OpenMP offloading with Fortran

Soner Steiner

Intel Instructor
VSC Research Center, TU-WIEN
soner dot steiner at tuwien dot ac dot at
sonersteiner at gmail dot com

Intel oneAPI HPC Workshop- Agenda
Online 19th-20th September 2023

DAY 2 – THEME: PROGRAMMING WITH ONEAPI

Time	Session name / description	Presenter
09:30	Welcome	*Soner/Claudia*
09:40	Porting CUDA code to SYCL using the Compatibility Tool	*Georg*
10:30	HANDS-ON WITH COMPATIBILITY TOOL A hands-on lab session where you can try porting a CUDA code to oneAPI with the help of the Compatibility Tool	*Georg*
11:00	Coffee	
11:15	OFFLOADING WITH C/C++ and FORTRAN Offloading using OpenMP mainly in C/C++ Offloading using OpenMP in FORTRAN Automatic offloading using DO CONCURRENT	*Soner*
12:00	Lunch	
13:00	LAB3: HANDS-ON OFFLOADING WITH OPENMP	*Soner*
14:30	Coffee	
14:45	LAB4: HANDS-ON VTUNE A hands-on lab session where you can use the Vtune and Advisor profilers to assess the performance of some example codes.	*Soner*
16:00	End of Day 2	

Agenda

- Automatic offloading with DO CONCURRENT
- Offloading with OpenMP
- OpenMP Target Construct
- Managing Device Data
- Implementing Parallelism

DO CONCURRENT

- *Embarrassingly parallel problems:*
  Can be distributed across several cores with little to no effort.
  E.g.: large number of independent data records, graphics rendering, …
  DO CONCURRENT can be used!

- *Non embarrassingly parallel problems:*
  Any parallel problem with interdependency between processes, requires communication and synchronization.

DO CONCURRENT

- What is do concurrent*?*
  It is a promise from the programmer to compiler that the code inside loop can be safely parallelized.

- What do concurrent is not*?*
  It is not a guarantee that the loop will run in parallel, maybe the compiler determines that a serial execution would be more efficient.

DO CONCURRENT

do [,] concurrent ([type-spec ::] index-spec-list [, scalar-mask-expr])

- type-spec (if present) specifies the type and kind of the index variables
- index-spec-list is a list of index specifications
  do concurrent (i=1:n, j=1:m)

DO CONCURRENT, Examples

```
do concurrent( i = 1:n )
 c(i) = a(i) + b(i)
end do
```

```
do concurrent(i = 1:grid_size)
  h(i) = exp(-decay * (i - icenter)**2)
end do
```

Demo at the end, ... demos/00_demo/

OpenMP Offload Constructs

Be aware:

Offloading to a device with oneAPI icx, icpx and ifx works only with Intel GPUs at the moment.

We are expecting it in the coming year

Just like the backend SW we have used with SYCL for the nvidia GPUs .

OpenMP Offload Constructs

Device Code

omp target *[clause[[,]clause]…] structured-block*
omp declare target *[function-definitions-or-declarations]*
omp declare target [variable-definitions-or-declarations]

Worksharing

omp teams *[clause[[,]clause]…] structured-block*
omp distribute *[clause[[,]clause]…] for-loops*

Memory operations

map *([[map-type-modifier[,]]map-type:] list) map-type := **alloc** | **tofrom** | **to** | **from** | **release** | **delete** map-type-modifier := **always***
omp target data *clause[[[,] clause]…] structured-block*
omp target enter data *clause[[[,]clause]…]*
omp target exit data clause[[[,]clause]…]
omp target update clause[[[,]clause]…]

OpenMP Offload Language

C/C++	Fortran
#pragma omp target *[clause[[,]clause]…]* *structured-block*	**!$omp target** *[clause[[,]clause]…]* *structured-block* **!$omp end target**
#pragma omp target data *[clause[[,]clause]…]* *structured-block*	**!$omp target** *[clause[[,]clause]…]* *structured-block* **!$omp end target data**
#pragma omp teams *[clause[[,]clause]…]* *structured-block*	**!$omp teams** *[clause[[,]clause]…]* *structured-block*
#pragma omp distribute *[clause[[,]clause]…]* *structured-block*	**!$omp distribute** *[clause[[,]clause]…]* *structured-block*

OpenMP 4.0 for Devices - Constructs

#pragma omp target !$omp target

- **target** construct transfer control and data from the host to the device
- Syntax (C/C++)

```
#pragma omp target [clause[[,] clause],…]
structured-block
```

- Syntax (Fortran)

```
!$omp target [clause[[,] clause],…]
structured-block
!$omp end target
```

