CUDA and nvcc: using the preprocessor to choose between float or double

Asked 13 years, 4 months ago Modified 13 years, 4 months ago Viewed 25k times

8

The problem:

Having a .h, I want to define real to be double if compiling for c/c++ or for cuda with computing capability >= 1.3. If compiling for cuda with computing capability < 1.3 then define real to be float.

After many hours I came to this (which does not work)

```
#   if defined(__CUDACC__)

#       warning * making definitions for cuda

#       if defined(__CUDA_ARCH__)
#           warning __CUDA_ARCH__ is defined
#       else
#           warning __CUDA_ARCH__ is NOT defined
#       endif

#       if (__CUDA_ARCH__ >= 130)
#                       define real double
#                       warning using double in cuda
#       elif (__CUDA_ARCH__ >= 0)
#               define real float
#               warning using float in cuda
#               warning how the hell is this printed when __CUDA_ARCH__ is not defined?
#       else
#               define real
#               error what the hell is the value of __CUDA_ARCH__ and how can I print it
#       endif

#   else
#       warning * making definitions for c/c++
#       define real double
#       warning using double for c/c++
#   endif
```

when I compile (note the -arch flag)

```
nvcc -arch compute_13  -Ilibcutil testFloatDouble.cu
```

I get

```
* making definitions for cuda
__CUDA_ARCH__ is defined
```

```
using double in cuda

* making definitions for cuda
warning __CUDA_ARCH__ is NOT defined
warning using float in cuda
how the hell is this printed if __CUDA_ARCH__ is not defined now?

Undefined symbols for architecture i386:
  "myKernel(float*, int)", referenced from: ....
```

I know that files get compiled twice by nvcc. The first one is OK (**CUDACC** defined and **CUDA_ARCH** >= 130) but what happens the second time? **CUDA_DEFINED** but **CUDA_ARCH** undefined or with value < 130? Why ?

Thanks for your time.

cuda preprocessor nvcc

Share Improve this question Follow

edited Jan 10, 2012 at 11:04
talonmies
72.5k 35 203 289

asked Jan 9, 2012 at 22:57
cibercitizen1
21.5k 17 78 100

2 Answers

Sorted by: Highest score (default)

39

It seems you might be conflating two things - how to differentiate between the host and device compilation trajectories when nvcc is processing CUDA code, and how to differentiate between CUDA and non-CUDA code. There is a subtle difference between the two. `__CUDA_ARCH__` answers the first question, and `__CUDACC__` answers the second.

Consider the following code snippet:

```
#ifdef __CUDACC__
#warning using nvcc

template <typename T>
__global__ void add(T *x, T *y, T *z)
{
    int idx = threadIdx.x + blockDim.x * blockIdx.x;

    z[idx] = x[idx] + y[idx];
}

#ifdef __CUDA_ARCH__
#warning device code trajectory
#if __CUDA_ARCH__ > 120
#warning compiling with double precision
template void add<double>(double *, double *, double *);
#else
#warning compiling with single precision
```

```
template void add<float>(float *, float *, float *);
#else
#warning nvcc host code trajectory
#endif
#else
#warning non-nvcc code trajectory
#endif
```

