# OpenACC* 2 OpenMP* migration tool

Harald Servat PhD & Giacomo Rossi PhD and support of many Intel colleagues
AXG / DEE / TCE / XCSS

September 29th, 2021

# Directive-based parallel programming in OpenMP and OpenACC

- Shared-memory multi-core programming and heterogenous computing
- Industry standards
- C/C++/Fortran
- OpenMP (v5.1)
  - Not linked to a particular vendor, nor accelerator type
    - At runtime, all accelerators of the same kind
  - Multiple vendors and open-source platforms are coming
  - Shared multi-processors initially and supporting accelerators since 2013
- OpenACC (v3.1)
  - Works on NVIDIA GPUs
  - Adopted by many HPC users for heterogenous computing
  - Accelerator-centric since 2011 and adding multi-core support since 2018
- Interoperability between OpenACC and OpenMP?

# OpenACC 2 OpenMP migration tool

- A prototype for a source-2-source tool for translating OpenACC into OpenMP
  - Initially planned for Fortran, later extended to C/C++
  - Python-based
    - processing OpenACC pragmas / statements
    - not using AST generators currently
  - Assumes syntactically and functionally correct input
  - Resulting code can be compiled w/ either OpenACC or OpenMP compilers
    - Compilers can apply their own heuristics & optimizations
    - Easy to adopt forthcoming specification extensions
  - User can (has to) visually inspect for the translation correctness

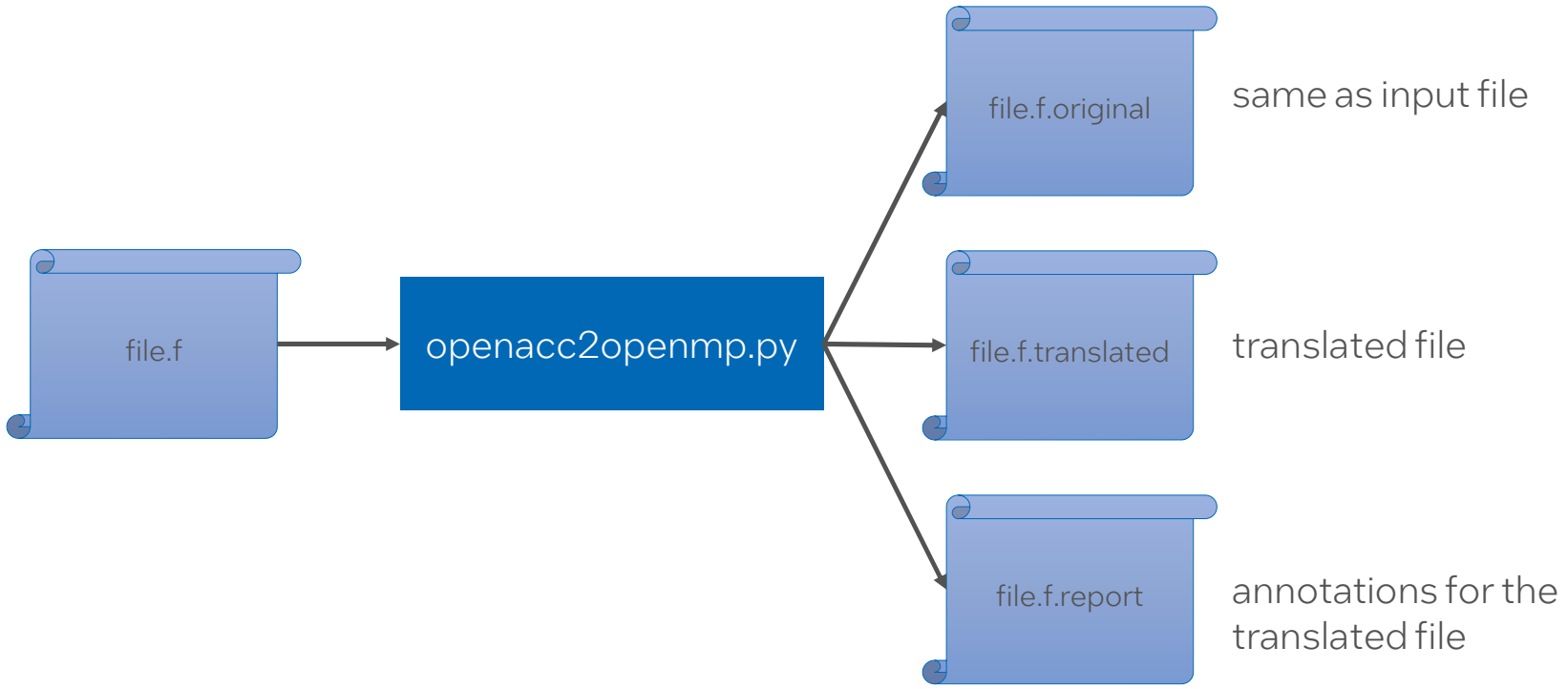
# Alternative OpenACC migration tools

- ACC2OMP by N. Romero [https://github.com/naromero77/ACC2OMP]
  - Fortran only
  - Did not succeed translating some mini-applications

