FENIX

RESEARCH INFRASTRUCTURE

How to exploit ICEI scalable computing services

Sadaf Alam

Swiss National Supercomputing Centre (CSCS)

Webinar

December 10, 2019

The ICEI project has received funding from the European Union's Horizon 2020 research and innovation programme under the grant agreement No 800858.

Piz Daint—A user’s perspective

- A multi-faceted supercomputer
  - **Scalable** computing services (with heterogenous multi-core and hybrid GPUs)
  - **Interactive** computing services (with multi-core and GPUs—JupyterHub and visualisation applications)
  - A Linux cluster environment with several, widely-used **code development** tools
  - A slurm cluster configuration for **running jobs**
  - Common **storage** interfaces for file transfers
  - HPC **container** services for containerized workloads
  - User and quota **management** tools

Piz Daint specifications

Model	Cray XC40/XC50
XC50 Compute Nodes	Intel® Xeon® E5-2690 v3 @ 2.60GHz (12 cores, 64GB RAM) and NVIDIA® Tesla® P100 16GB - 5704 Nodes
XC40 Compute Nodes	Two Intel® Xeon® E5-2695 v4 @ 2.10GHz (2 x 18 cores, 64/128 GB RAM) - 1813 Nodes
Login Nodes	Intel® Xeon® CPU E5-2650 v3 @ 2.30GHz (10 cores, 256 GB RAM)
Interconnect Configuration	Aries routing and communications ASIC, and Dragonfly network topology
Scratch capacity	8.8 PB

FENIX RI

Piz Daint programming environment

- C, C++ and Fortran compilers including multi-threading support
  - Cray, Intel, GNU and PGI
- MPI library
  - Optimized library from Cray, based on MPICH
- GPU
  - Nvidia CUDA tools
  - PGI OpenACC compiler
- Libraries:
  - Numerical libraries
  - Parallel IO libraries

Available using a module framework:
https://user.cscs.ch/computing/compilation/

```
module load daint-gpu
module load daint-mc
```

FENIX RI

Piz Daint resource management & scheduling system

- Slurm batch system for the submission, control and management of user jobs (https://user.cscs.ch/access/running)

Name	Max time	Max nodes	Brief Description
debug	30 min	4	Quick turnaround for test jobs (one per user)
large	12 h	4400	Large scale work, by arrangement only
long	72 h	4	Maximum 5 long jobs in total (one per user)
normal	24 h	2400(gpu)/ 512(mc)	Standard queue for production work
prepost	30 min	1	High priority pre/post processing
xfer	24h	1	Data transfer queue

FENIX RI

Piz Daint storage & data services

- Multiple file systems
  - Performance characteristics
  - Functional characteristics
- Data transfer services (internal and external)

	/scratch (Piz Daint)	/scratch (Clusters)	/users	/project	/store
Type	Lustre	GPFS	GPFS	GPFS	GPFS
Quota	Soft quota 1 M files	None	10 GB/user 100K files	Maximum 50K files/TB	Maximum 50K files/TB
Expiration	30 days	30 days	Account closure	End of the project	End of the contract
Data Backup	None	None	90 days	90 days	90 days
Access Speed	Fast	Fast	Slow	Medium	Slow
Capacity	8.8 PB	1.4 PB	86 TB	4.7 PB	3.6 PB

FENIX RI

Piz Daint HPC container service

- Tools for running a Linux container on HPC systems:
  - Sarus
  - Singularity
- Sarus is a software to run Linux containers:
  - Security oriented to HPC systems
  - Compatibility with the Open Container Initiative (OCI) standards
  - Compatibility with the presence of a workload manager
  - Creation of container filesystems tailored for diskless nodes and parallel filesystems https://user.cscs.ch/tools/containers/sarus/

```
module load daint-gpu # or daint-mc
module load sarus
```

