Interoperability Between OpenACC and OpenMP4.5

Lixiang Luo, IBM Research

Introduction

This study explores a practical way to mix two primary paradigms for accelerators – OpenACC and OpenMP4.5 – in the same code. There are several reasons to do so:

- Incremental porting without starting over from the host codes (error-prone, and sometimes impossible)
- To use libraries written in another paradigm
- Performance studies for different paradigms
- Debugging

OpenACC and OpenMP4.5 share a similar programming model, and many directives have a one-to-one mapping between the two paradigms. Unfortunately, no compiler can fully support both paradigms interchangeably, therefore the interoperability is a prerequisite for any code which uses both paradigms.

The original OpenACC code

For simplicity, this study is focused on PGI (for OpenACC) and IBM XL (for OpenMP4.5) on IBM "Minsky" systems (POWER8, Nvidia P100). The original OpenACC Fortran code looks like:

Main <main-pgi. F90>

```
program mixaccomp4
use modsub1
use modsub2
implicit none

integer, parameter :: n=10
real*8 :: arr(n)
integer :: i

forall(i=1:n) arr(i)=i
multiple=3.

!$acc data copy(arr)
call sub1(n,arr)
call sub2(n,arr)
!$acc end data

write(*,'(10F6.1)') arr

end program mixaccomp4
```

Sub1 <modsub1-pgi.F90>

```
module modsub1
implicit none

real*8 :: delta=0.1

contains

subroutine sub1(n,a)
implicit none
integer, intent(in) :: n
real*8, dimension(n) :: a
integer :: i

!$acc kernels present(a)
do i=1,n
  a(i)=a(i)+delta
end do
!$acc end kernels

end subroutine sub1

end module modsub1
```

Sub2 <modsub2-pgi.F90>

```
module modsub2
implicit none

real*8 :: multiple=2.0

contains

subroutine sub2(n,a)
implicit none
integer, intent(in) :: n
real*8, dimension(n) :: a
integer :: i

!$acc kernels present(a)
do i=1,n
  a(i)=a(i)*multiple
end do
!$acc end kernels

end subroutine sub2

end module modsub2
```

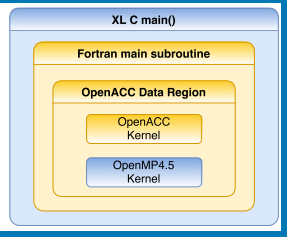
The challenges come from three fronts:

- Interoperability of Fortran implementations and runtimes
- Interoperability of OpenACC and OpenMP4.5 runtimes
- Complications from compilation and source files

Incremental porting

To demonstrate the possibility of incremental porting, only one of the two OpenACC kernels are ported to OpenMP4.5. Data management is still handled by OpenACC.

Incompatible Fortran object name mangling by the two compiler suits can be resolved by using the BIND(C) attribute, introduced in Fortran 2003, on all objects that need to be used by codes compiled by both compilers ("delta" in this example).



Original	Ported

Sub1 <modsub1-pgi.F90>
…
real*8 :: delta=0.1
…
!$acc kernels present(a)
do i=1,n
  a(i)=a(i)+delta
end do
!$acc end kernels
end subroutine sub1
…
	Sub1 <modsub1-pgi.F90>
	…
	real*8, bind(c) :: delta=0.1
	…
	interface
	 subroutine sub1omp4(nn,aa) bind(c)
	  integer :: nn
	  real*8 :: aa(*)
	 end subroutine
	end interface
	!$acc data present(a)
	call xlomp4_assoc(a,a,n)
	call sub1omp4(n,a)
	call xlomp4_disassoc(a)
	!$acc end data
	end subroutine sub1
	
	Sub1 OpenMP4 <ksub1omp4-xl.F90>
	module modsub1omp4
	real*8, bind(C) :: delta
	contains
	subroutine sub1omp4(n,a) bind(C)
	implicit none
	integer :: n
	real*8, dimension(n) :: a
	integer :: i
	!$omp target teams distribute parallel do simd
	do i=1,n
	  a(i)=a(i)+delta
	end do
	end subroutine sub1omp4
	end module modsub1omp4

GPU kernels generated by OpenACC and OpenMP4.5 can work together in the same code. Also, OpenMP4.5 provides a method to manipulate its present table. Along with a method to extract device pointer from OpenACC, we can duplicate the OpenACC present table in OpenMP4.5, so that OpenACC variables can be directly used in OpenMP4.5 kernels, as if they are managed by OpenMP4.5 all along.