- CLACC [FLACC?] by ORNL [https://csmd.ornl.gov/project/clacc]
  - Traditional compilation translates OpenACC source to an executable (using OpenMP runtime), and
  - Source-to-source mode translated OpenACC source to OpenMP source
  - What is the status of FLACC?

- GPUFORT by AMD [https://github.com/ROCmSoftwarePlatform/gpufort]
  - Fortran + OpenACC and CUDA fortran → Fortran + OpenMP 4.5+
  - Requires additional OpenACC runtime component based on GCC LIBGOMP and HIP++

# Migration process



# Migration tool knobs

```
$ ./openacc2openmp.py -help
Expected parameters: <Options> file

Where options:
 -h // -help                  : Shows this help screen
 -[no-]conditional-define     : If enabled, wraps translated code with #ifdef OPENACC2OPENMP_TRANSLATION. (enabled)
 -conditional-define=DEF      : If enabled, wraps translated code with #ifdef DEF.
 -[no-]force-backup           : If enabled, enforce writing a backup of the original file.               (disabled)
 -[no-]generate-report        : Enables/Disables report generation about the translation.                (enabled)
 -[no-]generate-multidimensional-alternate-code :                                                        (enabled)
                                Provides implementation suggestions for ACC ENTER/EXIT DATA constructs to be employed
                                if the multi-dimensional data is non-contiguous
 -loop=<ignore,collapse,keep> : Specifies how to treat ACC LOOP constructs.                              (ignore)
                                - collapse refers to turn ACC LOOP into COLLAPSED clauses (experimental).
                                - keep refers to translate ACC LOOP into OMP LOOP constructs (requires OpenMP 5.1+ compiler).
 -present=<alloc,tofrom,keep,hasdeviceaddr> : Specifies how to treat ACC LOOP constructs.               (alloc)
                                - alloc refers to mimic PRESENT clauses with MAP(ALLOC:)
                                - tofrom refers to mimic PRESENT clauses with MAP(TOFROM:)
                                - keep refers to use OMP PRESENT clauses (requires OpenMP 5.1+ compiler).
                                - hasdeviceaddr refers to use OMP HAS_DEVICE_ADDR clauses (requires OpenMP 5.1+ compiler).
 -async=<ignore,nowait>       : Specifies how to treat ACC ASYNC clauses.                                (nowait)
                                - ignore refers to not translate the ASYNC clauses.
                                - nowait translates ACC ASYNC clauses into nowait.
 -[no-]supress-openacc        : Enables/Disables supression of OpenACC translated statements in result.  (disabled)
 -[no-]overwrite-input        : Enables/Disables overwriting the original file with the translation.     (disabled)
```

# Demonstration

- openacc2openmp.git [bbcc727e3d64b91c1c6480680ad22d6e2f73d0d1]
- POT3D [r3.1.0] https://github.com/predsci/POT3D [42984a8ce428d54036d6b8a0732f05a046e8f840]
  - Edit Makefile
    - `FC = mpiifort -fc=ifx`
    - `FFLAGS = -fiopenmp -fopenmp-targets=spir64="-mllvm -vpo-paropt-enable-64bit-opencl-atomics=true" -O -I/nfs/home2/hservatg/apps/HDF5/1.12.1/include`
    - `LDFLAGS = -L/nfs/home2/hservatg/apps/HDF5/1.12.1/lib -Wl,-rpath -Wl,/nfs/home2/hservatg/apps/HDF5/1.12.1/lib -lhdf5_fortran -lhdf5_hl_fortran -lhdf5 -lhdf5_hl`
  - ~/src/openacc2openmp.git/src/openacc2openmp.py -loop=keep -async=ignore -no-conditional-define *.f
  - Use vimdiff to compare original and translated file
  - Modify cgdot reduction (file pot3d.f, line ~ 5550) to run reduction on the host and prepend
    - !$omp target update from(x,y)
  - Copy translated into original
  - Run make
  - Go into ../testsuite/validation/input
  - Run mpirun –np 2 ../../../src/pot3d
  - Compare pot3d.out against ../validation/pot3d.out
- ifx (IFORT) dev.x.0 Mainline 20210923

