Designing and running turbulence transport simulations using a distributed multiscale computing approach

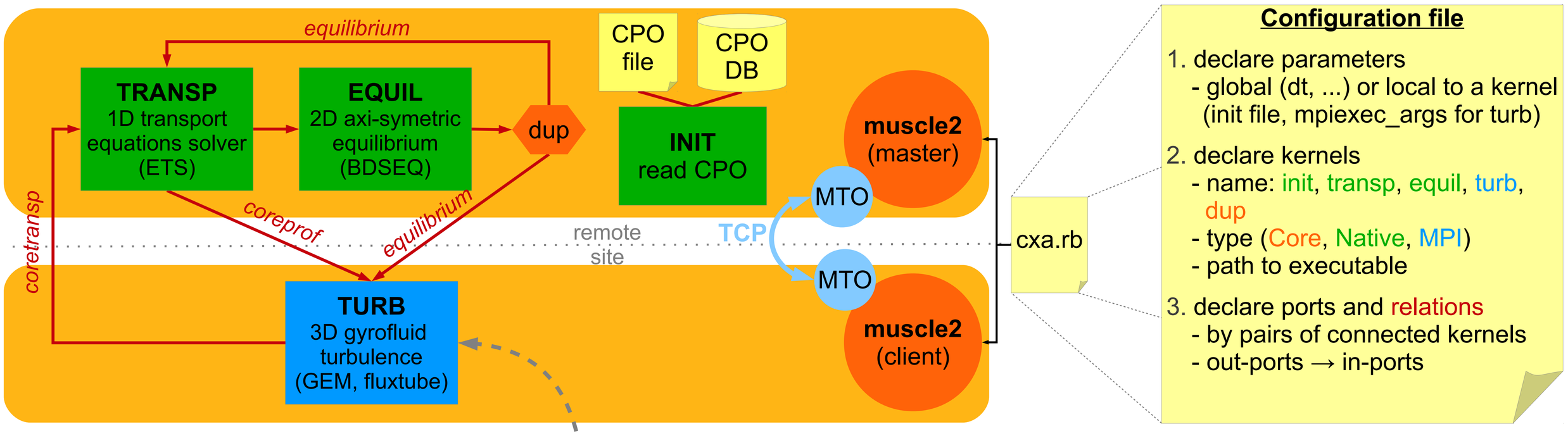
O. Hoenen[1], L. Fazendeiro[2], B D. Scott[1], J. Borgdorff[3], A G. Hoekstra[3], P. Strand[2], D P. Coster[1]
[1] Max-Planck-Institut für Plasmaphysik, Boltzmannstr. 2, D-85748, Garching, Germany [2] Chalmers University of Technology, SE-412 96 Göteborg, Sweden
[3] Institute of Informatics, University of Amsterdam, 1090 GH, The Netherlands

Introduction

- Multiscale simulation involving slow transport and fast turbulent scales is a challenging computational problem.
- Modelling a multiscale problem as a set of coupled single scale submodels prevents complexity of monolithic codes.
- Scales can be spatio-temporal domains or multi-physics.
- Such approach requires using some coupling framework.
- Interface agreement, data exchange:
  → submodels using ITM's generic datastructure, CPO [1]
- Need for distributed simulation capabilities:
  → execute a submodel on a specific hardware (accelerators or bigger HPC systems)
  → access to a local database (simulation or experiment)
- Target workflow: Transport-Equilibrium-Turbulence

MAPPER

- **M**ultiscale **APP**lications on **E**uropean e-Inf**R**astructure
  http://www.mapper-project.eu
- Provides a formal framework, tools, software and services for building and running distributed multiscale applications [2]
- Assists scientists developing multiscale applications from scratch or from legacy codes
- Two types of application:
  - loosely-coupled: acyclic, data exchange with file
  - tightly-coupled: cyclic, data exchange with coupling library
- Software stack on three levels:
  - high-level tools: web-based GUI, design and execution
  - middleware: distributed resource manager (grid and HPC)
  - coupling library (MUSCLE2): schedules work, transfers data