OpenMP4 device pointer association <memassoc-xl.F90>

```
#include <omp.h>
void xlomp4_assoc(double *a_h, double *a_d, int *size)
{
  omp_target_associate_ptr(a_h,a_d,*size*sizeof(double),0,0);
}

void xlomp4_disassoc(double *a_h)
{
  omp_target_disassociate_ptr(a_h,0);
}
```

PGI CUDA Fortran wrapper for device pointer association <memassoc_wrappers-pgi.F90>

```
module omp4_wrappers
interface
  subroutine xlomp4_assoc(a_h,a_d,nv) bind(c)
    real*8, dimension(*) :: a_h
    real*8, dimension(*), device :: a_d
    integer :: nv
  end subroutine

  subroutine xlomp4_disassoc(a_h) bind(c)
    real*8, dimension(*) :: a_h
  end subroutine
end interface
end module omp4_wrappers
```

Currently, the XL Fortran compiler must be used as the host linker. The original PGI main program is turned into a subroutine, and then called by a wrapper XL C main program. A dummy OpenMP4.5 kernel is placed before the PGI main subroutine, so that the OpenMP4.5 offloading environment is initialized before the initialization of OpenACC. If not done, OpenMP4.5 kernels will fail inside OpenACC data regions.

Original	Ported

Main program <main-pgi.F90>
program mixaccomp4
…
end program mixaccomp4
	Main subroutine <main-pgi.F90>
	subroutine mixaccomp4
	…
	end subroutine mixaccomp4
	
	New main program <main-xl.c>
	int main(int argc,char *argv[])
	{
	  int i;
	#pragma omp target teams
	  i=0;
	  mixaccomp4_();
	  return 0;
	}

Mix-compiling and results

PGI Fortran and OpenACC runtime libraries need to be linked manually. Also, "nordc" must be specified when compiling OpenACC kernels. The Makefile looks like:

```
modules := memassoc_wrappers-pgi.o modsub1-pgi.o modsub2-pgi.o ksub1omp4-xl.o
objects := main-pgi.o main-xl.o memassoc-xl.o

acclibs := -L$(PGI_ROOT)/lib -laccapi -laccg -laccn -laccg2 -ldl -lcudadevice -lpthread -lpgc -lpgkomp -lomp -lpgf90 -lpgf902 -lpgf90_rpm1 -lpgf90rtl -laccnc

.PHONY : xlomp4
xlomp4 : run_xlomp4

$(objects) : $(modules)

run_% : $(objects) $(modules)
  xlf_r -O2 -qsmp=omp -qoffload -o $@ $(objects) $(modules) $(acclibs)

%-pgi.o : %-pgi.F90
  pgf90 -Mcuda -ta=tesla:nordc -c $< -o $@

%-xl.o : %-xl.F90
  xlf_r -O2 -qsmp=omp -qoffload -c $< -o $@

%-pgi.o : %-pgi.c
  pgcc -O0 -ta=tesla:nordc -c $< -o $@

%-xl.o : %-xl.c
  xlc_r -O0 -qsmp=omp -qoffload -c $< -o $@
```

The expected output is

```
   1.0   2.0   3.0   4.0   5.0   6.0   7.0   8.0   9.0  10.0
   1.1   2.1   3.1   4.1   5.1   6.1   7.1   8.1   9.1  10.1
   3.3   6.3   9.3  12.3  15.3  18.3  21.3  24.3  27.3  30.3
```

Conclusion and known limitations

- It is possible to mix OpenACC and OpenMP4.5 in one code.
- OpenACC variables can be used in OpenMP4.5 kernels.
- Careful use of vender-independent coding practice allows incremental porting from either direction.

Several known limitations

- Implementation-dependent data objects (such as OOP objects) cannot be shared across compiler boundary.
- PGI main program cannot access command line arguments
- "nordc" disables the use of external device functions.