SC18 OpenMP® BoF Report (BoF 109)

Jim Cownie, Michael Klemm 28 November 2018

Summary

The OpenMP BoF was held on Wednesday 15 November 5:15pm-7:00pm.

There were over 100 attendees, a slight drop from last year; maybe there were a larger number of interesting BoFs in the same slot this year.

Oral and Twitter feedback was positive, for instance:-

> "I certainly enjoyed the BoF. ... I think the BoF structure was a good mix of structure and chaos, apt for the audience. Keep up the great work."[1]

SC no longer seems to be requesting online BoF feedback, so we do not have any to report.

A large component (approaching 75%) of the BoF involved audience questions and discussion.

We believe that the BoF achieved the aims set out by the SC organizers of education, community building and audience interaction.

Technical Presentations

Bronis R. de Supinski (chair of the language committee) presented details of the achievements of the committee which led to the release of the OpenMP 5.0 specification the week before SC18.

Leading technical contributors to the OpenMP standard then outlined key developments in their areas. Topics presented covered features for improved support for heterogeneous computing and accelerators, meta-directives to allow easier choice of optimized code for different contexts, the new, descriptive, `loop` directive, improved memory allocation control using "allocators", and improvements to the capabilities of tasking such as reductions, scans, and a new task dependency type.

Technical Discussion

Questions included:-

- How can we transition from OpenMP 4.5 to OpenMP 5.0?
  OpenMP is backwards compatible, so the transition is transparent. You do not have to change your code. (With the minor exception that the change in OpenMP 5.0 which makes the default dynamic schedule nonmonotonic, which might break user code. For details see Jim Cownie's booth presentation at https://www.openmp.org/wp-content/uploads/SC18-BoothTalks-Cownie.pdf)
- How do we move data from one GPU to another GPU?
  This is something for consideration in OpenMP 6.0.
- What is the difference between descriptive and prescriptive?

[1] https://twitter.com/tkphd/status/1063228847825526784

Prescriptive directives define a specific behaviour. Descriptive directives pass information about code semantics to the compiler to allow it to optimize without specifying how it might choose to do that.

OpenMP State of the Union

Michael Klemm, OpenMP CEO, gave a short presentation on the vision and schedule for OpenMP 6, and the changes in OpenMP membership structure which make it relatively cheap for academic institutions to join the OpenMP Architecture Review Board (ARB). (As ever, the critical commitment here is not the money, but the time to participate). He also showed the list of OpenMP events planned for next year, with conferences in Edinburgh (UK), and Auckland (NZ).

ARB Panel

We finished with a series of questions to members of the ARB.

The questions included:

- What is the compiler uptake for OpenMP 5.0?
  Please have a look at the compiler page on the OpenMP website[2]. This page is kept up to date. There is also a page where you can compare compilers with respect with device offload support[3].
- Will there be OpenMP 5.0 examples available?
  We are working on an OpenMP 5.0 examples document. As soon as it is finished it will be announced and published at openmp.org .
- What is the status of the PGI compiler?
  The PGI compiler will support a subset of the OpenMP 5.0 features.
- How can we make it easier for new users to take up OpenMP?
  The OpenMP website has a section with resources for new users (https://www.openmp.org/resources/). This section includes a link to videos of a beginners' course on OpenMP, to example guides, and to books on OpenMP. Furthermore courses on OpenMP are regularly given at universities and supercomputer centres.
- Can OpenMP help more with memory layout (e.g. array of structures to structure of array transformations)?
  This seems generally hard to impossible, since it affects deep properties assumed by the underlying language (at least in C/C++).
- Why are there so many pages in the OpenMP API specification? Is there a maximum number of pages?
  The standard has to be precise; that requires consideration of many corner cases, which leads to prolixity. The size of the standard is bounded by available on-line storage!

Presentations

[2] https://www.openmp.org/resources/openmp-compilers-tools
[3] https://crpl.cis.udel.edu/ompvvsollve/

OpenMP®

Enabling HPC since 1997

SC18 Birds of a Feather

Schedule

Overview of OpenMP 5.0 (21 minutes)

Bronis R. de Supinski, James Beyer, Christian Terboven, Stephen Olivier

OpenMP 5.0 technical questions/discussion (35 minutes)

Get your questions ready! This is your chance to ask the experts.

