Published under CC BY-SA 4.0

DOI: 10.5281/zenodo.1412256

Adding CUDA® Support to Cling: JIT Compile to GPUs

S. Ehrig[1,2], A. Naumann[3], and A. Huebl[1,2]

[1] Helmholtz-Zentrum Dresden - Rossendorf

[2] Technische Universität Dresden

[3] CERN

ROOT Users' Workshop

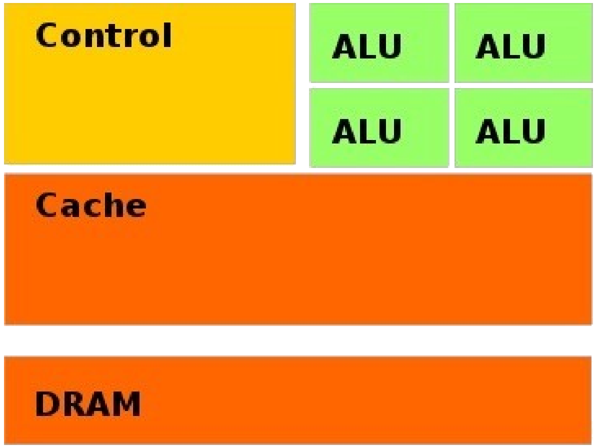
Parallelism, Heterogeneity and Distributed Data Processing