- Clauses

```
device(scalar-integer-expression)
map([{alloc | to | from | tofrom}:] list)
if(scalar-expr)
```

Target construct

```
1  program target_construct_1
2    implicit none
3
4    integer :: i
5    integer :: a(100), b(100), c(100)
6
7    do i=1, 100
8      a(i) = 3
9      b(i) = 7
10   end do
11
12   !$omp target
13     do i=1, 100
14         c(i) = a(i) + b(i)
15     end do
16   !$omp end target
17
18   do i=1, 100
19     write(*,*), a(i), b(i), c(i)
20   end do
21
22 end program target_construct_1
```

Device code

`target [clause]`

- Offloads a code region to a target device
- Sequential and synchronous by default

clause : device, private, firstprivate, map, allocate

Sync : nowait, depend

2.12.5 `target` Construct

Device Clause

```
!$omp target device(i)
```

- Specify which device to offload to in a multi-device environment

  - Device number an integer
    - Assignment is implementation-specific
    - Usually start at 0 and sequentially increments

  - Works with **`target`**, **`target data`**, **`target enter/exit data`**, **`target update`** directives

Device Clause

Target device construct
Integer numbers, 0, 1, 2
specify the device

```
:>$ sycl-ls
[opencl:acc:0] Intel(R) FPGA Emulation Platform for OpenCL(TM), Intel(R) FPGA
[opencl:cpu:1] Intel(R) OpenCL, Intel(R) Core(TM) i9-10900K CPU @ 3.70GHz 3.0
[opencl:gpu:2] Intel(R) OpenCL HD Graphics, Intel(R) UHD Graphics 630 [0x9bc5]
```

```
ifx -qopenmp -fopenmp-targets=spir64 -o target_construct_7.x target_construct_7.f90
:>$ OMP_TARGET_OFFLOAD="MANDATORY"
:>$ LIBOMPTARGET_PLUGIN=LEVEL0
:>$ LIBOMPTARGET_DEBUG=1
```

```
program target_construct_2
  use omp_lib
  implicit none

  integer :: i
  integer, parameter :: n=1000000
  integer :: a(n), b(n), c(n)

  do i=1, n
    a(i) = 3
    b(i) = 7
  end do

  !$omp target device(2)
    do i=1, n
        c(i) = a(i) + b(i)
    end do
  !$omp end target

  do i=1, n
    write(*,*), a(i), b(i), c(i)
  end do

end program target_construct_2
```

Device code

Target device construct

`target device`

- Specify which device to offload to in a multi-device environment
- Device number is an integer, usually starts at 0 and sequentially increments

- Works with **target, target data, target enter \ exit data, target update** directives

2.12.5 `target` Construct

OpenMP* Device Parallelism

```
5   double precision :: start_time, end_time
6   integer :: i
7   integer, parameter :: n=100000000
8   integer :: a(n), b(n), c(n)
9
10  do i=1, n
11    a(i) = 3
12    b(i) = 7
13  end do
14
15  start_time = omp_get_wtime()
16  !$omp target device(2)
17    !$omp parallel do
18        do i=1, n
19            c(i) = a(i) + b(i)
20        end do
21    !$omp end parallel do
22  !$omp end target
23  end_time = omp_get_wtime()
24
25  do i=1, n
26    write(*,*), a(i), b(i), c(i)
27  end do
28  write(*,*) "Work took: ", end_time - start_time, " seconds"
```

Device code

target [clause]

- Offloads a code region to a target device
- Sequential and synchronous by default

Why NOT parallel for?

- CPU parallelism differs from GPU – shared memory systems
- omp parallel for threads will use only 1 Streaming Multiprocessor (SM) to synchronize
- Need a different level of parallelism to step over multiple SM

OpenMP* Device Parallelism

```
5   double precision :: start_time, end_time
6   integer :: i
7   integer, parameter :: n=100000000
8   integer :: a(n), b(n), c(n)
9
10  do i=1, n
11    a(i) = 3
12    b(i) = 7
13  end do
14
15  start_time = omp_get_wtime()
16  !$omp target device(2)
17    !$omp teams distribute parallel do
18        do i=1, n
19            c(i) = a(i) + b(i)
20        end do
21    !$omp end teams distribute parallel do
22  !$omp end target
23  end_time = omp_get_wtime()
24
25  !  do i=1, n
26  !    write(*,*), a(i), b(i), c(i)
27  !  end do
28  write(*,*) "Work took: ", end_time - start_time, " seconds"
```

Device code

target [clause]

