- Manage memory allocations (High Bandwidth, Low Latency, Accelerator)
- memories) with traits (pinned memory, etc)
- Data movement of complicated data structures (e.g., deep copy)
- Support for latest C++ and Fortran standads
- Interoperability with libraries
- Performance portable directives
- Task parallelism for asynchronous execution to orchestrate work between CPUs and Accelerators
- SIMD directives (to support SIMD parallelism)
- **Focus on continuity of technology and early access to users**


E3SM


LQCD


CANDLE


QMCPACK


NWCHEM

OpenMP Offload in QMCPack

Tests from miniQMC

Compiler	Clang 9	AOMP 0.7-4	XL 16.1.1-3	Cray 9.0	GCC 9.2	GCC 10
device	NV	AMD	NV	NV	NV	AMD
math header conflict	F	P	P	P	P	-
math linker error	P	P	P	F	P	-
declare target static data	P	P	P	P	F	-
static linking	F	P	P	P	F	-
Async tasking	F	F	P	F	F	-
multiple stream	F	P	P	F	F	-
check_spo	FR	FW	P	P	FL	-
check_spo_batched	FR	P	P	P	FL	-
miniqmc_sync_move	FR	P	P	P	FL	-

```
P pass
F fail
FL fail in linking
FR fail in run
FW fail with wrong results
- not tested yet
```

XL is the only survival
Other compilers need further improvements

Figure 7: Test results impacting performance in MiniQMC

Performance with OpenMP Impls.

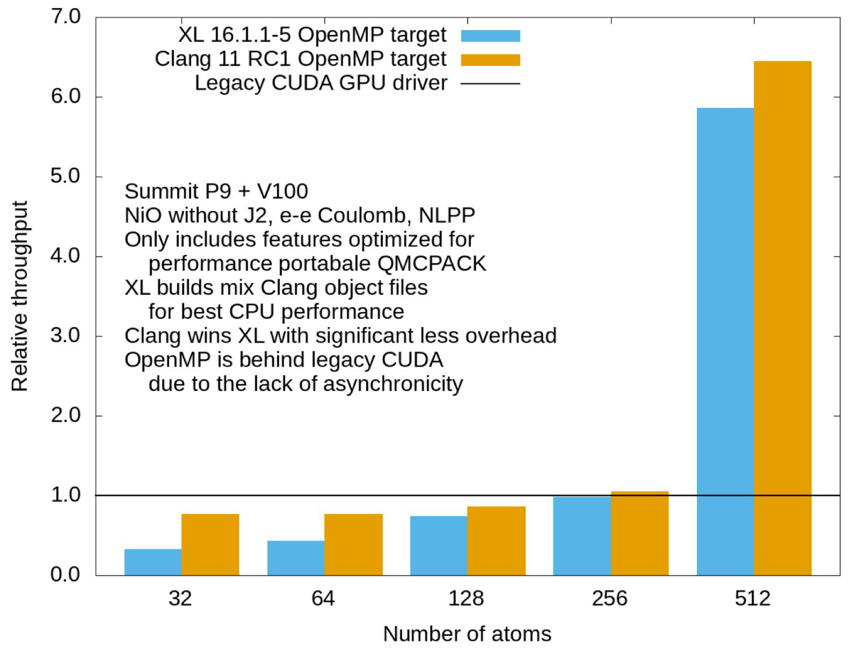


Figure 8: Performance of QMCPack with IBM OpenMP

Other Implementations

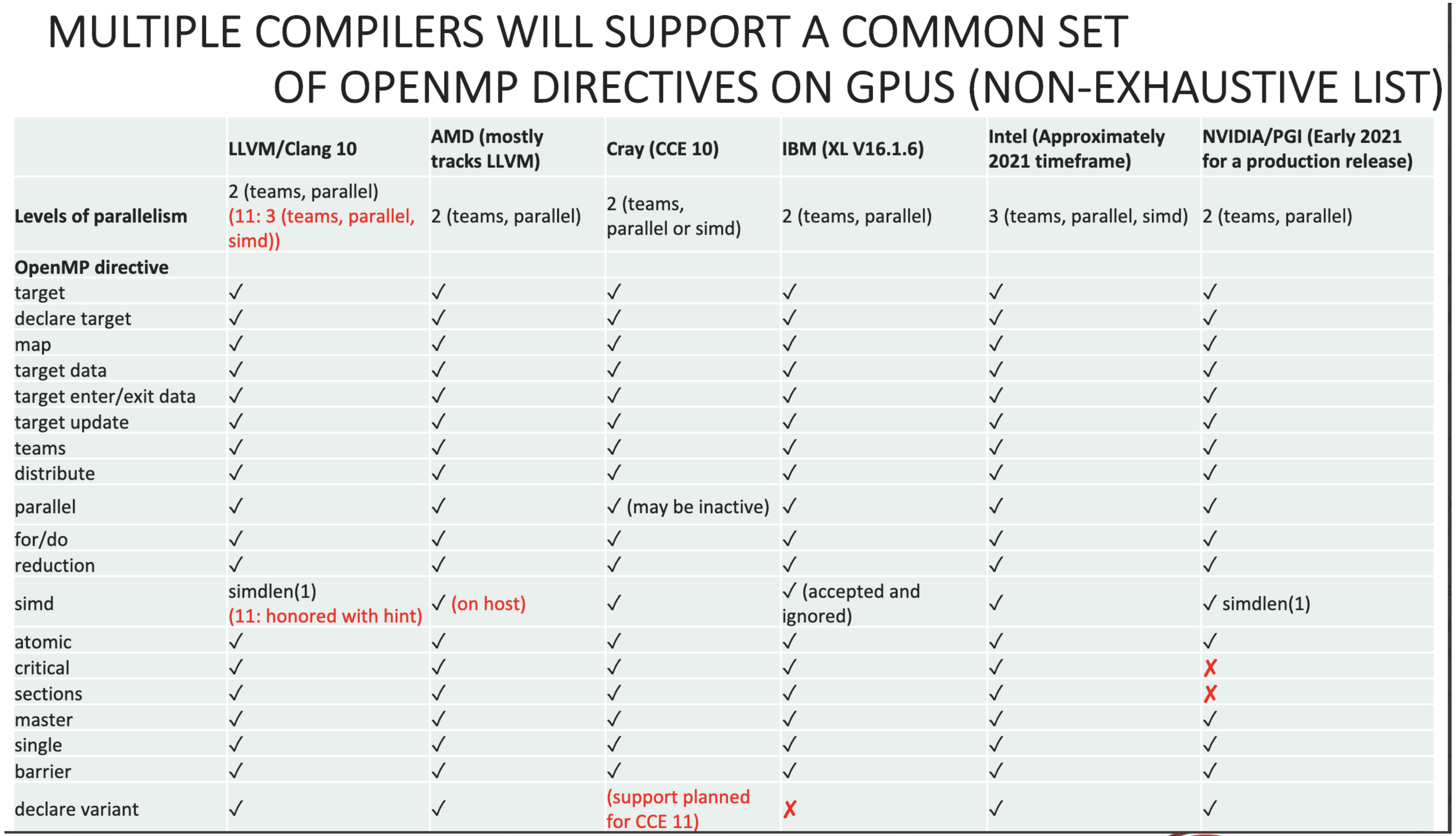
- Recent work (last few months) with clang to improve it, e.g., on target region-to-stream scheduling, support for std::complex shows promise for performance.
- Still can't show clang result due to unique-to-Summit CUDA driver problem soon to be fixed, but clang OpenMP estimated to have 0.75 performance of IBM XL.
- Also have run with Cray clang and AMD AOMP correctly. These show promise though don't have all feature support of clang.
- Got code to work with oneAPI.

