SCALEXA MSA Workshop

JSC's Setup

19 June 2023 | Andreas Herten, Sebastian Achilles | Jülich Supercompting Centre

Outline

- MSA at JSC
  - Motivation
  - DEEP
  - JSC Production Systems
  - JUWELS Systems
    - Specs
    - Network
- Using an MSA System
  - xenv
  - env
  - renv
  - Use All The Tools
- Exercises
  - Overview
  - Infrastructure

MSA at JSC

Motivation

- CPU-only systems: versatile, but power-limited
- Accelerated nodes: efficient, but only for sub-set of workloads

→ Match system designs to workloads *(not only for compute!)*

- MSA: Combine disjoint systems of distinct components to ***super*-system** → enable *mid*-granular, heterogeneous workloads

⇒ **Modular Supercomputing**

- Concept developed in **DEEP projects** targeting production
- Modular supercomputers at JSC: DEEP, JURECA, JUWELS*, JUPITER*
- Other modular supercomputers: MeluXina, Leonardo, MareNostrum5

The hardware Prototypes

2015

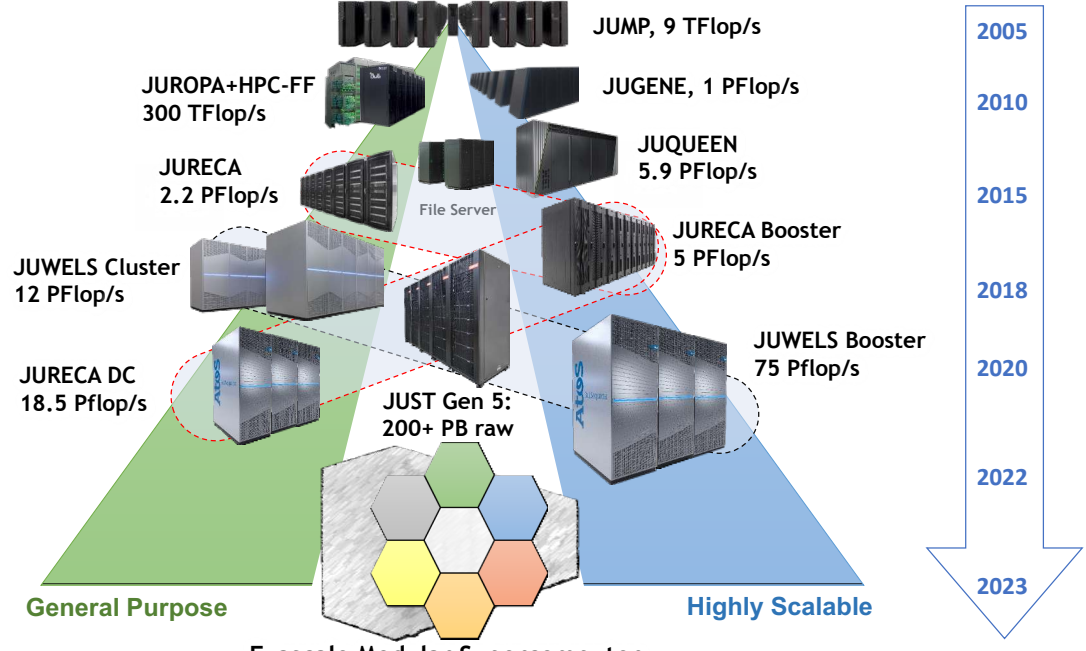
DEEP Prototype
128 Xeon + 284 KNC nodes
InfiniBand + 1.5Gbit Extoll
550 TFlop/s

2016

DEEP-ER Prototype
16 Xeon + 8 KNL nodes
100Gbit Extoll
40 TFlop/s

2020

DEEP-EST Prototype
55 Cluster + 75 Booster + 16 Data Analytics
100 Gbit Extoll + InfiniBand + Eth
800 TFlop/s


JUMP, 9 TFlop/s
JUROPA+HPC-FF
300 TFlop/s
JUGENE, 1 PFlop/s
JUQUEEN
5.9 PFlop/s
JURECA
2.2 PFlop/s
File Server
JURECA Booster
5 PFlop/s
JUWELS Cluster
12 PFlop/s
JUWELS Booster
75 Pflop/s
JURECA DC
18.5 Pflop/s
JUST Gen 5:
200+ PB raw
2005
2010
2015
2018
2020
2022
2023
General Purpose
Highly Scalable
Exascale Modular Supercomputer