- Offloads a code region to a target device
- Sequential by default

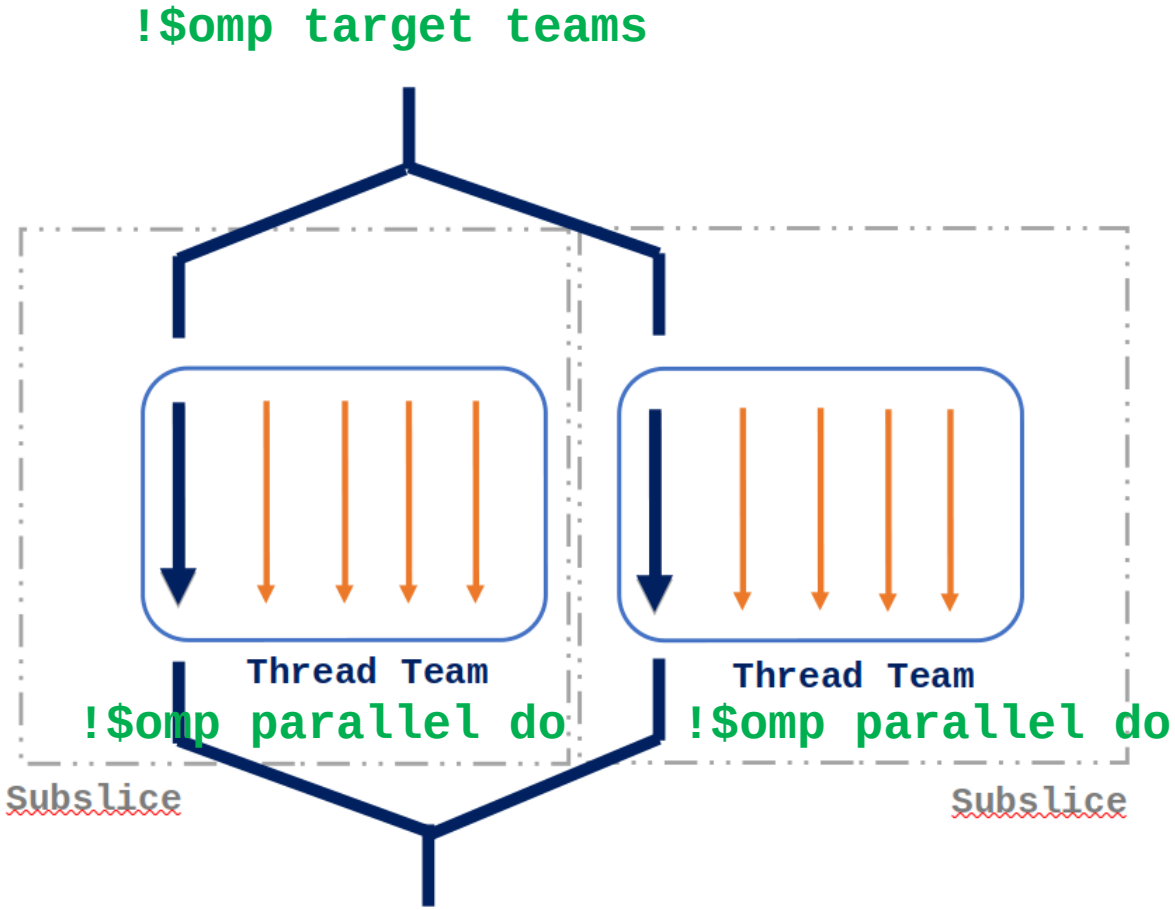
Target teams

- creates a league of teams where the primary thread of each team executes the teams region

[2.13.21 target teams Construct]

Teams Construct

OpenMP	GPU Hardware
SIMD	SIMD Lane (Channel)
Thread	SIMD Thread mapped to an EU
Team	Group of threads mapped to a Subslice
League	Multiple Teams mapped to a GPU



OpenMP* Worksharing

```
5   double precision :: start_time, end_time
6   integer :: i
7   integer, parameter :: n=100000000
8   integer :: a(n), b(n), c(n)
9
10  do i=1, n
11    a(i) = 3
12    b(i) = 7
13  end do
14
15  start_time = omp_get_wtime()
16    !$omp target teams distribute parallel do
17        do i=1, n
18            c(i) = a(i) + b(i)
19        end do
20  !$omp end target teams distribute parallel do
21  end_time = omp_get_wtime()
22
23  !  do i=1, n
24  !    write(*,*), a(i), b(i), c(i)
25  !  end do
26    write(*,*) "Work took: ", end_time - start_time, " seconds"
27
28  end program target_construct_5
29
30
```

Device code

`target teams distribute`

shortcut for specifying a target construct containing a teams distribute construct and no other statements.

