How to get access to NVIDIA V100 GPUs on the ÅA-UTU joint computer cluster dione.abo.fi

1 . You already have a user ID for your ÅA or UTU unix account. What is your user ID? Mine is jawester . I know this from the short version of my mail address. The long version is jan.westerholm@abo.fi and the short version is **jawester**@abo.fi.

2. Submit a request for a user account on dione.abo.fi from the link on

https://p55cc.utu.fi/

Ask for the same username that you already have. On p55cc.utu.fi, whatever applies to UTU applies to ÅA as well.

3. When your account request has been approved you can log on to the **front nod**e of dione.abo.fi:

```
ssh username@dione.abo.fi
```

Edit your source files using your favourite text editor and name your files with the extension `.cu` . GPU program compilation is done with `nvcc` on the front node. First load the cuda module and then a reasonably new version of GCC (note the use of capital and small letters here):

```
module load cuda
module load GCC
```

This will load CUDA 10.1 and gcc, version 8.3.0. Compile your CUDA program yourcode.cu with

```
nvcc -arch=sm_70 yourcode.cu -o yourexe -lm
```

The switch `-lm` means that the math library libm is linked with the program. Now you have an executable program on the front node, in this case the result of the compilation is stored as `yourexe`, for a name of your own choice, use -o your_executable when compiling. In order to run your program you must use the batch job system SLURM to put your job into the job queue for the dione computer cluster with 38 nodes and 8 GPUs. You can submit your program to SLURM with srun:

srun -p gpu –mem=10G -t 1:00:00 ./your_executable data_100k_arcmin.dat flat_100k_arcmin.dat

Here your program your_executable runs on the GPU partition of the cluster (-p gpu, only two of the nodes on dione have GPUs, most nodes do not have GPUs), reserves up to 10 GB of main memory for the CPU on the node, runs for at most 1 hour, and reads data from two files.

You then wait for the job to execute and finish. After your program has run you can repeat the editing, compilation and job queue sequence. **Do not run your program on the front node since other people are editing and compiling their jobs on the front node.** Thus, don't write

./your_executable data_100k_arcmin.dat flat_100k_arcmin.dat

(actually it won't run on the front node since there are no GPUs on the front node …**).**

You don't need this for the GPU course, but an OpenMPI job mympi.exe is run on two nodes with in total 80 cores (-n 80) by first loading the module OpenMPI and then running on two nodes (-N 2)

module load OpenMPI
srun -N 2 -n 80 ./mympi.exe

Note: there is no `mpirun -np #`. You may see an error message concerning the use of the Infiniband network between the nodes. You should get rid of this error message by specifying OpenMPI to use Infiniband. This is done by setting the following parameter to true:

export OMPI_MCA_btl_openib_allow_ib=1

Similarly, an OpenMP program is compiled with the flag -fopenmp and run with -n to indicate the number of OpenMP threads (max 80):

srun -n 40 ./myopenmp.exe

4. The module system

As described above, before compiling a program on dione you need to load specific resources or **modules** for your compilation. For instance, there are several versions of the gcc compilers available. Here is a list of commands related to the use of modules:

module avail	Show available modules on dione
module list	Show loaded modules
module unload <module>	Unload a module
module load <module>	Load a module
module load GCC/7.3.0-2.30	Load version 7.3.0 of gcc
module purge	unload all modules

Load a specific module, like gcc version 7.3.0, as listed by module avail

```
module load GCC/7.3.0-2.30
```

or load the default module

```
module load GCC
```

in which case you will obtain gcc version 8.3.0, marked by (D) in the list provided by module avail.

5. Executing jobs in the cluster

You may not run your jobs on the login node. All jobs must be dispatched to the cluster using SLURM commands. You can write `srun` with suitable switches on the command line as above, or you can use a script to define the job and the parameters for SLURM. There is a large number of parameters and environment variables that can be used to define how the jobs should be executed, please look at the SLURM manual for a complete list.

A typical script for starting the jobs may look as follows (name:batch.job):

```
#-------------------------------------------------------------------------------------------------------------------
#!/bin/bash
#SBATCH --job-name=test
#SBATCH -o result.txt
#SBATCH –p gpu                                  # run your program on the gpu partiton
#SBATCH --workdir=<Workdir path>
#SBATCH -c 1                                    # run with one core
#SBATCH -t 10:00                                # max run time is 10 mnutes
#SBATCH --mem=10M                               # program uses at most 10 MB
module purge                                    # Remove all loaded modules for a clean start
module load <desired modules if needed>         # You can either inherit the module environment from your present
                                                # working environment , or insert a new one here

srun <executable>
#-------------------------------------------------------------------------------------------------------------------
```

The script is run with

```
sbatch batch.job
```

The script defines several parameters that will be used for the job.

--job-name	defines the name
-o result.txt	redirects whatever is written to the standard output to the file results.txt
-p gpu	runs the job on the nodes in dione which have GPUs. –p normal has no GPUs!
--workdir	defines the working directory when running the program
-c 1	sets the number of cpus needed to 1
-t 10:00	the time limit of the job is set to 10 minutes. After that the process is stopped
--mem=100M	the memory required for the task is 100MB.

