Vlaams Supercomputer Centrum

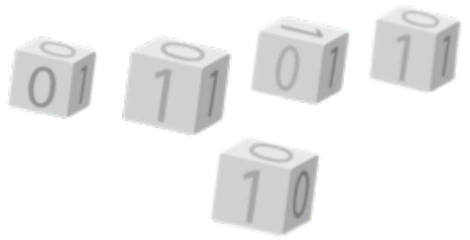
Transitioning to Leibniz and CentOS 7

Fall 2017

Overview

- Introduction: some important hardware properties of leibniz
- Working on leibniz:
  - Logging on to the cluster
  - Selecting software: toolchains
  - Activating software: modules
  - Starting jobs: changes in job scripts
- Special-purpose hardware in leibniz
  - Visualisation node
  - GPU nodes
  - Xeon Phi node

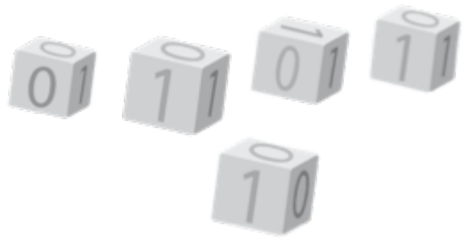
UAntwerp hardware



hopper (2014)	leibniz (2017)
• 168 compute nodes	• 152 compute nodes
○ 144*64GB, 24*256GB	○ 144*128GB, 8*256GB
○ 2 2.8GHz 10-core Ivy Bridge CPUs (E5-2680v2)	○ 2 2.4GHz 14-core Broadwell CPUs (E5-2680v4)
○ ±400 GB for swap and /tmp combined	○ No swap, 10GB local tmp
	• 2 GPU compute nodes (dual Tesla P100)
	• 1 Xeon Phi coprocessor node (KNL)
• 4 login nodes	• 2 login nodes
	• 1 visualisation node (hardware accelerated OpenGL)

- The storage is a joint setup for hopper and leibniz with 100TB capacity spread over a number of volumes

The processor



	hopper	leibniz	
CPU	Ivy Bridge E5-2680v2	Broadwell E5-2680v4	
Base clock speed	2.8 GHz	2.4 GHz	-14%
Cores/node	20	28	+40%
Vector instruction set	AVX	AVX2+FMA	
DP flops/core/cycle	8	16 (e.g., FMA instruction)	+100%
Memory speed	8*64b*1866 MHz DDR3	8*64b*2400MHz DDR4	+29%
DGEMM Gflops 50k x 50k	456 Gflops	974 Gflops	+113%
GMPbench	3875	3230/4409	-17%/+14%

- Despite IPC improvements, some single core applications will run slower on leibniz
  - Face it: CPU performance is going down in the future unless you exploit parallelism (SIMD/vector and multi-core) as performance/Watt is what matters nowadays
- Recompile needed to use all features of the leibniz CPU

The processor (2)

- Compared to the Westmere CPUs in part of turing:
  - Potential doubling of floating performance by using AVX instructions on Sandy/Ivy Bridge CPUs instead of SSE4 (hopper)
    - Vector width of 256 bits instead of 128 bits
  - Another potential doubling by using AVX2+FMA on haswell/broadwell CPUs (leibniz)
    - FMA does 2 flops per vector element in one operation, basically a*v1 + v2.
  - Another potential doubling by using AVX512 on Xeon Phi and some future Skylake CPUs (Xeon Phi node, future cluster extension?)
    - Vector width of 512 bits instead of 128 bits
- This is why we prefer to compile applications from source rather than installing from binary, as too often the binary does not support newer CPU generations.
- *See also third lecture in our HPC introductory courses ("supercomputers for starters").*

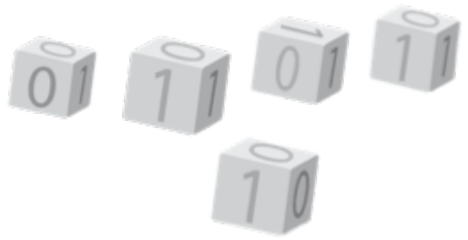
Logging on to leibniz

- Same procedure as for hopper
  - No new keys needed
  - But login.hpc.uantwerpen.be will still take you to the login nodes of hopper
  - login-leibniz.uantwerpen.be will take you to one of the login nodes of leibniz
- Specific login nodes:
  - login1-leibniz.uantwerpen.be
  - login2-leibniz.uantwerpen.be
- Internal names (on the cluster): ln1 and ln2 (long name ln1.leibniz.antwerpen.vsc etc.)