FENIX RI

Piz Daint scalable tools and frameworks

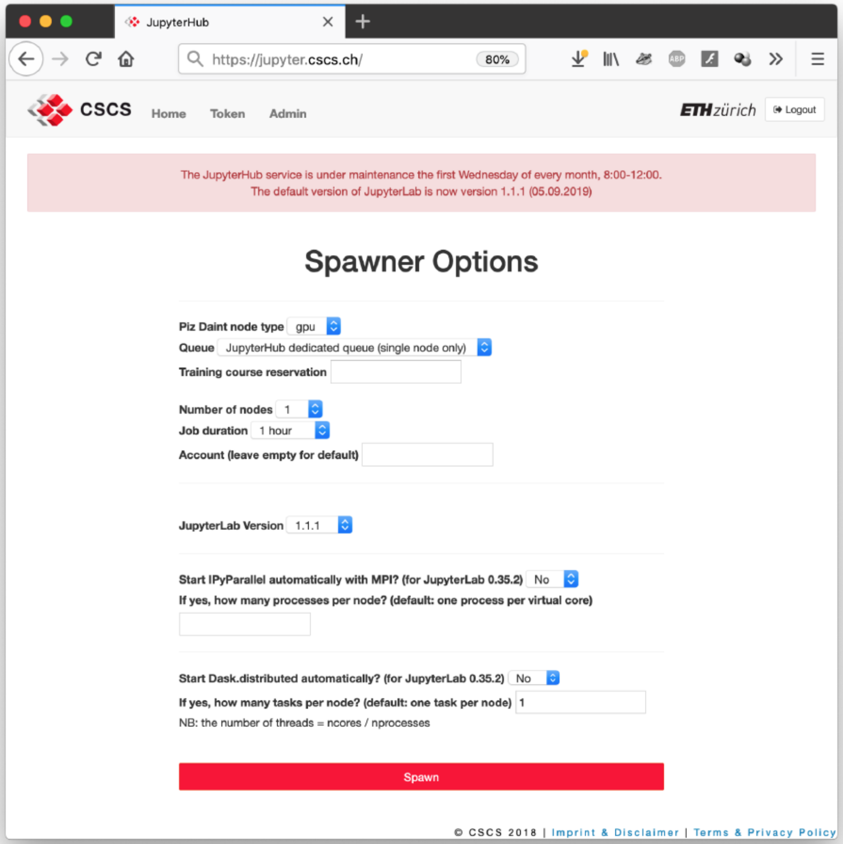
- Debugging and performance analysis tools
  - https://user.cscs.ch/computing/analysis/
  - Debug (Multi-core, GPU & MPI enabled)
    - https://user.cscs.ch/computing/analysis/ddt/
    - Nvidia debugger
  - Performance (Multi-core, GPU & MPI enabled)
    - Cray performance tools
    - Score-p/Vampir/Scalasca
    - Nvidia performance tools
- Data science frameworks
  - https://user.cscs.ch/computing/data_science/
    (frameworks e.g. Python, TensorFlow, Theano, Spark, Dask, etc.)

FENIXRI

Piz Daint interactive computing services

- https://user.cscs.ch/tools/interactive/
- https://user.cscs.ch/computing/visualisation/

- **https://jupyter.cscs.ch**
- JupyterHub provides Jupyter servers on demand for users of Piz Daint
- Log in with your CSCS credentials
- Spawns a server on a dedicated compute node of Piz Daint (gpu or mc)
- Launch time should be 5 minutes maximum
- To load modules or activate virtual environments, add these commands to `$HOME/.jupyterhub.env`



The JupyterHub service is under maintenance the first Wednesday of every month, 8:00-12:00.
The default version of JupyterLab is now version 1.1.1 (05.09.2019)

Spawner Options

Piz Daint node type: gpu
Queue: JupyterHub dedicated queue (single node only)
Training course reservation:

Number of nodes: 1
Job duration: 1 hour
Account (leave empty for default):

JupyterLab Version: 1.1.1

Start IPyParallel automatically with MPI? (for JupyterLab 0.35.2): No
If yes, how many processes per node? (default: one process per virtual core)

Start Dask.distributed automatically? (for JupyterLab 0.35.2): No
If yes, how many tasks per node? (default: one task per node): 1
NB: the number of threads = ncores / nprocesses

Spawn

© CSCS 2018 | Imprint & Disclaimer | Terms & Privacy Policy

How to access Piz Daint

- Start with a valid account
  - Link: https://www.cscs.ch/user-lab/applying-for-accounts/ & https://account.cscs.ch (assuming a valid allocation or grant to use resources)
- Front-end ela (external login interface)
  - Minimal Linux environment (`$ ssh ela.cscs.ch`)
  - ssh Piz Daint from ela (`$ ssh daint.cscs.ch`)
  - Start file transfer if needed
  - Is not equipped with programming environment and tools
  - Piz Daint scratch is not available

FENIX RI

How to compile code

- Check your environment
  - `$ module list`
- Module loaded at login
  - XC50 (Haswell and P100) `daint-gpu`
  - XC40 (Broadwell) `daint-mc`
  - `$ module avail`
  - `$ module swap`

`cc` for C code, `CC` for C++ code and `ftn` for Fortran code

- Compiler options (check man pages for details)
  - Cray, GNU, Intel and PGI (wrappers for C, C++ and Fortran)
  - GPU: Nvidia and directive based programming

