# GPU application porting strategies

LUMI Advanced Workshop

March 5–7, 2025

# Agenda

- Approaches to Accelerate Applications
- C / Fortran tips and tricks
- OpenMP / OpenACC

# Approaches to Accelerate Applications

## Accelerated Libraries

- The easiest solution, just link the library to your application without in-depth knowledge of GPU programming
- Many libraries are optimized by GPU vendors, eg. algebra libraries

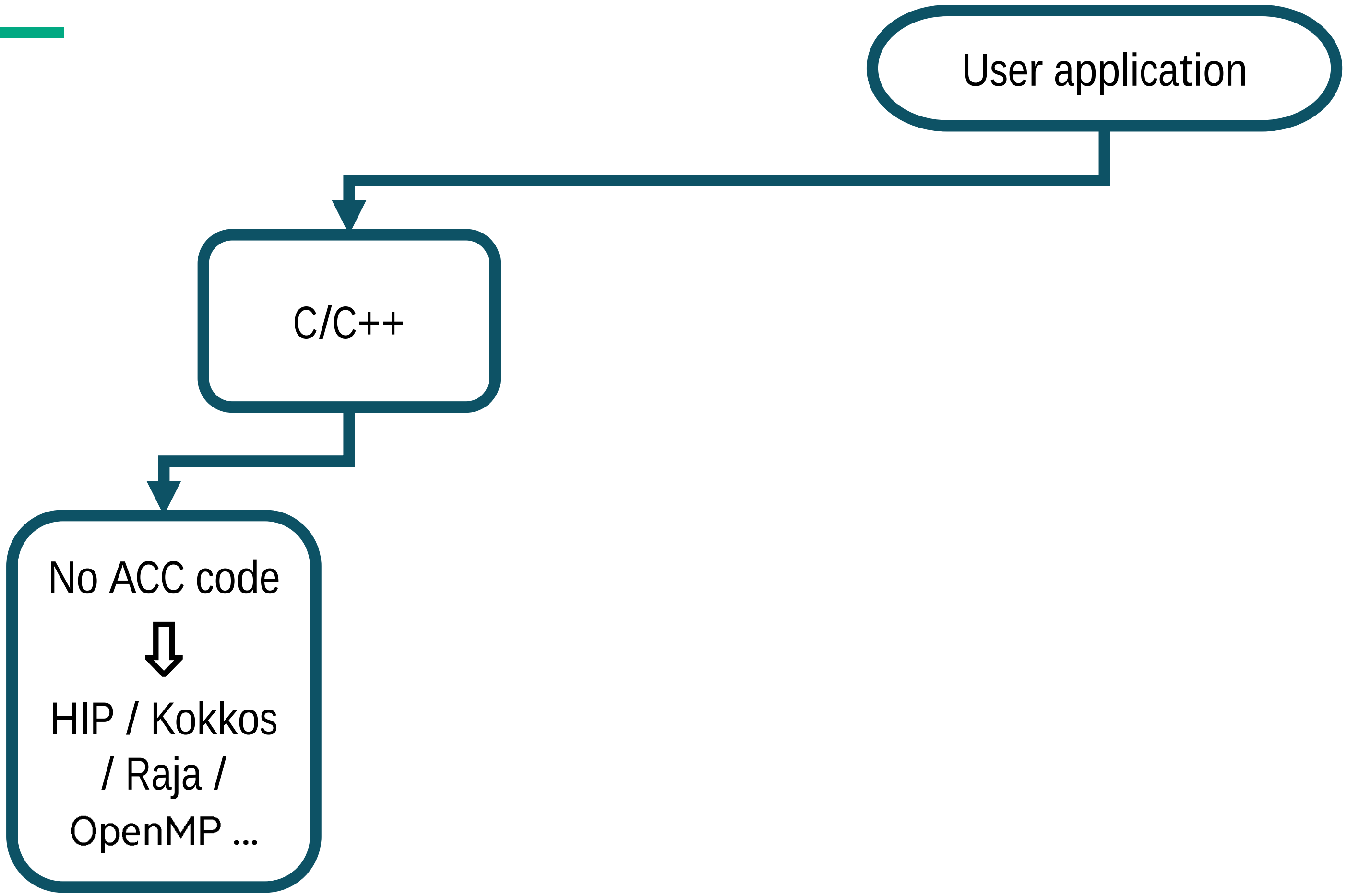
## Directive based methods

- Add acceleration to your existing code (C, C++, Fortran)
- Can reach good performance with somehow minimal code changes
- OpenACC, OpenMP