Available software

- The 2017a toolchain has been carried over to leibniz
  - All programs that had been used in the past two years before we started building the toolchain, have been installed except for a few that were no longer supported by the developers
- Software that did not compile correctly with the 2017a toolchain or showed a large performance regression, is installed in the 2016b toolchain
- We will not reinstall old versions of packages in the available toolchains unless there is a very very very good reason to keep using these older packages
  - They may not support newer CPUs well, even when recompiled
- See VSC web site: “User Portal” section “Available hardware” and go to the pages of the UAntwerp clusters.
  - There is a page listing all software installed in the 2017a toolchain

Modules



- We have switched to newer module management software
  - However, old commands still work almost the same
- One major distinction: It is no longer possible to load two modules with the same name but different versions
  - So no longer building up a toolchain combination by loading several hopper/* or leibniz/* modules
- Toolchains:
  - System toolchain:
    - Software compiled with the system compilers and libraries
    - Software installed from binaries
    - Refreshed with major operating system updates
  - Compiler toolchains:
    - Intel: Intel compilers, GNU compilers and Intel libraries
    - FOSS: GNU compilers, OpenMPI, OpenBLAS/Lapack/FFTW
    - Up to 2 annual refreshes (2016b, 2017a)

Modules (2)

- Cluster modules: Enable groups of application modules
  - leibniz/supported: Enables all currently supported application modules: up to 4 toolchain versions and the system toolchain modules
  - hopper/2017a: Enables only the 2017a compiler toolchain modules and the system toolchain modules
  - `module purge` will also unload this module, so you'll need to load a new cluster module first!
  - So it is a good practice to always load the appropriate cluster module first before loading any other module!
  - Hint: `module load $VSC_INSTITUTE_CLUSTER/supported`
- 2 types of application modules
  - Built with a specific version of a compiler toolchain, e.g., 2017a
    - Modules in a subdirectory that contains the toolchain version in the name
    - Try to support a toolchain for 2 years
  - Applications installed in the system toolchain
    - Modules in subdirectory centos7.

Modules
New commands

- New command:
  - `module spider flex` : Will show a list of all modules named “flex”
  - `module spider flex/2.6.4` : Will show extended information about the flex/2.6.4 module, including other modules that you might have to load to access this module
  - `module help` : Overview of all module subcommands
- We try to provide more information via “module help” as before
  - module help vsc-vnc/0.1

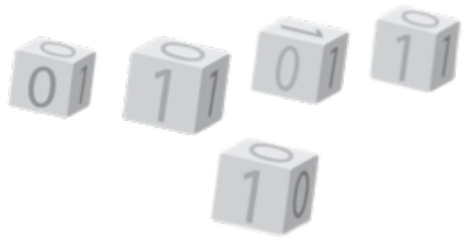
Resource specifications

- Change in the way memory is managed on CentOS 7 nodes
  - Very weak enforcement of memory limits on SL6 nodes
  - In CentOS 7, a new mechanism is enabled and used by the resource manager: cgroups
    - Better control of physical and virtual memory possible
    - Though currently hit by several bugs in the resource manager software
  - Be prepared for tougher control as soon as the bugs are corrected
    - Better stability of the cluster – fewer crashed nodes (jobs will crash without causing node crashes)
    - Better performance guarantee – swapping slows down cluster nodes too much
    - Protection of our hardware as SSDs don’t like swapping to them

Resource specifications

- Evolution in cluster architecture:
  - Hierarchical resource structure of compute node
    - 2 sockets
    - 2 NUMA domains per socket on some types of CPU (2 clusters of 7 cores on leibniz and BrENIAC)
    - Simultaneous MultiThreading (“hyperthreading”): 2 hardware threads per core (currently disabled on leibniz)
  - Don’t know these terms? *Third lecture of the HPC introduction course (“supercomputers for starters”)*
  - Old resource syntax didn’t work well with this hierarchy
- So we encourage to use the “resource syntax 2.0” in scripts from now on