→ IBM OpenMP is shown reasonably performant though rapid development of LLVM OpenMP has shown significant promise to allow for better performance over IBM offload.
→ QMCPack will continue to track performance of latest OpenMP implementations available on ECP systems.

Courtesy Paul Kent (ORNL) and Ye Luo (ANL) from QMCPack team.

Functionality Status of Features in OpenMP Implementations

Shows the features that are commonly supported across OpenMP Implementations



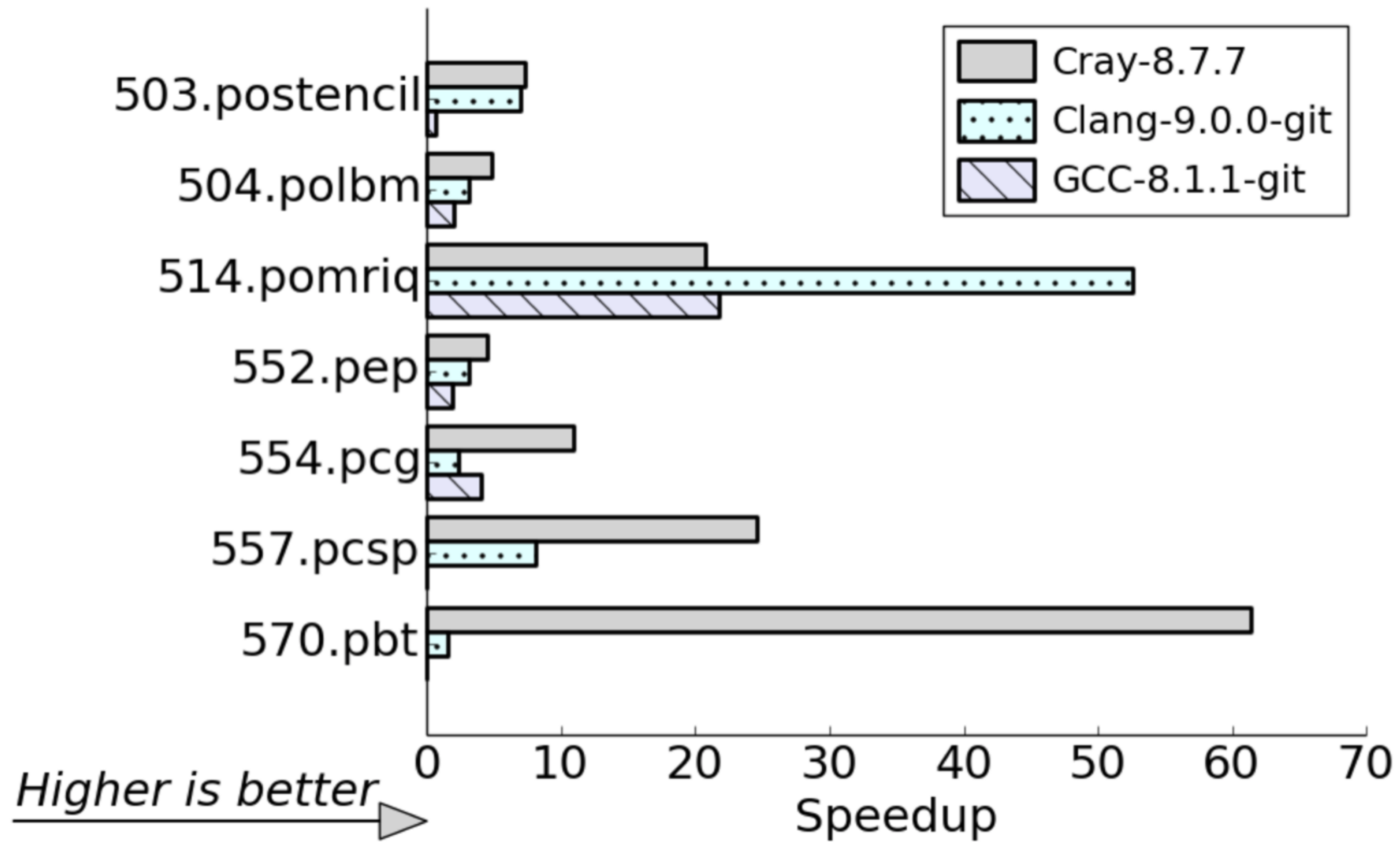
MULTIPLE COMPILERS WILL SUPPORT A COMMON SET OF OPENMP DIRECTIVES ON GPUS (NON-EXHAUSTIVE LIST)