## Programming Languages

- Maximum flexibility, require in-depth knowledge of GPU programming and code rewriting (especially for Fortran)
- Kokkos, RAJA, Alpaka, CUDA, HIP, OpenCL, SYCL

# Use-cases



# C/C++ - No ACC code – HIP programming

- Profile applications to identify which parts to offload
- Start writing HIP code
  - Data movement
  - Kernels
- Repeat the procedure to optimize and offload more

  (this procedure is valid for any porting technique)

# Portability AMD – NVIDIA

- HIP can run on NVIDIA, with some exceptions:
  - Wavefront size is 64 for AMD, 32 for NVIDIA
  - Dynamic parallelism not supported on AMD
- Because HIP is (almost) 1:1 to CUDA, you can use macros to support both in the same source baseline
  – Used in the DBCSR library, https://github.com/cp2k/dbcsr, `src/acc` directory
  – E.g.

```
ACC_API_CALL(SetDevice, (device_id)); // Only specify function “root” name
```

where

```
#if defined(__CUDA) // compile time flag, i.e. -D__CUDA
#define ACC(x) cuda##x
#elif defined(__HIP) // // compile time flag, i.e. -D__HIP
#define ACC(x) hip##x
#endif
#define ACC_API_CALL(func, args)                                                  \
  {                                                                               \
    ACC(Error_t) result = ACC(func) args;                                         \
    if (result != ACC(Success)) {                                                 \
      printf("\nACC API error %s (%s::%d)\n", ACC(GetErrorName)(result), __FILE__, __LINE__);  \
      exit(1);                                                                    \
    }                                                                             \
  }
```