Resource specifications
Syntax 2.0 - encouraged

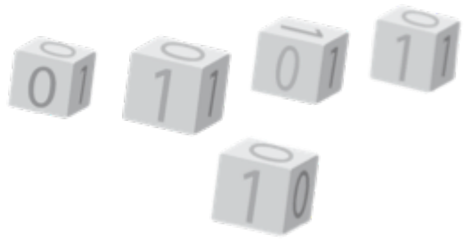


- Replaces CPU and memory specifications
- `-L` instead of `-l`
- Concepts:
  - **Task**: A job consists of a number of tasks.
    - Think of it as a process (e.g., one rank of a MPI job)
    - Must run in a single node (exception: SGI shared memory systems)
  - **Task group**: A group of tasks that need the same resources.
    - Most jobs have only one task group
    - E.g., all processes of a MPI run
  - **Logical processor**: Each task uses a number of logical processors
    - Hardware threads or cores
    - Specified per task
    - In a system with hyperthreading enabled, an additional parameter allows to specify whether the system should use all hardware threads or run one lproc per core.

Resource specifications
Syntax 2.0 - encouraged

- Concepts (Cntd):
  - **Placement**: Set locality of hardware resources for a task (the hierarchy on the node: socket – NUMA node – core – hardware thread)
  - **Feature**: Each compute node can have one or more special features. We use this for nodes with more memory
  - **Physical/resident memory**: Each task gets the specified amount of RAM allocated to it
    - In our setup, not enforced on a per-task but per-node basis
    - Tasks won't crash if you exceed this, but will start to use swap space
    - Related to the mem/pmem parameters of syntax 1.0
  - **Virtual memory**: This is RAM + swap space, again per task
    - In our setup, not enforced on a per-task but per-node basis
    - Tasks will crash if you exceed this.
    - Related to vmem/pvmem parameters of syntax 1.0
    - We will enforce the use of the parameter related to virtual memory!

Resource specifications
Syntax 2.0 - encouraged



- Specifying tasks and lprocs:
  `#PBS -L tasks=4:lprocs=7`
  - Each `-L` line contains all resource specifications for a single task group
  - Requests 4 tasks using 7 lprocs each (cores in the current setup of hopper and leibniz)
  - The scheduler should be clever enough to put these 4 tasks on a single 28-core node and assign cores to each task in a clever way
- Some variants:
  - `#PBS -L tasks=1:lprocs=all:place=node` : Will give you all cores on a node.
    - Does not work without `place=node`!
  - `#PBS -L tasks=4:lprocs=7:usecores` : Stresses that cores should be used instead of hardware threads
- Rather than thinking of processes (tasks) and threads in a process (lprocs) you can also think of a task as a node and lprocs the number of cores per node

Resource specifications
Syntax 2.0 - encouraged

- Adding physical memory:
  ```
  #PBS -L tasks=4:lprocs=7:memory=20gb
  ```
  - Requests 4 tasks using 7 lprocs each and 20gb memory per task
- Adding virtual memory:
  ```
  #PBS -L tasks=4:lprocs=7:memory=20gb:swap=25gb
  ```
  - Confusing name! This actually requests 25gb of virtual memory, not 25gb of swap space (which would mean 20gb+25gb=45gb of virtual memory).
  - We have no swap space on leibniz, so it is best to take swap=memory (at least if there is enough physical memory).
  - And in this case there is no need to specify memory as the default is memory=swap
    ```
    #PBS -L tasks=4:lprocs=7:swap=20gb
    ```
  - Specifying swap will be enforced!

