CONCLUDING REMARKS
BRENT LEBACK, MEMBER OF THE NVIDIA HPC SDK TEAM
PROGRAMMING THE NVIDIA PLATFORM
CPU, GPU, and Network

ACCELERATED STANDARD LANGUAGES
ISO C++, ISO Fortran

std::transform(par, x, x+n, y, y,
    [=](float x, float y) { return y + a*x; })

do concurrent (i = 1:n)
    y(i) = y(i) + a*x(i)
endo

import cunumeric as np
... def saxpy(a, x, y):
    y[:] += a*x

#pragma acc data copy(x,y) {
...

#pragma omp target data map(x,y) {
...

INCERMENTAL PORTABLE OPTIMIZATION
OpenACC, OpenMP

GLOBAL
void saxpy(int n, float a,
    float *x, float *y) {
    int i = blockIdx.x*blockDim.x + threadIdx.x;
    if (i < n) y[i] += a*x[i];
}

int main(void) {
...

PLATFORM SPECIALIZATION
CUDA

__global__
void saxpy(int n, float a,
    float *x, float *y) {
    int i = blockIdx.x*blockDim.x + threadIdx.x;
    if (i < n) y[i] += a*x[i];
}

int main(void) {
...

cudaMemcpy(d_x, x, ...);
cudaMemcpy(d_y, y, ...);
saxpy<<<(N+255)/256,256>>>(...);
cudaMemcpy(y, d_y, ...);

ACCELERATION LIBRARIES
Core Math Communication Data Analytics AI Quantum
WHAT DO WE MEAN BY INTEROPERABLE?

Different programming models can appear in the same source file

Objects from different programming models can be linked together into the same program

One programming model can use data declared/defined/initialized in a different programming model

One programming model can call kernels or device functions written in another programming model

Programming models can share attributes of the device, such as the current device, current context, and streams
SOME SOURCE CODE SHORTCUTS

% cat t1.f90
$ print *,"Compiled for OpenMP"
!@acc print *,"Compiled for OpenACC"
!@cuf print *,"Compiled for CUDA Fortran"
stop
end
% for op1 in "" -mp; do for op2 in "" -acc; do for op3 in "" -cuda; do nvfortran $op1 $op2 $op3 t1.f90; .;/a.out; done; done; done
FORTRAN STOP
Compiled for CUDA Fortran
FORTRAN STOP
Compiled for OpenACC
FORTRAN STOP
Compiled for CUDA Fortran
FORTRAN STOP
Compiled for OpenMP
FORTRAN STOP
Compiled for CUDA Fortran
FORTRAN STOP
Compiled for OpenMP
FORTRAN STOP
Compiled for CUDA Fortran
FORTRAN STOP
Compiled for OpenMP
Compiled for OpenACC
FORTRAN STOP
Compiled for CUDA Fortran
FORTRAN STOP
Compiled for OpenMP
Compiled for OpenACC
FORTRAN STOP
Compiled for CUDA Fortran

% cat t2.F90
!This is equivalent
#ifdef _OPENMP
print *,"Compiled for OpenMP"
#endif
#ifdef _OPENACC
print *,"Compiled for OpenACC"
#endif
#ifdef _CUDA
print *,"Compiled for CUDA Fortran"
#endif
stop
end
Use nvcc to compile CUDA C/C++
Use nvc or nvc++ to compile OpenMP
Calling CUDA Libraries with host-side interfaces does not require nvcc; use the -cuda and -cudalib options for easier compiling and linking.
By default, nvc and nvc++ generate relocatable device code (rdc). The nvcc compiler does not. Be aware of that.
We are working on C++ stdpar interoperability with pragma-based data directives. It is a hard problem.
FORTRAN OPEN[MP | ACC], STDPAR + <EVERYTHING ELSE>

Use nvfortran to compile for all models

Fortran calling C is well-defined, as is CUDA Fortran + CUDA C.

The NVIDIA HPC SDK contains Fortran modules for interfacing to the CUDA Libraries. Use the -cudalib option because some interfaces require extra wrapper libraries.

Because OpenMP defines a host-fallback mode, some cases which work with OpenACC+CUDA are not quite right yet with OpenMP+CUDA. We are working on it.

We need to define/decide if/how we allow non-Fortran features in do concurrent.