FENIXRI

How to run code/submit a batch job

- Slurm is the batch scheduling system that allows users to run jobs with specific settings

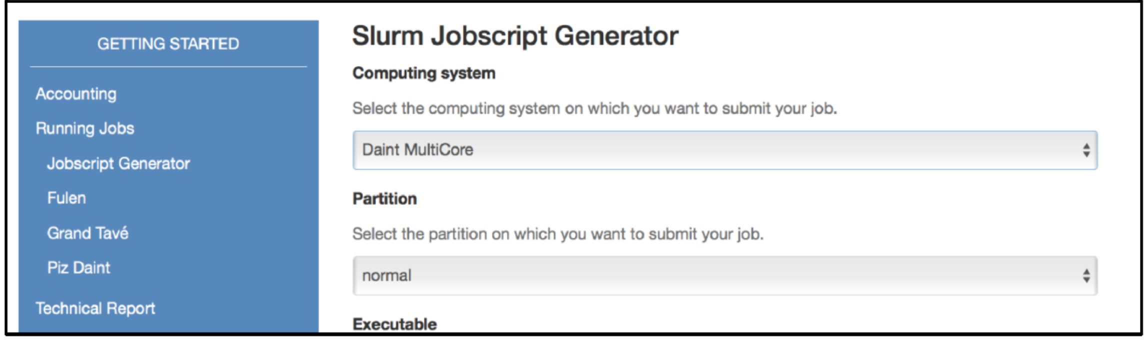
job.sh

```
#!/bin/bash -l
#SBATCH --nodes=10
#SBATCH --time=0:30:00
#SBATCH --partition=normal
#SBATCH --constraint=gpu
[…]

srun myprogram
```

```
$ sbatch job.sh
```

Slurm Jobscript Generator

https://user.cscs.ch/access/running/jobscript_generator/



Slurm Jobscript Generator

Computing system

Select the computing system on which you want to submit your job.

Daint MultiCore

Partition

Select the partition on which you want to submit your job.

normal

Executable

Monitoring your job and system status

- Watch your jobs in queues with

```
$ squeue -u ${USER}
```

```
daint103:~$ squeue -u simbergm
   JOBID     USER ACCOUNT            NAME  ST REASON    START_TIME               TIME  TIME_LEFT NODES CPUS
11942503 simbergm csstaff hpx-3662-gcc-7   R None      16:36:57                30:26    5:29:34     1   24
11945966 simbergm csstaff hpx-3712-gcc-7   R None      16:44:24                22:59    5:37:01     1   72
11947200 simbergm csstaff hpx-3229-clang  PD BeginTime 17:34:15                 0:00    6:00:00     1    1
11947180 simbergm csstaff hpx-3684-gcc-7  PD BeginTime Tomorr 00:19             0:00    6:00:00     1    1
```

- Observe state of queues with

```
$ sinfo -o"%P %.5a %.10l %.6D %.6t"
```

```
daint103:~$ sinfo
PARTITION AVAIL JOB_SIZE  TIMELIMIT   CPUS  S:C:T   NODES STATE      NODELIST
debug     up    1-4           30:00     72 2:18:2      2 allocated  nid00[448-449]
debug     up    1-4           30:00    24+ 1+:12+     14 idle       nid0[0008-0011,0450-0451,3508-3511,4276-
4279]
xfer      up    1        1-00:00:00      9  9:1:1      2 idle       nordend0[3-4]
uftp      up    1        1-00:00:00      0  0:0:0      0 n/a
cscsci    up    1        1-00:00:00    24+ 1+:12+      7 down$      nid0[0125,0299,3541-3543,4579,5967]
cscsci    up    1        1-00:00:00    24+ 1+:1+:     28 maint      nid0[0124,0126,1144-1147,1804-1807,3492-
3495,3576-
```