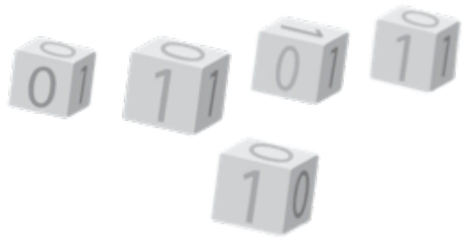
Resource specifications
Syntax 2.0 - encouraged

- Adding additional features:
  ```
  #PBS -L tasks=4:lprocs=7:memory=50gb:feature=mem256
  ```
  - Requests 4 tasks using 7 lprocs each and 60gb memory per task, and instructs the scheduler to use nodes with 256GB of memory so that all 4 tasks can run on the same node rather than using two regular nodes and leave the cores half empty.
  - Most important features listed on the VSC website (UAntwerp hardware pages)
  - As before we ask to help the scheduler and specify feature=mem256 if you need more than 4gb of memory per core (leibniz) or 2.75gb per core (hopper) to make sure nodes are used as efficiently as possible.
- Wall time etc. is specified in the same way as before (-l)
  - Maximum job wall time of 3 days on leibniz!

Resource specifications
Syntax 1.0 - deprecated

- This is the `-l` option as used on turing and hopper before
- Strongly discouraged as it turns out that the implementation is extremely buggy
  - Some bugs remained unnoticed since our users didn't use some of the features
  - Others remained unnoticed since users worked around them but didn’t tell us so we couldn’t report them
  - It works best when you do the opposite of what the manual advises you to do
- When adapting job scripts: Remember that each node of leibniz has 28 cores!
- Map onto version 2.0 syntax: One option is nodes → tasks and ppn → lprocs

Resource specifications
Syntax 1.0 - deprecated



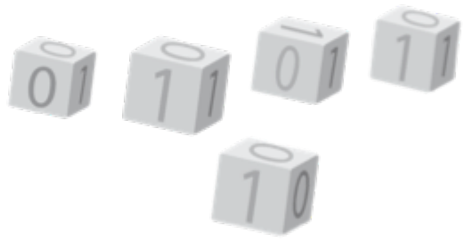
- Memory parameters: This is the buggy part

	Per job	Per core
Physical memory	mem	pmem
Virtual memory	vmem	pvmem

- If you want to run a single multithreaded process: mem and vmem will work for you (e.g., Gaussian).
- If you want to run a MPI job with one process per core: Any of the above will work most of the time, but remember that mem/vmem is for the whole job, not one node!
- Hybrid MPI/OpenMP job: Best to use mem/vmem
- So when enforcing, we'll likely enforce vmem.
- Source of trouble: ulimit are often set to a value which is only relevant for non-threaded jobs, or even smaller (the latter due to another bug), and don't correspond to the (correct) values for the cgroups.

Environment variables in job scripts

Passing variables



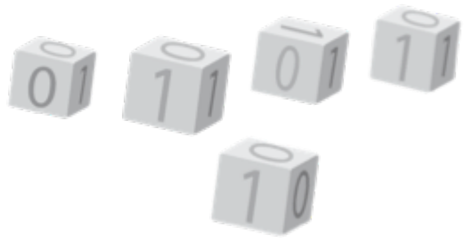
- With the `-v` argument, it is possible to pass variables from the shell to a job or define new variables.
- Example:

  ```
  $ export myvar="This is a variable”
  $ myothervar="This is another variable"
  $ qsub -I -L tasks=1:lprocs=1 -l walltime=10:00 -v myvar,myothervar,multiplier=1.005
  ```

  Next on the command line once the job starts:

  ```
  $ echo $myvar
  This is a variable
  $ echo $myothervar

  $ echo $multiplier
  1.005
  ```
- What happens?
  - myvar is passed from the shell (export)
  - myothervar is an empty variable since it was not exported in the shell from which qsub was called
  - multiplier is a new variable, its value is defined on the qsub command line
- qsub doesn’t like multiple -v arguments, only the last one is used