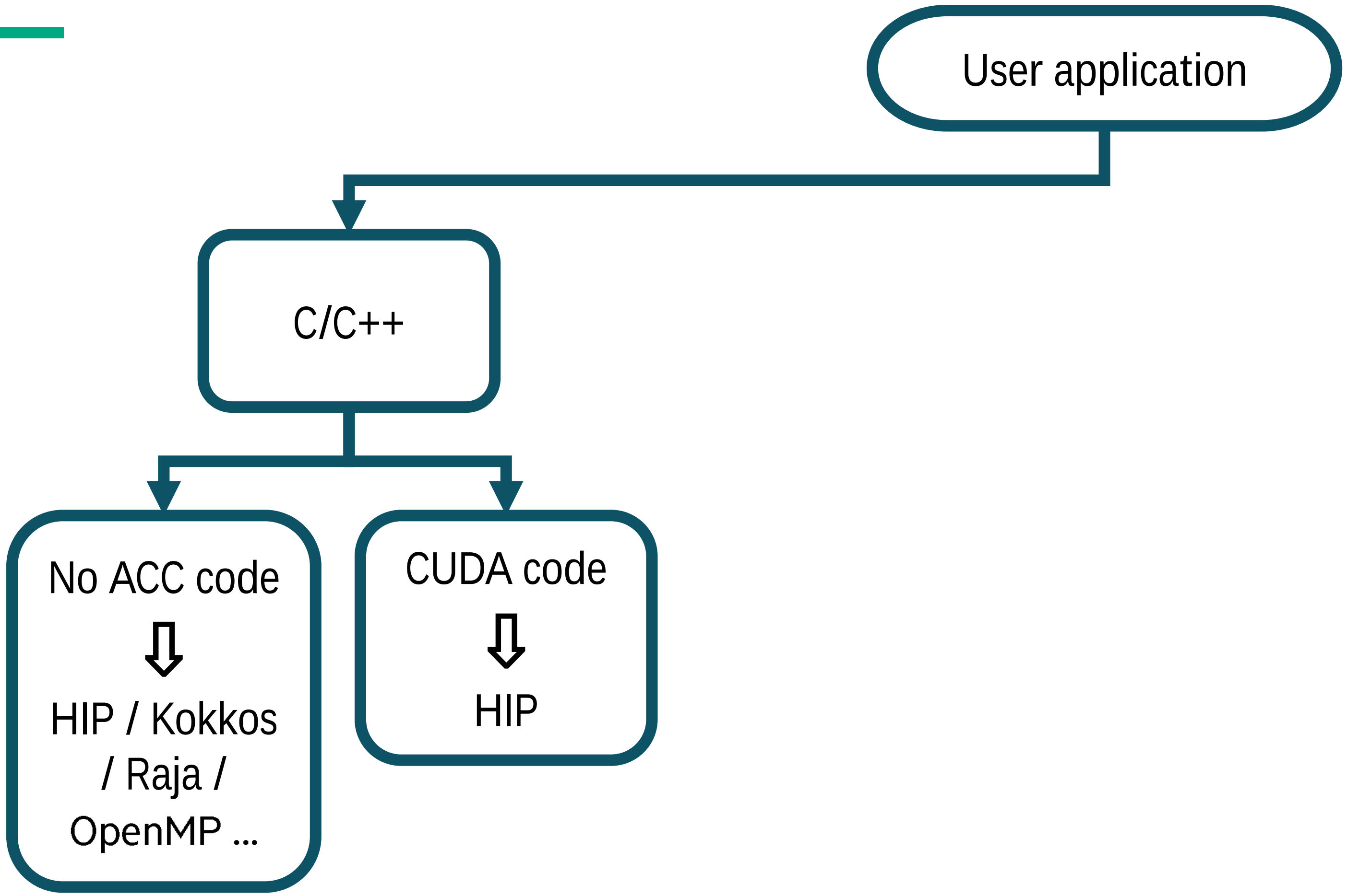
 

# C/C++ - No ACC code – Portable approaches

- Kokkos, Raja, Alpaka
  - High-level C++ libraries to provide performance portability across accelerators
  - Built on top of specific hardware backends (including CPU), .e.g CUDA, HIP, HPX, OpenMP, C++ threads
  - Well-supported
- SYCL
  - Standard developed by Khronos Group (https://www.khronos.org/sycl/)
  - Requires hipSYCL to compile for the AMD GPU backend (https://lumi-supercomputer.github.io/LUMI-EasyBuild-docs/h/hipSYCL/)
- Directive based approach with OpenMP offload
- Offload C++ Standard Parallelism (stdpar)
  - Evolving support as part of the C++ standard
  - https://github.com/ROCm/roc-stdpar
  - https://rocm.blogs.amd.com/software-tools-optimization/hipstdpar/README.html
  - https://arxiv.org/pdf/2401.02680.pdf
- OpenCL (not very popular nowadays)

Interoperability of the techniques is possible, but requires special care

# Use-cases


User application
C/C++
No ACC code
⇩
HIP / Kokkos / Raja / OpenMP ...
CUDA code
⇩
HIP

# C/C++ - CUDA code

- ROCm provides a tool to “hipify” CUDA code
  - hipify-clang
    - Compiler (clang) based translator
    - Handles very complex constructs
    - Prints an error if not able to translate
    - Supports clang options
    - Requires CUDA
  - hipify-perl
    - Perl script
    - Relies on regular expressions
    - May struggle with complex constructs
    - Does not require CUDA

➢ https://github.com/ROCm-Developer-Tools/HIPIFY

➢ https://rocmdocs.amd.com/projects/HIPIFY/en/latest/

# Hipify-perl example (1)

- `hipify-perl –examin <file>.cu`
  - For initial assessment
  - No replacements done
  - Prints basic statistics and the number of replacements