Sarajevo, September 10th 2018

Introduction

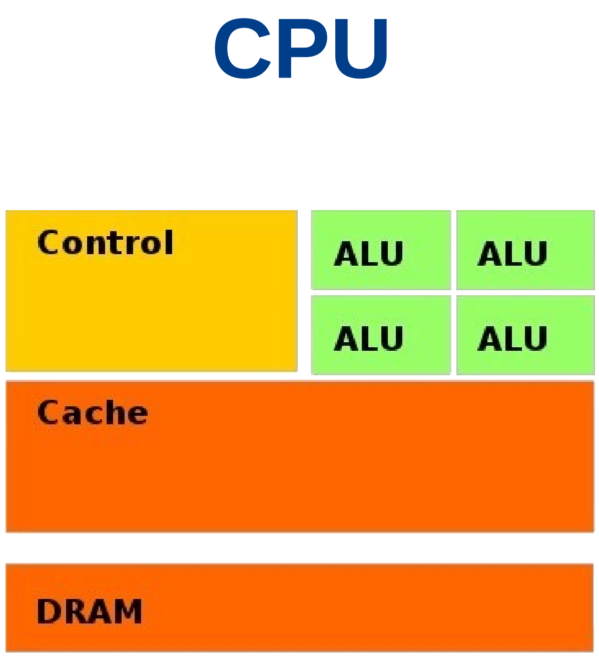
CPU/GPU Model

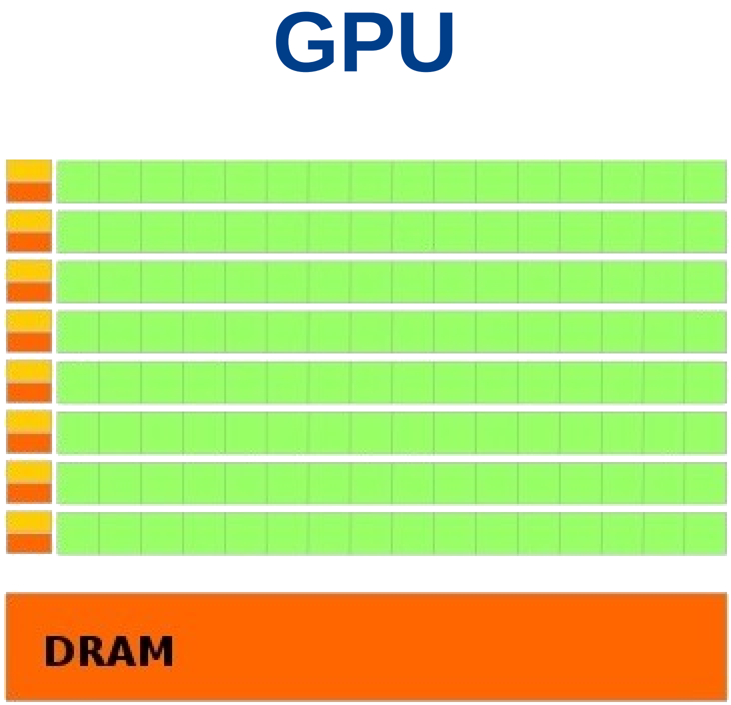
CPU



- Sources: Nvidia. *CUDA Reference Guide*

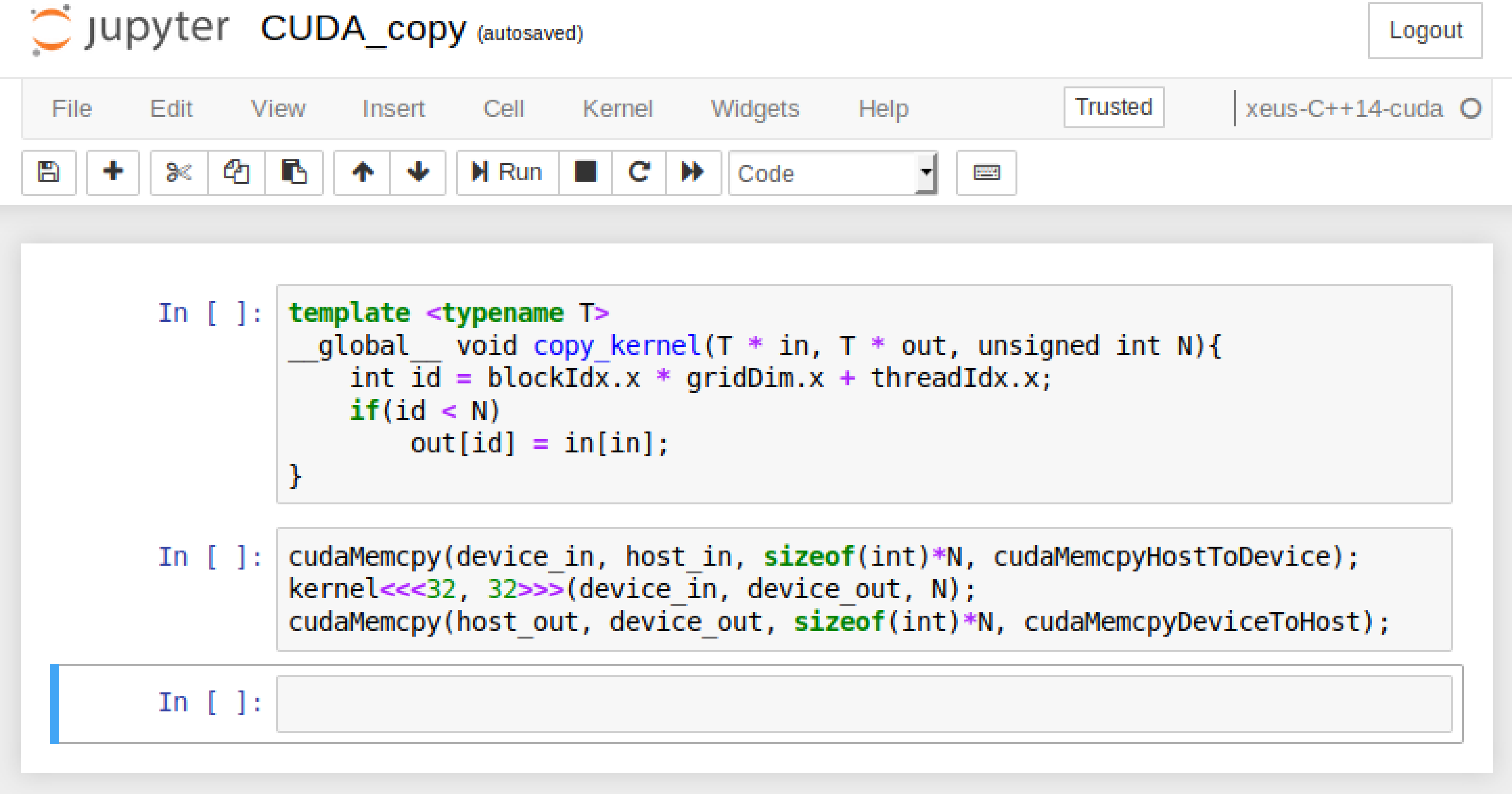
CPU/GPU Model





- Sources: Nvidia. *CUDA Reference Guide*

CUDA C++ in a Notebook: Runtime API



In [ ]:
```
template <typename T>
__global__ void copy_kernel(T * in, T * out, unsigned int N){
    int id = blockIdx.x * gridDim.x + threadIdx.x;
    if(id < N)
        out[id] = in[in];
}
```

In [ ]:
```
cudaMemcpy(device_in, host_in, sizeof(int)*N, cudaMemcpyHostToDevice);
kernel<<<32, 32>>>(device_in, device_out, N);
cudaMemcpy(host_out, device_out, sizeof(int)*N, cudaMemcpyDeviceToHost);
```

In [ ]:

How to use CUDA®

CUDA® source-code example

```
//function, which will run on GPU
template <typename T>
__global__ void copy_kernel(T * in, T * out, unsigned int N){
        int id = blockIdx.x * gridDim.x + threadIdx.x;
        if(id < N)
                out[id] = in[id];
}


int main(){

        // …

        // copy memory from cpu to gpu
        cudaMemcpy(device_in, host_in, sizeof(int) * N, cudaMemcpyHostToDevice);

        // start function on GPU with 32 threads an 10 blocks
        kernel<int><<<32, 10>>>(a, b, c);

        // copy memory from gpu to cpu