# OpenACC vs OpenMP discussion

# #pragma acc kernels

- #pragma acc kernels construct is a hint (vs #pragma acc parallel which is an assertion)
  - compiler automatically extracts parallelism from the code
  - typically, each loop nest will be a distinct kernel
  - gangs, workers and vector length may be different for each kernel

- Migration issues
  - no semantically equivalent in OpenMP
  - not necessarily applied on top of a loop
  - loop siblings and loop nests may need to be parallelized independently
    - AST may help but there is also work to identify parallelization opportunities

```
#pragma acc kernels
{
        for (i = 0; i < N; ++i)
        {
                y[i] = 0.0f;
                x[i] = (float)(i+1);
        }
        for (i = 0; i < N; ++i)
        {
                y[i] = 2.0 * x[i] + y[i];
        }
}
```

```
!$acc kernels
do i = 1, N
        y(i) = 0
        x(i) = i
enddo
do i = 1, N
        y(i) = 2.0 * x(i) + y(i)
enddo
!$acc end kernels
```

Code from OpenACC Programming and Best Practices Guide

# #pragma acc kernels

- #pragma acc kernels gets translated into target teams (serial)
  - … except if combined with loop

```
#pragma acc kernels
#pragma omp target
{
    for (i = 0; i < N; ++i)
    {
        y[i] = 0.0f;
        x[i] = (float)(i+1);
    }
    for (i = 0; i < N; ++i)
    {
        y[i] = 2.0 * x[i] + y[i];
    }
}
```

```
#pragma acc kernels loop
#pragma omp target teams distribute parallel for
for (i = 0; i < N; ++i)
{
  y[i] = 2.0 * x[i] + y[i];
}
```

# Alignment between the two languages (present)

- #pragma acc … present(X)
  - -present=keep
    - Use OpenMP 5.1 present clause but not supported on Intel compilers yet
  - -present=alloc
    - semantically different
    - allocate if not already allocated
  - -present=tofrom
    - semantically different (host-centric)
    - perf overhead
    - for debugging purposes?
  - -present=has_device_addr
    - use OpenMP 5.1 has_device_addr but not fully supported on Intel compilers yet
    - more to come in OpenMP 5.2
- default(present)
  - not currently supported but could map to defaultmap(X)

```
#pragma acc parallel loop present(X)
```

```
[tentative/5.1]
#pragma omp target teams distribute parallel for
map(present,tofrom:X)
```

```
#pragma omp target teams distribute parallel for
map(alloc:X)
```

```
#pragma omp target teams distribute parallel for
map(tofrom:X)
```

```
[tentative/5.1+]
#pragma omp target teams distribute parallel for
has_device_addr(X)
```

# Alignment between the two languages (loop)

- #pragma acc loop
  - Similar construct in OpenMP 5.0
    - use –loop=keep
  - On compilers supporting OpenMP 4.5 or older
    - -loop=collapse → tries to collapse loops but has some limitations
    - -loop=ignore → ignores the loop constructs
- #pragma acc loop can specify num gangs, num workers and vector size if not specified by parent construct
  - not available in OpenMP's loop, though

`-loop=keep`

```
#pragma acc parallel loop
#pragma omp target teams distribute parallel for
for (auto i = 0; i < N; ++i)
        #pragma acc loop
        #pragma omp loop
        for (auto j = 0; j < M; ++j)
        {
                vec[i][j[] += 1.0;
        }
```

`-loop=collapse`

```
#pragma acc parallel loop
#pragma omp target teams distribute parallel for collapse(2)
for (auto i = 0; i < N; ++i)
        #pragma acc loop
        for (auto j = 0; j < M; ++j)
        {
                vec[i][j[] += 1.0;
        }
```

# Asynchronism

- OpenACC's async(x) clause
  - if x is non-negative, then is used to select the queue (stream) on the current device onto which to enqueue an operation
    - successive clauses on the same queue will be executed sequentially
  - if x is negative, the behavior is implementation defined
    - acc_async_noval, acc_async_sync

- OpenMP's async mechanisms
  - #pragma omp target … nowait → may be sufficient to mimic on simple cases
  - depend clause → express chain of dependencies and mimic OpenACC's async
    - turn #pragma acc … async(1) into #pragma omp target .. depend (inout: dep_async_1)
    - turn #pragma acc wait (1) into #pragma omp target .. depend (in:dep_async_1)

