Computational Science on Many-Core Architectures

360.252

Karl Rupp

Institute for Microelectronics
Vienna University of Technology
http://www.iue.tuwien.ac.at

Zoom Channel 95028746244
Wednesday, January 13, 2021

Agenda for Today

Exercise 9 Recap

HIP

SYCL

Exercise 10

Exercise 9 Recap

Kernel

- How was your experience?

Exercise 9 Recap

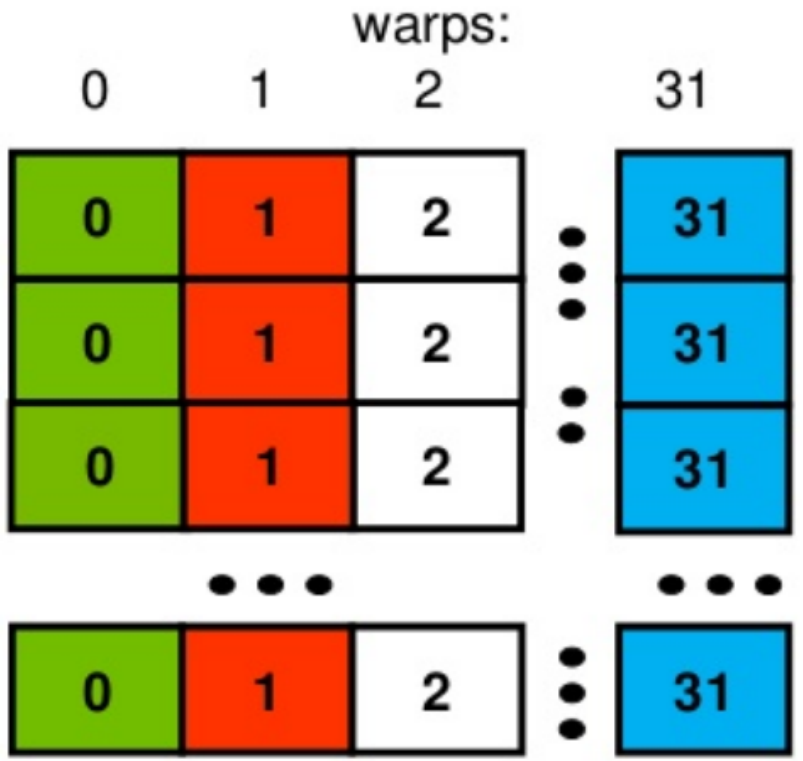
Kernel

- How was your experience?