Dependent jobs



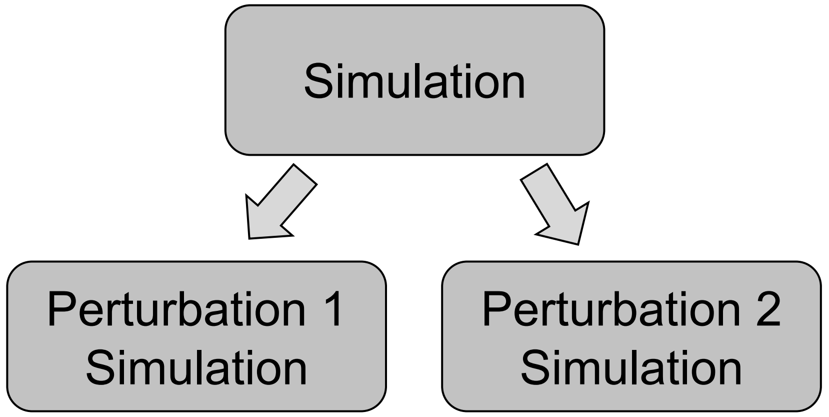
- It is also possible to instruct Torque/Moab to only start a job after finishing one or more other jobs:

  ```
  qsub -W depend=afterany:jobid1:jobid2 jobdepend.sh
  ```

  will only start the job defined by jobdepend.sh after the jobs with jobid1 and jobid2 have finished
- Many other possibilities, including
  - Start another job only after a job has ended successfully
  - Start another job only after a job has failed
  - And many more, check the Moab manual for the section "Job Dependencies"
- Useful to organize jobs, e.g.,
  - Need to run a sequence of simulations, each one using the result of the previous one, but bumping into the 3-day walltime limit so not possible in a single job
  - Need to run simulations using results of the previous one, but with a different optimal number of nodes
  - Extensive sequential pre- or postprocessing of a parallel job
  - Powerful in combination with -v

Dependent jobs

Example



job.sh

```
#!/bin/bash
#PBS -L tasks=1:lprocs=1:swap=1gb
#PBS -l walltime=30:00

cd $PBS_O_WORKDIR

echo "10" >outputfile
sleep 300

multiplier=5
mkdir mult-$multiplier ; pushd mult-$multiplier
number=$(cat ../outputfile)
echo $(($number*$multiplier)) >outputfile; sleep 300
popd

multiplier=10
mkdir mult-$multiplier ; pushd mult-$multiplier
number=$(cat ../outputfile)
echo $(($number*$multiplier)) >outputfile; sleep 300
```

Dependent jobs

Example

- A job whose result is used by 2 other jobs

job_first.sh

```
#!/bin/bash
#PBS -L tasks=1:lprocs=1:swap=1gb
#PBS -l walltime=10:00

cd $PBS_O_WORKDIR

echo "10" >outputfile
sleep 300
```

job_depend.sh

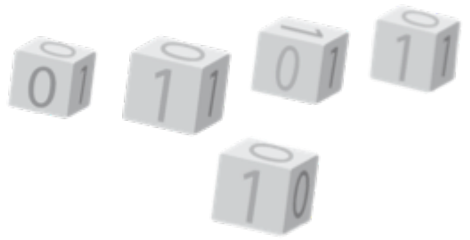
```
#!/bin/bash
#PBS -L tasks=1:lprocs=1:swap=1gb
#PBS -l walltime=10:00

cd $PBS_O_WORKDIR

mkdir mult-$multiplier ; cd mult-$multiplier
number=$(cat ../outputfile)
echo $(($number*$multiplier)) >outputfile
sleep 300
```

job_launch.sh

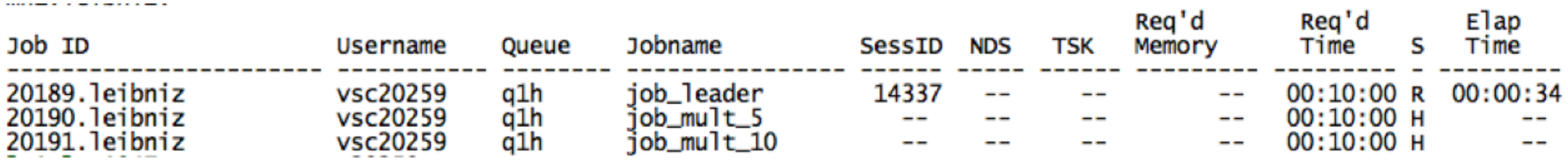
```
#!/bin/bash
first=$(qsub -N job_leader job_first.sh)
qsub -N job_mult_5  -v multiplier=5  -W depend=afterany:$first job_depend.sh
qsub -N job_mult_10 -v multiplier=10 -W depend=afterany:$first job_depend.sh
```