JUWELS Overall Architecture

JUWELS Cluster (2018)

- 2511 compute nodes (2× Skylake)
- 48 GPU nodes (4× V100 w/ NVLink2)
- Mellanox EDR 100 Gbit/s network, fat-tree topology (1:2@L1)
- 12 PFLOP/s

JUWELS Booster (2020)

- 936 compute nodes (2× AMD Rome, 4× A100 w/ NVLink3)
- Mellanox HDR 200 Gbit/s network, DragonFly+ topology
- 73 PFLOP/s

JUWELS

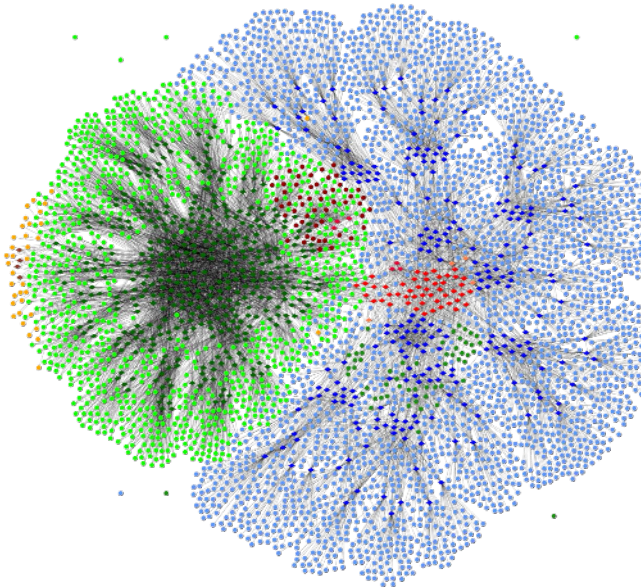
Cluster Booster Integration

Fully integrated system: **JUWELS** with Cluster and Booster modules

- File system: GPFS
- Network: InfiniBand
- Workload management: *PS*Slurm
- Resource management: ParaStation / ParaStation Slurm

Picture legend:

- Cluster CPU node
- Cluster GPU node
- Cluster switch
- Cluster gateway
- Top-level switch
- Booster node
- Booster switch
- Booster gateway
- JUST
- Service node



Graphic provided by D. Alvarez

Using an MSA System

Usage Guidelines

- Base functionality: Slurm's **hetjobs**

  Allocation Separate options of job components with `#SBATCH hetjob` (batch script) or `:` (interactive); most options propagated from first job component, some not, can be overwritten

Usage Guidelines

- Base functionality: Slurm's **hetjobs**

  Allocation Separate options of job components with `#SBATCH hetjob` (batch script) or `:` (interactive); most options propagated from first job component, some not, can be overwritten

  Execution Separate job components of `srun` with `:` (same communicator); target individual components with `--het-group=0-2,4` for job steps (separate communicators); suffixed `$SLURM_..._HET_GROUP_N` environment variables

Usage Guidelines

- Base functionality: Slurm's **hetjobs**

  Allocation: Separate options of job components with `#SBATCH hetjob` (batch script) or `:` (interactive); most options propagated from first job component, some not, can be overwritten

  Execution: Separate job components of `srun` with `:` (same communicator); target individual components with `--het-group=0-2,4` for job steps (separate communicators); suffixed `$SLURM_..._HET_GROUP_N` environment variables

  → `slurm.schedmd.com/heterogeneous_jobs.html`

Usage Guidelines

- Base functionality: Slurm's **hetjobs**

  Allocation: Separate options of job components with `#SBATCH hetjob` (batch script) or `:` (interactive); most options propagated from first job component, some not, can be overwritten

  Execution: Separate job components of `srun` with `:` (same communicator); target individual components with `--het-group=0-2,4` for job steps (separate communicators); suffixed `$SLURM_..._HET_GROUP_N` environment variables

  → `slurm.schedmd.com/heterogeneous_jobs.html`

- Heterogeneous jobs: possibly entirely different applications for components

```
$ srun -n 1 --gpus-per-task 4 ./gromacs : -n 24 ./qe
```

