MODULAR SUPERCOMPUTING: a system-wide orchestration of heterogeneous resources

ADAC, 24.01.2022 | Estela Suarez (JSC)
OUTLINE

• System architecture
  - From dual architecture to the Modular Supercomputing Architecture (MSA)
  - Hardware implementations of MSA

• Software
  - Software stack
  - ParaStation Modulo
  - Scheduler

• Application experience

• Conclusions and next steps
Can one combine the best of these two worlds into a single system?
THE DEEP PROJECTS

2011-2021: The DEEP projects

• DEEP (2011 – 2015)
  - Introduced Cluster-Booster architecture

• DEEP-ER (2013 – 2017)
  - Added I/O and resiliency functionalities

• DEEP-EST (2017 – 2021)
  - Modular Supercomputer Architecture

2021-2024 The SEA projects

• DEEP-SEA, IO-SEA, RED-SEA
HOMOGENEOUS

General Purpose Cluster

Nodes contain only CPUs

+: Easy to use
+: Very flexible
-: Power hungry
HETEROGENOUS MONOLITHIC

Every node contains accelerators (e.g. GPUs)

- Every node contains CPU(s) and some accelerator
- All nodes are equal → “monolithic”

+: Energy efficient
+: Easy management
-: Static assignment of accelerators to CPUs
-: Difficult to efficiently share resources
HETEROGENEOUS MODULAR

Different nodes are grouped in “modules”

- All nodes within one module are equal
- Different modules have different configurations → “modular”

+ Energy efficient
+ Better scalability
+ High flexibility
+ Dynamic resource assignment
- Complexity

Cluster

Booster

Network
MODULAR SUPERCOMPUTING ARCHITECTURE

Composability of heterogeneous resources

• Cost-effective scaling

---


MODULAR SUPERCOMPUTING ARCHITECTURE

Composability of heterogeneous resources

- Cost-effective scaling
- Effective resource-sharing
- Match application diversity
  - Large-scale, complex workflows


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
MODULAR SUPERCOMPUTER JUWELS

**JUWELS Cluster**  #44

- Intel Xeon (Skylake) processor
- InfiniBand network
- 2,500 compute nodes
- 10 PFLOP/s

**JUWELS Booster**  #7

- AMD EPYC Rome 7402 processor
- 3,700 NVIDIA A100 GPUs
- InfiniBand HDR DragonFly+
- 70 PFLOP/s peak (GPU-based)

**TOP500:**
- Rank 7 World
- Rank 1 Europe

**Green500:**
- Rank 1 in TOP250

**TOP5 AI:**
- Rank 4 (3 in 2021)

JUWELS is designed for simulation and large-scale machine learning

Funded through SiVeGCS (BMBF, MWK-NRW)

Entry in Nov’20
JSC MSA DEPLOYMENTS

- **2022**: JUNIQ
- **2020**: JUWELS
- **2018**: JUQUEEN
- **2017**: JURECA
- **2015**: JUROPA
- **2012**: DEEP

Architecture innovation started in DEEP

Modular Supercomputing
MELUXINA

Cluster: 570 CPU nodes
- AMD EPYC 7H12, 2× 64C @2.6GHz, 512 GB (~ 4 GB / core)

Booster: 200 GPU nodes
- AMD EPYC 7452, 2× 32C @ 2.35GHz, 512 GB (~ 8 GB / core)
- 4× NVIDIA A100 Ampere, 40GB HBM2

Smaller partitions
20 Large Memory nodes: CPU node with 4096 GB, 1.92 TB SSD
20 FPGA nodes: CPU node with, 2× Stratix FGPA10MX (16GB HBM)
20 Cloud nodes: CPU node with 4096 GB, 1.92 TB SSD

System-wide
- NVIDIA/Mellanox InfiniBand HDR 200 Gb/s
- Atos BullSequana XH2000
- ParTec ParaStation Modulo Software

Source: Valentin Plugaru
OUTLINE

• System architecture
  - From dual architecture to the Modular Supercomputing Architecture (MSA)
  - Hardware implementations of MSA

