e-infrastructure

CAPACITIES

Tightly-coupled workflows using MUSCLE2

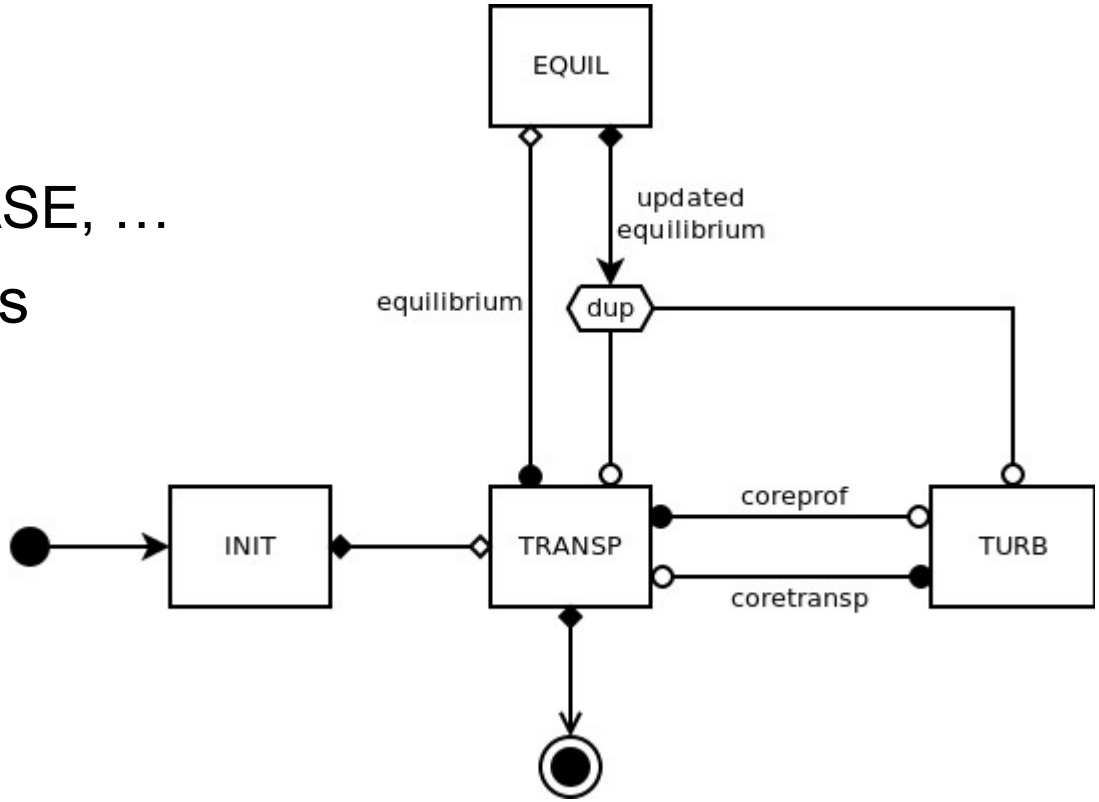
22/11/2013

Olivier Hoenen

The Mapper project receives funding from the EC's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° RI-261507.