OpenMP's Future Vision (4 minutes)

Michael Klemm

Architecture Review Board Questions/Discussion (rest of the time)

Greg Rodgers, Bronis R. de Supinski, James Beyer, Michael Klemm, Barbara Chapman, Sunita Chandrasekaran

An Overview of Current and Future OpenMP® Directions

Bronis R. de Supinski
Chair
OpenMP Language Committee

November 14, 2018

LLNL-PRES-760847
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 5.0 was ratified last week

- Addressed several major open issues for OpenMP
- Did not break (most?) existing code
  - → One possible issue: `nonmonotonic` default
- Includes 293 passed tickets
  - → Focused on major tickets initially
  - → Built on three comment drafts (TR4, TR6 and TR7)
  - → All tickets after TR7 arose from final quality control pass

Major new features in OpenMP 5.0

- Significant extensions to improve usability
  - → OpenMP contexts, `metadirective` and `declare variant` — **James**
  - → Addition of `requires` directive, including support for unified shared memory — **James**
  - → Memory allocators and support for deep memory hierarchies — **Christian**
  - → Descriptive `loop` construct — **James**
  - → Ability to quiesce OpenMP threads
  - → Support to print/inspect affinity state
  - → Release/acquire semantics added to memory model
  - → Support for C/C++ array shaping
- First (OMPT) and third (OMPD) party tool support

Major new features in OpenMP 5.0

- Some significant extensions to existing functionality
  - → Verbosity reducing changes such as implicit `declare target` directives — **James**
  - → User defined mappers provide deep copy support for map clauses — **James**
  - → Support for reverse offload — **James**
  - → Support for task reductions , including on `taskloop` construct, task affinity, new dependence types, depend objects and detachable tasks — **Stephen**
  - → Allows `teams` construct outside of `target` (i.e., on host)
  - → Supports collapse of non-rectangular loops
  - → Scan extension of reductions — **Stephen**
- Major advances for base language normative references
  - → Completed support for Fortran 2003
  - → Added Fortran 2008, C11, C++11, C++14 and C++17

Clarifications and minor enhancements

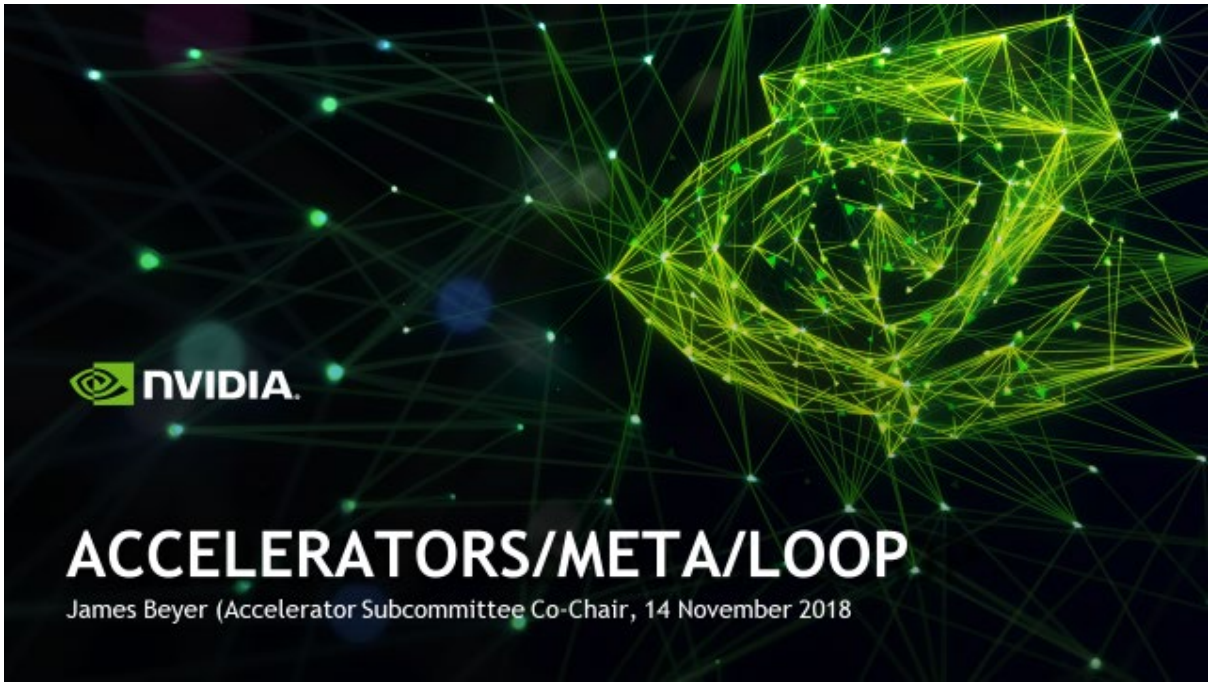
- Supports collapse of imperfectly nested loops
- Supports `!=` on C/C++ loops
- Adds `conditional` modifier to `lastprivate`
- Support use of any C/C++ *lvalue* in `depend` clauses
- Permits `declare target` on C++ classes with virtual members
- Clarification of `declare target` C++ initializations
- Adds `task` modifier on many `reduction` clauses
- Adds `depend` clause to `taskwait` construct