Note on Shared Memory Banks

- Shared memory is organized in 32 banks of 32/64 bits each

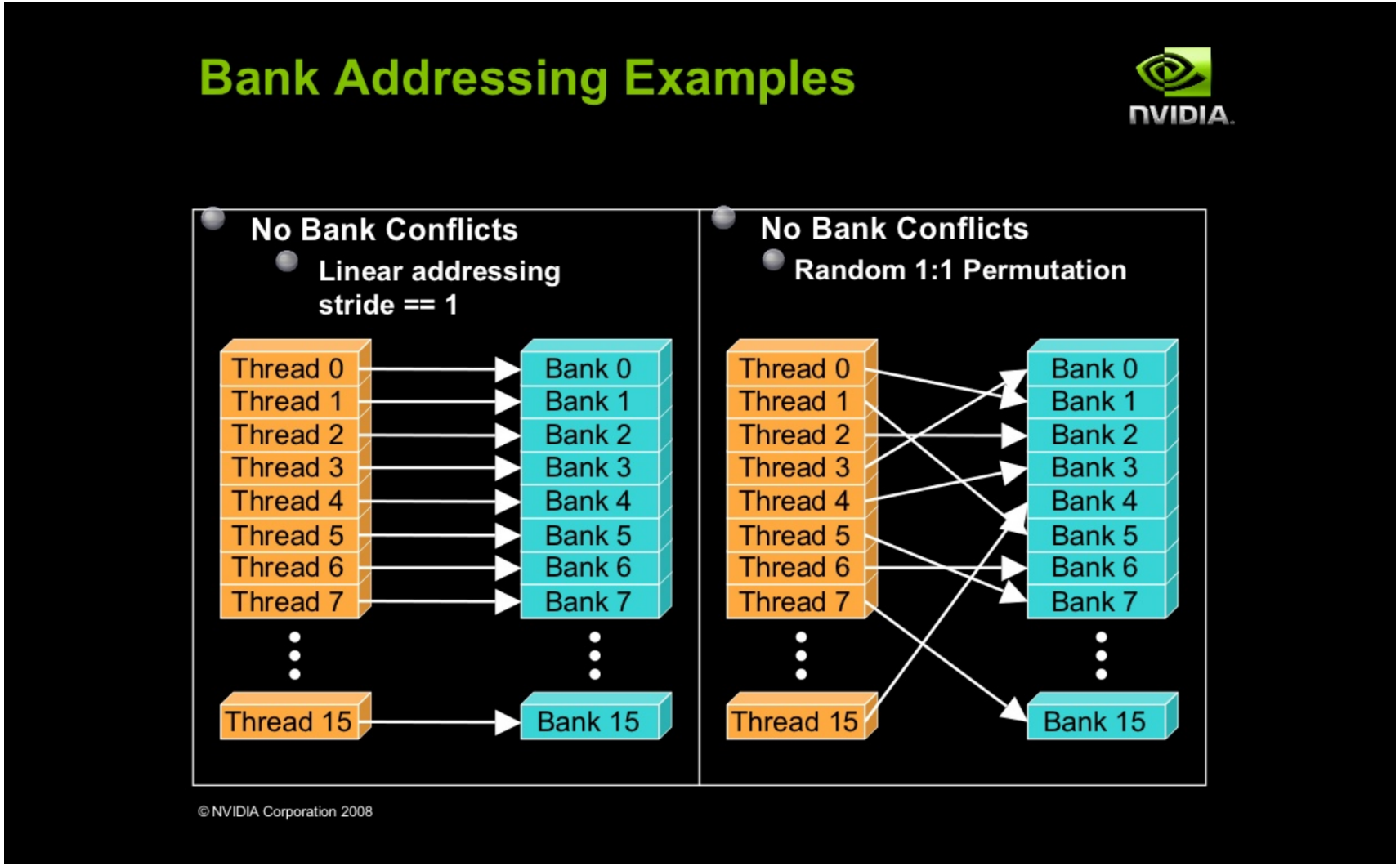
Bank 0
Bank 1
...
Bank 31



Exercise 9 Recap

The Good

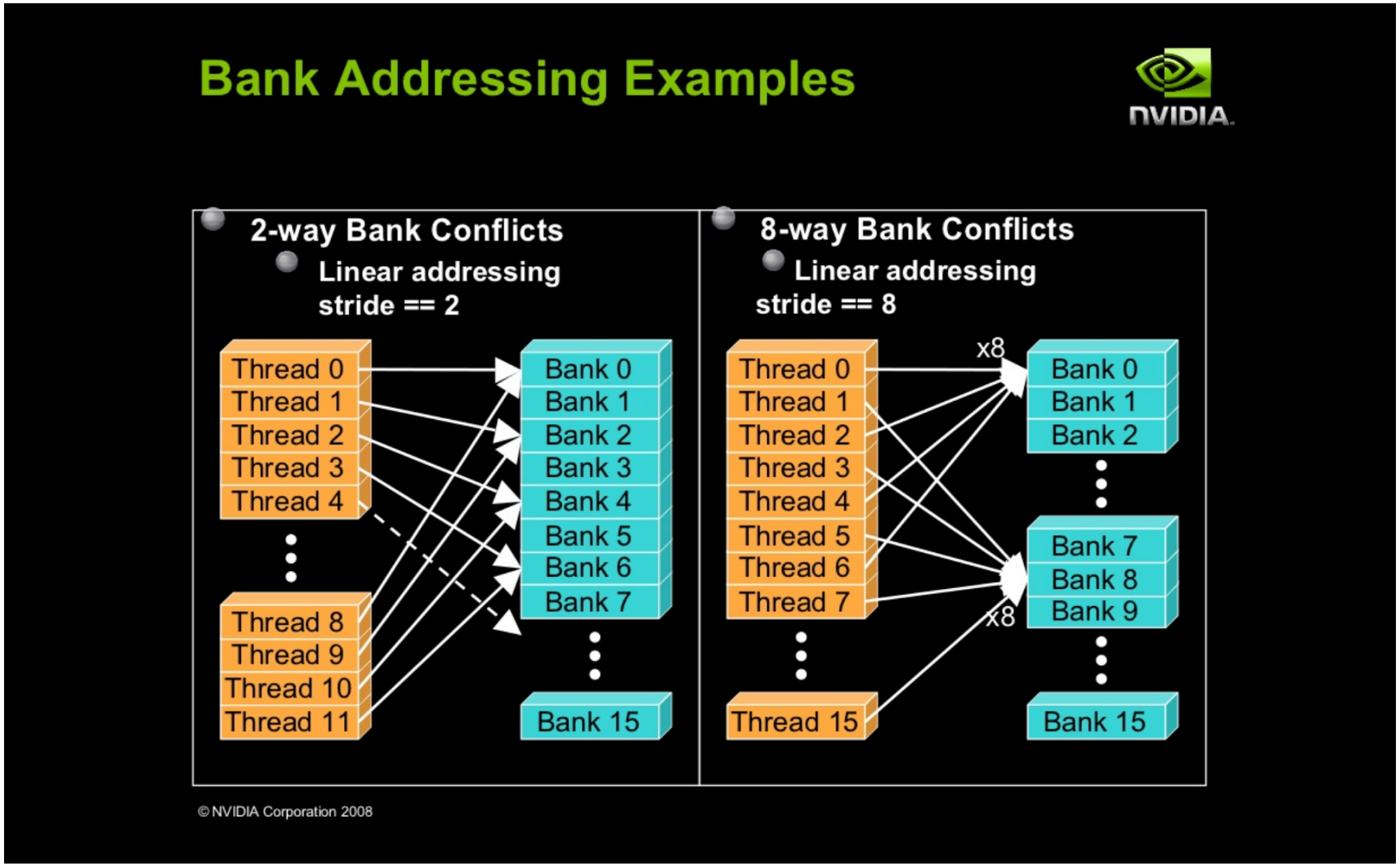
- Access to different banks is parallel



Exercise 9 Recap

The Bad

- Access to the same bank is serialized



Exercise 9 Recap

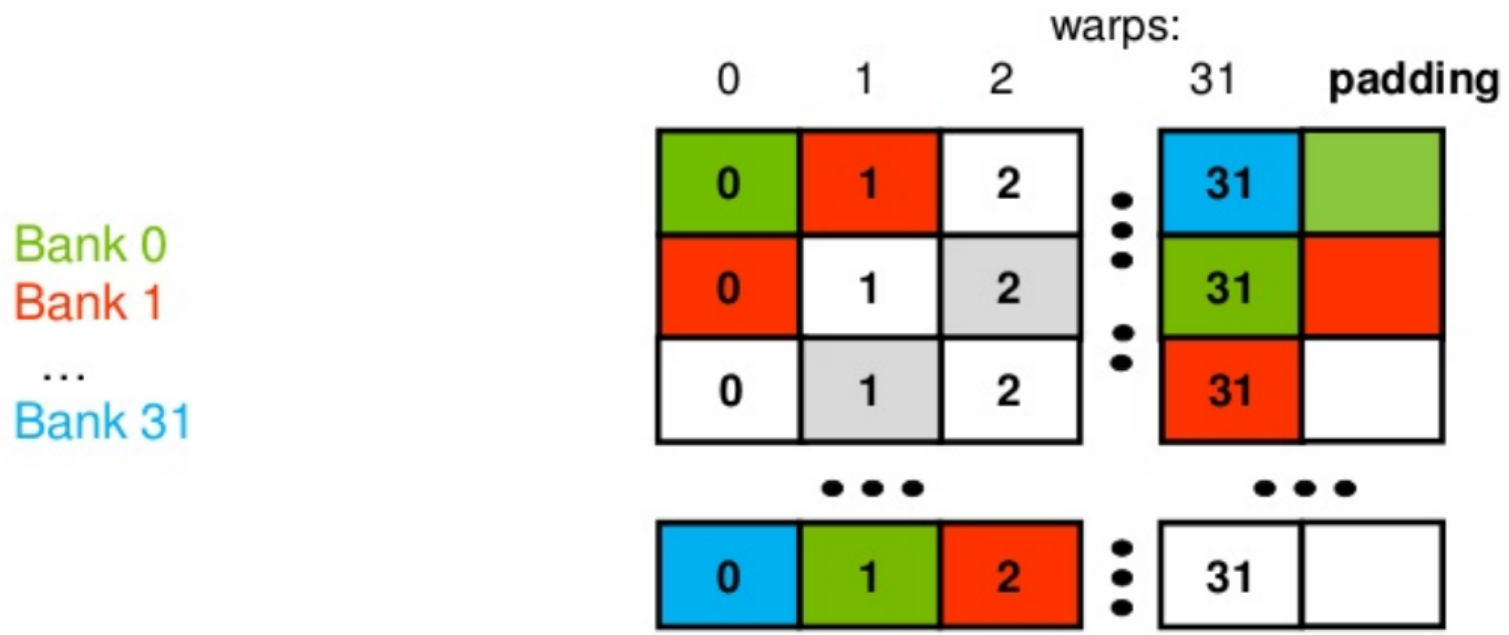
The Solution

- Instead of

```
__shared__ double tile[TILE_DIM][TILE_DIM];
```

use

```
__shared__ double tile[TILE_DIM][TILE_DIM+1];
```



Exercise 8 Recap

Reference Solution: Thrust

```
thrust::transform(x.begin(), x.end(), y.begin(),
                  x_plus_y.begin(), thrust::plus<double>());
thrust::transform(x.begin(), x.end(), y.begin(),
                  x_minus_y.begin(), thrust::minus<double>());

dot = thrust::inner_product(x_plus_y.begin(), x_plus_y.end(),