	LLVM/Clang 10	AMD (mostly tracks LLVM)	Cray (CCE 10)	IBM (XL V16.1.6)	Intel (Approximately 2021 timeframe)	NVIDIA/PGI (Early 2021 for a production release)
Levels of parallelism	2 (teams, parallel) (11: 3 (teams, parallel, simd))	2 (teams, parallel)	2 (teams, parallel or simd)	2 (teams, parallel)	3 (teams, parallel, simd)	2 (teams, parallel)
OpenMP directive						
target	✓	✓	✓	✓	✓	✓
declare target	✓	✓	✓	✓	✓	✓
map	✓	✓	✓	✓	✓	✓
target data	✓	✓	✓	✓	✓	✓
target enter/exit data	✓	✓	✓	✓	✓	✓
target update	✓	✓	✓	✓	✓	✓
teams	✓	✓	✓	✓	✓	✓
distribute	✓	✓	✓	✓	✓	✓
parallel	✓	✓	✓ (may be inactive)	✓	✓	✓
for/do	✓	✓	✓	✓	✓	✓
reduction	✓	✓	✓	✓	✓	✓
simd	simdlen(1) (11: honored with hint)	✓ (on host)	✓	✓ (accepted and ignored)	✓	✓ simdlen(1)
atomic	✓	✓	✓	✓	✓	✓
critical	✓	✓	✓	✓	✓	✗
sections	✓	✓	✓	✓	✓	✗
master	✓	✓	✓	✓	✓	✓
single	✓	✓	✓	✓	✓	✓
barrier	✓	✓	✓	✓	✓	✓
declare variant	✓	✓	(support planned for CCE 11)	✗	✓	✓

Figure 1: Feature support of OpenMP directives in different OpenMP Implementations

Thanks to Colleen Bertoni, JaeHyuck Kwack and organizers of Feb 2020 ECP AM OpenMP Vendor BoF

Cray compiler has highest performance in 6/7 C benchmarks (unofficial SPEC results)



LLVM/Clang is 39x slower than Cray on 570.pbt!

OpenMP loop construct placement hurts LLVM/Clang performance

From Christopher Daley NERSC

How is this being addressed?

- LLVM implementations
- OpenMP performance benchmarks

OpenMP Version 5.0

- OpenMP 5.0 introduced powerful features to improve programmability

Task Reductions

Memory Allocators

Detachable Tasks

Initial C11, C++11, C++14 and C++17 support

Dependence Objects

Tools APIs

Complete Fortran 2003 Support, Initial Fortran 2008 Support

Unified Shared Memory

`loop` Construct

Improved Affinity Support

Collapse Non-Rectangular Loops

Task-to-data Affinity

Multi-Level Parallelism

Data Serialization for Offload (Deep Copy)

Meta-Directives

Function Variants

Reverse Offload

Parallel Scan

Interoperability and Usability Enhancements

Improved Task Dependences

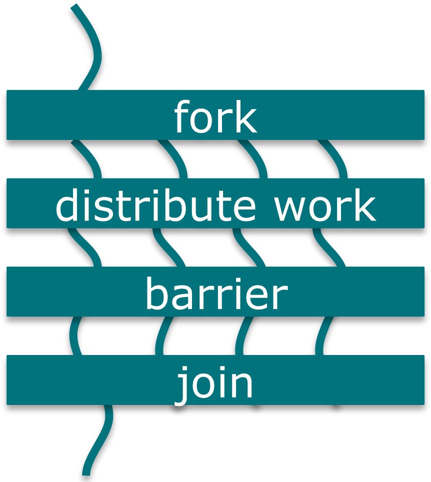
What are the important features are important to applications?

- **Loop construct**
- Unified shared memory support
- Accelerator data management
  - Non-contiguous data mappings
- Memory allocators
- Metadirective and variants
- Tasks
  - Detach
  - Reductions
- Deep copy
- C++ virtual methods

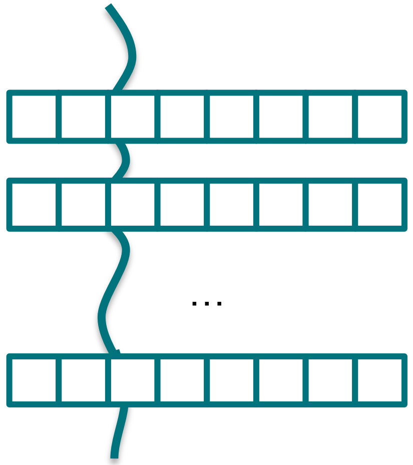
`loop` Construct

- Existing loop constructs are tightly bound to execution model:

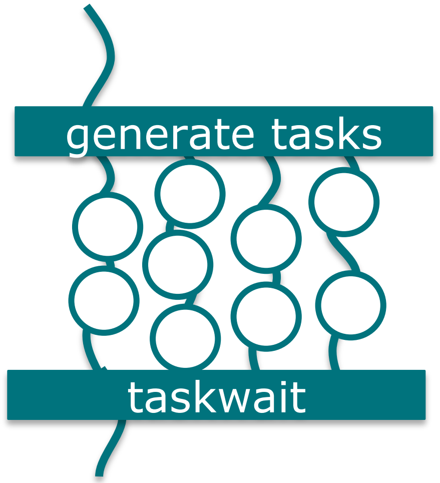
```
#pragma omp for
for (i=0; i<N;++i) {…}
```



```
#pragma omp simd
for (i=0; i<N;++i) {…}
```



```
#pragma omp taskloop
for (i=0; i<N;++i) {…}
```



- The `loop` construct is meant to let the OpenMP implementation pick choose the right parallelization scheme.

How to use OpenMP on Accelerators

```
#pragma omp target teams
#pragma omp distribute
for (i=0; i<N; ++i) {
#pragma omp parallel for
  for (j=0; j<N; ++j) {
    x[j+N*i] *= 2.0;
  }
}
```

- The **target** construct offloads the enclosed code to the accelerator
- The **teams** construct creates a league of teams
- The **distribute** construct distributes the outer loop iterations between the league of teams
- The **parallel for** combined construct creates a thread team for each team and distributes the inner loop iterations to threads

working now

How to use modern OpenMP – Execution Example

```
#pragma omp target
#pragma omp loop bind(thread) \
                 collapse(2)
for (i=0; i<N; ++i) {
  for (j=0; j<N; ++j) {
    x[j+N*i] *= 2.0;
  }
}
```