    → need to define dep_async_X list item or depobj in some headers / modules

```
6070 !$acc parallel loop default(present) collapse(3) async(1)$
6071 !$omp target teams distribute parallel do collapse(3) nowait$
6072       do k=1,np$
6073         do j=1,nt$
6074           do i=1,nrm1$
6075             br(i,j,k)=(phi(i+1,j,k)-phi(i,j,k))/dr(i)$
6076           enddo$
6077         enddo$
6078       enddo$
6079 c$
6080 c ****** Get Bt.$
6081 c$
6082 !$acc parallel loop default(present) collapse(3) async(2)$
6083 !$omp target teams distribute parallel do collapse(3) nowait$
6084       do k=1,np$
6085         do j=1,ntm1$
6086           do i=1,nr$
6087             bt(i,j,k)=(phi(i,j+1,k)-phi(i,j,k))/(rh(i)*dt(j))$
6088           enddo$
6089         enddo$
6090       enddo$
6091 c$
6092 c ****** Get Bp.$
6093 c$
6094 !$acc parallel loop default(present) collapse(3) async(3)$
6095 !$omp target teams distribute parallel do collapse(3) nowait$
6096       do k=1,npm1$
6097         do j=1,nt$
6098           do i=1,nr$
6099             bp(i,j,k)=(phi(i,j,k+1)-phi(i,j,k))/(rh(i)*sth(j)*dp(k))$
6100           enddo$
6101         enddo$
6102       enddo$
6103 !$acc wait$
6104 !$omp taskwait$
```

# Non-contiguous data-mapping

- OpenACC supports data mapping of non-contiguous arrays (e.g. pointers to pointers)
  - At what performance cost?
- Not supported directly in OpenMP with Intel compiler
  - OpenMP 5.0 defines *iterators* but not implemented on Intel compiler yet

    → migration tool suggests an alternative code that transfers data on per-row basis

```
127     #pragma acc enter data copyin(this[0:1])$
128 #pragma omp target enter data map(to:this[0:1])$
129     #pragma acc enter data copyin(_filter[0:_size][0:_size])$
130 #pragma omp target enter data map(to: filter[0: size][0: size])$
131 // ATTENTION! The following suggested code is an alternative reference implementation$
132 // ATTENTION! that could be used if _filter is a non-contiguous allocated multi-dimensional array$
133 // #pragma omp target enter data map(alloc:_filter[0:_size])$
134 // for (auto _idx0 = 0; _idx0 = _size; ++_idx0)$
135 // {$
136 //   #pragma omp target enter data map(to:_filter[_idx0][0:_size])$
137 // }$
```

# Seq clause

- OpenACC supports the seq clause on loop and routine constructs
  - Override any automatic parallelization or vectorization
    - Not to be executed in parallel by gangs/threads
    - Not to be executed in a vector fashion
- Nothing similar in OpenMP
  - Cannot specify thread_limit nor simdlen on #pragma omp loop and #pragma declare target
- Currently ignored

```
#pragma acc routine seq
float func(float, float);

#pragma acc loop seq
for (auto i = 0; i < N; ++i)
  data[k] += _data[k][i];
```

# Performance portability

- OpenACC 3-levels of parallelism (gangs, workers, vector ) can be mapped to similar clauses in OpenMP (teams, threads and vector)
  - Mapping on hardware with different characteristics

```
#pragma acc parallel loop num_gangs(NG)
num_workers(NW)
#pragma omp target teams distribute num_teams(NG)
thread_limit(NW)
for (auto i = 0; i < N; ++i)
{
  #pragma acc loop gang
  #pragma omp loop bind(teams)
  for (auto j = 0; j < M; ++j)
  {
  ..
  }
  #pragma acc loop worker
  #pragma omp loop bind(thread)
  for (auto j = 0; j < M; ++j)
  {
  ..
  }
}
```

# Interoperability with the runtime

- OpenACC API supports interaction with CUDA/OpenCL
  - e.g.
    - acc_get_current_cuda_device/context,
    - acc_set/get_cuda_stream,
    - acc_get_current_opencl_device/context,
- OpenMP 5.0 includes “interop” constructs into the spec
  - https://github.com/OpenMP/Examples/blob/v5.1/program_control/sources/interop.1.c
    - OpenMP and CUDA interaction
  - https://github.com/argonne-lcf/HPC-Patterns/blob/main/interop_omp_ze_sycl.cpp
    - OpenMP and SYCL interaction
- Applications using these features?

# Conclusions and future work

- We've shown a prototype tool for helping migrate Fortran and C/C++ OpenACC codes into OpenMP

- We're gathering feedback for more complex applications
  - Extend support for additional OpenACC constructs, applications
    - AST likely to help, but what AST generator for Fortran?
- Performance suggestions/evaluations
  - Guidance on perf hints (gangs, workers, vector length, collapse, transfers…) ?
  - OpenACC vs translated OpenMP using different compilers & hw (HPC SDK, clang/LLVM on NVidia hw and Intel compilers)


intel