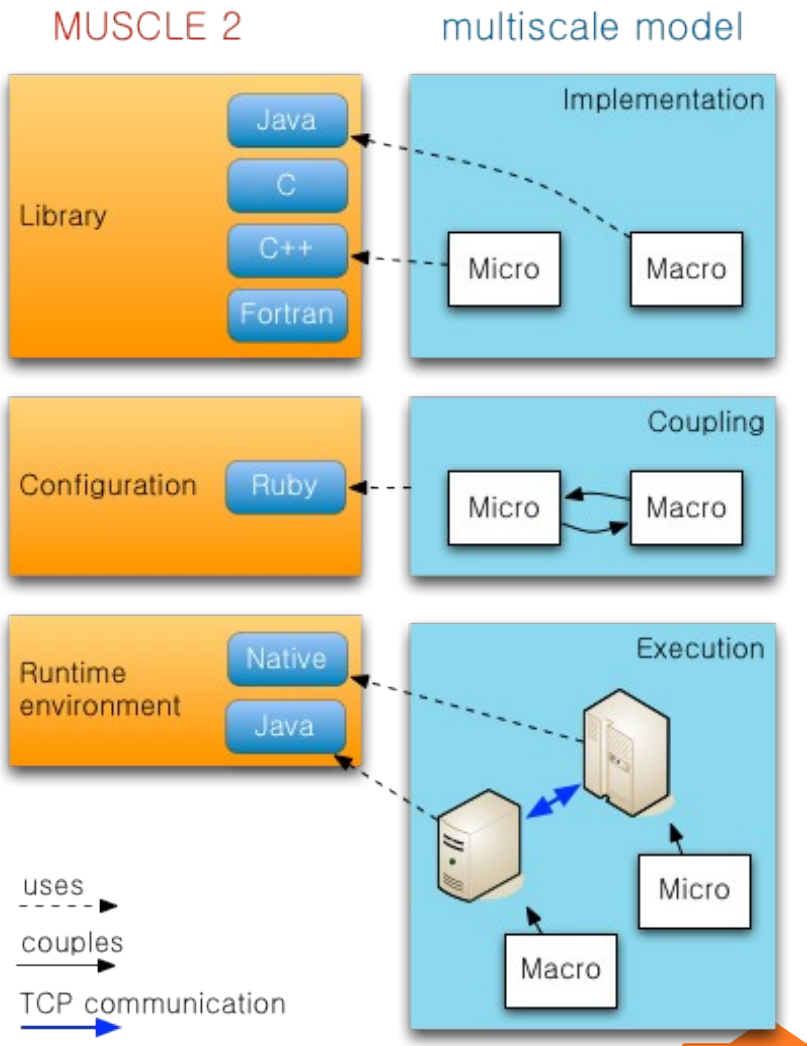
Use case

- Tightly-coupled (cyclic)
  - Transport solver
    ETS
  - Equilibrium code
    Helena, Bdseq, CHEASE, …
  - Transport coefficients
    Bohmgb, GEM, ...



MUSCLE 2

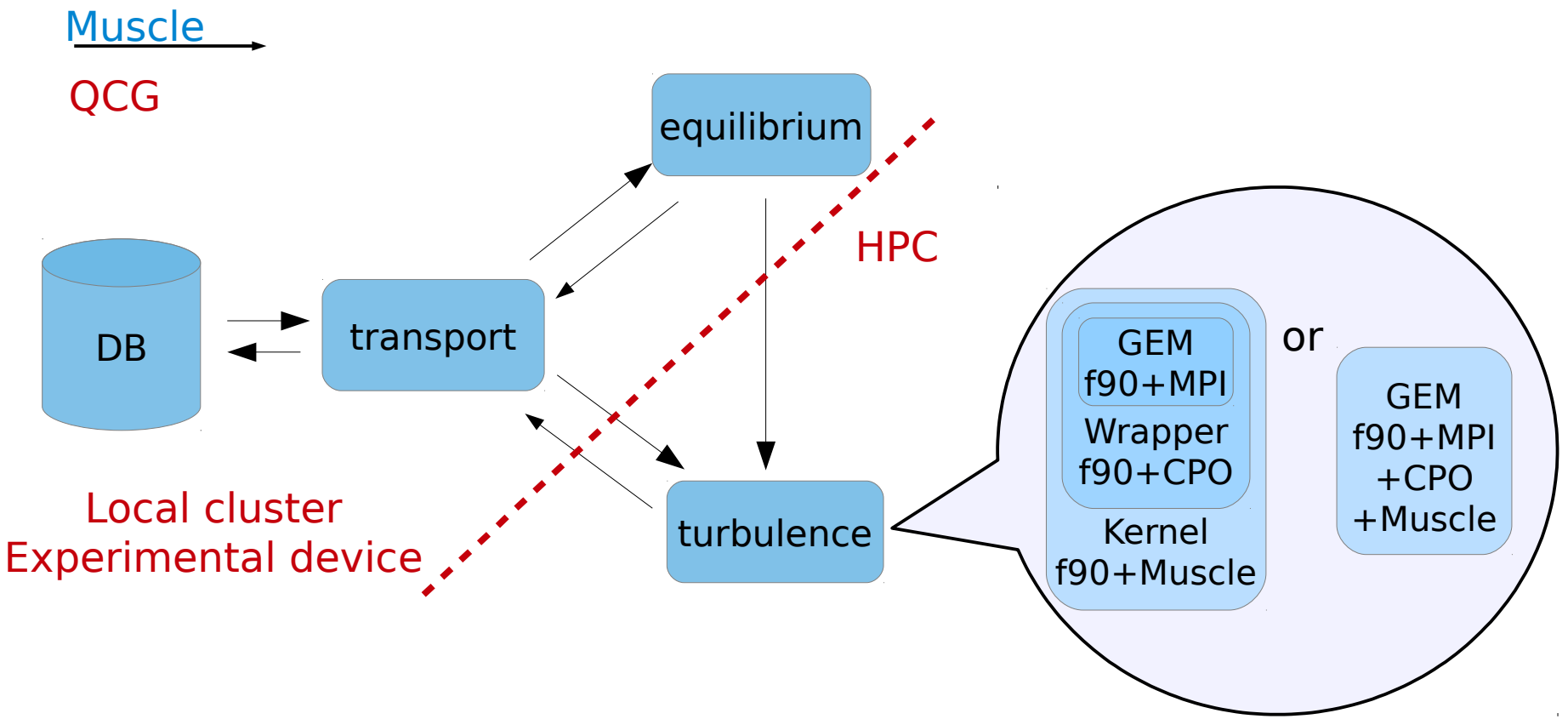
- Coupling library
  - Core in Java (now also in C++)
  - API in C/C++, F90 (Python, Scala)
  - Based on “kernels”
- Handle time evolution
  - Reflects MML terminology
- Workflow described in Ruby (script)
- Inter-sites communiations
  - MTO (TCP, MPWide)
    http://apps.man.poznan.pl/trac/muscle