• Software
  - Software stack
  - ParaStation Modulo
  - Scheduler

• Application experience

• Conclusions and next steps
MATCHING APPLICATIONS AND HARDWARE
SOFTWARE ENVIRONMENT

- **Low-level SW:** Inter-network bridging
- **Scheduler:** Slurm, psslurm (ParaStation Modulo)
- **Filesystem:** BeeGFS, GPFS
- **Compilers:** Intel, GCC, NVIDIA HPC SDK
- **Debuggers:** Intel Inspector, TotalView
- **Programming:** ParaStation MPI, OpenMP, OmpSs, CUDA
- **Performance analysis tools:** Scalasca, Score-P, Extrae/Paraver, Vampir, Intel Advisor, VTune…
- **Benchmarking tools:** JUBE
- **I/O Libraries:** SIONlib, SCR, HDF5,…

---

- Clauss et al., *Dynamic Process Management with Allocation-internal Co-Scheduling towards Interactive Supercomputing*, COSH@HiPEAC, (2016)
ParaStation Modulo

• ParaStation ClusterTools
  - Tools for system provisioning and system management

• ParaStation HealthChecker & TicketSuite
  - Automated error detection & error handling
  - Ensuring integrity of the computing environment
  - Keeping track of issues
  - Powerful analysis tools

• ParaStation Process Management & ParaStation MPI
  - Runtime environment tuned for the largest distributed memory supercomputers
  - Optimally support the Modular Supercomputing Architecture

Source: Thomas Moschny
ParaStation Process Manager

- Scalable network of process management daemons
  - Process startup and control, I/O forwarding, …
  - Precise resource monitoring
  - Proper cleanup after jobs
  - Daemons run on the compute nodes

- psslurm: full integration with Slurm
  - Plugin to ParaStation Management
  - Reduce number of daemons on compute nodes
    - Replace node-local Slurm daemon
  - Integration with ParaStation HealthChecker
  - Possible to fix problems and add unique features
ParaStation MPI Library

- **Based on MPICH 3.4.2** (MPI-3.1 compliant)
  - Supports MPICH tools (tracing, debugging, …)
  - MPICH layers beneath ADI3 replaced by ParaStation PSP Device
  - Powered by pscom low-level communication library
  - Maintains MPICH ABI compatibility

- **Support for various transports and protocols via pscom plugins**
  - Support for InfiniBand, Omni-Path, Extoll, (soon BXI)
  - Multiple transports / plugins can be used concurrently
  - Gateway capability via PSGW plugin
  - CUDA awareness via GPUDirect

- **Proven scaling up to ~3,500 nodes and ~140,000 procs. / job**

Source: Thomas Moschny
ParaStation Global MPI for MSA

- One can also start two parts of a code and connect them via MPI_Connect()
- Or have one single common MPI_COMM_WORLD and split it into subcommunicators via MPI_Comm_Split()

• An MPI application can run:
  – Using only Cluster nodes
  – Using only Booster nodes
  – Distributed over Cluster and Booster
    ▪ In this case two executables are created
    ▪ Collective offload process
    ▪ Transparent data exchange via MPI

• ParaStation Global MPI
  – Uses MPI_Comm_spawn()
    ▪ Collective spawn groups of processes from Cluster to Booster (or vice-versa)
  – Inter-communicator
    ▪ Connects the 2 MPI_COMM_WORLD
    ▪ Contains all parents on one side and all children on the other
      ▪ Returned by MPI_Comm_spawn for the parents
      ▪ Returned by MPI_Get_parent by the children

Clauss et al., Dynamic Process Management with Allocation-internal Co-Scheduling towards Interactive Supercomputing, COSH@HiPEAC, (2016)
**COMPILE AND RUN**

- **Compilation**
  - Creates two executables (if different CPU architecture)
    - One for CLUSTER code
    - One for BOOSTER code

- **Batch system**
  - Reserves required resources

- **Execution**
  - Script starts Booster code
  - This code calls MPI_Comm_spawn() with Cluster executable
  - Optional: xenv to load suitable environment modules