- The **target** construct offloads the enclosed code to the accelerator
- The **loop** construct allows concurrent execution of the associated loops

working soon

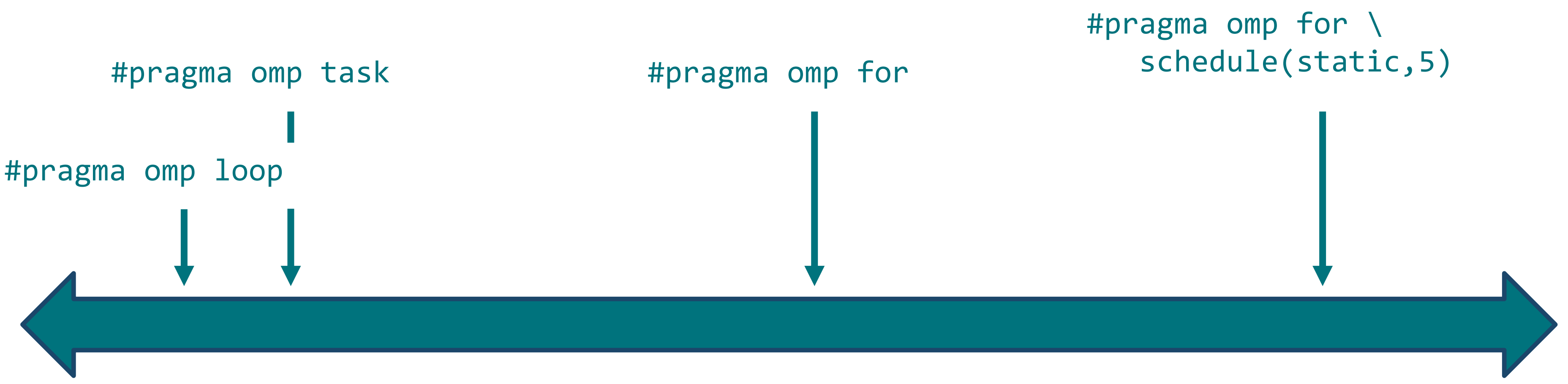
How to use modern OpenMP – Execution Example

```
#pragma omp target teams
#pragma omp loop bind(teams)
for (i=0; i<N; ++i) {
#pragma omp loop bind(thread)
  for (j=0; j<N; ++j) {
    x[j+N*i] *= 2.0;
  }
}
```

- The **target** construct offloads the enclosed code to the accelerator
- The **teams** construct creates a league of teams
- The **loop** construct allows concurrent execution of the associated loops, iterations are "logically" spread across the OpenMP threads in the binding thread set

working very soon

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

What are the important features are important to applications?

- Loop construct
- **Unified shared memory support**
- Accelerator data management
  - Non-contiguous data mappings
- Memory allocators
- Metadirective and variants
- Tasks
  - Detach
  - Reductions
- Deep copy
- C++ virtual methods

Unified Virtual Memory Support

- Single address space over CPU and GPU memories
- Data migrated between CPU and GPU memories transparently to the application - no need to explicitly copy data

```
#pragma omp requires unified_shared_memory
for (k=0; k < NTIMES; k++)
{
// No data directive needed for pointers a, b, c
#pragma omp target teams distribute parallel for
  for (j=0; j<ARRAY_SIZE; j++) {
    a[j] = b[j] + scalar * c[j];
  }
}
```

What are the important features are important to applications?

- Loop construct
- Unified shared memory support
- Accelerator data management
  - **Non-contiguous data mappings**
- Memory allocators
- Metadirective and variants
- Tasks
  - Detach
  - Reductions
- Deep copy
- C++ virtual methods

Non-contiguous data updates and mappings

```
allocate( a(nx, ny) )
!$OMP TARGET DATA MAP(to: a(1:nx/2, 1:ny) )
…
!$OMP TARGET TEAMS DISTRIBUTE
    !  a(1:nx/2, 1:ny) = a(1:nx/2, 1:ny)/nx
!$OMP END TARGET TEAMS DISTRIBUTE
…
!$OMP END TARGET DATA
```

What are the important features are important to applications?

- Loop construct
- Unified shared memory support
- Accelerator data management
  - Non-contiguous data mappings
- **Memory allocators**
- Metadirective and variants
- Tasks
  - Detach
  - Reductions
- Deep copy
- C++ virtual methods

How to use modern OpenMP – Data Placement

```
#pragma omp target teams dist...
{ double Scratchpad[PartitionSize];
   pragma omp allocate(Scratchpad) \
     allocator(omp_pteam_mem_alloc)
}
// OR
double Scratchpad[PartitionSize];
#pragma omp target teams dist... \
     private(Scratchpad)\
     allocator(omp_pteam_mem_alloc)
{ // Do stuff
}
```

- The **allocate** directive allows to place variables in different memory regions, e.g., omp_pteam_mem_alloc will put variables into "shared GPU memory"
- The **omp_alloc** runtime call allocates memory dynamically using a specified allocator, e.g., omp_pteam_mem_alloc

Example: Using Memory Allocators

```
void allocator_example(omp_allocator_t *my_allocator) {
    int a[M], b[N], c;
    #pragma omp allocate(a) allocator(omp_high_bw_mem_alloc)
    #pragma omp allocate(b) // controlled by OMP_ALLOCATOR and/or omp_set_default_allocator
    double *p = (double *) omp_alloc(N*M*sizeof(*p), my_allocator);

    #pragma omp parallel private(a) allocate(my_allocator:a)
    {
        some_parallel_code();
    }

    #pragma omp target firstprivate(c) allocate(omp_const_mem_alloc:c)  // on target; must be compile-time expr
    {
         #pragma omp parallel private(a) allocate(omp_high_bw_mem_alloc:a)
         {
             some_other_parallel_code();
          }
    }

    omp_free(p);
}
```

What are the important features are important to applications?

- Loop construct
- Unified shared memory support
- Accelerator data management
  - Non-contiguous data mappings
- Memory allocators
- **Metadirective and variants**
- Tasks
  - Detach
  - Reductions
- Deep copy
- C++ virtual methods

Metadirective

```
#pragma omp target teams
{
#pragma omp metadirective \
  when(device={kind(nohost)}: distribute parallel for) \
  default (parallel for)
 for(int i=0; i<N; i++)
     C[i] = A[i]+B[i];
}
```

Begin declare variant

```
// Nvidia
#pragma omp begin declare variant match(device={arch(nvptx)}, \
                                  implementation={score(1):vendor(llvm,ibm)})
float fast_sqrt(float __x) { return __nv_sqrt(__x); }
#pragma omp end declare variant
// Intel
#pragma omp begin declare variant match(device={arch(haswell)}, \
                                  implementation={score(1):vendor(intel)})
float fast_sqrt(float __x) { return intel_asm_sqrt(__x); }
#pragma omp end declare variant
// Default
float fast_sqrt(float __x) { return slow_sqrt(__x); }
```

What are the important features are important to applications?