OpenMP 5.1 will be released in November 2020

- Proceedings of the IEEE article on vision: "The Ongoing Evolution of OpenMP"
  - → Broadly support on-node performant, portable parallelism
  - → OpenMP 5.0 fits within that vision
  - → OpenMP 5.1 will refine how OpenMP 5.0 realizes it
  - → OpenMP 6.0 will be a major step to further realizing it
- Expect issues from detailed implementation and use of OpenMP 5.0, which is big and will require time to implement
- Guarantee OpenMP 5.1 will not break existing code
- Clarifications, corrections and maybe some small extensions
  - → Improved native device support (e.g., CUDA streams)
  - → May add `taskloop` dependences
  - → Other small extensions must entail small implementation burden

OpenMP 6.0 will be released in November 2023

- Deeper support for descriptive and prescriptive control
- More support for memory affinity and complex hierarchies
- Support for pipelining, other computation/data associations
- Continued improvements to device support
  - → Extensions of deep copy support (serialize/deserialize f'ns)
- Task-only, unshackled or free-agent threads
- Event-driven parallelism
- Completing support for new normative references
- 38 5.1 tickets already; 2 tickets already deferred to 6.0



ACCELERATORS

Subcommittee report

- **Requires** - reverse_offload, unified_shared_memory*, atomic_default_mem_order(...)
- **Implicit declare target**
- **Reverse offload** - ancestor only
- **User defined mappers** - declare mapper(*T* v) map(tofrom:v)
- **Teams outside of target** - non-offload targets can benefit from this as well!
- **Pointer attachment** - Pointers on the device get translated initial value
- **Defaultmap** - more classes plus NONE
- OMP_TARGET_OFFLOAD = MANDATORY | DISABLED | DEFAULT
- **Support for C++ classes with virtual members**

META DIRECTIVE

The directive directive

- Started life many years, at least 5, ago as the super_if
- Especially important now that we have target constructs
- A metadirective is a directive that can specify multiple directive variants of which one may be conditionally selected to replace the metadirective based on the enclosing OpenMP context.

```
#pragma omp metadirective when( device={kind(gpu)}: target teams\
                                                    distribute )\
                                    default( parallel for ))
for (i= lb; i< ub; i++)
   v3[i] = v1[i] * v2[i];
   ...
```

META DIRECTIVE
The directive directive

- Started life many years, at least 5, ago as the super_if
- Especially important now that we have target constructs
- A metadirective is a directive that can specify multiple directive variants of which one may be conditionally selected to replace the metadirective based on the enclosing OpenMP context.

```
#pragma omp target teams distribute
for (i= lb; i< ub; i++)
   v3[i] = v1[i] * v2[i];
   ...
```

When compiling for a gpu

META DIRECTIVE
The directive directive

- Started life many years, at least 5, ago as the super_if
- Especially important now that we have target constructs
- A metadirective is a directive that can specify multiple directive variants of which one may be conditionally selected to replace the metadirective based on the enclosing OpenMP context.

When compiling for a anything that is not a gpu!

```
#pragma omp parallel for
for (i= lb; i< ub; i++)
   v3[i] = v1[i] * v2[i];
   ...
```

META DIRECTIVE

The directive directive

- Started life many years, at least 5, ago as the super_if
- Especially important now that we have target constructs
- A metadirective is a directive that can specify multiple directive variants of which one may be conditionally selected to replace the metadirective based on the enclosing OpenMP context.