`target teams distribute parallel do`

parallel worksharing-loop construct is a shortcut for specifying a target construct containing a teams distribute parallel worksharing-loop construct and no other statements

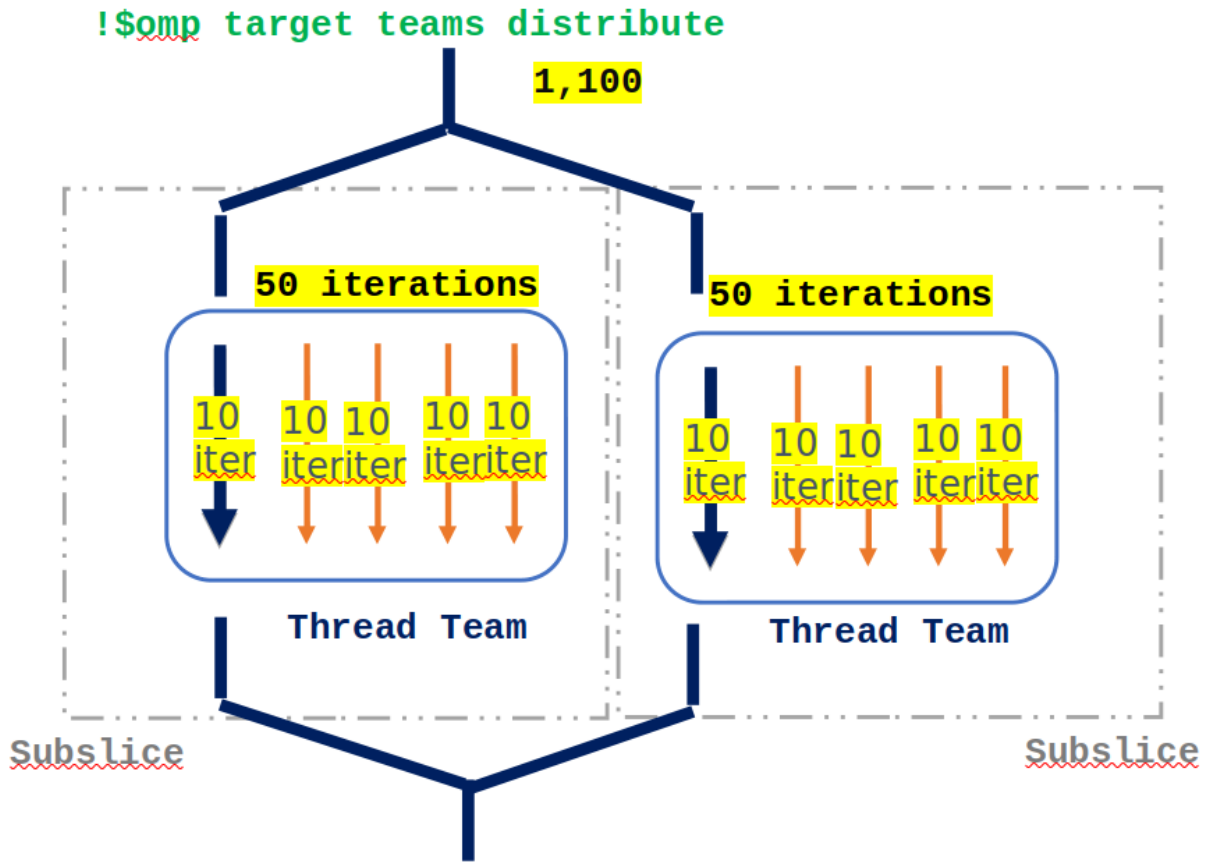
2.13.11 `teams distribute` **Construct**
2.13.25 Target Teams Distribute Parallel Worksharing-Loop Construct

Teams Distribute Construct

```
!$omp target teams
distribute parallel do
    do k=1,100

       c(k) =  a(k) + b(k)

    end do
!$omp end target teams
distribute parallel do
```



Calling functions inside Target region

```
subroutine devicefunc()
!$omp declare target
device_type(device)
…
end subroutine

program main
!$omp target
   call devicefunc()
!$omp end target
end program
```

`declare target`

compiles a version of the function/subroutine for the target device

Function compiled for both host execution and target execution by default

`device_type(host | nohost | any)`

2.12.7 `declare target` Directive

Managing Device Data

Data Offload

- The host and devices have separate memory spaces.
- When code is offloaded, data needs to be mapped.
- By default, copied to the device on entry and from the device to the host on exit.