        cudaMemcpy(host_out, device_out, sizeof(int) * N, cudaMemcpyDeviceToHost);

        // …
}
```

CUDA® source-code example

Device-Code

```
//function, which will run on GPU
template <typename T>
__global__ void copy_kernel(T * in, T * out, unsigned int N){
        int id = blockIdx.x * gridDim.x + threadIdx.x;
        if(id < N)
                out[id] = in[id];
}
```

Host-Code

```
int main(){

        // …

        // copy memory from cpu to gpu
        cudaMemcpy(device_in, host_in, sizeof(int) * N, cudaMemcpyHostToDevice);

        // start function on GPU with 32 threads an 10 blocks
        kernel<int><<<32, 10>>>(a, b, c);

        // copy memory from gpu to cpu
        cudaMemcpy(host_out, device_out, sizeof(int) * N, cudaMemcpyDeviceToHost);

        // …
}
```

CUDA® C/C++-APIs

Driver API

- C/C++-conform
- Host and Device-Code separated
- Compiling kernels via library functions during runtime
- Modifiable kernel possible

CUDA® C/C++-APIs

Driver API

- C/C++-conform
- Host and Device-Code separated
- Compiling kernels via library functions during runtime
- Modifiable kernel possible

→ Works on Cling without modification

CUDA® C/C++-APIs

Driver API

- C/C++-conform
- Host and Device-Code separated
- Compiling kernels via library functions during runtime
- Modifiable kernel possible

