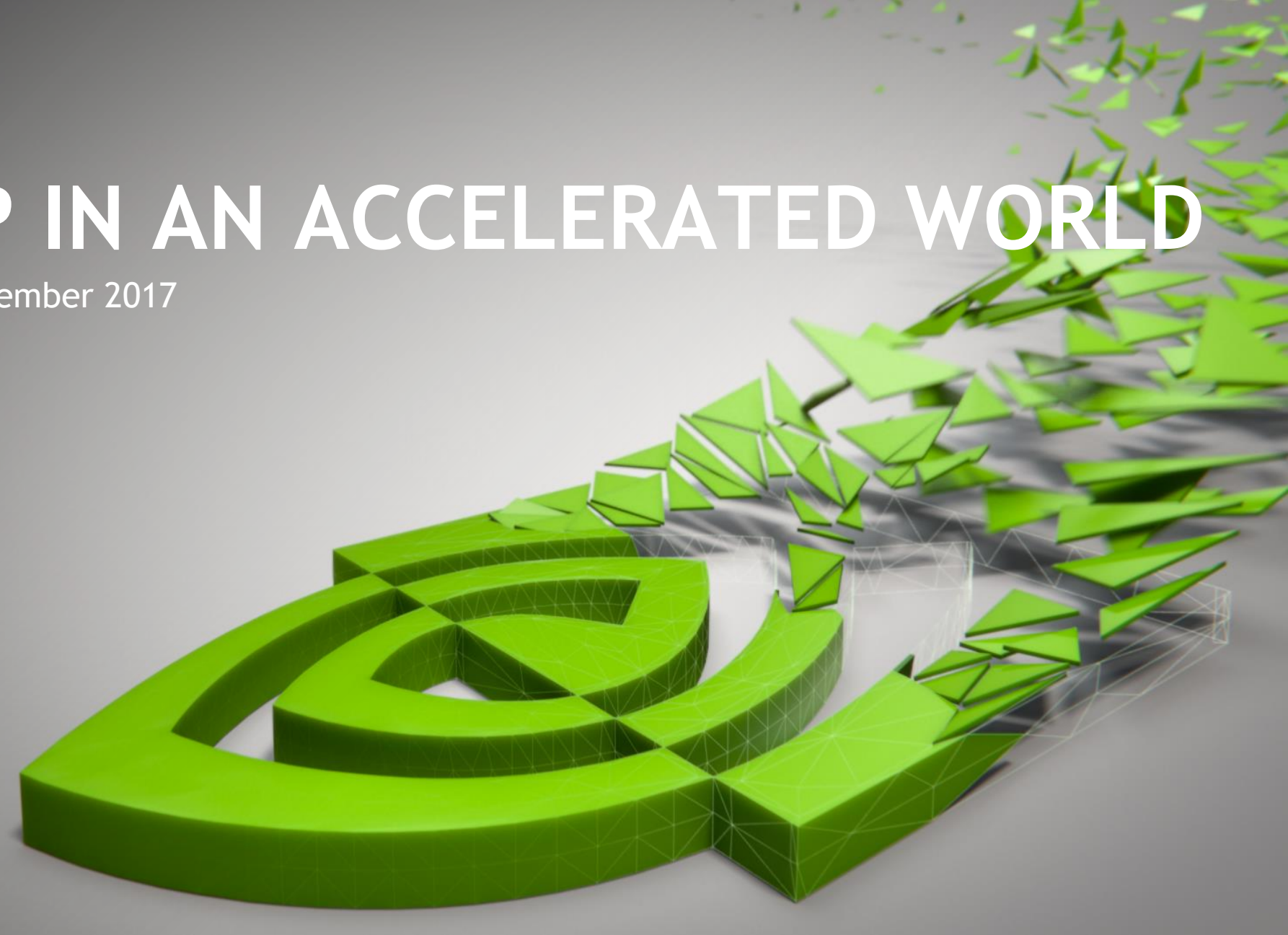
OPENMP IN AN ACCELERATED WORLD

James Beyer, November 2017

NVIDIA



OPENMP ON ACCELERATORS?

Why?