- Data environment is **lexically scoped**
- Data environment is destroyed at closing curly brace
- Allocated buffers/data are automatically released

Default Behaviour:

```
5   double precision :: start_time, end_time
6   integer :: i
7   integer, parameter :: n=100000000
8   integer :: a(n), b(n), c(n)
9
10  do i=1, n
11    a(i) = 3
12    b(i) = 7
13  end do
14
15  start_time = omp_get_wtime()
16  !$omp target device(2)
17    !$omp parallel do
18        do i=1, n
19            c(i) = a(i) + b(i)
20        end do
21    !$omp end parallel do
22  !$omp end target
23  end_time = omp_get_wtime()
24
25  do i=1, n
26    write(*,*), a(i), b(i), c(i)
27  end do
28  write(*,*) "Work took: ", end_time - start_time, " seconds"
```

The compiler identifies variables that are used in the target region.

All accessed arrays are copied from host to device and back

Copying a and b back is not necessary: it was not changed.

Target map construct, map clause

- Use **map** clause to manually determine how an original variable in a data environment is mapped to a corresponding variable in a device data environment
  - omp target map (*map-type: list*)
  - Available map-type
    - to : alloc and assign value of original variable on target region entry
    - from : alloc and assign value to original variable on target region exit
    - tofrom: default, both to and from

Using MAP to refine behaviour:

```
5   double precision :: start_time, end_time
6   integer :: i
7   integer, parameter :: n=1000
8   integer :: a(n), b(n), c(n)
9
10  do i=1, n
11    a(i) = 3
12    b(i) = 7
13  end do
14
15  start_time = omp_get_wtime()
16  !$omp target map(to:a) map(to:b) map(tofrom:c)
17    !$omp parallel do
18        do i=1, n
19            c(i) = a(i) + b(i)
20        end do
21    !$omp end parallel do
22  !$omp end target
23  end_time = omp_get_wtime()
24
25  do i=1, n
26    write(*,*) c(i)
27  end do
28  write(*,*) "Work took: ", end_time - start_time, " seconds"
```

Unnecessary to copy
a and b back to the host

OpenMP Offload Constructs

- Device Code
  - **omp target** *[clause[[,]clause]…] structured-block*
  - **omp declare target***[function-definitions-or-declarations]*
  - **omp declare target** [variable-definitions-or-declarations]
- Worksharing
  - **omp teams** *[clause[[,]clause]…] structured-block*
  - **omp distribute** *[clause[[,]clause]…] for-loops*
- Memory operations
  - **map** *([[map-type-modifier[,]]map-type:] list) map-type := **alloc** | **tofrom** | **to** | **from** | **release** | **delete** map-type-modifier := **always***
  - **omp target data** *clause[[[,] clause]…] structured-block*
  - **omp target enter data** *clause[[[,]clause]…]*
  - **omp target exit data** clause[[[,]clause]…]
  - **omp target update** clause[[[,]clause]…]

Compiling and running on your laptop

```
$ ifx -fopenmp  -fopenmp-targets=spir64 -O2 -g
-o 00-hello_world_omp.x 00-hello_world_omp.f90
```

build

```
$ ./00-hello_world_omp.x
```

run

Setting the number of threads used via the shell variable

```
$ env OMP_NUM_THREADS=8 ./00-Hello.x
```

```
$ g++ -fopenmp -O2 -g  -o 00-Hello.x 00-Hello.cpp
```

Using the gcc compiler

Or using a simple makefile

```
1  # Compiler settings
2  FXX = ifx
3  #FXXFLAGS = -qopenmp -fopenmp-targets=spir64 -qopt-report
4  FXXFLAGS = -qopenmp -fopenmp-targets=spir64 -O2 -g
5
6  # Setting the source and binary files
7  SRC = $(wildcard *.f90)
8  BIN = $(SRC:.f90=)
9
10 # Rules
11 default: 00_hello_world_omp
12
13 00_hello_world_omp: 00_hello_world_omp.f90
14     $(FXX) $(FXXFLAGS) -o 00_hello_world_omp.x 00_hello_world_omp.f90
15
16 clean:
17     @$(RM) $(BIN)
18
19 # Setting the independent commands
20 .PHONY: default clean
```

Makefile saved as makefile_simple.mak

Invoke for compiling

$ make -f makefile_simple.mak


intel

Notices & Disclaimers

Performance varies by use, configuration, and other factors. Learn more at www.Intel.com/PerformanceIndex.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See configuration disclosure for details.

Your costs and results may vary.

Intel technologies may require enabled hardware, software or service activation.