Usage Guidelines

- Base functionality: Slurm's **hetjobs**

  Allocation: Separate options of job components with `#SBATCH hetjob` (batch script) or `:` (interactive); most options propagated from first job component, some not, can be overwritten

  Execution: Separate job components of `srun` with `:` (same communicator); target individual components with `--het-group=0-2,4` for job steps (separate communicators); suffixed `$SLURM_..._HET_GROUP_N` environment variables

  → `slurm.schedmd.com/heterogeneous_jobs.html`

- Heterogeneous jobs: possibly entirely different applications for components

```
$ srun -n 1 --gpus-per-task 4 ./gromacs : -n 24 ./qe
```

- Cross-compilation might be hard, easiest: compile each application on targeted system

Usage Guidelines

- Cross-compilation inconvenient, but necessary, **and possible**
- But what about dynamically linked libraries?
  - Might be distinct per system!
  - Usually, environment modules loaded in Slurm batch script before `srun`
  - But: Executed on *first* Slurm host, independent of architecture!

Usage Guidelines

- Cross-compilation inconvenient, but necessary, **and possible**
- But what about dynamically linked libraries?
  - Might be distinct per system!
  - Usually, environment modules loaded in Slurm batch script before `srun`
  - But: Executed on *first* Slurm host, independent of architecture!
  - The following will load GCC module on host of `cpu` partition, and propagate that environment also to `gpu` partition

```
#SBATCH --partition=cpu
#SBATCH hetjob
#SBATCH --partition=gpu
module load GCC
srun --label printenv EBROOTGCC : printenv EBROOTGCC
# result: twice the GCC for "cpu"
```

Usage Guidelines

- Cross-compilation inconvenient, but necessary, **and possible**
- But what about dynamically linked libraries?
  - Might be distinct per system!
  - Usually, environment modules loaded in Slurm batch script before `srun`
  - But: Executed on *first* Slurm host, independent of architecture!
  - The following will load GCC module on host of `cpu` partition, and propagate that environment also to `gpu` partition

```
#SBATCH --partition=cpu
#SBATCH hetjob
#SBATCH --partition=gpu
module load GCC
srun --label printenv EBROOTGCC : printenv EBROOTGCC
# result: twice the GCC for "cpu"
```

→ Use custom wrapper script or **small helper tool**

xenv

Cross Environment Manipulation

- `xenv`: Small helper tool for cross-module environment handling
- Developed by JSC *(and possibly only available there)*
- Creates **ad-hoc** subshell with certain modules loaded/unloaded

xenv

Cross Environment Manipulation

- xenv: Small helper tool for cross-module environment handling
- Developed by JSC *(and possibly only available there)*
- Creates **ad-hoc** subshell with certain modules loaded/unloaded
- Usage

```
$ printenv EBROOTGCC || echo 'Not set'
Not set
$ xenv -L GCC printenv EBROOTGCC || echo 'Not set'
/p/software/jurecadc/stages/2023/software/GCCcore/11.3.0
$ srun --label xenv -L GCC printenv EBROOTGCC : xenv -L GCC
  printenv EBROOTGCC
# result: two seperate GCCs compiled for each architecture
```

xenv

Cross Environment Manipulation

- xenv: Small helper tool for cross-module environment handling
- Developed by JSC *(and possibly only available there)*
- Creates **ad-hoc** subshell with certain modules loaded/unloaded
- Usage

```
$ printenv EBROOTGCC || echo 'Not set'
Not set
$ xenv -L GCC printenv EBROOTGCC || echo 'Not set'
/p/software/jurecadc/stages/2023/software/GCCcore/11.3.0
$ srun --label xenv -L GCC printenv EBROOTGCC : xenv -L GCC
  printenv EBROOTGCC
# result: two seperate GCCs compiled for each architecture
```