- Loop construct
- Unified shared memory support
- Accelerator data management
  - Non-contiguous data mappings
- Memory allocators
- Metadirective and variants
- Tasks
  - **Detach**
  - Reductions
- Deep copy
- C++ virtual methods

What are the important features are important to applications?

- Loop construct
- Unified shared memory support
- Accelerator data management
  - Non-contiguous data mappings
- Memory allocators
- Metadirective and variants
- Tasks
  - Detach
  - **Reductions**
- Deep copy
- C++ virtual methods

Task Reductions

- Task reductions extend traditional reductions to arbitrary task graphs
- Extend the existing `task` and `taskgroup` constructs
- Also work with the `taskloop` construct

```
int res = 0;
node_t* node = NULL;
...
#pragma omp parallel
{
   #pragma omp single
   {
      #pragma omp taskgroup task_reduction(+: res)
      {
         while (node) {
            #pragma omp task in_reduction(+: res) \
                             firstprivate(node)
            {
               res += node->value;
            }
            node = node->next;
         }
      }
   }
}
```

What are the important features are important to applications?

- Loop construct
- Unified shared memory support
- Accelerator data management
  - Non-contiguous data mappings
- Memory allocators
- Metadirective and variants
- Tasks
  - Detach
  - Reductions
- **Deep copy**
- C++ virtual methods
- Interoperability with GPU streams

OpenMP 5.0 Improves Using Devices: Deep Copy Support

- Not all devices support shared memory so requiring it makes a program less portable
- Painstaking care was required to map complex data before 5.0
- OpenMP 5.0 adds deep copy support so that programmer can ensure that compiler correctly maps complex (pointer-based) data

```
typedef struct mypoints {
    int len;
    double *needed_data;
    double useless_data[500000];
} mypoints_t;

// no declare target needed
int do_something_with_p(mypoints_t *p);

#pragma omp declare mapper(mypoints_t v)\
        map(v.len, v.needed_data,       \
            v.needed_data[0:v.len])

mypoints_t * p = create_array_of_mypoints_t(N);

#pragma omp target map(p[:N])
{
    do_something_with_p(p);
}
```

What are the important features are important to applications?

- Loop construct
- Unified shared memory support
- Accelerator data management
  - Non-contiguous data mappings
- Memory allocators
- Metadirective and variants
- Tasks
  - Detach
  - Reductions
- Deep copy
- **C++ virtual methods**
- Interoperability with GPU streams

Classes with virtual methods

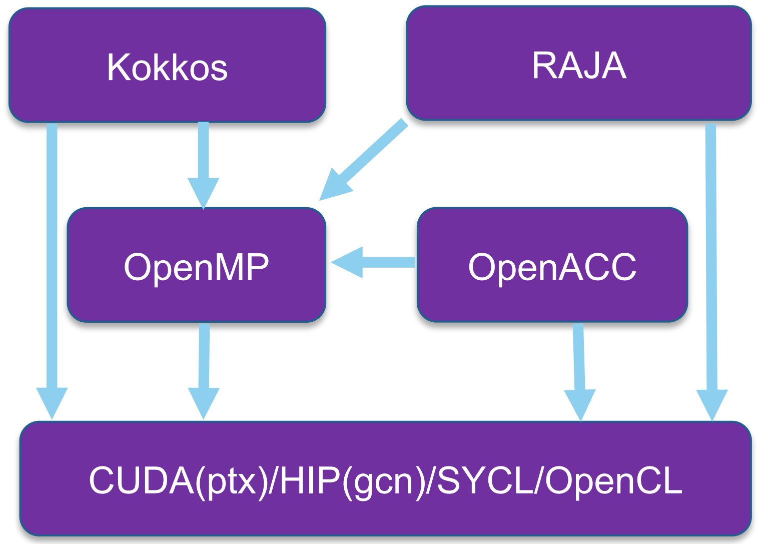
```
class Base {
  virtual void something() = 0;
  virtual void mapSelf() = 0;
}
class Derived : public Base {
  void something() override { /* do logic */ }
  void mapSelf() override {
 #pragma omp target enter data map(to:this[0])
  }
}
```

```
void foo() {
  Derived d;
  d.mapSelf();
  bar(&d);
}
void bar(Base *b) {
  #pragma omp target
  b->something();
}
```

OpenMP 5.0 will support other C++ accelerator frameworks

- Number of related technologies: Kokkos, RAJA, OpenACC, CUDA/HIP, SYCL
- Goal is to deliver enhanced OpenMP to address increasing heterogeneity and complexity of systems (e.g. accelerator offloading, tasks)

	CUDA / HIP	Kokkos	OpenACC	OpenMP 5.0	RAJA	SYCL
Languages	C/C++	C/C++	C/C++/ Fortran	C/C++/ Fortran	C/C++	C/C++
Prog. Style		Template Meta-programming, C++11 lambdas	Directive-based	Directive-based	C++11 lambdas	Template Meta-programming, C++11 lambdas
Parallelism	SIMT	OpenMP, Pthreads, CUDA, HIP	SIMD, Fork-Join, CUDA, HIP	SPMD, SIMD, Tasks, Fork-Join, CUDA, HIP	OpenMP, CUDA, HIP,	OpenCL
Licensing/ Accessibility	Proprietary	Open-sourced	Few compilers Not on all arch.	Open-sourced	Open-sourced	Royalty-Free
Abstraction Level	Low	Medium	High	High	Medium	Medium



Service Layers

HIP targets are work-in-progress activities

What are the important features are important to applications?