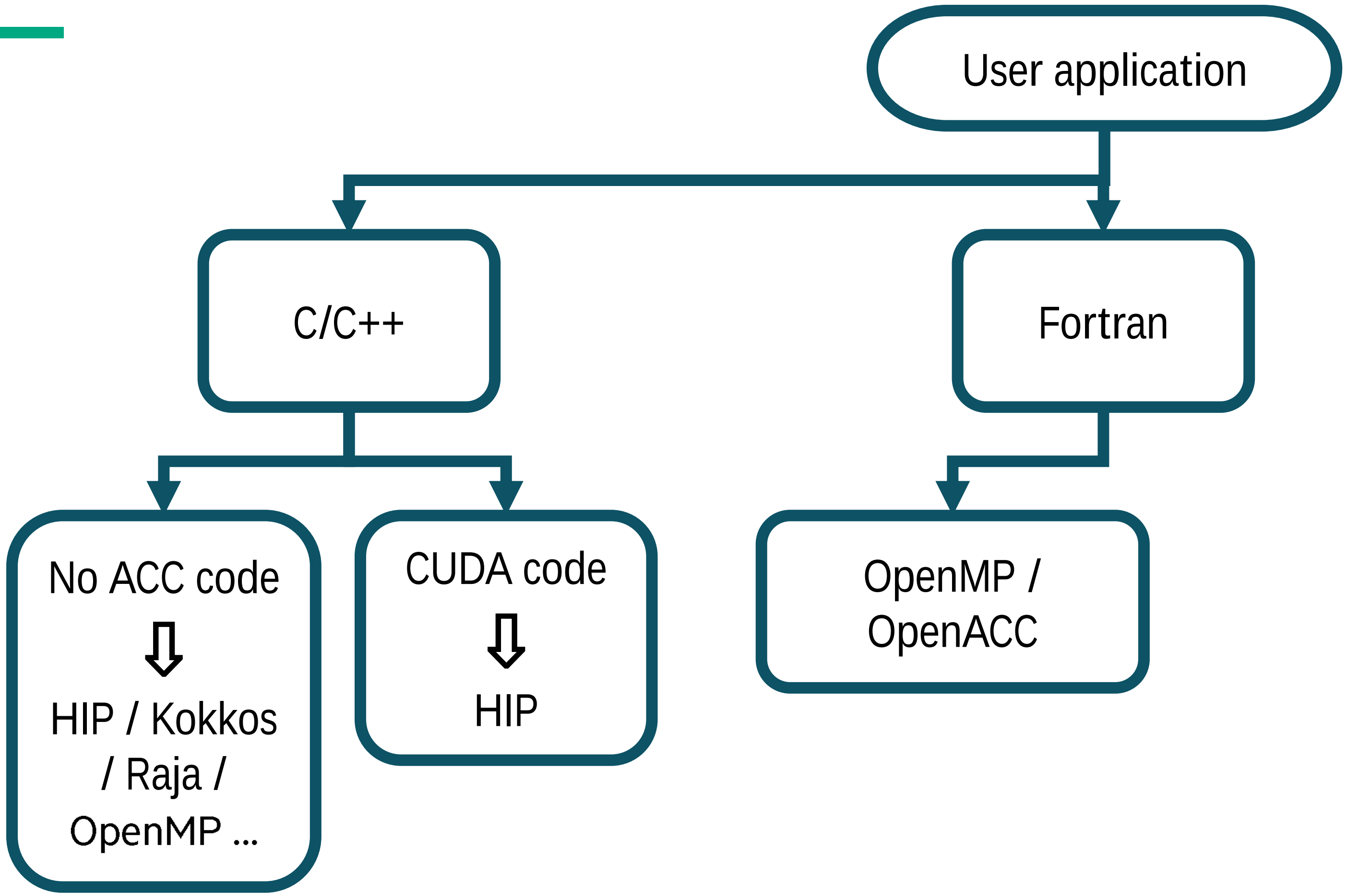
```
> hipify-perl -examin example_slide.cu

[HIPIFY] info: file 'example_slide.cu' statisitics:
  CONVERTED refs count: 18
  TOTAL lines of code: 56
  WARNINGS: 0
[HIPIFY] info: CONVERTED refs by names:
  cuda.h => hip/hip_runtime.h: 1
  cudaError_t => hipError_t: 1
  cudaFree => hipFree: 3
  cudaGetErrorName => hipGetErrorName: 1
  cudaMalloc => hipMalloc: 3
  cudaMemcpy => hipMemcpy: 3
  cudaMemcpyDeviceToHost => hipMemcpyDeviceToHost: 1
  cudaMemcpyHostToDevice => hipMemcpyHostToDevice: 2
  cudaSuccess => hipSuccess: 1
```

# Hipify-perl example (2)

- `hipify-perl <file>.cu`
  - Translating a file to standard output

- Other options
  - `-inplace` Backup the input file in .prehip file, modify the input file inplace
  - Recursively do folders

# Use-cases


User application
C/C++
Fortran
No ACC code
⇩
HIP / Kokkos / Raja / OpenMP ...
CUDA code
⇩
HIP
OpenMP / OpenACC

# Fortran – OpenMP / OpenACC

- Incremental parallelism, minimal changes in the code (in principle)

| | CCE Compiler | AMD Compiler | GNU Compiler (v13+) | NVIDIA Compiler |
|---|---|---|---|---|
| OpenACC | X | | X | X |
| OpenMP | X | X | X | X |

  - Offload GNU compiler and NVIDIA compiler not available on ARCHER2
  - GNU reference at https://gcc.gnu.org/wiki/Offloading
  - Other info at:
    - https://docs.archer2.ac.uk/user-guide/gpu/#compilation-strategies-for-gpu-offloading
    - https://www.lumi-supercomputer.eu/offloading-code-with-compiler-directives/

- OpenACC well established in some communities (e.g. Climate)
- OpenMP gaining traction for new porting projects
  - Tool to migrate OpenACC to OpenMP: https://github.com/intel/intel-application-migration-tool-for-openacc-to-openmp