→ Works on Cling without modification

Runtime API

- Special syntax and semantic

CUDA® source-code example

```
//function, which will run on GPU
template <typename T>
__global__ void copy_kernel(T * in, T * out, unsigned int N){
    int id = blockIdx.x * gridDim.x + threadIdx.x;
    if(id < N)
        out[id] = in[id];
}


int main(){

    // …

    // copy memory from cpu to gpu
    cudaMemcpy(device_in, host_in, sizeof(int) * N, cudaMemcpyHostToDevice);

    // start function on GPU with 32 threads an 10 blocks
    kernel<int><<<32, 10>>>(a, b, c);

    // copy memory from gpu to cpu
    cudaMemcpy(host_out, device_out, sizeof(int) * N, cudaMemcpyDeviceToHost);

    // …
}
```

```
//special kernel launch syntax!
kernel<int><<<32, 10>>>(a, b, c)
```

CUDA® C/C++-APIs

Driver API

- C/C++-conform
- Host and Device-Code separated
- Compiling kernels via library functions
  during runtime
- Modifiable kernel possible

→ Works on Cling without modification

Runtime API

- Special syntax and semantic
- Single-Source-Design

CUDA® source-code example

```
//function, which will run on GPU
template <typename T>
__global__ void copy_kernel(T * in, T * out, unsigned int N){
        int id = blockIdx.x * gridDim.x + threadIdx.x;
        if(id < N)
                out[id] = in[id];
}


int main(){

        // …

        // copy memory from cpu to gpu
        cudaMemcpy(device_in, host_in, sizeof(int) * N, cudaMemcpyHostToDevice);

        // start function on GPU with 32 threads an 10 blocks
        kernel<int><<<32, 10>>>(a, b, c);

        // copy memory from gpu to cpu
        cudaMemcpy(host_out, device_out, sizeof(int) * N, cudaMemcpyDeviceToHost);

        // …
}
```

main.cu

CUDA® C/C++-APIs

Driver API

- C/C++-conform
- Host and Device-Code separated
- Compiling kernels via library functions
  during runtime
- Modifiable kernel possible

→ Works on Cling without modification

Runtime API

- Special syntax and semantic
- Single-Source-Design
- Compiling kernels during compiletime
- Modifiable Kernels not designated

CUDA® C/C++-APIs

Driver API

- C/C++-conform
- Host and Device-Code separated
- Compiling kernels via library functions
  during runtime
- Modifiable kernel possible

→ Works on Cling without modification

Runtime API

- Special syntax and semantic
- Single-Source-Design
- Compiling kernels during compiletime
- Modifiable Kernels not designated

→ Cling needs modification

Implementation

Implementation

- Handle special syntax, semantic and **single-source design**
  - Enable Clang CUDA frontend[1] in Cling

[1] GPUCC - An Open-Source GPGPU Compiler CGO 16; nowadays mainline in Clang

Implementation

- Handle special syntax, semantic and **single-source design**
  - Enable Clang CUDA frontend[1] in Cling

- Generating Device-Code during runtime
  - Develop second compiler pipeline
  - Rely on Clang CUDA Toolchain **up to PTX**
  - Couple via Nvidia “fatbinary”
  - Generate SASS code on Nvidia driver side