- Loop construct
- Unified shared memory support
- Accelerator data management
  - Non-contiguous data mappings
- Memory allocators
- Metadirective and variants
- Tasks
  - Detach
  - Reductions
- Deep copy
- C++ virtual methods
- **Interoperability with GPU streams**

Interop: get stream/queue/etc.

```
omp_interop_t o = OMP_INTEROP_NONE;  intptr_t type;
#pragma omp interop tasksync init obj(o) depend(inout: a)
omp_get_interop_property(o, OMP_INTEROP_TYPE, &type);
if (type == OMP_INTERFACE_CUDA) {
  cudaStream_t s;
  omp_get_interop_property(o, OMP_INTEROP_TASKSYNC, &s);
  cublasSetStream(s);
  call_cublas_async_stuff();
} else  {
  // handle other cases
}
#pragma omp interop tasksync destroy obj(o) depend(inout: a)
```

Tuning OpenMP target : Thread Blocking Effects

```
#pragma omp target
#pragma omp teams distribute num_teams(nblocks) thread_limit(nthreads)
for(int ss=0; ss<nblocks; ss++) {
#pragma omp parallel for
    for(int tt=0; tt<nthreads; tt++) {
      auto tmp = eval(ss*nthreads+tt,expr);
      vstream(me[ss*nthreads+tt],tmp);
    }
}
```

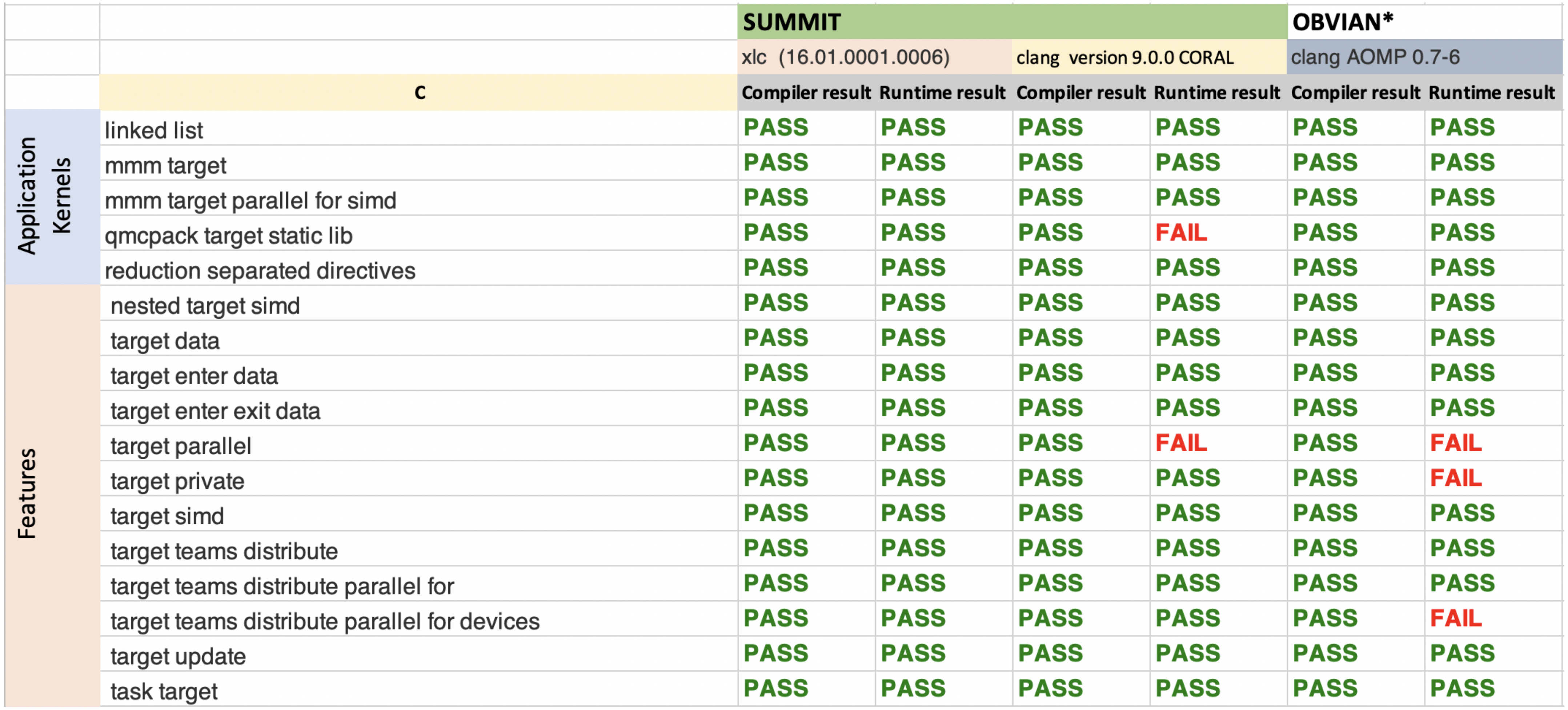
Code from GridMini in ECP's Lattice QCD

nblocks	nthreads	GB/s
Default	Default	240
65536	8	162
32768	32	252
640	128	289
4096	256	306

Functionality of OpenMP C Implementations Based on SOLLVE's V&V

Results for tests based on QMCPack

→ Feature support in OpenMP runtimes needs to be improved.



		SUMMIT				OBVIAN*	
		xlc (16.01.0001.0006)		clang version 9.0.0 CORAL		clang AOMP 0.7-6	
	C	Compiler result	Runtime result	Compiler result	Runtime result	Compiler result	Runtime result
Application Kernels	linked list	PASS	PASS	PASS	PASS	PASS	PASS
	mmm target	PASS	PASS	PASS	PASS	PASS	PASS
	mmm target parallel for simd	PASS	PASS	PASS	PASS	PASS	PASS
	qmcpack target static lib	PASS	PASS	PASS	FAIL	PASS	PASS
	reduction separated directives	PASS	PASS	PASS	PASS	PASS	PASS
Features	nested target simd	PASS	PASS	PASS	PASS	PASS	PASS
	target data	PASS	PASS	PASS	PASS	PASS	PASS
	target enter data	PASS	PASS	PASS	PASS	PASS	PASS
	target enter exit data	PASS	PASS	PASS	PASS	PASS	PASS
	target parallel	PASS	PASS	PASS	FAIL	PASS	FAIL
	target private	PASS	PASS	PASS	PASS	PASS	FAIL
	target simd	PASS	PASS	PASS	PASS	PASS	PASS
	target teams distribute	PASS	PASS	PASS	PASS	PASS	PASS
	target teams distribute parallel for	PASS	PASS	PASS	PASS	PASS	PASS
	target teams distribute parallel for devices	PASS	PASS	PASS	PASS	PASS	FAIL
	target update	PASS	PASS	PASS	PASS	PASS	PASS
	task target	PASS	PASS	PASS	PASS	PASS	PASS