Options

- `-L module load`
- `-U module use`
- `-P module purge`
- `-R module restore`

See `man xenv`*!*

env

Environment Manipulation

- xenv modeled after Unix's env
- env: Two *modes*

  No parameter Dump environment

  Parameter Change environment in ad-hoc subshell

env

Environment Manipulation

- xenv modeled after Unix's env
- env: Two *modes*

  No parameter Dump environment

  Parameter Change environment in ad-hoc subshell

- Usage

```
$ env MYVAR=hey printenv MYVAR
hey
$ env -u EBROOTGCC printenv EBROOTGCC || echo 'Not set'
Not set
```

env

Environment Manipulation

- xenv modeled after Unix's env
- env: Two *modes*

  No parameter Dump environment

  Parameter Change environment in ad-hoc subshell

- Usage

```
$ env MYVAR=hey printenv MYVAR
hey
$ env -u EBROOTGCC printenv EBROOTGCC || echo 'Not set'
Not set
```

Options

`ENV=VAL`	Set variable to value
`-i`	Ignore environment
`-u ENV`	Unset variable
`-C pth`	Chang working directory

See `man env`*!*

renv

Environments with Rank IDs

- `renv`: My development *(alpha state)*
- ***R**ank-aware* env: Set variable according to MPI rank

renv

Environments with Rank IDs

- `renv`: My development *(alpha state)*
- ***R**ank-aware* env: Set variable according to MPI rank
- Usage

```
$ srun --label -n 2 renv ABC printenv ABC
0: 0
1: 1
$ srun --label -n 2 renv --map '1:0,0:1' ABC printenv ABC
0: 1
1: 0
$ srun --label -n 2 renv -g CUDA_VIS printenv CUDA_VIS : -n 2
  renv -g CUDA_VIS printenv CUDA_VIS
0: 0
1: 1
2: 2
3: 3
```

renv

Environments with Rank IDs

- `renv`: My development *(alpha state)*
- ***R**ank-aware* env: Set variable according to MPI rank
- Usage

```
$ srun --label -n 2 renv ABC printenv ABC
0: 0
1: 1
$ srun --label -n 2 renv --map '1:0,0:1' ABC printenv ABC
0: 1
1: 0
$ srun --label -n 2 renv -g CUDA_VIS printenv CUDA_VIS : -n 2
  renv -g CUDA_VIS printenv CUDA_VIS
0: 0
1: 1
2: 2
3: 3
```

Options

`ENV`	Set ENV to rank ID
`-g`	Use global task IDs (instead of node-local)
`-f`	Also overwrite pre-existing vars
`-m`	Map rank IDs to values

See `github.com/FZJ-JSC/renv/`*!*

All Together Now

```
srun --label \
  -n 1 --cpus-per-task=48 --hint=nomultithread \
  xenv -P -L GCC -L ParaStationMPI \
    env TYPE=CPU \
      ./cpu.app
: \
  -n 4 --cpu-bind=verbose,map_ldom=3,1,7,5 \
  xenv -P -L GCC -L CUDA -L ParaStationMPI -l MPI-Setting/CUDA \
    env TYPE=GPU \
        renv CUDA_VISIBLE_DEVICES \
          ./gpu.app
```

Exercises

MSA Exercises

- Three exercises to learn MSA execution
- Increasing complexity
- Style: Fill-in-the-blanks with TODOs, solutions given
- Exercises
  1. Hetjob *Hello World*
     TODO: Complete job script
  2. CPU-GPU printf *Hello World*
     TODO: Receive MPI message in GPU buffer
  3. CPU-GPU ping-pong
     TODO: Use GPU buffers on GPU-side

Exercises Infrastructure

- Find material at `$PROJECT_training2317/material/` or `go.fzj.de/msa-hello`
- Login to JUWELS, source project environment

```
$ source $PROJECT_training2317/env.sh
```

- Sync material to home folder: `jsc-material-sync`
- One folder per exercise, each including
  - Fill-in-the-blanks files with TODOs
  - Solutions
  - README instructions
  - Auxiliary scripts
- Reservation of 8+8 nodes for today, automatically applied when in `training2317` project

Exercises Infrastructure

- Find material at `$PROJECT_training2317/material/` or `go.fzj.de/msa-hello`
- Login to JUWELS, source project environment

```
$ source $PROJECT_training2317/env.sh
```

- Sync material to home folder: `jsc-material-sync`
- One folder per exercise, each including
  - Fill-in-the-blanks files with TODOs
  - Solutions
  - README instructions
  - Auxiliary scripts
- Reservation of 8+8 nodes for today, automatically applied when in `training2317` project

Happy Hacking!