FENIX RI

Data and storage orchestration

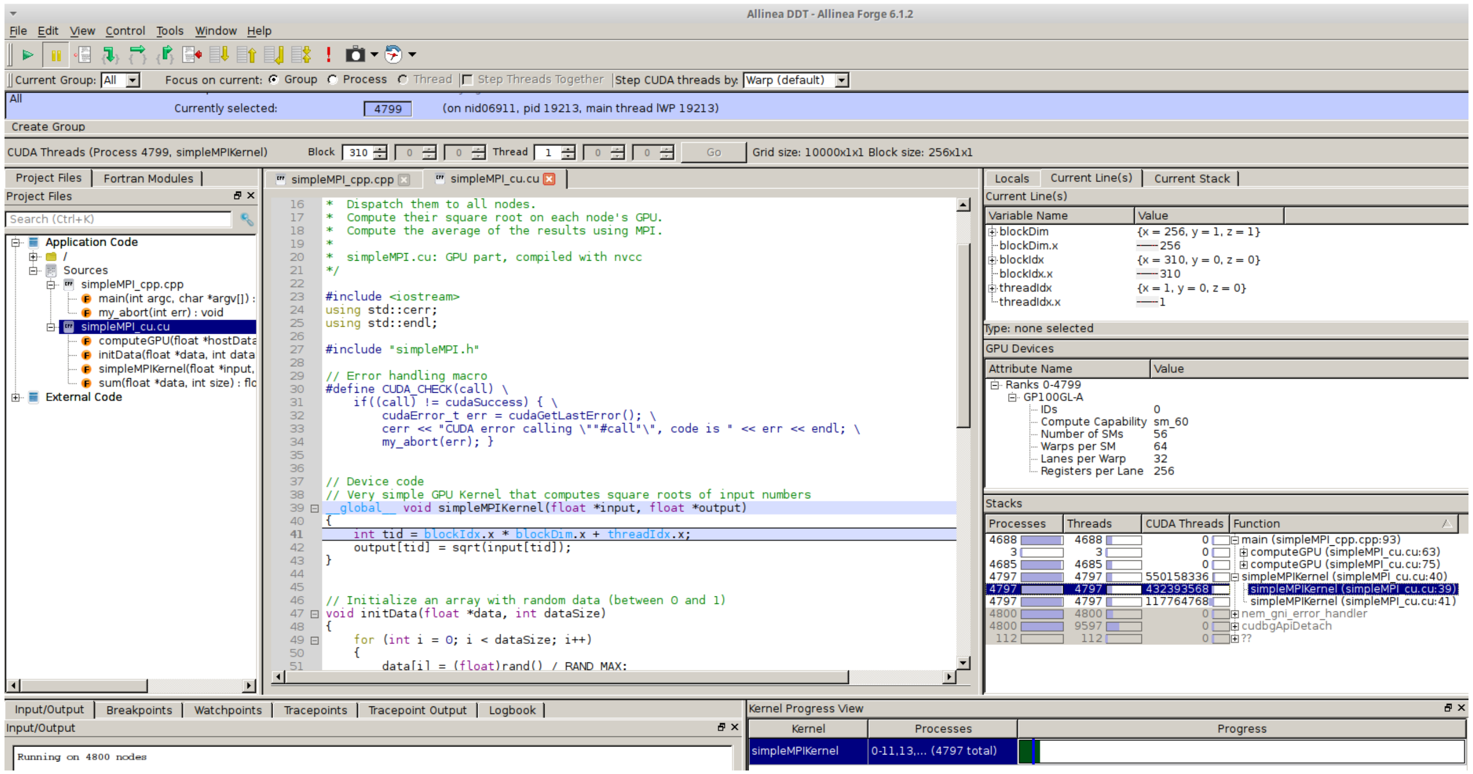
- Transfer queue
  - to address data transfers between internal CSCS file systems (/user, /project, /store, and /scratch)
- Outside CSCS:
  - Classic file transfer service
  - Support for Fenix archival data repositories (OpenStack Swift)

```bash
#!/bin/bash -l
#
#SBATCH --time=02:00:00
#SBATCH --ntasks=1
#SBATCH --partition=xfer

module unload xalt
command="rsync -av"
echo -e "$SLURM_JOB_NAME started on $(date):\n $command $1 $2\n"
srun -n $SLURM_NTASKS $command $1 $2
echo -e "$SLURM_JOB_NAME finished on $(date)\n"

if [ -n "$3" ]; then
 # unset memory constraint enabled on xfer partition
 unset SLURM_MEM_PER_CPU
 # submit job with dependency
 sbatch --dependency=afterok:$SLURM_JOB_ID $3
fi
```

FENIX RI

How to troubleshoot

- Frequently asked questions: https://user.cscs.ch/access/faq/
- Correctness, performance and scaling issues
  - https://user.cscs.ch/computing/analysis/



Currently selected: 4799 (on nid06911, pid 19213, main thread lWP 19213)