* Obivan is a HPC cluster @ UDel

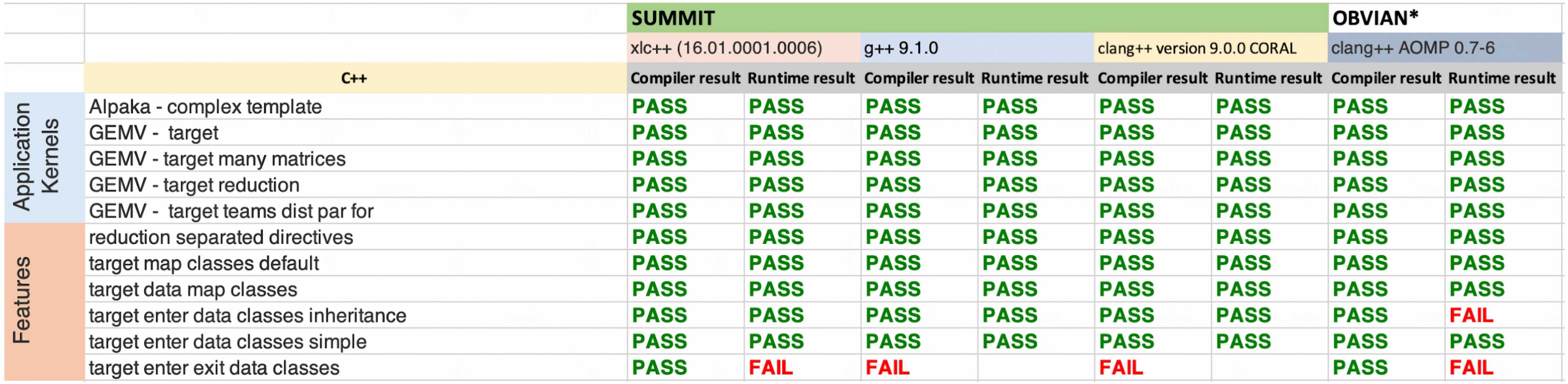
Figure 1: Table of test results for OpenMP C Implementations

Courtesy Swaroop Pophale and David Bernholdt

For further information, e.g., understanding of failures, visit: https://crpl.cis.udel.edu/ompvvsollve/results/

Functionality of OpenMP C++ Implementations

Tests based on GEMV from QMCPack.



		SUMMIT						OBVIAN*	
		xlc++ (16.01.0001.0006)		g++ 9.1.0		clang++ version 9.0.0 CORAL		clang++ AOMP 0.7-6	
	C++	Compiler result	Runtime result	Compiler result	Runtime result	Compiler result	Runtime result	Compiler result	Runtime result
Application Kernels	Alpaka - complex template	PASS	PASS	PASS	PASS	PASS	PASS	PASS	PASS
	GEMV - target	PASS	PASS	PASS	PASS	PASS	PASS	PASS	PASS
	GEMV - target many matrices	PASS	PASS	PASS	PASS	PASS	PASS	PASS	PASS
	GEMV - target reduction	PASS	PASS	PASS	PASS	PASS	PASS	PASS	PASS
	GEMV - target teams dist par for	PASS	PASS	PASS	PASS	PASS	PASS	PASS	PASS
Features	reduction separated directives	PASS	PASS	PASS	PASS	PASS	PASS	PASS	PASS
	target map classes default	PASS	PASS	PASS	PASS	PASS	PASS	PASS	PASS
	target data map classes	PASS	PASS	PASS	PASS	PASS	PASS	PASS	PASS
	target enter data classes inheritance	PASS	PASS	PASS	PASS	PASS	PASS	PASS	FAIL
	target enter data classes simple	PASS	PASS	PASS	PASS	PASS	PASS	PASS	PASS
	target enter exit data classes	PASS	FAIL	FAIL		FAIL		PASS	FAIL

Figure 3: Table of test results for OpenMP C++ Implementations.

→ Most OpenMP offload features in all OpenMP implementations work.

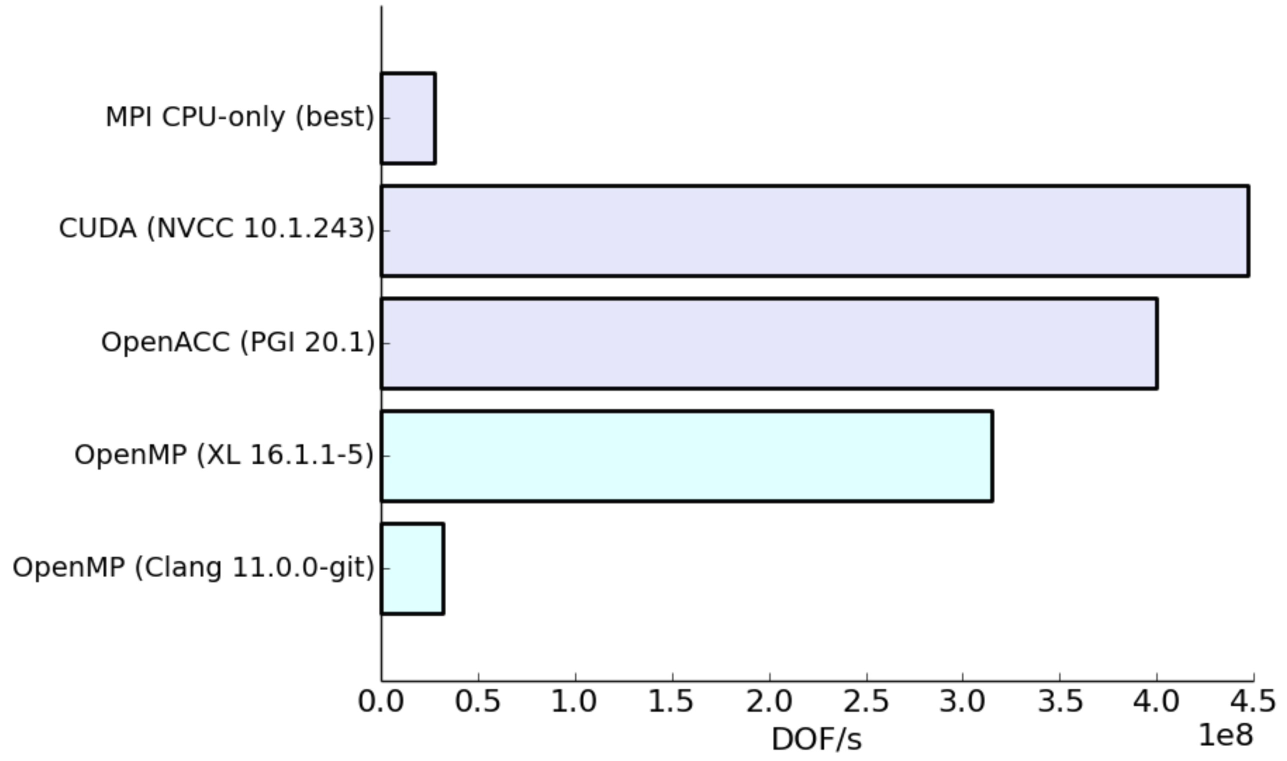
→ target enter exit data isn't supported properly across any OpenMP implementations.

