Coming Next:

**Modular Supercomputing on JUWELS**

*ESM workshop*

06.02.2019 | E. Suarez (JSC)
THE DEEP PROJECTS

Objectives

• Flexible association of heterogeneous resources
• Increase system performance & energy efficiency
• Address diverse application needs (HPC, HPDA, ML)
• Co-design production-quality HW & SW prototypes
• Build a strong, sustainable European ecosystem

Project data

• 27 partners
• Time frame: 2011 - 2020
• Total EU funding: 30 M€
OUTLINE

• Modular Supercomputing Architecture
• Software environment
  – Programming environment
  – Scheduling
• Applications
  – First results on code partitions
  – Workflows
• Observations
OUTLINE

• Modular Supercomputing Architecture
  • Software environment
    – Programming environment
    – Scheduling
  • Applications
    – First results on code partitions
    – Workflows
  • Observations
ROADMAP
2012-2018

Cluster-Booster

6 PF
JUQUEEN

7.2 PF
JURECA

12 PF
 Modular Supercomputing

2018
2017
2015
2012

JÜLICH
Forschungszentrum
• Interconnect any number of specialized modules
  – Address diverse application needs
### Cluster vs. Booster

<table>
<thead>
<tr>
<th>Cluster</th>
<th>Booster</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Processor</strong></td>
<td><strong>Intel Xeon (Haswell)</strong></td>
</tr>
<tr>
<td><strong>Interconnect</strong></td>
<td><strong>InfiniBand EDR</strong></td>
</tr>
<tr>
<td><strong>Node count</strong></td>
<td><strong>1,872</strong></td>
</tr>
<tr>
<td><strong>Peak Perf. (PFlops)</strong></td>
<td><strong>1,8 (CPU) + 0.4 (GPU)</strong></td>
</tr>
</tbody>
</table>
JUWELS

a) JUWELS Cluster

- Processor: Intel Xeon (Skylake)
- Interconnect: InfiniBand EDR
- Node count: 2,500
- Peak Perf. (PFlops): \(10.4 \text{ (CPU)} + 1.6 \text{ (GPU)}\)

b) JUWELS Booster

- Interconnect: InfiniBand HDR
- Node count: TBD
- Peak Perf. (PFlops): >50 PFlop/s

Installation expected 2020
JUWELS GPU Node

JUWELS Cluster configuration → expect improvements in the Booster

(b) JUWELS Cluster compute node (GPU partition).
Architecture Schema

JUWELS

Cluster

Booster

High-speed Network InfiniBand
OUTLINE

• Modular Supercomputing Architecture

• Software environment
  – Programming environment
  – Scheduling

• Applications
  – First results on code partitions
  – Workflows

• Observations
PROGRAMMING ENVIRONMENT

- One 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

- ParaStation Global MPI
  - Enables distributing code
  - Uses **MPI_Comm_spawn()**
    - *Collective spawn groups of processes from Cluster to Booster (or vice-versa)*
  - Inter-communicator
    - *Connects the 2 MPI_Comm_worlds*

- 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()**
“DOUBLE” OFFLOAD

1) From Cluster-to-Booster, 2) From Booster-CPU to Booster-GPU
LAUNCHING A JOB – with SLURM

- Job only on **Cluster**
  - 4 nodes with 2 MPI tasks per node
  ```bash
  salloc --partition=cluster -N 4
  srun -N 4 -n 8 ./hi_cluster
  ```

- Job only on **Booster**
  - 12 nodes with 1 MPI task per node
  ```bash
  salloc --partition=booster -N 12
  srun -N 12 -n 12 ./hi_booster
  ```

- Job on **Cluster + Booster**
  - Case 1: Two executables
  ```bash
  salloc --partition=cluster -N 4 : --partition=booster -N 12
  srun -N 4 -n 8 ./hi_cluster : -N 12 -n 12 ./hi_booster
  ```

  - Case 2: One Cluster executable (spawning itself processes on the Booster)
  ```bash
  salloc --partition=cluster -N 4 : --partition=booster -N 12
  srun --pack-group=0 -N 4 -n 8 ./hi_cluster
  ```
OUTLINE