The top 16 machines on the Green 500 use an accelerator![1]

Exascale subcommittee report[2] “challenges”

Reducing Power Requirements
Coping with Run-time errors
Exploiting massive parallelism

1. https://www.top500.org/green500/list/2017/06/
2. “The opportunities and challenges of exascale computing.”

OPENMP ON ACCELERATORS?

Why?

The top 16 machines on the Green 500 use an accelerator![1]

Exascale subcommittee report[2] “challenges”

- Reducing Power Requirements
- Coping with Run-time errors
- Exploiting massive parallelism

1. https://www.top500.org/green500/list/2017/06/
2. “The opportunities and challenges of exascale computing.”

OPENMP ON ACCELERATORS?

Why?

The top 16 machines on the Green 500 use an accelerator![1]

Exascale subcommittee report[2] “challenges”

- Reducing Power Requirements
- Coping with Run-time errors
- Exploiting massive parallelism

OpenMP is the *de facto* standard for Parallel programming on shared memory systems*

- Wait! What’s this about shared memory?
- Relaxed shared memory model** — extremely relaxed now with accelerators

1. https://www.top500.org/green500/list/2017/06/
2. “The opportunities and challenges of exascale computing.”

PROLOG

OpenMP has supported accelerators since version 4.0

OpenMP 4.5 added significant features to the accelerator support

OpenMP 5.0 is going to add even more features
many to make programming accelerators easier

This talk will concentrate on a few usability features present in TR6, OpenMP 5.0 preview.

CHALLENGES

OpenMP on Accelerators

Array of structures containing arrays (pointers)

ARRAY OF STRUCTURES CONTAINING ARRAYS

```
typedef struct mystruct {
    char *p;
    int a;
} mystruct_t;
mystruct_t S;
S.p = malloc(100);

#pragma omp target data map(S)
#pragma omp target map(S.p[:100])
{
  for (int i ... )
    DEF(S.p[i])
  foo(S);
}

free(S.p);

int foo(...) {
  for (int i ...)
      USE(S.p[i])
}
```

Problem?

```
typedef struct mystruct {
    char *p;
    int a;
} mystruct_t;
mystruct_t S;
S.p = p_ptr = malloc(100);

#pragma omp target data map(S)
#pragma omp target map(p_ptr[:100])
{
  S.p = p_ptr;
  for (int i ... )
    DEF(S.p[i])
  foo(S);
}

free(S.p);

int foo(...) {
  for (int i ...)
      USE(S.p[i])
}
```

Fixup code

Problem?

ARRAY OF STRUCTURES CONTAINING ARRAYS

```
typedef struct mystruct {
    char *p;
    int a;
} mystruct_t;
mystruct_t S;
S.p = malloc(100);

#pragma omp target data map(S)
#pragma omp target map(S.p[:100])
{
  for (int i ... )
    DEF(S.p[i])
  foo(S);
}

free(S.p);

int foo(...) {
  for (int i ...)
      USE(S.p[i])
}
```

Problem?

```
typedef struct mystruct {
    char *p;
    int a;
} mystruct_t;
mystruct_t S;
S.p = save_p = p_ptr = malloc(100);

#pragma omp target data map(S)
#pragma omp target map(p_ptr[:100])
{
  S.p = p_ptr;
  for (int i ... )
    DEF(S.p[i])
  foo(S);
  S.p = save_p;
}
...
```

Fixup code

OPENMP TR6

Solutions to Challenges

Pointer Attachment

Top down attachment of newly transferred objects to associated pointers

POINTER ATTACHMENT

Example

```
typedef struct mystruct {
    char *p;
    int a;
} mystruct_t;
mystruct_t S;
S.p = malloc(100);

#pragma omp target data map(S)
#pragma omp target map(S.p[:100])
{
  for (int i ... )
    DEF(S.p[i])
  foo(S);
}
free(S.p);

int foo(...) {
  for (int i ...)
      USE(S.p[i])
}
```

```
attach(S.p) = device_malloc(100);
```

```
device_free(S.p[:100]), detach(S.p);
```

OPENMP TR6

Solutions to Challenges

Pointer Attachment

Top down attachment of newly transferred objects to associated pointers

User-defined Mappers

Define a mapper for a structure and then let compiler do the proper transfer and attachment