For further information, e.g., understanding of failures, visit: https://crpl.cis.udel.edu/ompvvsollve/results/

Courtesy Swaroop Pophale (ORNL) and David Bernholdt (ORNL)

EXASCALE COMPUTING PROJECT

OpenMP Offload in HPGMG



Higher is better

Figure 5: Performance of HPGMG with different Implementations.

- HPGMG is a DOE benchmark which may be included in the SPEC HPC 2020 benchmark suite
- The plot compares CUDA, OpenACC, and OpenMP performance on 1 socket of the Summit supercomputer using 3 MPI ranks and 3 GPUs for the Unified Memory version of HPGMG run on Summit.
- Results for the explicit data management version of HPGMG will be shown at a later date when the IBM compiler fixes a reported bug and the CCE compiler supports OpenMP pointer attachment

→ IBM xl's OpenMP offload performance shown with HPGMG is encouraging

Performance of SU3 LQCD Benchmark with OpenMP Libraries

- Developed benchmark code representative of applications in ECP Application Project LQCD. The code is at https://bitbucket.org/dwdoerf/su3_bench.
- Ran with three different OpenMP libraries, with CUDA and with PGI's OpenACC.
- Note that the peak GF/s in plots refers to the theoretical floating point performance based on the Arithmetic Intensity of the offloaded kernel. A Volta GPU has a peak GF/s of 7800 GF/s for kernels which are not bound by memory bandwidth.

```
#pragma omp target teams distribute
for(int i=0; i<1048576; ++i) {

#pragma omp parallel for collapse(3)
  for(int j=0; j<4; ++j) {
    for(int k=0; k<3; k++) {
      for(int l=0; l<3; l++) {
        Complx cc;
        for(int m=0; m<3; m++) {
          cc += d_a[i].link[j].e[k][m] * d_b[j].e[m][l];
        }
        d_c[i].link[j].e[k][l] = cc;
      }
    }
  }
}
```

Use teams parallelism for the ~1 million sites

Use thread parallelism for the matrices associated with the 4 "links" per site

Listing 1: GPU Computation region of SU3 benchmark

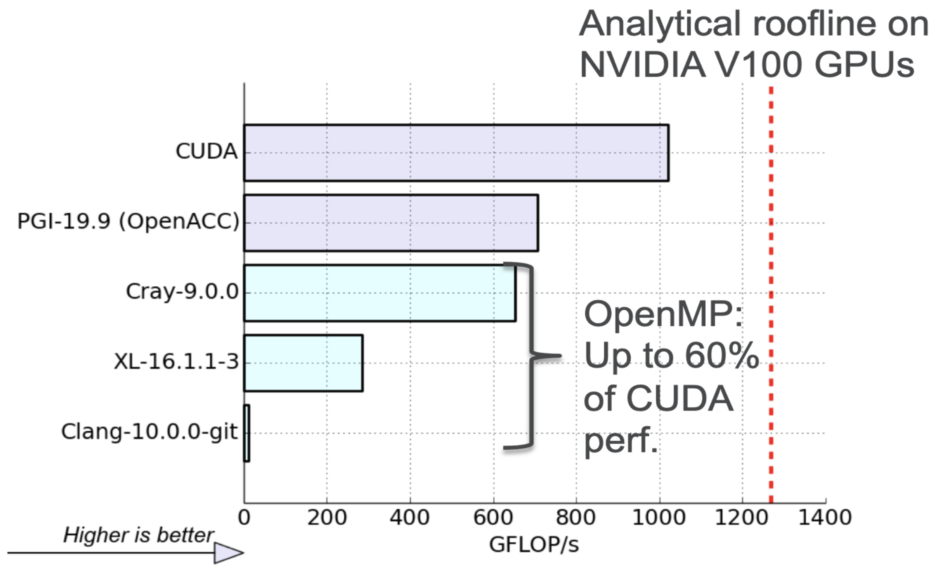


Figure 7: Performance of SU3 with different Implementations

→ Results for SU3 benchmark run on NVIDIA Tesla V100 with different OpenMP libraries (left plot) shows how clang provides best performance of 640 GFLOP/s
→ The performance of clang OpenMP is 3% of peak and is very low compared to other Compilers. However, manual SPMDization of code can reduce implicit memory flushes and increases performance to 401 GFLOP/s.
→ Ongoing changes in clang OpenMP can provide better performance over the other OpenMP vendor libraries.

Thanks to Chris Daley and Doug Doerfler at NERSC.

Conclusions

- On ECP Systems (particularly Summit) compilers are ready for device offload. Fundamental features are available. Still, tests could be improved to handle real-world data structures with pointers.
- Applications can move towards OpenMP offload using clang/LLVM OpenMP as it has support for many new OpenMP 5.0 offload features.
- IBM's support for OpenMP offload for C is mature. Could be improved for Fortran.
- Performance of IBM OpenMP offload is 70% of CUDA performance in HPGMG.
- While QMCPack currently relies on IBM xl OpenMP for offload, it's recently (a) shown to potentially have good performance from using LLVM clang OpenMP offload support and (b) works with other vendor compilers
- The SPEC HPC 2020 benchmark suite is under active development. The benchmarks will be pruned over the next few months based on benchmark readiness and formally meeting benchmark suite requirements (no new benchmarks will be considered at this stage).