GPU Course Cheat Sheet

Workstation

Logging in

Username: `train0XX`
Password: *See slip of paper*

Editing

kate provides remote editing. Use `sftp://juron/` to access your files via JURECA and edit them locally.
Alternative: Mount your remote home directory via `fish://juron`

Editing

We are doing most of the work in the command line. Start a terminal window from your desktop using `ALT+F2` and entering `konsole`

Some Commands

`cd` *`dir`* Changes working directory to *`dir`*
`ls` Lists files in the current directory
`ls -l` Like above, but gives more detail
`mkdir` *`dir`* Creates a new subdirectory named *`dir`*
`rm` *`file`* Removes a file named *`file`* (Can not be undone!)
`less` *`file`* Shows the content of *`file`*

Supercomputers

We will be working on **JURON**.

Logging In

Add SSH key to SSH agent: type `ssh-add` and enter password from the slip of paper.
Login: type `ssh juron.fz-juelich.de`

Environment

JURON uses a module system to provide different software. List available modules with `module avail`
On JURON, CUDA can be loaded with
`module load cuda/9.1.85`

JURON Compute Nodes

JURON is accessed through a login node. This frontend node can be used for development, but to run your code on a GPU you need to access the compute nodes via the batch system.

To submit a GPU program to the batch queue, use
`bsub -U cuda18_0 -Is -R "rusage[ngpus_shared=1]" prog`

`-U cuda18_0` Reservation of the day
`-Is` Launch interactive job (`-I`), returning output to screen directly (`-s`)
`-R "..."` Resource specification: One GPU slot

CUDA

Allocate Memory

```
cudaMallocManaged(T** pointer, size_t nbytes)
cudaMalloc(T** pointer, size_t nbytes)
```

Free Memory

```
cudaFree(pointer)
```

Transfer Data

Needed, when not using Unified Memory with `cudaMallocManaged()`
`cudaMemcpy(void* dst, void* src, size_t nbytes, enum cudaMemcpyKind dir)`
The last argument can be one of `cudaMemcpyHostToDevice`, `cudaMemcpyDeviceToHost`, `cudaMemcpyDeviceToDevice`, `cudaMemcpyDefault`

Call Kernel

```
kernel_name<<<dim3 grid, dim3 block>>>(...)
```

Device Functions

Kernel:
`__global__ void kernel_name([args]) {...}`
Device-side Function:
`__device__ void device_function([args])`
All kernel functions have access to
`dim3 gridDim, blockDim, blockIdx, threadIdx`

CUDA Documentation

Lots of documentation about CUDA can be found online
`https://docs.nvidia.com/cuda/`