University of Amsterdam
BSc Informatica

Distributed and Parallel Programming

DAS5 Tutorial

Becoming Familiar with the Supercomputing Infrastructure

dr. Adam Belloum

October 2023

In this first tutorial you will learn some basic features of the DAS5 supercomputer. This tutorial will focus on the details needed in the first three assignments of the Distributed and Parallel Programming course, namely:

- How to set up your working environment.
- The rules of using the DAS5 systems.
- How to run a simple application using DAS5.
- How to use queuing systems.
- How to run simple CUDA programs on the DAS5 special nodes.

For more details about DAS5 consult the DAS5 web page: `https://www.cs.vu.nl/das5/` and the DAS5 instructions as published on Canvas.

Questions

1. DAS5 login and data copying

   a) **How do you log in on DAS5? What is needed to login from home/from outside the UvA?**
   To log in, an SSH connection is established to the front server of the UvA DAS-5 cluster:

   ```
   ssh dpp2567@fs2.das5.science.uva.nl
   ```

   When connecting from outside the UvA network (e.g. from home), a VPN connection `UvA eduVPN` must first be activated to reach the internal network. Login then proceeds normally using the assigned username and password.

b) **Where should students change their password?**
After the first successful login, passwords are changed directly on the front server `fs2` using:

```
passwd
```

The user first enters the current temporary password and then provides the new one (for this report the new password was set to `Dnpp-12345678`).

c) **How do you copy a file from your local file system to the DAS5? How do you copy a folder from DAS5 to the local file system?**
To transfer files to the DAS5 can be used spc command.

```
scp "<Local Directory of hello_cuda.cu>" \
    "<Local Directory of Makefile>" \
    dpp2567@fs2.das5.science.uva.nl:~/
```

To copy data back from the cluster to the local machine, the command direction is simply reversed:

```
scp -r dpp2567@fs2.das5.science.uva.nl:~/cuda-hello  .
```

(the option `-r` copies folders recursively).

2. Usage policy

a) **What is the default run time for jobs?**
The default runtime for jobs on DAS5 is 15 minutes unless specified otherwise with a runtime flag.

b) **What is the maximum run-time for a job during the day?**
From 8:00am to 8:00pm the limit is also 15 minutes.

c) **How and when can one execute long-running programs?**
It is possible for you to run longer jobs during the night, where the limit is 2 hours for all your jobs.

d) **What are the permitted actions on the head node and on the regular nodes?**
The head node (`fs2`) is reserved for light tasks such as compiling code, editing files, and submitting jobs. Computational programs must not be executed directly on it. All CPU- or GPU-intensive computations must be dispatched to regular compute nodes through the job scheduler using `prun` or `srun`.

e) **What are the consequences of not following the rules?**
Users who violate the usage policy risk losing the access to their accounts.

3. Job Execution

a) **How is the Prun user interface used? Please follow the Prun/MPI example. Modify the example to run on 1 node with 1 process each. How do you know (based on the output) you have succeeded to change the execution configuration?**
The `prun` command provides a convenient user interface on top of the SLURM scheduler. The number of compute nodes is specified with the flag `-np`, and the number of processes per node with `-X`. For example, to run the standard `cpi.c` MPI program on a **single node with one process**, the following commands are used:

```
module load prun
mpicc -O2 cpi.c -o cpi
prun -np 1 -1 -script $PRUN_ETC/prun-openmpi ./cpi
```

From the program output it can be verified that the configuration was correct: only one process reports its rank and host, for example

```
pi is approximately 3.1416009869231254, error is 0.0000083333333323
wall clock time = 0.000023
Process 0 on node207
```

The presence of exactly one rank and one node name confirms that the execution ran on **1×1** (one node, one process).

b) **Assume you have just compiled your application, into "assign1", in the current directory, and you are ready to execute it. What would be the command to run it on the headnode (which, as you know, you should never do)? What about running it on a single regular node?**
Running directly on the head node (which is *not allowed*) would be done by:

```
./assign1
```

The correct and permitted way to run on a single node is to use `prun`:

```
module load prun
prun -np 1 -script $PRUN_ETC/prun-openmpi ./assign1
```

4. GPU computing

a) **How do you setup the environment for running CUDA programs?**
To compile and execute CUDA programs on the DAS-5 cluster, the CUDA toolkit and the `prun` interface must first be loaded.

```
module load cuda12.3/toolkit
module load prun
```

Note here I used cuda 12.3 instead of 10.3 stated in the instruction. The latter would cause a GCC error when `make`. The CUDA compiler version can be verified with:

```
nvcc --version
```

After compilation, GPU programs must be launched on compute nodes equipped with GPUs (TitanX nodes as it's what UvA got allocated) through the scheduler via `prun`.

b) **Use the files you copied (`hello_cuda.cu` and `Makefile`) to build the GPU application (just use `make`). Run this GPU application using `prun`. What is the output?**
From the home directory, the GPU application was compiled using:

```
make
```

which internally calls `nvcc` to produce the executable `hello_cuda`. The program was then executed on a TitanX GPU node through the `prun` interface as follows:

```
prun -v -np 1 -native '-C TitanX' ./hello_cuda
```

The `-C TitanX` constraint ensures that the job is scheduled on one of the GPU nodes (`node205` or `node206`) that each have two TitanX GPUs. The option `-np 1` requests a single node for the run. A typical output from this execution is:

```
Reservation number 12128: Reserved 1 hosts for 900 seconds
Run on 1 hosts for 960 seconds from Fri Oct 31 12:27:26 2025
: node202/0
Testing CUDA!
sresults OK!
```

This confirms that the program was executed successfully on a GPU-equipped node and that the CUDA kernel produced the expected output.