                            x_minus_y.begin(), 0);
```

Apparent Problem: Temporary vectors!

Exercise 8 Recap

Reference Solution: Thrust

```
transform(x.begin(), x.end(), y.begin(),
          x_plus_y.begin(), plus<double>());
transform(x.begin(), x.end(), y.begin(),
          x_minus_y.begin(), minus<double>());

dot = inner_product(x_plus_y.begin(), x_plus_y.end(),
                    x_minus_y.begin(), 0);
```

Apparent Problem: Temporary vectors!

Exercise 8 Recap

Intermediate Step: Additional Temporary

```
transform(x.begin(), x.end(), y.begin(),
          x_plus_y.begin(), plus<double>());
transform(x.begin(), x.end(), y.begin(),
          x_minus_y.begin(), minus<double>());
transform(
make_zip_iterator(make_tuple(x_plus_y.begin(),x_minus_y.begin())),
make_zip_iterator(make_tuple(x_plus_y.end(),x_minus_y.end())),
xpy_xmy.begin(), zipped_multiplies<double>()
);

dot = reduce(xpy_xmy.begin(), xpy_xmy.end(), 0, plus<double>());
```

Exercise 8 Recap

Intermediate Step: Additional Temporary

```
transform(x.begin(), x.end(), y.begin(),
          x_plus_y.begin(), plus<double>());
transform(x.begin(), x.end(), y.begin(),
          x_minus_y.begin(), minus<double>());
transform(
make_zip_iterator(make_tuple(x_plus_y.begin(),x_minus_y.begin())),
make_zip_iterator(make_tuple(x_plus_y.end(),x_minus_y.end())),
xpy_xmy.begin(), zipped_multiplies<double>()
);

dot = reduce(xpy_xmy.begin(), xpy_xmy.end(), 0, plus<double>());
```

```
struct zipped_multiplies
  : public unary_function<tuple<double, double>,double>
{
  __host__ __device__
  double operator()(tuple<double, double> zipped_xy) {
    return (get<0>(zipped_xy) * get<1>(zipped_xy));
  }
};
```