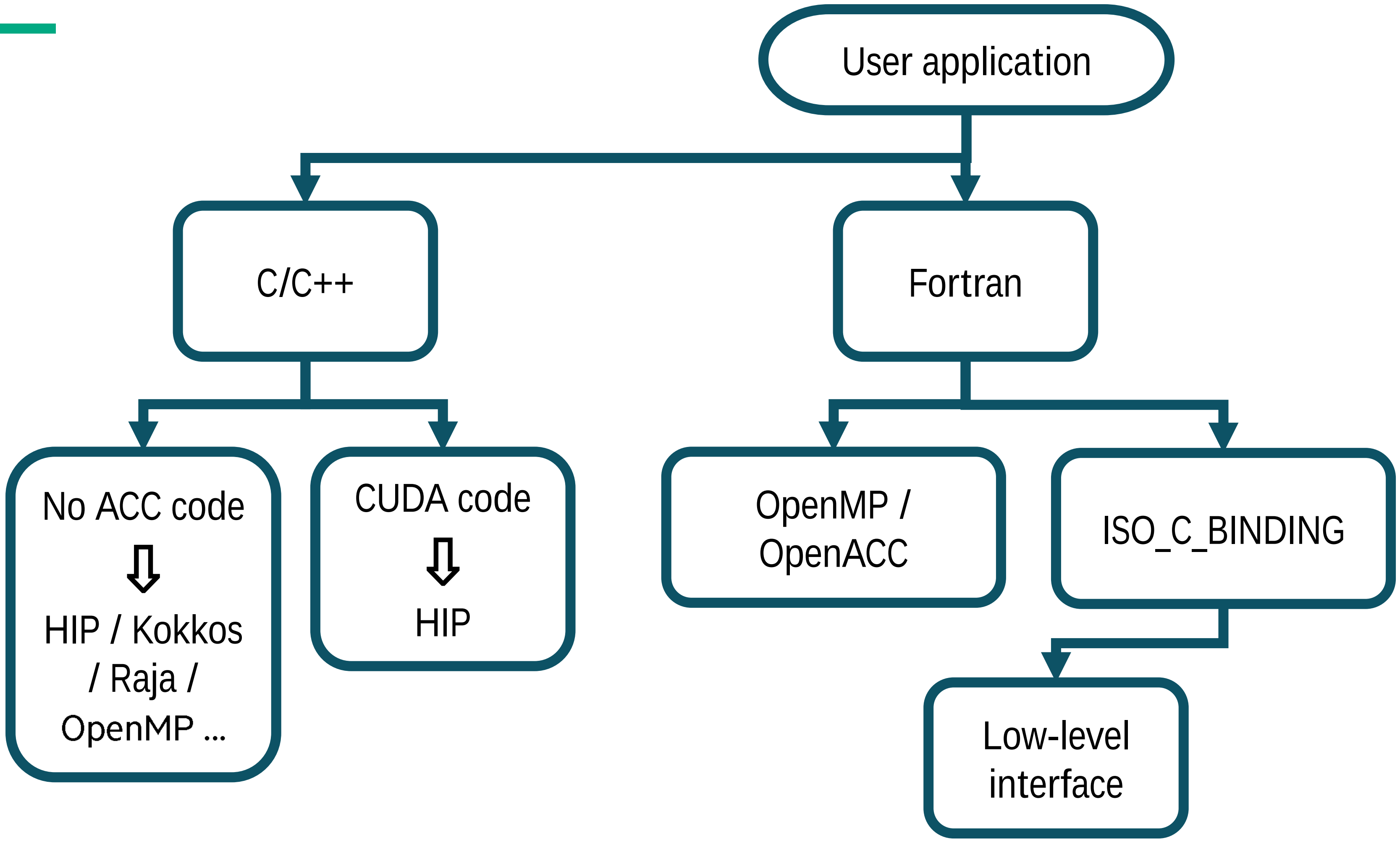
# Fortran – OpenMP / OpenACC – Performance

- You should not expect any difference in performance
  - Can use both in the same code baseline for comparison (and debugging) purpose
  - Device-side directives can be selected at compile time via macro
  - E.g. Yambo (https://github.com/yambo-code/yambo)

```
!DEV_OMP target enter data map(to: B, C) map(alloc: A)
!DEV_ACC enter data copyin(B, C) create(A)
!DEV_OMP target teams distribute parallel do simd
!DEV_ACC parallel loop
  do i = 1, n
    A(i) = B(i) + scalar * C(i)
  end do
!DEV_OMP end target teams distribute parallel do simd
!DEV_ACC end parallel loop
!DEV_OMP target exit data map(from: A)
!DEV_ACC exit data copyout(A)
```

```
#if defined _OPENACC
#  define DEV_ACC $acc
#else
#  define DEV_ACC !!!!
#endif

#if defined _OPENMP
#  define DEV_OMP $omp
#else
#  define DEV_OMP !!!!
#endif
```