Dependent jobs

Example

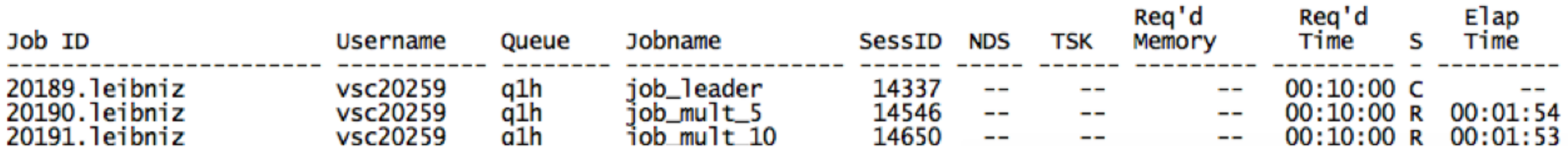


- After start of the first job – `qstat -u vsc20259`



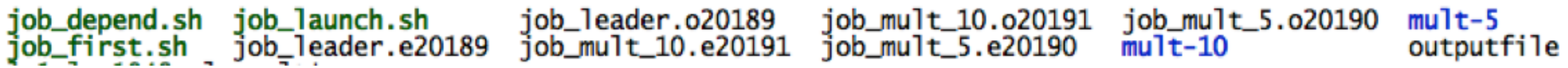
```
                                                                      Req'd       Req'd       Elap
Job ID                  Username    Queue    Jobname          SessID  NDS   TSK   Memory      Time    S   Time
----------------------- ----------- -------- ---------------- ------ ----- ------ ---------- --------- - ---------
20189.leibniz           vsc20259    q1h      job_leader        14337   --     --         --  00:10:00 R  00:00:34
20190.leibniz           vsc20259    q1h      job_mult_5           --   --     --         --  00:10:00 H        --
20191.leibniz           vsc20259    q1h      job_mult_10          --   --     --         --  00:10:00 H        --
```

- Some time later:



```
                                                                      Req'd       Req'd       Elap
Job ID                  Username    Queue    Jobname          SessID  NDS   TSK   Memory      Time    S   Time
----------------------- ----------- -------- ---------------- ------ ----- ------ ---------- --------- - ---------
20189.leibniz           vsc20259    q1h      job_leader        14337   --     --         --  00:10:00 C        --
20190.leibniz           vsc20259    q1h      job_mult_5        14546   --     --         --  00:10:00 R  00:01:54
20191.leibniz           vsc20259    q1h      job_mult_10       14650   --     --         --  00:10:00 R  00:01:53
```

- When finished: Output of ls:



```
job_depend.sh  job_launch.sh     job_leader.o20189   job_mult_10.o20191  job_mult_5.o20190  mult-5
job_first.sh   job_leader.e20189 job_mult_10.e20191  job_mult_5.e20190   mult-10            outputfile
```

- `cat outputfile` 10

  `cat mult-5/outputfile` 50

  `cat mult-10/outputfile` 100

Visualization node

- Leibniz has one visualization node with GPU and VirtualGL
  - Access via TurboVNC
  - Simple desktop environment with the xfce window manager
  - Start OpenGL programs via vglrun. Otherwise the OpenGL libraries and graphics accelerator are not found.
- The same software stack minus the OpenGL libraries is also available on all login nodes of hopper and leibniz and can be used with programs that only use 2D graphics or a built-in 3D renderer.
- You need a VNC client on your PC. Some give better results than others.
- You really need a basic understanding of networks and ssh to get this to work; it is *not a solution that gives you the ease-of-use of a PC on a supercomputer*.

