OMP5.1: The Dispatch Construct

Ravi Narayanaswamy, Intel
AGENDA

• MOTIVATION
• DECLARE VARIANT DIRECTIVE
• DISPATCH CONSTRUCT
• EXAMPLE
Enable OpenMP interoperability with libraries which may or may not use OpenMP offloading.
DECLARE VARIANT (C/C++)

**#pragma omp declare variant**(variant-func-id) clause [[[[,]
clause] ... ] new-line

where clause is one of the following:

- **match**(context-selector-specification)
- **adjust_args**(adjust-op : argument-list)
- **append_args**(append-op[, append-op ... ])

where adjust-op is one of the following:

- **nothing**
- **need_device_ptr**

where append-op is one of the following:

- **interop**(interop-type[, interop-type ...])

where variant-func-id is the name of a function variant that is either a base language identifier or, for C++, a template-id.

**#pragma omp begin declare variant** clause new-line

declaration-definition-seq

**#pragma omp end declare variant** new-line
DECLARE VARIANT (FORTRAN)

```fortran
!$omp declare variant
expression-stmt
(base-proc-name:)[variant-proc-name) clause [[[;] clause] ... ]
```

where clause is one of the following:

- `match(context-selector-specification)`
- `adjust_args(adjust-op : argument-list)`
- `append_args(append-op[, append-op] ... )`

where adjust-op is one of the following:

- `nothing`
- `need_device_ptr`

where append-op is one of the following:

- `interop(interp-type[, interp-type ...])`

where variant-proc-name is the name of a function variant that is a base language identifier.
MATCH

• Choose specific target calls to replace
• Ability to elide code

eg:

• `match(context={dispatch})`
  replace call in omp dispatch region

• `match(device={arch(gen)})`
  skip the variant región if the target arch is not gen

• `match(device={kind(nohost)})`
  skip the variant región if compiling for host

• `match(device={kind(host)})`
  skip the variant región if compiling for device
**APPEND_ARGS**

**Allows to pass additional arguments to the variant function**

- the interop operation constructs an argument of type `omp_interop_t` (C/C++) or `omp_interop_kind` (Fortran) from the interoperability requirement set of the encountering task.
- the argument is constructed as if an interop construct with an init clause of interop-types was specified.
- the arguments are passed in the same order in which they are specified in the append_args clause.
- the arguments are destroyed after the call to the selected variant returns
ADJUST_ARGS

Allows to adjust the argument passed to the variant.

- if the adjust-op modifier is `need_device_ptr`,
  - argument is not a device pointer then will behave as a `use_device_ptr` is applied
  - if argument is device pointer, it is passed without being modified.
  - if the argument cannot be converted into a device pointer then a null value will be passed
- if the adjust-op modifier is nothing,
  - the argument is passed without being modified.
**DISPATCH DIRECTIVE (C/C++)**

```c
#pragma omp dispatch [clause[ [,] clause] ...] new-line
```

where `expression-stmt` is an expression statement with one of the following forms:

```c
expression = target-call ([expression-list]);
target-call ([expression-list]);
```

where `clause` is one of the following:

- `device(integer-expression)`
- `depend([depend-modifier,] dependence-type : locator-list)`
- `nowait`
- `novaraints(scalar-expression)`
- `nocontext(scalar-expression)`
- `is_device_ptr(list)`
#pragma omp dispatch [clause[ [,] clause] ... ]

where stmt is an expression statement with one of the following forms:

- **expression** = target-call ([arguments]);
- **call** target-call ([arguments]);

where clause is one of the following:

- **device**(scalar-integer-expression)
- **depend**([[depend-modifier,] dependence-type : locator-list])
- **nowait**
- **novariants**(scalar-logical-expression)
- **nocontext**(scalar-logical-expression)
- **is_device_ptr**(list)
Dispatch clause

- If `novariants` clause is present and evaluates to true no variant substitution occurs
- If `nocontext` clause is present and evaluates to true, dispatch construct is not added to the `construct set` of OpenMP context
- `Is_device_ptr` indicates that its list items are device pointers.
- `no_wait` clause if present will be added to `interoperability requirement set`
- If `depend` clause are present, they are added to `interoperability requirement set`
using IFDEF

```c
#ifdef host
    static int omp_is_initial_device() { return 1; }
#else
    static int omp_is_initial_device() { return 0; }
#endif
```

- Need to standardize the define macro
- Need defines to indicate different conditions like architecture, devices ...
- All call sites would call the same function, cannot selectively chose call sites
using DECLARE VARIANT

#pragma omp begin declare variant match(device={kind(host)})

    static int omp_is_initial_device() { return 1; }

#pragma omp end declare variant

#pragma omp begin declare variant match(device={kind(nohost)})

    static int omp_is_initial_device() { return 0; }

#pragma omp end declare variant
my_math.h:

......

void sgemm_gpu(float * a, float *b, float *c, int size, omp_interop_t *obj);

#pragma omp declare variant(sgemm_gpu) \ 
   match(context={dispatch}, \ 
      offload_device={arch(gen)}) \ 
   adjust_arg(need_device_ptr(a,b,c)) \ 
   append_params(interop(...))

void sgemm (float * a, float *b, float *c, int size);

......

AT *, INT, FLOAT, FLOAT *, INT);
EXAMPLE (user code)

```c
#include my_math.h

int main() {
    int size;
    float *a, *b, *c, *d;

    allocate_init(a,b,c,d); // initialize data
    sgemm (a, b, c, size); // call cpu version

#pragma omp target data
    map(to:a[0:sizec],b[0:sized]) map(tofrom:c[0:sizea],d[0:sizeb]) device(dev_id)
{
    #pragma omp dispatch device(dev_id)
        sgemm (a, b, c, size); // call gpu version
    #pragma omp dispatch device(dev_id)
        sgemm (a, b, d, size); // call gpu versión
}
```