# Use-cases


User application
C/C++
Fortran
No ACC code
⇩
HIP / Kokkos / Raja / OpenMP ...
CUDA code
⇩
HIP
OpenMP / OpenACC
ISO_C_BINDING
Low-level interface

# Fortran – Low-level interface

- Low-level interface approach limits HIP (C++) to relevant functions calls and kernels
  - Fortran code to implement the application logic
- Bind HIP functions and types via the ISO_C_BINDING module
  - Approach used by NEKO code, https://github.com/ExtremeFLOW/neko, file `src/device/hip_intf.F90`
- Example of interface

```
interface
   integer (c_int) function hipMalloc(ptr_d, s) &
        bind(c, name='hipMalloc')
     use, intrinsic :: iso_c_binding
     implicit none
     type(c_ptr) :: ptr_d
     integer(c_size_t), value :: s
   end function hipMalloc
end interface
```

- Kernels declared in C++
  - Fortran interfaces via ISO_C_BINDING module
- No equivalent of CUDA Fortran, which is not standard at all!

# Fortran – Hipfort (1)

- ROCm provides `hipfort`
  - Interfaces to main HIP and ROCm libraries
  - F2003 and F2008 interfaces

```c
#include <hip/hip_runtime.h>

__global__ void vector_add(double *out, double const *a,
                           double const *b, int N)
{
  size_t index = blockIdx.x * blockDim.x + threadIdx.x;

  if (index<N)
    out[index] = a[index] + b[index];
}


extern "C"
{
  void launch(double **dout, double **da, double **db, int N)
  {
    int num_threads = 256;
    int num_blocks = (N+num_threads-1)/num_threads;
    vector_add<<<num_blocks, num_threads>>>(*dout, *da, *db, N);
  }
}
```

```fortran
  use iso_c_binding
  use hipfort
  use hipfort_check

  implicit none

  interface
     subroutine launch(out,a,b,N) bind(c)
       use iso_c_binding
       implicit none
       type(c_ptr) :: a, b, out
       integer, value :: N
     end subroutine
  end interface

  type(c_ptr) :: da = c_null_ptr
  type(c_ptr) :: db = c_null_ptr
  type(c_ptr) :: dout = c_null_ptr

  integer, parameter :: N = 1000000
  integer, parameter :: bytes_per_element = 8 !double precision
  integer(c_size_t), parameter :: Nbytes = N*bytes_per_element

  ! Plain real should be equivalent to float
  double precision,allocatable,target,dimension(:) :: a, b, out

  integer :: i

  ! Allocate host memory
  allocate(a(N)) ; allocate(b(N)) ; allocate(out(N))

  ! Initialize host arrays
  a(:) = 1.0 ; b(:) = 2.0

  ! Allocate array space on the device
  call hipCheck(hipMalloc(da,Nbytes)) ; call hipCheck(hipMalloc(db,Nbytes)) ; call hipCheck(hipMalloc(dout,Nbytes))

  ! Transfer data from host to device memory
  call hipCheck(hipMemcpy(da, c_loc(a(1)), Nbytes, hipMemcpyHostToDevice))
  call hipCheck(hipMemcpy(db, c_loc(b(1)), Nbytes, hipMemcpyHostToDevice))

  call launch(dout,da,db,N)

  ! Transfer data back to host memory
  call hipCheck(hipMemcpy(c_loc(out(1)), dout, Nbytes, hipMemcpyDeviceToHost))

  call hipCheck(hipFree(da)) ; call hipCheck(hipFree(db)) ; call hipCheck(hipFree(dout))

  ! Deallocate host memory
  deallocate(a) ; deallocate(b) ; deallocate(out)

end program vecadd
```

- Example adapted from `https://github.com/ROCm/hipfort/tree/develop/test/f2003/vecadd`

# Fortran – Hipfort (2)

- Compile and link with `PrgEnv-amd`, adding `hipfort` includes and libraries, e.g.