Here we have a templated CUDA kernel with CUDA architecture dependent instantiation, a separate stanza for host code steeered by `nvcc`, and a stanza for compilation of host code not steered by `nvcc`. This behaves as follows:

```
$ ln -s cudaarch.cu cudaarch.cc
$ gcc -c cudaarch.cc -o cudaarch.o
cudaarch.cc:26:2: warning: #warning non-nvcc code trajectory

$ nvcc -arch=sm_11 -Xptxas="-v" -c cudaarch.cu -o cudaarch.cu.o
cudaarch.cu:3:2: warning: #warning using nvcc
cudaarch.cu:14:2: warning: #warning device code trajectory
cudaarch.cu:19:2: warning: #warning compiling with single precision
cudaarch.cu:3:2: warning: #warning using nvcc
cudaarch.cu:23:2: warning: #warning nvcc host code trajectory
ptxas info    : Compiling entry function '_Z3addIfEvPT_S1_S1_' for 'sm_11'
ptxas info    : Used 4 registers, 12+16 bytes smem

$ nvcc -arch=sm_20 -Xptxas="-v" -c cudaarch.cu -o cudaarch.cu.o
cudaarch.cu:3:2: warning: #warning using nvcc
cudaarch.cu:14:2: warning: #warning device code trajectory
cudaarch.cu:16:2: warning: #warning compiling with double precision
cudaarch.cu:3:2: warning: #warning using nvcc
cudaarch.cu:23:2: warning: #warning nvcc host code trajectory
ptxas info    : Compiling entry function '_Z3addIdEvPT_S1_S1_' for 'sm_20'
ptxas info    : Used 8 registers, 44 bytes cmem[0]
```

The take away points from this are:

- `__CUDACC__` defines whether `nvcc` is steering compilation or not
- `__CUDA_ARCH__` is *always* undefined when compiling host code, steered by `nvcc` or not
- `__CUDA_ARCH__` is only defined for the device code trajectory of compilation steered by `nvcc`

Those three pieces of information are always enough to have conditional compilation for device code to different CUDA architectures, host side CUDA code, and code not compiled by `nvcc` at all. The `nvcc` documentation is a bit terse at times, but all of this is covered in the discussion on compilation trajectories.

Share Improve this answer Follow

answered Jan 10, 2012 at 20:17

talonmies
72.5k 35 203 289

Very good contribution, I keep it. Unfortunately for my actual problem I should check if it is feasible. We are porting/extending an existing c/c++ library (a lot of .h and .c) written with double to be usable for cuda (float and double). We replaced double to real and now want to have a consistent definition of real to be double or to be float depending on the situation. We have to think about how mechanically translate the current function headers to be templates and, most important, whether that is acceptable for users wanting to use pure C. Thanks. – cibercitizen1 Jan 10, 2012 at 20:34

Also, I didn't realize that your definition of add() is *inside* the #ifdef **CUDACC**. BUT it should be also available to c/c++ code for using it. – cibercitizen1 Jan 10, 2012 at 21:10

On the last point - no. By definition, device code and the host code which will call a cuda kernel using the kernel<<<>>> syntax must be compiled with nvcc. And a kernel definition must be available to both the host and device trajectories of the compilation, because the host side requires an internally generated entry stub function for the kernel call. If you want to call a kernel from regular C or C++ code, you either need a C/C++ wrapper inside a .cu file, or use the driver API to load a cubin or JIT compile PTX. – talonmies Jan 10, 2012 at 21:30

Apart of the double/float choice, the kernels are not a problem for us. The library functions are converted, in cuda, to DEVICE functions (being DEVICE **device** #if **CUDACC** and nothing otherwise). That way, such functions are both available to c/c++ or to cuda. – cibercitizen1 Jan 10, 2012 at 21:53

1 I believe that `__CUDA_ARCH__` is defined when `nvcc` is parsing any code, even host code. It seems to be defined to `0` in host code. – Jared Hoberock Jan 10, 2012 at 22:30

3

For the moment the only practical solution I see is using a custom define:

```
#   if (!defined(__CUDACC__) ||  defined(USE_DOUBLE_IN_CUDA))
#       define real double
#       warning defining double for cuda or c/c++
#   else
#       define real float
#       warning defining float for cuda
#   endif
```

and then

```
nvcc -DUSE_DOUBLE_IN_CUDA -arch compute_13  -Ilibcutil testFloatDouble.cu
```

As it outputs the for the two compilations:

```
#warning defining double for cuda or c/c++
#warning defining double for cuda or c/c++
```

and

```
nvcc  -Ilibcutil testFloatDouble.cu
```

does