MUSCLE2

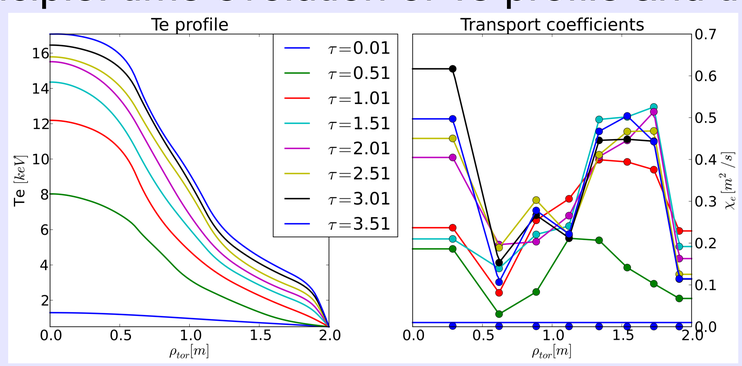
- Java-based library
- Simple API in C/C++ and F90:
  - `MUSCLE_init/finalize`
  - `MUSCLE_get_property`
  - `MUSCLE_send/recv`
- Communication: blocking, inter-operable types, byte streams
- Coupled simulation requires:
  → `muscle2` bootstrap program
  → submodels implementation (kernels)
  → configuration file

Muscle kernel
- init
- get workflow parameters
- time loop
  - receive data from in-ports
  - **I/O wrapper**
    - deserialize CPO
    - **Physics routine**
      - CPO as interface
      - XML specific params
    - serialize CPO
  - send data to out-ports
- finalize

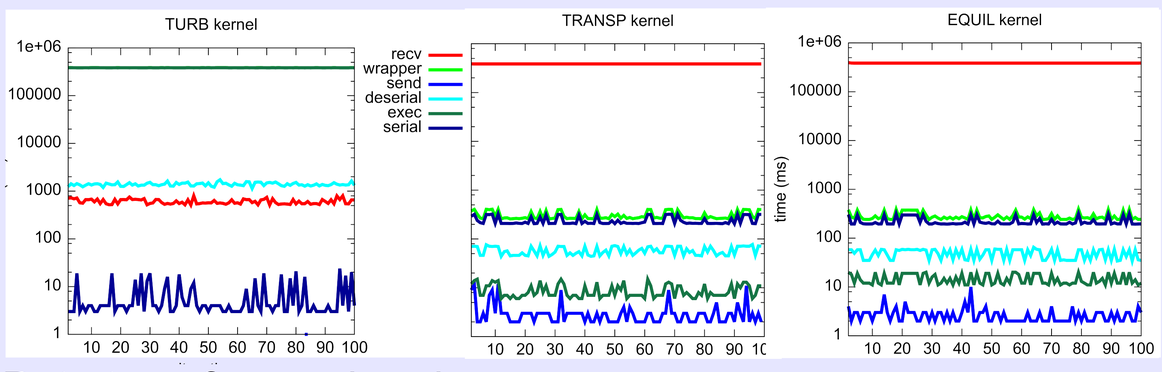
- Transparent remote transfer (clients just need master IP:port)
- Firewall traversal through Muscle Transfer Overlay (MTO)

Results

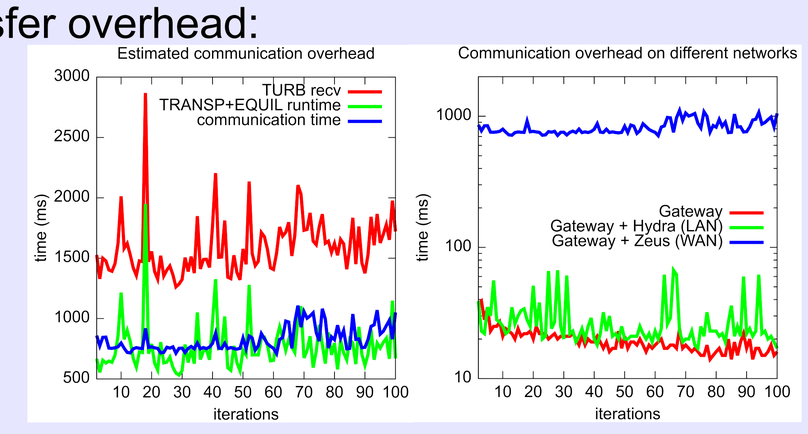
Proof of principle: time evolution of Te profile and transport coefficients



Muscle's kernel serialization/deserialization overhead:



Data transfer overhead:



Conclusion

- Simple to use, non-intrusive, adaptable through a config file
- MPI: only main process interacts with muscle ⇒ broadcast
- Serialization overhead is not negligible for fast kernels

References:
[1] F. Imbeaux, Computer Physics Communications **181**, 987 (2010)
[2] J. Borgdorff, et al., Procedia Computer Science **9**, 596 (ICCS 2012)

Acknowledgement: The research leading to these results has received funding from the European Union Seventh Framework Programme under grant agreement n°261507 (the MAPPER project)