CUDA Threads (Process 4799, simpleMPIKernel) Block 310 0 0 Thread 1 0 0 Grid size: 10000x1x1 Block size: 256x1x1

```
Application Code
  /
  Sources
    simpleMPI_cpp.cpp
      main(int argc, char *argv[]) :
      my_abort(int err) : void
    simpleMPI_cu.cu
      computeGPU(float *hostData
      initData(float *data, int data
      simpleMPIKernel(float *input,
      sum(float *data, int size) : flc
External Code
```

simpleMPI_cpp.cpp | simpleMPI_cu.cu

```
*  Dispatch them to all nodes.
*  Compute their square root on each node's GPU.
*  Compute the average of the results using MPI.
*
*  simpleMPI.cu: GPU part, compiled with nvcc
*/

#include <iostream>
using std::cerr;
using std::endl;

#include "simpleMPI.h"

// Error handling macro
#define CUDA_CHECK(call) \
    if((call) != cudaSuccess) { \
        cudaError_t err = cudaGetLastError(); \
        cerr << "CUDA error calling \""#call"\", code is " << err << endl; \
        my_abort(err); }


// Device code
// Very simple GPU Kernel that computes square roots of input numbers
__global__ void simpleMPIKernel(float *input, float *output)
{
    int tid = blockIdx.x * blockDim.x + threadIdx.x;
    output[tid] = sqrt(input[tid]);
}


// Initialize an array with random data (between 0 and 1)
void initData(float *data, int dataSize)
{
    for (int i = 0; i < dataSize; i++)
    {
        data[i] = (float)rand() / RAND_MAX;
```

Current Line(s)

Variable Name	Value
blockDim	{x = 256, y = 1, z = 1}
blockDim.x	256
blockIdx	{x = 310, y = 0, z = 0}
blockIdx.x	310
threadIdx	{x = 1, y = 0, z = 0}
threadIdx.x	1

Type: none selected

GPU Devices

Attribute Name	Value
Ranks 0-4799	
GP100GL-A	
IDs	0
Compute Capability	sm_60
Number of SMs	56
Warps per SM	64
Lanes per Warp	32
Registers per Lane	256

Stacks

Processes	Threads	CUDA Threads	Function
4688	4688	0	main (simpleMPI_cpp.cpp:93)
3	3	0	computeGPU (simpleMPI_cu.cu:63)
4685	4685	0	computeGPU (simpleMPI_cu.cu:75)
4797	4797	550158336	simpleMPIKernel (simpleMPI_cu.cu:40)
4797	4797	432393568	simpleMPIKernel (simpleMPI_cu.cu:39)
4797	4797	117764768	simpleMPIKernel (simpleMPI_cu.cu:41)
4800	4800	0	nem_gni_error_handler
4800	9597	0	cudbgApiDetach
112	112	0	??

Input/Output

```
Running on 4800 nodes
```

Kernel Progress View

Kernel	Processes	Progress
simpleMPIKernel	0-11,13,... (4797 total)	

User and project management tools

- Checking your computing budget
  - Group usage
    `sbucheck`
    reports group usage across various CSCS computing systems
  - Daily usage
    `monthly_usage`
    `monthly_usage -individual`
    usage per group and group member
  - Overview of resources with the accounting and resource tool (via browser)

List of HBP Projects using Piz Daint

Virtual Epileptic Patient (PI: V. Jirsa)
Full-scale hippocampus model (PI: M. Migliore)
Cerebellum modelling (PI: E. D'Angelo)
Neurorobotics Platform (NRP) development (PI: A. von Arnim)
Image segmentation toolkit (ilastik) workflow (PI: A. Kreshuk)
NEST network construction and simulation (PI: H. E. Plesser)

SimLab Neuroscience (PIs: A. Morrison, B. Orth)

Model validation Service (PI: A Davison)

Neuromorphic Computing front-end services (PI: A. Davison)

https://www.humanbrainproject.eu/en/follow-hbp/news/nine-projects-from-hbp-enabled-by-fenix-consortium-partner-eth-zuerich-cscs-e-infrastructure/

FENIX RI

References

- Fenix research infrastructure : https://fenix-ri.eu
- CSCS user portal: https://user.cscs.ch
- Piz Daint specifications: https://www.cscs.ch/publications/news/2017/factsheetpizdaintoneofthemostpowerfulsupercomputersintheworld/
- CSCS User Lab day: https://github.com/eth-cscs/UserLabDay
- CSCS training events (upcoming): https://www.cscs.ch/events/upcoming-events/
- CSCS training events (past—links to material): https://www.cscs.ch/events/past-events/
- CSCS service catalog: https://www.cscs.ch/services/service-catalog

FENIX

RESEARCH INFRASTRUCTURE

Contact Details and Additional Information

https://fenix-ri.eu/contact-us

https://fenix-ri.eu/media/webinars

https://fenix-ri.eu/infrastructure/resources

Thank you for your attention

FENIX RI