[1] GPUCC - An Open-Source GPGPU Compiler CGO 16; nowadays mainline in Clang

Implementation

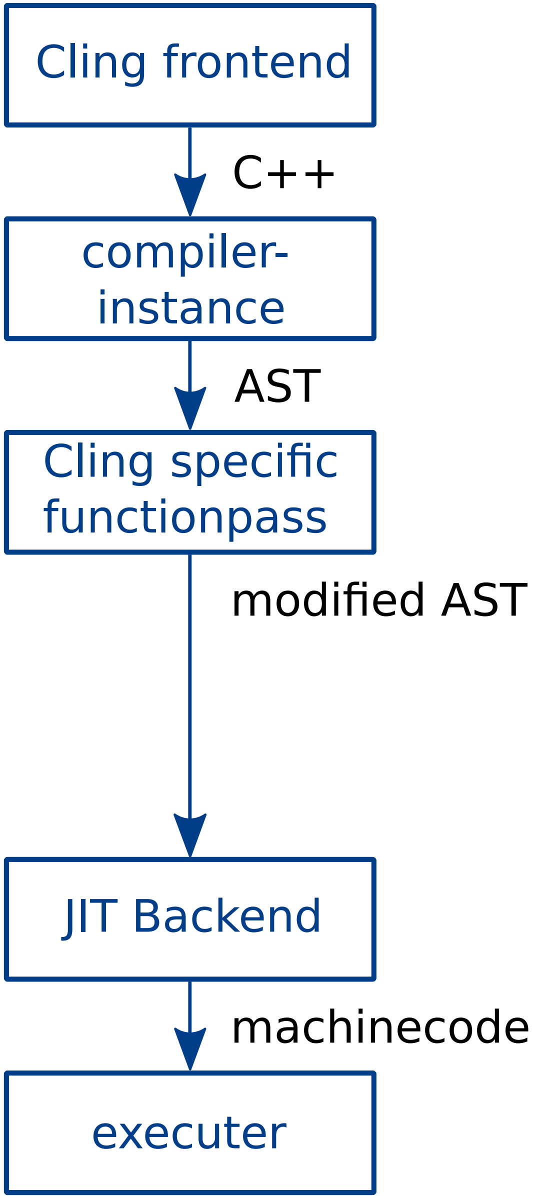
- Handle special syntax, semantic and **single-source design**
  - Enable Clang CUDA frontend[1] in Cling

- Generating Device-Code during runtime
  - Develop second compiler pipeline
  - Rely on Clang CUDA Toolchain **up to PTX**
  - Couple via Nvidia “fatbinary”
  - Generate SASS code on Nvidia driver side

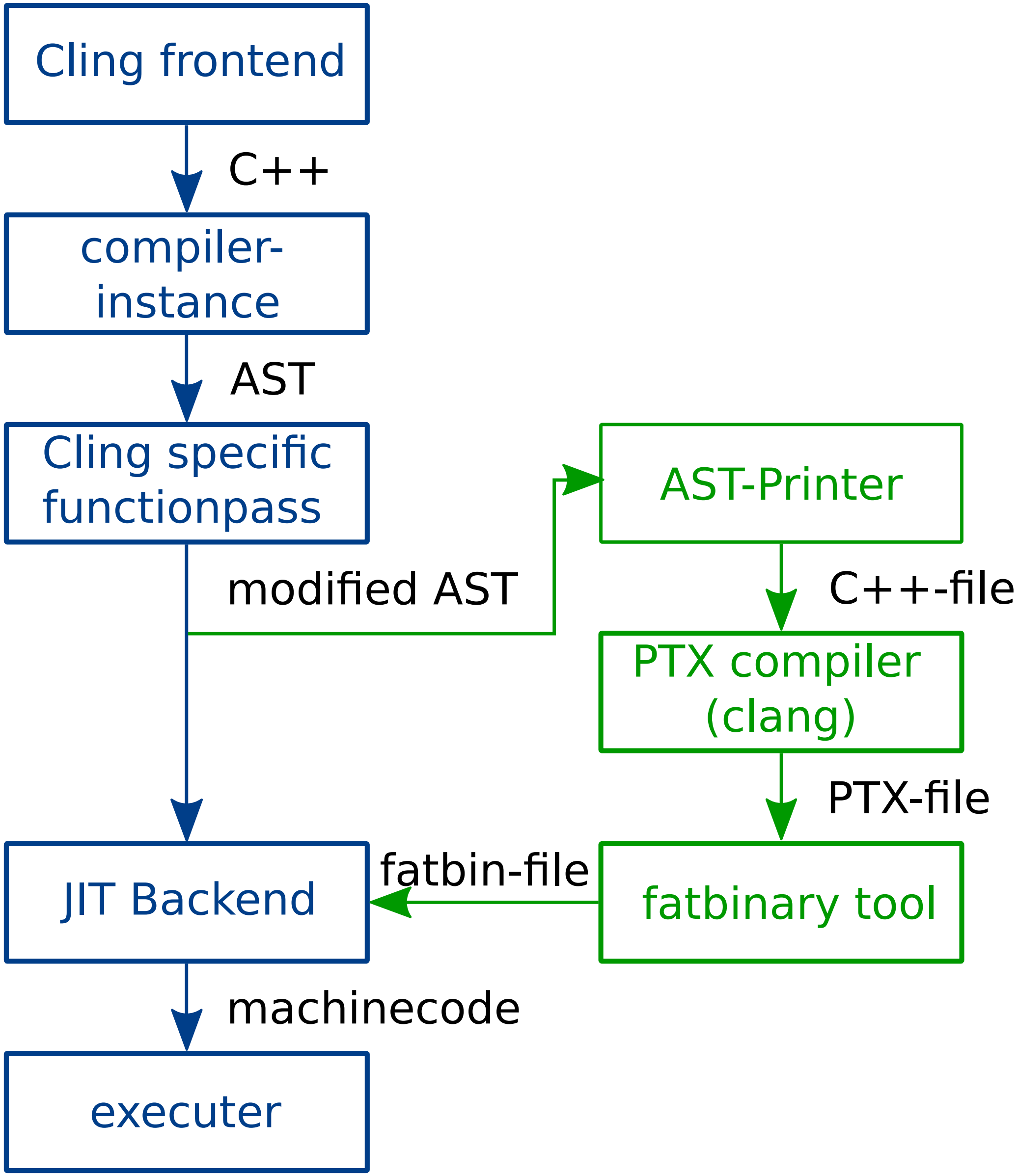
- Cling-CUDA in Jupyter Notebook
  - standard kernel of **cling -x cuda**
  - using **xeus-cling** (patch to be upstreamed)