Exercise 8 Recap

Without Temporary Vectors

```
dot = transform_reduce(
   make_zip_iterator(make_tuple(X.begin(), Y.begin())),
   make_zip_iterator(make_tuple(X.end(), Y.end())),
   xpy_xmy{}, 0, plus<double>());
```

Exercise 8 Recap

Without Temporary Vectors

```
 dot = transform_reduce(
    make_zip_iterator(make_tuple(X.begin(), Y.begin())),
    make_zip_iterator(make_tuple(X.end(), Y.end())),
    xpy_xmy{}, 0, plus<double>());
```

```
struct xpy_xmy : public unary_function<tuple<double, double>,
    double>
{
  __host__ __device__
  double operator()(tuple<double, double> zipped_xy) {
    return (get<0>(zipped_xy) + get<1>(zipped_xy))
         * (get<0>(zipped_xy) - get<1>(zipped_xy));
  }
};
```

85us without temporaries, 100us with temporaries (1e3 entries)
1.2ms without temporaries, 2.0ms with temporaries (1e6 entries)
10ms without temporaries, 17ms with temporaries (1e7 entries)

HIP

About HIP

- AMD's response to CUDA
- Part of the ROCm framework: `https://rocmdocs.amd.com/`
- *HIP code provides the same performance as native CUDA code, plus the benefits of running on AMD platforms.*

Important Tools

- `hipify` automatically converts CUDA code to HIP
- `hip-nvcc` generates CUDA code from HIP, then calls `nvcc`
- `hip-clang` generates binaries for AMD GPUs

Example of HIP in Action

```
// kernel
__global__ void matrixTranspose(float* out, float* in, const int width) {
    int x = hipBlockDim_x * hipBlockIdx_x + hipThreadIdx_x;
    int y = hipBlockDim_y * hipBlockIdx_y + hipThreadIdx_y;

    out[y * width + x] = in[x * width + y];
}

int main() {
  ...

  // allocate
  float* gpuMatrix,float* gpuTransposeMatrix;
  hipMalloc((void**)&gpuMatrix, NUM * sizeof(float));
  hipMalloc((void**)&gpuTransposeMatrix, NUM * sizeof(float));

  // copy to device
  hipMemcpy(gpuMatrix, Matrix, NUM * sizeof(float), hipMemcpyHostToDevice);

  // launch kernel
  hipLaunchKernelGGL(matrixTranspose,
      dim3(WIDTH / THREADS_PER_BLOCK_X, WIDTH / THREADS_PER_BLOCK_Y), // grid size
      dim3(THREADS_PER_BLOCK_X, THREADS_PER_BLOCK_Y),    // block size
      0, 0,    // no shared memory allocated, default stream
      gpuTransposeMatrix, gpuMatrix, WIDTH); // kernel arguments

  // copy from device
  hipMemcpy(TransposeMatrix, gpuTransposeMatrix, NUM * sizeof(float), hipMemcpyDeviceToHost);

  // clean up
  hipFree(gpuMatrix);
  hipFree(gpuTransposeMatrix);
}
```

HIP Conversion

	CUDA	HIP
Runtime Functions	`cudaMalloc` `cudaFree`	`hipMalloc` `hipFree`

Thread Management	`threadIdx.x`
	`blockIdx.x`
	`blockDim.x`
	`gridDim.x`
		`hipThreadIdx_x`
		`hipBlockIdx_x`
		`hipBlockDim_x`
		`hipGridDim_x`

Kernel Launch	`<<< >>>`	`hipLaunchKernelGGL`

Kernel Language	`__syncthreads`
	`atomicAdd`
	`__global__`
	etc.
		`__syncthreads`
		`atomicAdd`
		`__global__`
		etc.