• Modular Supercomputing Architecture
• Software environment
  − Programming environment
  − Scheduling
• Applications
  − First results on code partitions
  − Workflows
• Observations
CO-DESIGN APPLICATIONS

• 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 devel.(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]
EMAC – Climate Simulation (CYI)

Atmospheric Model (ECHAM)
- Base model
- Low scalability

Chemistry Model (MESSy)
- Embarrassingly parallel
- MECCA is its most time consuming part
EMAC – Climate Simulation (CYI)

EARLY 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

---

### xPic – ORIGINAL CONFIGURATION

```cpp
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();
}
```

- **fld**: Field Solver
- **pcl**: Particle Solver
- Copy information between solvers
```c
#define __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();
}
#define __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
```
COMPILE AND RUN

• **Compilation**
  - Creates two executables
    - *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 name of Cluster executable

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

---

```
salloc --partition=cluster -N 4
    : --partition=booster -N 12
srun -N 4 -n 8 ./hi_cluster
    : -N 12 -n 12 ./hi_booster
```

```c
int main (int argc, char *argv[]){
    /* ... */
    MPI_Comm_spawn("./xPic.Cluster", &argv[1],
                   nproc, MPI_INFO_NULL, 0, GRID_COMM_WORLD,
                   INTERCOMM, MPI_ERRCODES_IGNORE);
    /* ... */
}
```
xPic – (1-NODE) PERFORMANCE RESULTS

• **Field solver:**
  - 6 × faster on **Cluster**
    - *Limited code scalability*
    - *Frequent, global communication*

• **Particle solver:**
  - 1.35 × faster on **Booster**
    - *Highly parallel*
    - *Billions of particles*
    - *Long-range communication*

• **Overall performance gain:**
  - 1,28 × gain compared to Cluster alone
  - 1,21 × gain compared to Booster alone

  - 3%-4% overhead per solver for C+B communication (point to point)

---

**Kreuzer, A.; Eicker, N.; Amaya, J.; Suarez, E., “Application Performance on a Cluster-Booster System”**
2018 IEEE IPDPS Workshops (IPDPSW), Vancouver, Canada, p 69 - 78 (2018) [10.1109/IPDPSW.2018.00019]
xPic – weak scaling on DEEP-ER

• Performance gain grows with system size
  – 8-node Performance:

  1,38 × gain compared to Cluster only
  1,34 × gain compared to Booster only

• Better parallel efficiency:
  – C+B distribution: 85%
  – Cluster alone: 79%
  – Booster alone: 77%

Workflow Applications – E.g. Space Weather

**Data analytics:** Forecast solar wind conditions at L1 from remote image analysis of the Sun

**Simulations:** Detailed physics simulations of the Earth environment given the solar wind conditions at L1
Workflow Applications – E.g. Neuroscience

- **NEST** for large-scale network simulation
- **Arbor** for detailed neuron simulation
  - Designed for vectorised architectures
- **Elephant** for data analytics
OUTLINE

• Modular Supercomputing Architecture
• Software environment
  – Programming environment
  – Scheduling
• Applications
  – First results on code partitions
  – Workflows
• Observations
What works already & What not yet

On JURECA

Modular functionality available
Work ongoing on improving performance and usability

• Joint reservation of nodes on Cluster and Booster
• Script-based gateway reservation
  – Work ongoing to integrate gateway-reservation in scheduler
  • Not relevant for JUWELS – no gateways!
• Static assignment of resources
  – Plan to have dynamic assignment in future
• Execution of “modular” jobs
  – MPI_Comm_World() split over both partitions
  – MPI_Comm_Spawn() not supported yet – current modular users don’t need it
• Job-packs reserving resources over full runtime
  – Investigation on time-reservation of resources for each job on a pack
OBSERVATIONS

• The MSA (and in particular Cluster-Booster)
  – Allows scaling hardware in economical way
    • Booster \(\rightarrow\) Exascale
  – Can serve very diverse application profiles
  – Maximum flexibility for users, without taking anything away
    • You can still always use individual modules

• Benefit of partitioning applications in an MSA
  – Provide each part of the code a suitable hardware
    • Particularly interesting for multi-physics / multi-scale codes
  – Monolithic codes do not need to be divided