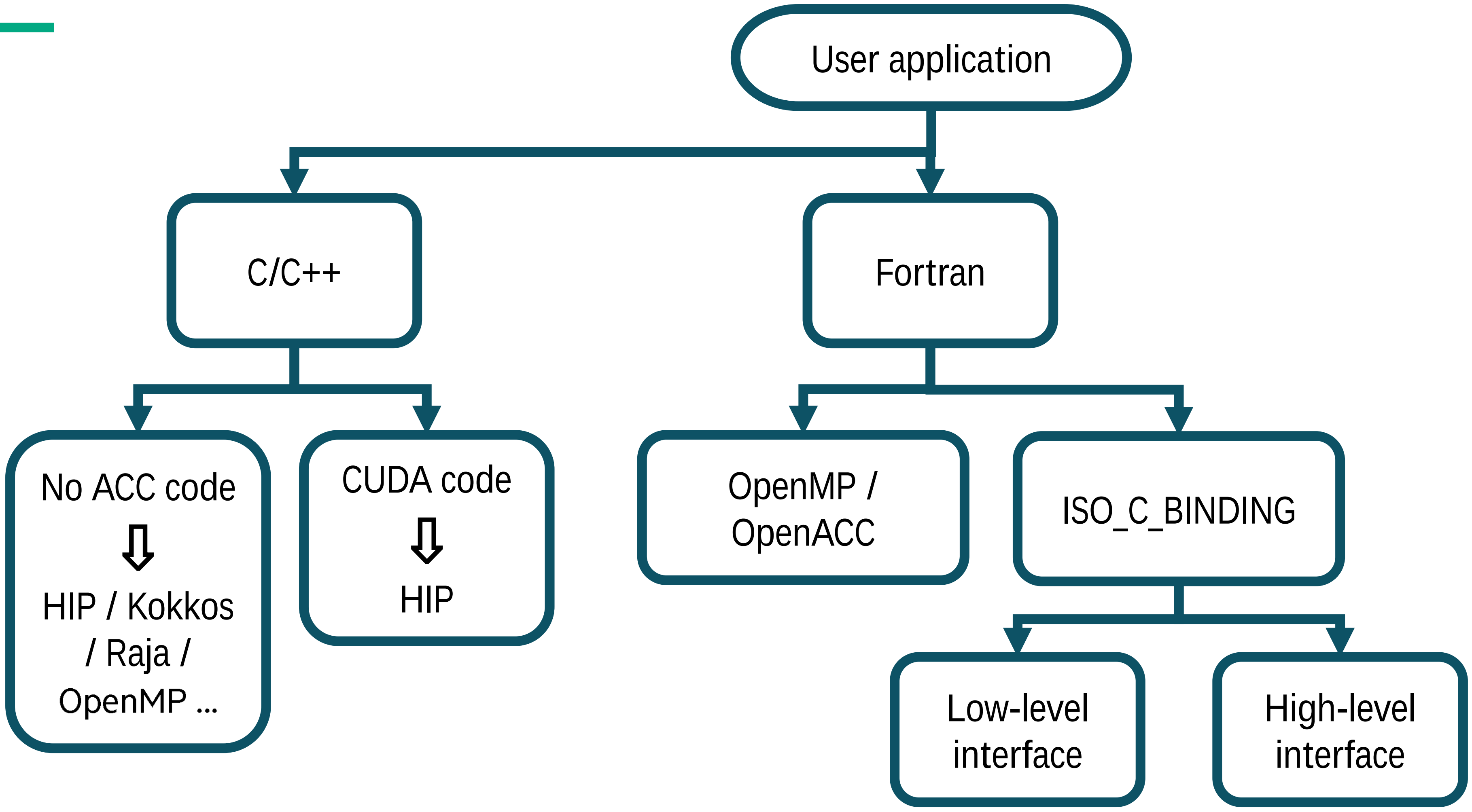
```
> CC -xhip -c vecadd_kernel.cpp
> ftn -I${ROCM_PATH}/include/hipfort/amdgcn vecadd.f03 vecadd_kernel.o \
  -L${ROCM_PATH}/lib -lhipfort-amdgcn -o vecadd.x
```

- Can use PrgEnv-cray but need to recompile hipfort with it to get compatible Fortran modules.