The command srun in the batch script starts a program. When starting the program SLURM gives it a job id which can be used to track its execution with e.g. the squeue command.

6. Useful commands in SLURM

sinfo shows the current status of the cluster.

`sinfo -p gpu`	Shows the status of the GPU-partition
`sinfo -O all`	Shows a comprehensive status report node per node
`sstat <job id>`	Shows information on your job
`squeue`	Lists the job queue
`squeue -u <username>`	Shows only your jobs
`srun <command>`	Dispatches jobs to the scheduler
`sbatch <script>`	Runs a script defining the jobs to be run
`scontrol`	Control your jobs in many aspects
`scontrol show job <job id>`	Show details about the job

`scontrol -u <username>`	Show only a certain users jobs
`scancel <job id>`	Cancel a job
`scancel -u <username>`	Cancel all your jobs

Some practical information can be found with the following commands:

To list all available modules (here only a partial list is shown):

```
-bash-4.1$ module avail
-------------------------------- /export/modules/modulefiles ---------------------------------
PrgEnv-amd/0.2(default)              cuda/4.2.9                          matplotlib/1.2.0(default)
PrgEnv-gnu/0.2(default)              cuda/5.0.35                         mpich/3.0.4(default)
PrgEnv-intel/0.2(default)            cuda/5.5.22(default)                mpip/3.3(default)
PrgEnv-pgi/0.2(default)              dp/5.3(default)                     mvapich2/1.9
abinit/7.2.1(default)                espresso/5.0.2(default)             mvapich2/2.0a
acml/4.4.0                           fftw/3.3.3(default)                 mvapich2/2.0b(default)
acml/5.3.0(default)                  gcc/4.6.4                           mvapich2-x/1.9b(default)
acml-fma4/5.3.0(default)             gcc/4.8.1                           netcdf/4.2.1.1(default)
acml-fma4-mp/5.3.0(default)          gcc/4.8.2(default)                  numpy/1.6.2(default)
acml-mp/4.4.0                        gcc/cilk-4.8.0-20121105             openmpi/1.6.4(default)
acml-mp/5.3.0(default)               ghc/7.4.2                           petsc/3.2-p7
amd-app-sdk/2.8.1(default)           ghc/7.6.1(default)                  petsc/3.3-p5
amd-codexl/1.2.2484(default)         gpaw/0.9.0.8965(default)            petsc/3.4.2(default)
amd-gdebugger/6.2.438(default)       gpaw-setups/0.9.9672(default)       pgi/13.6(default)
…
atlas/3.10.1(default)                haskell-platform/2012.2.0.0(default) scalapack/2.0.2(default)
atlas/3.8.4                          hdf5/1.8.10p1(default)              scalasca/1.4.2(default)
clamdblas/1.10.321(default)          hydra/3.0.3(default)                scipy/0.11.0
clamdfft/1.10.321(default)           intel/2013.2.146(default)           valgrind/3.8.1(default)
cp2k/2.3(default)                    intel-ocl-sdk/2012(default)         yambo/3.3.0r36(default)
```

The 'module load' and 'module remove' / 'module unload' commands can be used to load or remove modules. 'module purge' command will remove all loaded modules. If you have real troubles, try to logout and login again to reset all settings.

6. Sample GPU program

If nothing seems to work, try to compile and run the following simple program:

Hello_world.cu

```
#include <cuda.h>
#include <stdio.h>

const int N = 16;
const int blocksize = 16;

__global__ void hello(char *a, int *b)
{
	a[threadIdx.x] += b[threadIdx.x];
}

int main()
{
	char a[N] = "Hello \0\0\0\0\0\0";
	int b[N] = {15, 10, 6, 0, -11, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};

	char *ad;
	int  *bd;
	const int csize = N*sizeof(char);
	const int isize = N*sizeof(int);

	printf("%s", a);

	cudaMalloc( (void**)&ad, csize );
	cudaMalloc( (void**)&bd, isize );
	cudaMemcpy( ad, a, csize, cudaMemcpyHostToDevice );
	cudaMemcpy( bd, b, isize, cudaMemcpyHostToDevice );

	dim3 dimBlock( blocksize, 1 );
	dim3 dimGrid( 1, 1 );
	hello<<<dimGrid, dimBlock>>>(ad, bd);
	cudaMemcpy( a, ad, csize, cudaMemcpyDeviceToHost );
	cudaFree( ad );
	cudaFree( bd );

	printf("%s\n", a);
	return EXIT_SUCCESS;
}
```

Compile the program, submit it to the queue system, wait for it to finish, and the result printed “on screen” will be in out.txt:

```
-bash-4.1$ nvcc hello_world.cu -o hello_world
-bash-4.1$ srun -c 1 -t 10:00 -p=gpu --mem=10M -o out.txt -e err.txt ./hello.world
srun: job 384207 queued and waiting for resources
srun: job 384207 has been allocated resources

[1]+  Done
-bash-4.1$ less out.txt
Hello World!
```