Kernels

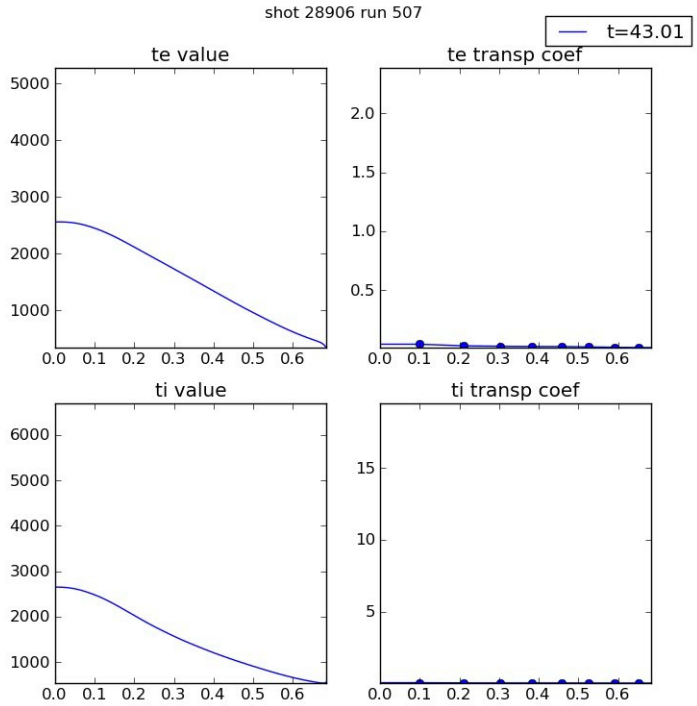
- API familiar to MPI users
  - `MUSCLE_Init, MUSCLE_Receive, MUSCLE_Get_Property, MUSCLE_Send, MUSCLE_Finalize`
- Receive/send handle accept basic C types
- Kernel contains time loop
  - `MUSCLE_Will_Stop` or hand-made
- Native source code → executable → concurrent exec
  - Scheduling is controlled by dataflow
- MPI kernels → only rank 0 calls MUSCLE