[1] GPUCC - An Open-Source GPGPU Compiler CGO 16; nowadays mainline in Clang

Cling-CUDA Compiler Pipeline



Cling-CUDA Compiler Pipeline



What is still missing

- Some C++ and CUDA statements, although supported by Clang 5.0 on CUDA 8.0
  - AST-Printer: variable __attributes__, structured bindings

AST-Printer failure examples

```
__device__ int var = 42;
```

↓

```
int var = 42 __attribute__((device));
```

```
// struct s { int x1 = 1; float x2 = 2.0f;}; s S;
auto [a, b] = S;
```

↓

```
auto = S
```

What is still missing

- Some C++ and CUDA statements, although supported by Clang 5.0 on CUDA 8.0
  - AST-Printer: variable __attributes__, structured bindings
  - CUDA __device__ globals, __constant__

What is still missing

- Some C++ and CUDA statements, although supported by Clang 5.0 on CUDA 8.0
  - AST-Printer: variable __attributes__, structured bindings
  - CUDA __device__ globals, __constant__

- Kernel unloading
  - in contact with Nvidia about further documentation

- Cleanup
  - e.g. semantic detection of CUDA device functions

Summary

Initial CUDA Support in Cling

- First interpreter for the CUDA **runtime** API
  - Based on Clang CUDA toolchain, not cudafe

Initial CUDA Support in Cling

- First interpreter for the CUDA **runtime** API
  - Based on Clang CUDA toolchain, not cudafe
- Most features already upstream in cling master

Initial CUDA Support in Cling

- First interpreter for the CUDA **runtime** API
  - Based on Clang CUDA toolchain, not cudafe

- Most features already upstream in cling master

- Easy access to HPC GPU systems via **Jupyter Notebook**
  - **Data analysis** in notebooks with GPUs
  - Big, **interactive simulation** with GPUs
  - **Teaching** GPU programming
  - Easing **development** and debugging
- **xeus-cling:** patched kernel for **cling -x cuda**