In a Nutshell

- Replace `cuda` with `hip` in runtime functions
- Special keywords like `__device__` do not require modification at all

HIP

Recommendations

- HIP covers a large subset of CUDA (even including warp shuffles!)
- Consider writing HIP code instead of CUDA code

HIP

Recommendations

- HIP covers a large subset of CUDA (even including warp shuffles!)
- Consider writing HIP code instead of CUDA code

Caveat

- AMD software toolchain is not always as robust as CUDA
- `hip-nvcc` is lightweight, but still an extra dependency

SYCL

About SYCL

- Aim: Lifting OpenCL to C++ in a standardized manner
- Foundation: OpenCL 2.0
- Nomenclature from OpenCL carries over (context, command queue, etc.)
- Maintained by the Khronos Group

SYCL

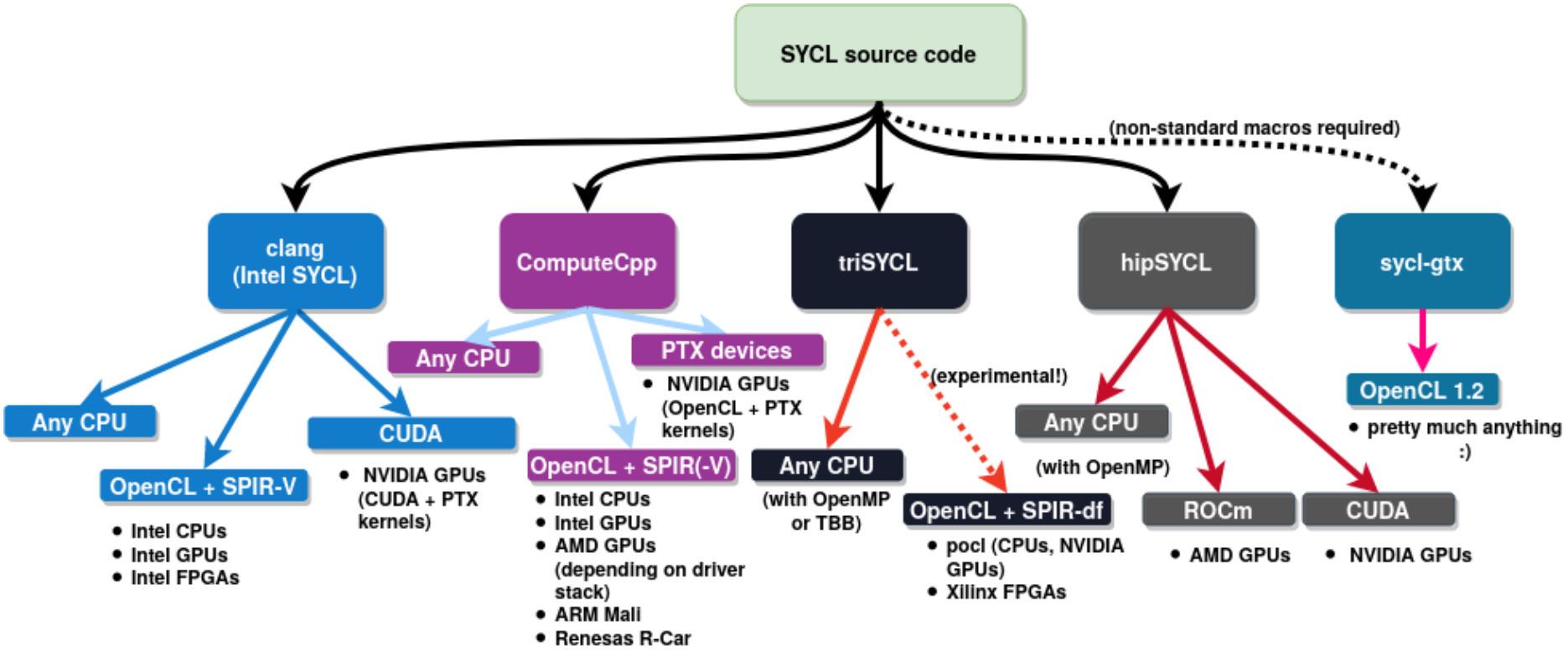
About SYCL

- Aim: Lifting OpenCL to C++ in a standardized manner
- Foundation: OpenCL 2.0
- Nomenclature from OpenCL carries over (context, command queue, etc.)
- Maintained by the Khronos Group