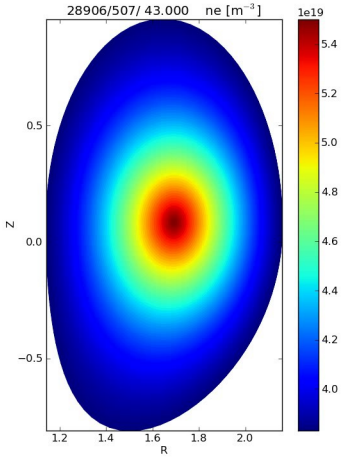
Distributed run

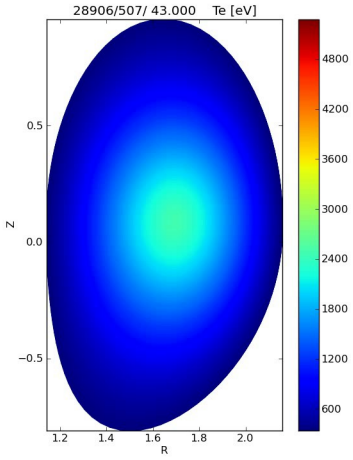


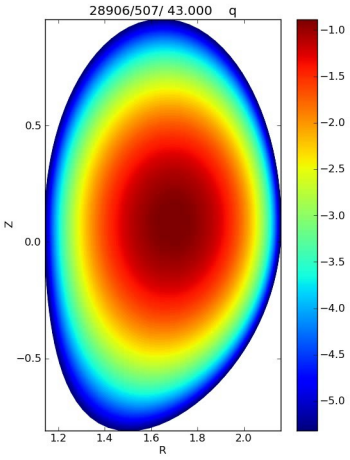
Some results

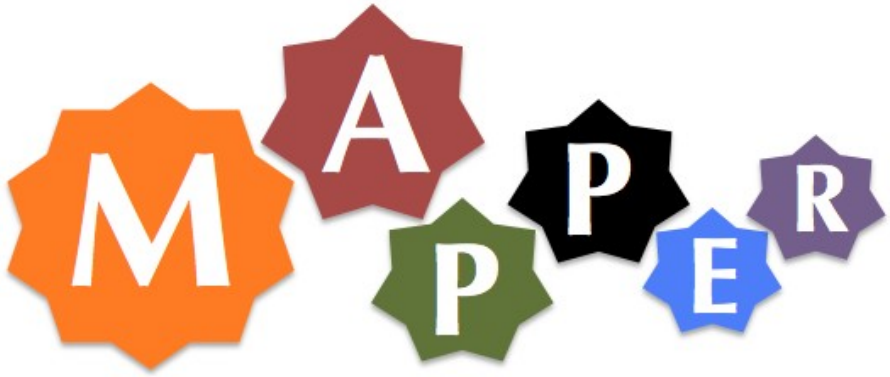
- Example on AUG shots:
  - 1s, transport dt=0.01, gyrofluid on 8 fluxtubes
  - 128 cores → wallclock = 10:36:26 (Gateway)











Questions?

Ruby CXA file

- Describes the workflow
- Kernels id and path
  - User defined: Native, MPI, Java
  - Utils: duplication mapper,...
- Variables
  - Global or local to a kernel
- Links between kernels
  - Attach kernels, attach ports
  - Add filters: conversion, compression, etc...

```
# configuration of commands
cxa.env["transp:command"]="ets_kernelB"
cxa.env["turb:command"]="gem_kernelB"


# global params
cxa.env["time_step"] = 0.1
# kernel specific params
cxa.env["init:cpo_file"] = "CPO_000004_000001"
cxa.env["transp:solver_type"] = 3
cxa.env["transp:sigma_source"] = 0

# declare kernels
cxa.add_kernel('transp',
'muscle.core.standalone.NativeKernel')
cxa.add_kernel('turb',
'muscle.core.standalone.MPIKernel')
cxa.add_kernel('dupEquil',
'muscle.core.kernel.DuplicationMapper')

cs.attach('transp' => 'turb') {
tie('coreprof_out','coreprof_in')
}
cs.attach('turb' => 'transp') {
tie('coretransp_out','coretransp_in')
}
```

QCG-Broker

- Resource management and brokering service
  - Assign kernels to site
  - Processes counts
  - Advance reservation
- Additional scripts
  - Preprocessing
  - Postprocessing
- Environnement
  - stderr/stdout
  - modules

```
<task persistent="true" taskId="task">
  <requirements>
    <topology>
      <processes masterGroup="true" processesId="transp">
              <processesCount> ...
        <candidateHosts>
                <hostName>inula.man.poznan.pl</hostName>
        </candidateHosts>
      </processes>
      <processes processesId="turb">
              <processesCount> ...
        <candidateHosts>
          <hostName>zeus.cyfronet.pl</hostName>
              </candidateHosts>
            </processes>
    </topology>
  </requirements>
  <execution type="mapper">
    <executable>
      <application name="muscle2"/>
    </executable>
    <arguments>
      <value>ParallelModelsB.cxa.rb</value>
    </arguments>
    <stdout> ...
    <stderr> ...
    <stageInOut> ...
    <environment> ...
  </execution>
  <executionTime> ...
</task>
```

On Helios

- First benchmark on Helios:
  - Scalability follows main component performance (GEM)
  - Speed-up loss from 512 to 1024 due to impact of serial components : ETS+CHEASE takes around 10-15s
  - At 2048 cores and above, non scalable behavior is due to non-scalable I/O usage (naïve implementation of serialization, etc...): to be investigated