GPU Course Cheat Sheet

Workstations

Logging in

Username: `train0??`, where ?? is the number assigned to you.
Password: 20Patc-CUDA16!

Editing

kate provides remote editing. Use sftp://jureca/homea/hpclab/train0?? to access your files on JURECA and edit them locally.

Command line

A lot of our work will be done on the command line. You can start a terminal window using <ALT>+<F2> (opens a window to enter commands) and then type konsole

A few commands

`cd <dir>` - switches the working directory to <dir>
`ls` - lists files in current directory
`ls -l` - same as `ls` but gives more detail
`mkdir -p <dir1>/<dir2>` - creates a new (subdirectory> <dir1>/<dir2>
`rm <file>` - deletes (removes) a file. Cannot be undone!
`less <file>` - shows the context of a file.

JURECA

Logging in:

Type `ssh-add` then `20Patc-CUDA16!`
`ssh jureca.fz-juelich.de`

Setting up the environment

Several compilers are available. To get a list use the module command
`module avail`
To load CUDA first load a toolchain, e.g,
`module load intel-para`
and then the module
`module load CUDA/7.5.18`

Accessing a compute node

The frontend nodes can be used for development but to run your code on a GPU you need to access one of the compute nodes. To get a node with four GPUs for 8 hours use

```
salloc --reservation=cuda -p gpus
 --nodes=1 --gres=gpu:4 --time=8:0:0
```

After some time you are returned to the prompt. To run a program on the compute node start it with
`srun <executable>`
To start an interactive session use

```
srun --forward-x --cpu_bind=none --
 pty /bin/bash -i
```

Allocate memory on device

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

Free memory on device

```
cudaFree(pointer)
```

Transfer data

```
cudaMemcpy(void* dst, void* src, size_t nbytes,
 enum cudaMemcpyKind dir)
```

Directions can be one of

```
cudaMemcpyHostToDevice, cudaMemcpyDeviceToHost,
 cudaMemcpyDeviceToDevice, cudaMemcpyDefault
```

Call kernel

```
kernel_name<<<dim3 grid, dim3 block>>>([args])
```

CUDA kernel

`__global__ void kernel_name([args]) {...}` - kernel function
`__device__ void device_function([args]) {...}` - function that can be called from kernel
`__shared__` - fast on-chip memory
All kernel functions have access to
`dim3 gridDim, blockDim, blockIdx, threadIdx`

CUDA Documentation

You can find lots of documentation at the Nvidia web page: http://docs.nvidia.com/cuda/index.html

Jan H. Meinke <j.meinke@fz-juelich.de>, Jülich Supercomputing Centre, Forschungszentrum Jülich GmbH, Jülich, Germany