- **Runtime + Scheduler + FS**
  - Detect ParaStation MPI calls
  - Distribute child binaries

```
salloc --partition=cluster -N 4
srun --het-group=1 -N 4 -n 8
./app_booster
```
Heterogeneity from user’s PoV

- **Slurm** supports the ability to submit **heterogeneous jobs** (since v 17.11)
  - form **job pack (het-job)** allocation using colon notation for `salloc`, `sbatch`, `srun`
  - even allowing different executables

  ```bash
  $ srun -N 1 -p part1 ./first \
  : -N 2 -p part 2 ./second
  ```

- **Full support for job packs** in **ParaStation psslurm**, with **unique features** for modular jobs:
  - Support for heterogeneous jobs with common MPI_COMM_WORLD
  - For each job in the job pack, resources can be specified individually
  - Support global resources (e.g. gateways): `psgw` plugin to `psmgmt` + spank plugin
    - Compensates for Slurm's inability to handle global resources
    - Extends `salloc`, `srun` and `sbatch`

- **Modular here means: Jobs across heterogeneous hardware**
  - Either with a common MPI_COMM_WORLD, or with separated / interconnected MPI_COMM_WORLDs

Source: Thomas Moschny
Explicit MSA-extensions to ParaStation MPI

- **API additions to retrieve topology information**
  - Querying the module ID: `MPI_Info_get(MPI_INFO_ENV, "msa_module_id", …, value, …);`
  - Splitting communicators according to the topology:
    ```c
    MPI_Comm_split_type(oldcomm, MPIX_COMM_TYPE_MODULE, …, &newcomm);
    ```

- **Modularity-aware MPI collectives**
  - Optimized patterns for collectives that take topology of the MSA system into account
  - Assumption: Inter-module communication is the bottleneck
  - Dynamic updates of the communication patterns supported, e.g. for malleable jobs (experimental)

Source: Thomas Moschny
Hierarchical collectives (MSA awareness)

- **General rules to optimize collectives** → execute these steps in order:
  - 1) Do all *module-internal* gathering and/or reduction operations (if required)
  - 2) Conduct the *inter-module operation* with a single process per module
  - 3) Perform a strict *module-local data distribution*

- **Multi-level hierarchy awareness**
  - Apply this set of rules recursively, i.e., node level, module level, system level

- **Performance heavily depends on concrete settings**, i.e.:
  - Number of processes / gateway nodes
  - Distribution of the ranks in the communicator
  - Message sizes (and hence the collective communication pattern)

Source: Thomas Moschny
Resource Management

Current behaviour

- CPU
- GPU
- Accel. 2
- Accel. n

Ideal behaviour

- CPU
- GPU
- Accel. 2
- Accel. n

Application execution parts

Resource reservation window

Application execution part = Resource reservation window
Improved Workflow Support (experimental)

- New parameter `--delay` introduced in `sbatch` command for job packs
  - Amount of time, the next job should wait after start of the first job in a job pack

- Goal: Overlapping job execution
  - Currently not supported by Slurm
    - Whole job pack either accepted or rejected
    - All jobs allocated and run in parallel
    - All jobs wait for allocation if any of the jobs cannot be allocated at the moment
EXTENDING THE SOFTWARE STACK

- Support for accelerators & memory
- Malleability
- Interoperability
- Composability
- Performance portability
- Resiliency

Application / Workflow

Programming Model
- DSL NabLab
- DaCe
- OpenMP
- OmpSs
- GPI
- MPI

Runtime
- Component-specific node-level support

Operating System

System Architecture
- CPU +DDR +NVM
- GPU +HBM
- Accel. 2 + mem
- ... Accel. N + mem
Quantum integration in MSA

- **New usage models**
  - Tightly coupled simulations: benefit from efficient data exchange
  - Hybrid quantum-HPC simulations: combining quantum and classical algorithms
  - Workflows comprising stages on the QPU, with pre- and post-processing on HPC modules

- **Integrate QPU and its front-end into the managements stack**
  - Low-latency connection to other modules via federated, high-speed network
  - Unified environment: Integrated in the user-, SW-, schedule- and resource- mgmt.
    - Provide “direct” access of the QPU via a web-based portal
    - Redirect portal requests through the global scheduler/resource manager
    - Pseudo-shared usage model as prerequisite