➢ https://github.com/ROCm/hipfort
➢ https://rocm.docs.amd.com/projects/hipfort/en/latest/

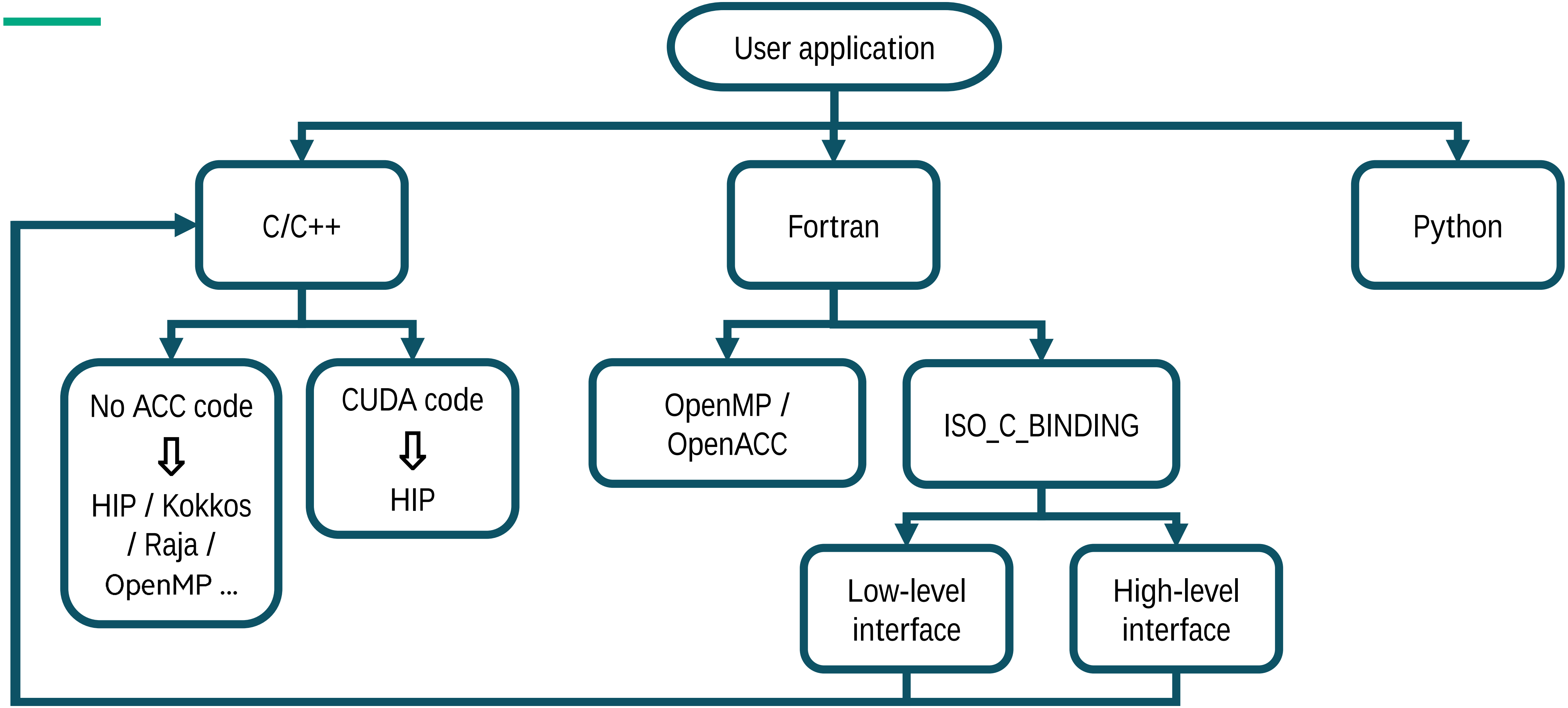
# Use-cases


User application
C/C++
Fortran
No ACC code
⇩
HIP / Kokkos / Raja / OpenMP …
CUDA code
⇩
HIP
OpenMP / OpenACC
ISO_C_BINDING
Low-level interface
High-level interface

# Fortran – High-level interface

- Implement GPU code and the related logic in C++
- Only few high-level interfaces provided in Fortran for GPU operations
- Good solution for complex existing Fortran codes
  - Separation of concerns between GPU (C/C++) and the existing Fortran code
- Example: DBCSR library (https://github.com/cp2k/dbcsr)

# Use-cases



User application
C/C++
Fortran
Python
No ACC code
⇩
HIP / Kokkos / Raja / OpenMP ...
CUDA code
⇩
HIP
OpenMP / OpenACC
ISO_C_BINDING
Low-level interface
High-level interface

# Python

- Low-level Python and Cython Bindings for HIP provided by HIP Python

➢ https://github.com/ROCm/hip-python
➢ https://rocm.docs.amd.com/projects/hip-python/en/latest/

There are many complex frameworks and libraries that implement GPU support

- CuPy (https://cupy.dev/) provides AMD support for computing with Python
- PyTorch

# Questions?