When compiling for a both

```
#pragma omp target teams distribute
for (i= lb; i< ub; i++)
   v3[i] = v1[i] * v2[i];
   ...
```

```
#pragma omp parallel for
for (i= lb; i< ub; i++)
   v3[i] = v1[i] * v2[i];
   ...
```


DECLARE VARIANT DIRECTIVE

- The **declare variant** directive declares a specialized variant of a base function and specifies the context in which that specialized variant is used. The **declare variant** directive is a declarative directive.
- Combines proposed extensions for DECLARE SIMD and DECLARE TARGET into one that works anywhere.
- Reuse context selector mechanism used by meta directive

DECLARE VARIANT DIRECTIVE

This may not be the supported name!

```
#pragma omp declare variant( int important_stuff(int x) ) \
                    match( context={target,simd} device={arch(nvptx)} )
int important_stuff_nvidia(int x){
      /* Specialized code for NVIDIA target */
}

#pragma omp declare variant( int important_stuff(int x) ) \
                    match( context={target, simd(simdlen(4))}, device={isa(avx2)}
)
 __m256i _mm256_epi32_important_stuff(__m256i x) {
     /* Specialized code for simdloop called on an AVX2 processor */
}
...
#pragma omp target ... SIMD
{
   ...
   int y =important_stuff(x);
```

When compiling for NVIDA GPUS the compiler translates this to important_stuff_nvidia(x);

DECLARE VARIANT DIRECTIVE

```
#pragma omp declare variant( int important_stuff(int x) ) \
                    match( context={target,simd} device={arch(nvptx)} )
int important_stuff_nvidia(int x){
      /* Specialized code for NVIDIA target */
}

#pragma omp declare variant( int important_stuff(int x) ) \
                    match( context={target, simd(simdlen(4))}, device={isa(avx2)}
)
 __m256i _mm256_epi32_important_stuff(__m256i x) {
     /* Specialized code for simdloop called on an AVX2 processor */
}
...
#pragma omp target ... SIMD
{
   ...
   int y =important_stuff(x);
```

When compiling for AVX2 the compiler translates this to __m256i _mm256_epi32_important_stuff (x);

#PRAGMA OMP LOOP

- Introduced as #PRAGMA OMP CONCURRENT in TR6
  - A **loop** construct specifies that the iterations of the associated loops may execute concurrently and permits the encountering thread(s) to execute the loop accordingly.
- Why?
  - It's descriptive!
  - Enables the compiler to make certain complex optimizations that would require dependency analysis
- Limitations
  - Not a complete replacement for do/for, yet!
  - User responsible for bindings, teams, parallel, thread, of orphaned constructs.

OMP LOOP EXAMPLE

Syntax:

```
#pragma omp teams (or parallel)
{
   #pragma omp loop
   for(int i = 0; i < N; i++) {
      for(int j = 0; j < N; j++) {
      }
   }
}
```

Generate Parallelism

Assert to the compiler that it is safe to parallelize the next loop

OMP LOOP FUTURE

- Replace do and for with a single loop directive
- Clauses tell the compiler how prescriptive the programmer chose to be

DECLARE VARIANT DIRECTIVE

```
#pragma omp declare variant( int important_stuff(int x) ) \
                    match( context={target,simd} device={arch(nvptx)} )
int important_stuff_nvidia(int x){
      /* Specialized code for NVIDIA target */
}

#pragma omp declare variant( int important_stuff(int x) ) \
                    match( context={target, simd(simdlen(4))}, device={isa(avx2)}
)
 __m256i _mm256_epi32_important_stuff(__m256i x) {
     /* Specialized code for simdloop called on an AVX2 processor */
}
...
int y =important_stuff(x);
```


SC18 Dallas, TX | hpc inspires.