Visualization node

- How does VNC work?
  - Basic idea: Don’t send graphics commands over the network, but render the image on the cluster and send images.
  - Step 1: Load the vsc-vnc module and start the VNC server. You can then log out again.
  - Step 2: Create an SSH tunnel to the port given when starting the server.
  - Step 3: Connect to the tunnel with your VNC client.
  - Step 4: When you’ve finished, don’t forget to kill the VNC server. Disconnecting will not kill the server!
    - We’ve configured the desktop in such a way that logging out on the desktop will kill the VNC server, at least if you started through the vnc-* commands shown in the help of the module (vnc-xfce).
- Full instructions: Page “Remote visualization @ UAntwerp” on the VSC web site (https://www.vscentrum.be/infrastructure/hardware/hardware-ua/visualization)

Visualization node

- Using OpenGL:
  - The VNC server currently emulates a X-server without the GLX extension, so OpenGL programs will not run right away.
  - On the visualisation node: Start OpenGL programs through the command vglrun
    - vglrun will redirect OpenGL commands from the application to the GPU on the node
    - and transfer the rendered 3D image to the VNC server
  - E.g.: `vglrun glxgears` will run a sample OpenGL program
- Our default desktop for VNC is xfce.
  GNOME is not supported due to compatibility problems

GPU compute nodes

- Goals
  - Experiment with GPU computing
  - Check efficiency of applications compared to pure CPU applications
    - So we are particularly interested in applications where the developer paid attention to efficiency on both CPU and GPU and not just one of these
    - Our GPU compute nodes are only economically efficient when an application runs at least 3 times faster than on a single dual socket CPU node
- Software stack based on the standard CUDA distribution
  - Additional packages installed on demand
  - GCC 7.X supports OpenMP and OpenACC offload to the GPU but is not yet installed

GPU compute nodes

- Getting access:
  - Mail hpc@uantwerpen.be and explain why you need the GPUs
  - Not yet into the job system
    - Once in the job system, the maximum wall time for a job will be 24h as we want to give everybody a chance to experiment
  - Feedback about experiences required
- Future purchases will depend on the efficiency of the investment
- Further instructions: Page “GPU computing@ UAntwerp” on the VSC web site (https://www.vscentrum.be/infrastructure/hardware/hardware-ua/gpu-computing)

Xeon Phi node

- Xeon Phi is Intel’s answer to GPU computing
  - Many (68) simple 64-bit x86 cores on a single chip
  - Compatible with regular Xeon processors
  - Performance from vector instructions (512-bit vectors)
  - 512-bit vector instruction set will also migrate to regular Intel CPUs
    - Already in Skylake Xeon and most X-series 7xxx.
  - 4 hardware threads/core to hide latency
  - The Xeon Phi can run a full operating system and thus run as the main and only processor of a compute node
- Goal: Experiment with Xeon Phi and study application behaviour
- Our version is still the coprocessor version but since your directories are mounted on the Xeon Phi expansion board you can run native applications
  - New Xeon Phi clusters run Xeon Phi as the main CPU

Xeon Phi node

- Software:
  - Compile with the regular Intel compilers with the right compiler options
  - GCC should also work, but not tested yet
  - Applications installed on demand
- Getting access:
  - Mail hpc@uantwerpen.be and explain why you need the Xeon Phi
  - Not yet into the job system
    - Once in the job system, the maximum wall time for a job will be 24h as we want to give everybody a chance to experiment
  - Feedback about experiences required
- Further instructions: Page “Xeon Phi testbed@ UAntwerp” on the VSC web site (https://www.vscentrum.be/infrastructure/hardware/hardware-ua/xeonphi)

Quid hopper?

- Hopper is also being updated to CentOS 7
  - Same application stack as leibniz
  - Old applications can be made available again on demand but no guarantee that they will still run
- Same modifications for the job scripts
  - But remember hopper has 20 cores per node and leibniz 28
- Hopper may be the better choice for applications that don’t sufficiently exploit vectorization or use few cores while consuming a lot of memory
  - Turbo mode which is enabled on hopper compute nodes somewhat compensates for underusing cores
- login3-hopper.uantwerpen.be already on CentOS 7; jobs submitted on that node will run on CentOS 7 nodes

Documentation

- Check the “User portal” on the VSC web site and in particular the UAntwerp pages of the “Available hardware” section.
- Documentation of Torque and Moab can be found on the Documentation pages of Adaptive Computing.
  - Moab Workload Manager Administrator Guide (we are on the 9.1.x versions)
  - Torque Administrator Guide (we are on the 6.1.x versions)