USER-DEFINED MAPPERS

Example

```
typedef struct mystruct {
    char *p;
    int a, len;
} mystruct_t;
mystruct_t S;
S.len = 100;
S.p = malloc(S.len);
#pragma omp declare mapper(mystruct_t s)\
        use_by_default map(s, s.a, s.p[:s.len])
#pragma omp target data map(S)
{
  for (int i ... )
    DEF(S.p[i])
  foo(S);
}
free(S.p);
int foo(...) {
  for (int i ...)
      USE(S.p[i])
}
```

CHALLENGES

OpenMP on Accelerators

Array of structures containing arrays (pointers)

Function availability
declare directive

FUNCTION AVAILABILITY

declare directive

```
#pragma omp declare target
int compute_plus( int ii, int jj) { return ii + jj;}
#pragma omp end declare target
…
#pragma omp declare target
int compute_mult_plus( int aa, int bb, int cc) {
  return compute_plus( aa*bb, cc);}
#pragma omp end declare target
…
#pragma omp target
{
   compute_mult_plus( a, b, c );
}
```

OPENMP TR6

Solutions to Challenges

Pointer Attachment

Top down attachment of newly transferred objects to associated pointers

User-defined Mappers

Define a mapper for a structure and then let compiler do the proper transfer and attachment

Inferred Function Mapping

No need to add directives for functions used on the device if they are visibly “defined”

INFERRED FUNCTION MAPPING

```
int compute_plus( int ii, int jj) { return ii + jj;}
…
int compute_mult_plus( int aa, int bb, int cc) {
  return compute_plus( aa*bb, cc);}
…
#pragma omp target
{
   compute_mult_plus( a, b, c );
}
```

This does not work if function definition is in another file!

CHALLENGES

OpenMP on Accelerators

Array of structures containing arrays (pointers)

Function availability
- declare directive

Data availability
- data clauses
- data directives
  - local objects
  - global objects

DATA AVAILABILITY

data clauses

map([*[map-type-modifier[,]] map-type* : *] list*)

map-type:
to
from
tofrom
alloc
release
delete

map-type-modifier:
always

DATA AVAILABILITY

data directives

#pragma omp target data *clause[[[,] clause] ...] new-line*
structured-block

#pragma omp target enter data *[clause[[,] clause]...] new-line*

#pragma omp target exit data *[clause[[,] clause]...] new-line*

#pragma omp target update *clause[[[,] clause] ...] new-line*

#pragma omp declare target …

#pragma omp target …

DATA AVAILABILITY

Example

```
int a;
#pragma omp declare target (a)

int b;
#pragma omp declare target to(b)

int big_array[100000];
#pragma omp declare target link(big_array)


#pragma omp target map(to: a, b, big_array[100:300])
…

#pragma omp target map(always, to: a, b, big_array[100:300])
…
```

Data movement for a and b ignored!
Data moved and attached for big_array!

Data movement for a and b not ignored.

OPENMP TR6

Solutions to Challenges

Pointer Attachment

Top down attachment of newly transferred objects to associated pointers

User-defined Mappers

Define a mapper for a structure and then let compiler do the proper transfer and attachment

Inferred Function Mapping

No need to add directives for functions used on the device if they are visibly “defined”

Device Profiles

What requirements does the code have of the implementation?

DEVICE PROFILES

#pragma omp requires unified_address |unified_shared_memory

unified_address

Asserts that the program assumes a unified address space, all addresses are unique!

unifed_shared_memory

Asserts that the program assumes all memory can be read by all processors.
Likely no data clauses or directives in the code.
Possible concern: Scalars are still firstprivate by default.

DATA AVAILABILITY

Example

```
int a;
int b;
int big_array[100000];


#pragma omp requires unified_shared_memory
…

#pragma omp target

…
```

Notice loss of big_array[100:300]

This may or may not be an issue, depending on how the system handles memory sharing

OPENMP ON ACCELERATORS

It’s getting easier!

Pointer Attachment

Easier to use user-defined data types on the device

User-defined Mappers

No possible to define named mappers which understand data structure that may be in unchangeable files.

Inferred Function Mapping

No more need to add extra directives just to get linking to work correctly.

Device Profiles

No more data directives when targeting machines that have shared memory. There will be a portability penalty! Does that matter?