
OpenACC
archer2
epcc

Reusing this material

Partners

Overview

- OpenACC is an alternative directives-based API for offloading to devices.
- Pre-dates OpenMP offloading, but now has fewer implementations than OpenMP offloading.
- OpenMP and OpenACC have very similar functionality, though OpenACC is a little more GPU-specific.
- OpenACC uses different terminology and syntax.
- OpenACC does not control parallelism on the host (CPU).
- OpenACC has a slightly less prescriptive approach than OpenMP.
    - intended to rely a bit more on compiler analysis capabilities

Syntax

- Very similar to OpenMP
- Sentinels are:

  Fortran: `!$ACC` instead of `!$OMP`

  C/C++: `#pragma acc` instead of `#pragma omp`

Execution model

- As with OpenMP, OpenACC exposes three layers of parallelism on the device (though we mainly use the first two for GPUs).

OpenMP	OpenACC	GPU hardware
team	gang	SM
thread	worker	hardware thread within an SM
simd	vector	vector instruction in a thread

Execution on the device

- The `acc parallel` construct is the basic offloading construct in OpenACC.
- During execution, when an `acc parallel` construct is encountered, the code it contains is executed on the device.
- By default, the code inside the `acc parallel` construct is executed by *every gang*.
  - Note this is different from OpenMP: `acc parallel` is equivalent to `omp target teams`.
  - If you really want serial execution on the device you can use `acc serial`.

```
// Block A: executed on host
#pragma acc parallel
{
    // Block B: executed on device
}
// Block C: executed on host
```

C/C++

```
! Block A: executed on host
!$ACC PARALLEL
    ! Block B: executed on device
!$ACC END PARALLEL
! Block C: executed on host
```

Fortran

Data transfers

- These clauses work in a very similar way to OpenMP, only the syntax is different.

OpenMP	OpenACC
`map(to:list)`	`copyin(list)`
`map(from:list)`	`copyout(list)`
`map(tofrom:list)`	`copy(list)`
`map(alloc:list)`	`create(list)`

Data regions

- Structured and unstructured data regions and updates also work in the same way as in OpenMP.

OpenMP	OpenACC
`omp target data`	`acc data`
`omp target [enter|exit] data`	`acc [enter|exit] data`
`omp target update to`	`acc update device`
`omp target update from`	`acc update self`

Data transfer example

Two arrays and one scalar

```
#pragma acc parallel \
copyin(B, C) copy(sum)
{
....
}
```

C/C++

```
!$ACC PARALLEL
!$ACC& COPYIN(B(1:N), C(1:N))
!$ACC& COPY(SUM)
....
!$ACC END PARALLEL
```

Fortran

Parallel loops

- In OpenACC the `acc loop` construct can be applied to loops to specify parallel execution on the device.
- Note: this does not exactly match the semantics of the OpenMP `omp loop` construct: `acc loop` is more prescriptive than `omp loop`
- The level(s) at which the loop should be parallelised is specified by one or more of the `gang`, `worker` and `vector` clauses.
- `collapse`, `reduction` and `private` clauses work just the same as in OpenMP.

OpenMP	OpenACC
`omp teams distribute`	`acc loop gang`
`omp parallel for/do`	`acc loop worker`
`omp simd`	`acc loop vector`
`omp teams distribute parallel for/do`	`acc loop gang worker`

Parallel loops

```
#pragma acc parallel loop gang worker\
copyin(B, C) copy(sum) reduction(+:sum)
for (int i=0; i<N; i++) {
  sum += B[i] + C[i];
}
```

C/C++

```
!$ACC PARALLEL LOOP GANG WORKER
!$ACC& COPYIN(B(1:N), C(1:N)) COPY(SUM)
!$ACC& REDUCTION(+:SUM)
DO i = 1,N
  sum = sum + B(i) + C(i)
ENDDO
!$ACC END PARALLEL LOOP
```

Fortran

Kernels

- OpenACC supports a more “hands-off” approach to offloading via the **`acc kernels`** construct.
- Specifies a block of code (which may contain multiple loops and other statements) to be offloaded to the device.
- Leaves it up to the compiler to decide how to do this.
  - Typically, each loop nest will be a distinct kernel.
- Can leave it to the compiler to automatically generate the required data transfers (or specify them explicitly).
- No guarantee the compiler will succeed!
- No direct counterpart in OpenMP – closest equivalent is **`omp target teams loop`**.

Kernels

```
#pragma acc kernels
for (int i=0; i<N; i++) {
  sum += B[i] + C[i];
}
```

C/C++

```
!$ACC KERNELS
DO i = 1,N
  sum = sum + B(i) + C(i)
ENDDO
!$ACC END KERNELS
```

Fortran

Called routines

- Routines called from inside offloaded regions must contain an `acc routine` directive to indicate that compilation for device is required.
  - Equivalent to `omp declare target` in OpenMP.
- Goes in the same place as in OpenMP (before routine spec in C/C++, inside routine in Fortran).
  - N.B. no equivalent to `omp end declare target` in C/C++: each routine needs a separate `acc routine` directive.
- `acc routine` directive can take a `gang|worker|vector` clause to indicate the level of parallelism.
  - E.g. use `acc routine gang` if the routine contains an `acc loop gang` directive.

Synchronisation

- `acc atomic` directive in OpenACC is exactly the same as `omp atomic` in OpenMP.
- Protects the memory location on the left-hand side from race conditions.
- Also has `read`, `write` and `capture` variants as well as `update` (default).

```
#pragma acc atomic
  a++;
```
C/C++

```
!$ACC ATOMIC
   a=a+1
```
Fortran

- Unlike OpenMP, OpenACC does not support other synchronisation constructs, such as `omp barrier` or `omp critical`, within a team/gang.

Asynchronous offloads

- As with OpenMP, offloaded code constructs in OpenACC are blocking on the host by default.
- Asynchronous offloads can be enabled with the `async` clause and the `acc wait` directive will block the host until all asynchronous kernels have completed.
- Equivalent to `omp target nowait` and `omp taskwait` in OpenMP, but OpenACC does not support a full tasking model on the host.
  - No equivalent to the `depend` clause, for example.
- On the other hand, OpenACC explicitly supports multiple queues on the device, which OpenMP does not.

Asynchronous offloads

```
#pragma acc kernels async
for (int i=0; i<N; i++) {
  sum += B[i] + C[i];
}
// do some computation on host at the same time
#pragma acc wait
```

C/C++

```
!$ACC KERNELS ASYNC
DO i = 1,N
  sum = sum + B(i) + C(i)
ENDDO
!$ACC END KERNELS
! do some computation on host at the same time
!$ACC WAIT
```

Fortran

Runtime Execution Environment Routines

OpenMP	OpenACC	Notes
`omp_get_num_devices`	`acc_get_num_devices`	
`omp_get_device_num`	`acc_get_device_num`	
`omp_get_num_teams` `omp_set_num_teams`	- `num_gangs` clause	Clause on `parallel` or `kernels` construct
`omp_get_team_num`	-	
`omp_get_teams_thread_limit` `omp_set_teams_thread_limit`	- `num_workers` clause	Clause on `parallel` or `kernels` construct