- **Exact requirements depend on the use case and are subject to research**
OUTLINE

- **System architecture**
  - From dual architecture to the Modular Supercomputing Architecture (MSA)
  - Hardware implementations of MSA

- **Software**
  - Software stack
  - ParaStation Modulo
  - Scheduler

- **Application experience**

- **Conclusions and next steps**
Architecture Use-Modes

Cluster-Booster use mode

Code partition
Workflow
I/O forward

- Kreuzer et al. The DEEP-ER project: I/O and resiliency extensions for the Cluster-Booster architecture. HPCC’18 proceedings (2018) [10.1109/HPCC/SmartCity/DSS.2018.00046]
- Christou et al., EMAC on DEEP, Geoscientific model deovel.(2016) [10.5194/gmd-9-3483-2016]
- Kumbhar et al., Leveraging a Cluster-Booster Architecture for Brain-Scale Simulations, Lecture Notes in Computer Science 9697 (2016) [10.1007/978-3-319-41321-1_19]
- Leger et al., Adapting a Finite-Element Type Solver for Bioelectromagnetics to the DEEP-ER Platform. ParCo 2015, Advances in Parallel Computing, 27 (2016) [10.3233/978-1-61499-621-7-349]
Application use case: xPic

- **Space Weather simulation**
  - Simulates plasma produced in solar eruptions and its interaction with the Earth magnetosphere
  - Particle-in-Cell (PIC) code
  - Authors: KU Leuven

- **Two solvers:**
  - **Field solver:** Computes electromagnetic (EM) field evolution
    - Limited code scalability
    - Frequent, global communication
  - **Particle solver:** Calculates motion of charged particles in EM-fields
    - Highly parallel
    - Billions of particles
    - Long-range communication

---

for (auto i=beg+1; i<=end; i++){
  fld.solver->calculateE();
  fld.cpyToArr_F();
}

pcl.cpyFromArr_F();
for (auto is=0; is<nspec; is++) {
  pcl.species[is].ParticlesMove();
  pcl.species[is].ParticleMoments();
}
pcl.cpyToArr_M();

fld.solver->calculateB();
fld.cpyFromArr_M();
```cpp
#ifdef __CLUSTER__
    for (auto i=beg+1; i<=end; i++){
        fld.solver->calculateE();
        fld.cpyToArr_F();
        ClusterToBooster();
        // Auxiliary computations
        ClusterWait();
        BoosterToCluster();
        BoosterWait();
        fld.solver->calculateB();
        fld.cpyFromArr_M();
    }
#endif

#ifdef __BOOSTER__
    for (auto i=beg+1; i<=end; i++){
        ClusterToBooster();
        ClusterWait();
        pcl.cpyFromArr_F();
        for (auto is=0; is<nspec; is++) {
            pcl.species[is].ParticlesMove();
            pcl.species[is].ParticleMoments();
        }
        pcl.cpyToArr_M();
        BoosterToCluster();
        // I/O and auxiliary computations
        BoosterWait();
    }
#endif
```
**xPic – (1-NODE) PERFORMANCE RESULTS**

- Field solver: 6× faster on *Cluster*
- Particle solver: 1.35 × faster on *Booster*
- Overall performance gain:
  - 1× node: 28% × gain compared to Cluster alone
  - 38% × gain compared to Cluster only
  - 21% × gain compared to Booster alone
  - 34% × gain compared to Booster only
  - 3%-4% overhead per solver for C+B communication (point to point)

---

**Compilation flags**
- Cluster: `-openmp, -mavx`
- Booster: `-xMIC-AVX512`

**Table**:  
<table>
<thead>
<tr>
<th>#cells per node</th>
<th>4096</th>
</tr>
</thead>
<tbody>
<tr>
<td>#particles per cell</td>
<td>2048</td>
</tr>
</tbody>
</table>

---

xPic – STRONG SCALING on JURECA

• Code portions can be scaled-up independently
  - Particles scale almost linearly on Booster
  - Fields kept constant on the Cluster (4CNs)