OpenMP Enabling HPC since 1997

Memory Management

Christian Terboven

IT Center | High Performance Computing | RWTH AACHEN UNIVERSITY

OpenMP Affinity Subcommittee

2016 - 2018 achievements

- November 2016 at this BoF:
  - →Concepts of Memory Management presented
  - →Release of a TR document promised for early 2017
- November 2017 at this BoF:
  - →Reported on TR5
  - →Overview about functionality expected for OpenMP 5.0
- November 2018: OpenMP 5.0 release w/ Memory Mgmt. ☺

Memory Management

- Allocator := an OpenMP object that fulfills requests to allocate and deallocate storage for program variables
- OpenMP allocators are of type `omp_allocator_handle_t`
- OpenMP 5.0 supports a set of memory allocators
  - → Enables the support of different kinds of memory
- OpenMP 5.0 supports custom allocators
  - → Provides opportunity to request specific properties

OpenMP allocators

- Selection of a certain kind of memory

Allocator name	Storage selection intent
omp_default_mem_alloc	use default storage
omp_large_cap_mem_alloc	use storage with large capacity
omp_const_mem_alloc	use storage optimized for read-only variables
omp_high_bw_mem_alloc	use storage with high bandwidth
omp_low_lat_mem_alloc	use storage with low latency
omp_cgroup_mem_alloc	use storage close to all threads in the contention group of the thread requesting the allocation
omp_pteam_mem_alloc	use storage that is close to all threads in the same parallel region of the thread requesting the allocation
omp_thread_local_mem_alloc	use storage that is close to the thread requesting the allocation

Using OpenMP allocators

- New clause on all constructs with data sharing clauses:
  - → `allocate( [allocator:] list )`
- Allocation:
  - → `omp_alloc(size_t size, omp_allocator_handle_t allocator)`
- Deallocation:
  - → `omp_free(void *ptr, const omp_allocator_handle_t allocator)`
- `allocate` directive: standalone directive for allocation, or declaration of allocation stmt.

OpenMP allocator traits

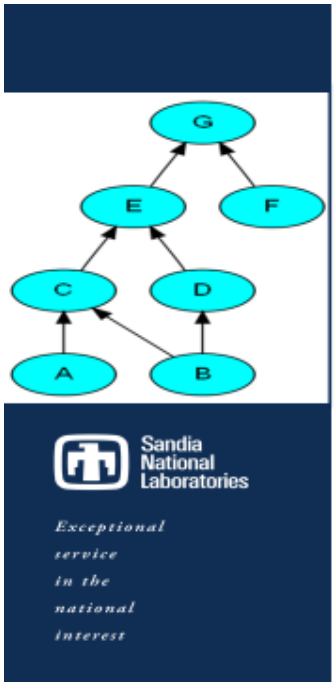
- Allocator traits control the behavior of the allocator

sync_hint	contended, uncontended, serialized, private default: contended
alignment	positive integer value that is a power of two default: 1 byte
access	all, cgroup, pteam, thread default: all
pool_size	positive integer value
fallback	default_mem_fb, null_fb, abort_fb, allocator_fb default: default_mem_fb
fb_data	an allocator handle
pinned	true, false default: false
partition	environment, nearest, blocked, interleaved default: environment

Using OpenMP allocator traits

- Construction of allocators with traits via
  - → `omp_allocator_handle_t omp_init_allocator(omp_memspace_handle_t memspace, int ntraits, const omp_alloctrait_t traits[]);`
  - → Selection of memory space mandatory
  - → Empty traits set: use defaults
- Allocators have to be destroyed with `*_destroy_*`
- Custom allocator can be made default with `omp_set_default_allocator(omp_allocator_handle_t allocator)`

OpenMP Affinity Subcommittee



OpenMP Tasks: New Features in 5.0

SC18 OpenMP BoF

Stephen Olivier
Center for Computing Research
Sandia National Labs (NM)

November 14, 2018

Sandia National Laboratories is a multi-mission laboratory managed and operated by National Technology & 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.

