CudaLinux

CUDA

NVIDIA CUDA C

The Nvidia CUDA-Toolkit contains CUDA and OpenCL (since version 3.0). To use it, you have to load the corresponding module, first:

```
module load cuda[/version]
```

Loading the module will, for instance, set your PATH and LD_LIBRARY_PATH variables to the correct location and furthermore will provide the variable $CUDA_ROOT which contains the root directory of the loaded toolkit. The currently loaded version can also be obtained by `nvcc --version`. Documentation can be found under $CUDA_ROOT/doc, e.g. a the Nvidia Programming Guide, the Best Practices Guide, Fermi Tuning Guide and more.

You can **compile** and link e.g. your CUDA-C program Pi.cu by:

```
nvcc pi.cu
```

For enabling double precision, you have to set the architecture to (at least) compute capability 1.3.

```
nvcc -arch=sm_13 pi.cu
```

For the Fermi architecture you can even set the compute capability to 2.0:

```
nvcc -arch=sm_20 pi.cu
```

For general information about the compatibility of Fermi GPUs see "NVIDIA_FermiCompatibilityGuide" in the documentation directory.

Combining MPI and CUDA is a possibility to scale over several nodes. For instance, you can run one process per machine and each process uses one (or if available two) GPUs. Another example for a two-GPU-machine is to specify that there should be two processes per node and each uses one GPU.
To use CUDA with MPI, our recommendation is to compile you CUDA code with nvcc, as usual, and then link with mpicxx by denoting the CUDA libraries. E.g.:

```
nvcc -arch=sm_20 -m64 -c foo.cu -o foo.o
$MPICXX -c bar.cpp -o bar.o
$MPICXX foo.o bar.o -o foobar.exe -L$CUDA_ROOT/lib64 -cudart
```

Run your program with $MPIEXEC: For information about its interactive usage, see "interactive access". Information about MPI usage in batch mode, you can find under "batch access".

Usually, `nvcc` uses the GNU compiler (check using `nvcc -v` or `nvcc -dryrun`). However, since CUDA Toolkit 3.2 the Intel 11.1 or 12 compiler shall be supported as well. To use the intel compiler, denote:

```
nvcc -ccbin [<path to intel compiler>/]icc ...
```

The Nvidia GPU Computing **SDK** provides a lot of examples in CUDA C. They can be used to verify the correct setup of the GPU (i.e. examples deviceQuery and bandwithTest), to give a starting point for your own application and to give you the idea how to implement certain algorithm on a GPU. You can find the howto of the SDK here.

The **CUDA Visual Profiler** is can be called by `nvvp` (in older CUDA Toolkit versions by `computeprof`). Please make sure that the CUDA module is loaded before you use it.
More information on our GPGPU-Tools-Wiki: Visual Profiler

The **CUDA GDB Debugger** (cuda-gdb) can be found in $CUDA_ROOT/bin. You have to compile your program using the following flags:

```
nvcc -g -G pi.cu
```

Afterwards you can execute cuda-gdb, set breakpoints to kernel functions, switch between threads and examine values.
More information on our GPGPU-Tools-Wiki: cuda-gdb A manual can be found in the documentation directory.

PGI CUDA Fortran

Coming soon (see PGI Accelerator Page for general information in the mean while)