• A configuration is reached where same time is spent on Cluster and Booster
  - Additional 2× time-saving is enabled via overlapping

<table>
<thead>
<tr>
<th>(4 Cluster nodes)</th>
<th>Number of Booster nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>#cells per node</td>
<td>36864</td>
</tr>
<tr>
<td>#particles per cell</td>
<td>1024</td>
</tr>
<tr>
<td>#blocks per MPI process</td>
<td>12, 32 or 64</td>
</tr>
<tr>
<td>Compilation flags</td>
<td>-mavx (Cluster)</td>
</tr>
<tr>
<td></td>
<td>-openmp, xMIC-AVX512 (Booster)</td>
</tr>
</tbody>
</table>
GROMACS: multi-module usage in MD simulations

- Best mapping on MSA depends on the problem size and aims at optimizing the computational load
  - <10^4 particles: only on Cluster (CPU)
  - ~10^5 particles: Booster or DAM (Data Analytics Module)
  - >10^6 particles (large macromolecules): pair interactions on GPU, run PME on CPUs
GROMACS: multi-module usage in MD simulations

- Best mapping on MSA depends on the problem size and aims at optimizing the computational load:
  - <10^4 particles: only on Cluster (CPU)
  - ~10^5 particles: Booster or DAM (Data Analytics Module)
  - >10^6 particles (large macromolecules): pair interactions on GPU, run PME on CPUs
  - Very large volume (>10^6 nm^3): Replace PME with FMM (Fast Multipole Method) running on ESB
GROMACS: multi-module usage in MD simulations

- **Best mapping on MSA depends on the problem size** and aims at optimizing the computational load
  - $<10^4$ particles: only on Cluster (CPU)
  - $\sim 10^5$ particles: Booster or DAM (Data Analytics Module)
  - $>10^6$ particles (large macromolecules): pair interactions on GPU, run PME on CPUs
  - **Very large volume ($>10^6$ nm$^3$): Replace PME with FMM running on ESB**
NextDBSCAN: multi-module usage in ML

- Parallel algorithm for density-based clustering of arbitrary data sets
  - Performance and flexibility gain by running on multiple modules

See presentation in ADAC applications group:

*DBSCAN Clustering and Modular Supercomputing: Lessons Learned*

Ernir Erlingsson (University of Iceland)
OUTLINE

• System architecture
  - From dual architecture to the Modular Supercomputing Architecture (MSA)
  - Hardware implementations of MSA

• Software
  - Software stack
  - ParaStation Modulo
  - Scheduler

• Application experience

• Conclusions and next steps
CONCLUSIONS

• The Modular Supercomputing Architecture (MSA)
  - Orchestrates heterogeneity at system level
  - Allows scaling hardware in economical way (Booster → Exascale)
  - Serves very diverse application profiles
    o Maximum flexibility for users, without taking anything away (still can use individual modules)

• Distribute applications on the MSA give each code-part a suitable hardware
  - Straight-forward implementation for workflows
  - Partition at MPI-level interesting for multi-physics / multi-scale codes
  - Monolithic codes do not need to be divided

• Current / Upcoming implementations of MSA
  - DEEP system, JURECA, JUWELS
  - MELUXINA (Luxembourg EuroHPC Petascale system)
  - EUPEX and HPCQS pilots
  - … Exascale!
MODULAR SUPERCOMPUTING TO EXASCALE

Pilot systems

Pre-Exascale @ Lux, @It

Petascale

EuroHPC

Projects

2024

2023

2022

2021

2020

2017

2011

FP7 - H2020

JURECA

IQUELS

MELU

INA

LEONARDO

Exascale

Suarez – 2022
The DEEP projects have received funding from the European Union’s Seventh Framework Programme (FP7) for research, technological development and demonstration and the Horion2020 (H2020) funding framework under grant agreement no. FP7-ICT-287530 (DEEP), FP7-ICT-610476 (DEEP-ER), H2020-FETHPC-754304 (DEEP-EST), and EuroHPC 955606 (DEEP-SEA).