Some 5.0 Tasking Features

- Iterator syntax for clauses and its use in depend clause
  - Expands to multiple values in a range
  - Expected to have more uses in future
- Task affinity based on data locations, similar to dependences
  - Unlike dependences, is a hint and does not impose ordering constraints
  - Can use iterators in this clause also
- The `mutexinoutset` dependence type
  - Enables a set of inout tasks to commute but not execute concurrently
- Allow the depend clause on the `taskwait` construct
- Allow any l-value in the depend clause

More 5.0 Tasking Features

- Depend objects
  - Portable representation of a task dependence
  - New construct to initialize, update, and destroy
  - Can then be supplied to the depend clause
- Detached tasks
  - Decouples completion of a task from completion of its structured block
  - Creates an event handle that can be passed in function calls
  - Calling `omp_fulfill_event` routine on a handle completes the task
  - Enables asynchronous interoperation with other programming models like CUDA and MPI

Reductions and Scans

- Reductions over tasks
  - Scoped using `task_reduction` clause on `taskgroup` construct or `reduction` clause on `taskloop` construct
  - Can also specify a `task` modifier in the `reduction` clause on a `parallel` or worksharing region to reduce over tasks in the region
  - Specify `in_reduction` clause on any `task`, `target` or `taskloop` construct to participate in the reduction
- Bonus: Parallel prefix scan (not a tasking feature)
  - `scan` directive in a loop / loop nest
  - `inscan` modifier on the reduction clause
  - Can specify inclusive or exclusive

Example: Target Task in Reduction

Sandia National Laboratories

```
int x = 0;

#pragma omp taskgroup task_reduction(+:x)
{
   #pragma omp target in_reduction(+:x) nowait   // offload
      ...
   #pragma omp target in_reduction(+:x) nowait   // offload
      ...
   #pragma omp task in_reduction(+:x)            // exec. on host
      ...
}
// combined value of x available at this point
```


OpenMP
Enabling HPC since 1997

OpenMP State of the Union

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.

OpenMP

Membership Structure

- ARB Member
  - Highest membership category
  - Participation in technical discussions and organizational decisions
  - Voting rights on organizational topics
  - Voting rights on technical topics (tickets, TRs, specifications)
- ARB Advisor & ARB Contributor
  - Contribute to technical discussions
  - Voting rights on technical topics (tickets, TRs, specifications)

Your organization can join and influence the direction of OpenMP.
Talk to me or send email to michael.klemm@openmp.org.

OpenMP

OpenMP UK User-group Meeting

OpenMP Users
DEVELOPER CONFERENCE & USER GROUP

- Conference dates:
  - June 4-5
- Location: Edinburgh, UK

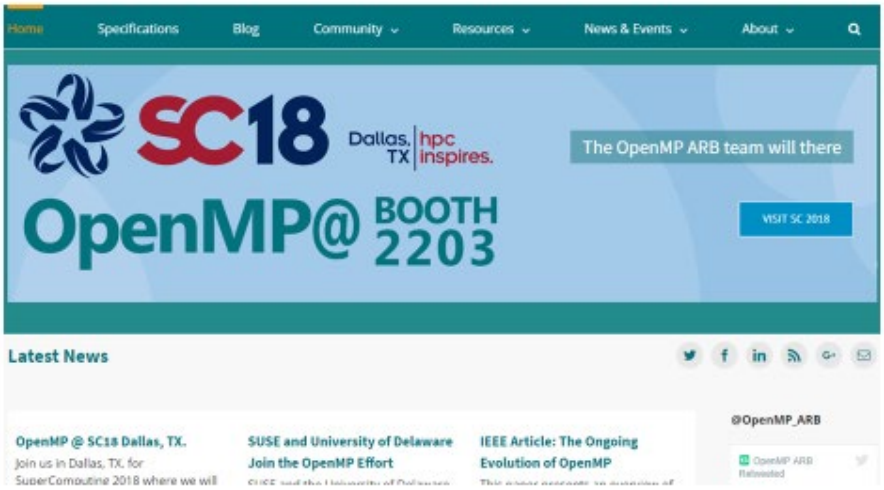
OpenMP

OpenMPCon & IWOMP 2019

OpenMPCon
IWOMP
INTERNATIONAL WORKSHOP

- Conference dates:
  - OpenMPCon: Sep 9-10
  - Tutorials: Sep 11
  - IWOMP: Sep 12-13
- Location: Auckland, NZ

Visit www.openmp.org for Information



ARB Panel Questions

AMD	Greg Rodgers
OpenMP, LLNL	Bronis R. de Supinski (Chair of Language Committee)
NVIDIA	James Beyer
OpenMP, intel	Michael Klemm (OpenMP CEO)
OMPanity	Barbara Chapman
University of Delaware	Sunita Chandrasekaran