SYCL Implementations

- hipSYCL: A research SYCL compiler on top of HIP/CUDA
- triSYCL: research project to experiment and to give feedback to the Khronos Group SYCL committee and also to the ISO C++ committee.
- ComputeCpp: Commercial compiler by Codeplay (in close collaboration with Khronos)
- Data Parallel C++: SYCL compiler by Intel, part of oneAPI initiative

SYCL


SYCL source code
(non-standard macros required)
clang (Intel SYCL)
ComputeCpp
triSYCL
hipSYCL
sycl-gtx
Any CPU
OpenCL + SPIR-V
• Intel CPUs
• Intel GPUs
• Intel FPGAs
CUDA
• NVIDIA GPUs (CUDA + PTX kernels)
Any CPU
PTX devices
• NVIDIA GPUs (OpenCL + PTX kernels)
OpenCL + SPIR(-V)
• Intel CPUs
• Intel GPUs
• AMD GPUs (depending on driver stack)
• ARM Mali
• Renesas R-Car
Any CPU
(with OpenMP or TBB)
(experimental!)
OpenCL + SPIR-df
• pocl (CPUs, NVIDIA GPUs)
• Xilinx FPGAs
Any CPU
(with OpenMP)
ROCm
• AMD GPUs
CUDA
• NVIDIA GPUs
OpenCL 1.2
• pretty much anything :)

SYCL Example

```
#include <CL/sycl.hpp>
...
using cl::sycl;  // to fit onto one slide

queue q;
...

buffer<data_type> buff_a(a.data(), a.size());
buffer<data_type> buff_b(b.data(), b.size());
buffer<data_type> buff_c(c.data(), c.size());

q.submit([&](handler& cgh){
   auto access_a = buff_a.get_access<access::mode::read>(cgh);
   auto access_b = buff_b.get_access<access::mode::read>(cgh);
   auto access_c = buff_c.get_access<access::mode::write>(cgh);

   // The parallel section (cf. OpenCL outer-for in kernel)
   cgh.parallel_for<class vector_add>(work_items,
                                      [=] (id<1> tid) {
     access_c[tid] = access_a[tid] + access_b[tid]; });
 });
...
```

SYCL

SYCL Notes

- Big bet by Intel on SYCL

SYCL

SYCL Notes

- Big bet by Intel on SYCL
- If Intel doesn't deliver good GPUs, SYCL adoption will be hampered

SYCL

SYCL Notes

- Big bet by Intel on SYCL
- If Intel doesn't deliver good GPUs, SYCL adoption will be hampered
- Heavy C++ implies a steep learning curve

SYCL

SYCL Notes

- Big bet by Intel on SYCL
- If Intel doesn't deliver good GPUs, SYCL adoption will be hampered
- Heavy C++ implies a steep learning curve
- SYCL has been around for years and hasn't gained any significant traction yet

SYCL

SYCL Notes

- Big bet by Intel on SYCL
- If Intel doesn't deliver good GPUs, SYCL adoption will be hampered
- Heavy C++ implies a steep learning curve
- SYCL has been around for years and hasn't gained any significant traction yet
- Still plenty of early adopter pitfalls

SYCL

SYCL Notes

- Big bet by Intel on SYCL
- If Intel doesn't deliver good GPUs, SYCL adoption will be hampered
- Heavy C++ implies a steep learning curve
- SYCL has been around for years and hasn't gained any significant traction yet
- Still plenty of early adopter pitfalls

SYCL Resources

- `https://tech.io/playgrounds/48226/introduction-to-sycl/introduction-to-sycl-2`
- `https://github.com/jeffhammond/dpcpp-tutorial`
- `https://www.khronos.org/sycl/resources`

Exercises

Environment

- `https://gtx1080.360252.org/2020/ex10/`
- (Might receive additional hints)
- Due: Tuesday, January 19, 2021 at 23:59pm

Hints and Suggestions

- Consider version control for locally developed code
- Please let me know of any bugs or issues