Parallel distributed-memory computing with Gridap ecosystem

PartitionedArrays.jl, GridapDistributed.jl, GridapP4est.jl, GridapPETSc.jl, GridapSolvers.jl

Dr. Alberto F. Martín - `alberto.f.martin@anu.edu.au`
School of Computing, Australian National University, Canberra, 29th Nov 2023

Synergy among HPC and CSE is crucial

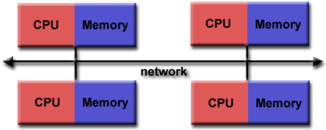
- We already find ourselves in the **Exascale** era ($\mathcal{O}(10^{18})$ FLOPs/s peak)
- **Frontier**: 1st Exascale supercomputer (Oak Ridge US National Labs) ($\sim$10M cores, 1.1EFLOPs/s, ranked #1 Jun, 2023 Top500 list)

- Performance boost mostly based on adding hardware parallelism (e.g., higher #cores/CPU) and heterogeneous hardware (CPUs, GPUs, . . .)
- To exploit such vast concurrency is a **formidable task** for CSE (breakthroughs in scalable algorithms and software innovations)

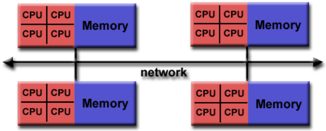
Parallel Distributed-memory multiprocessors (e.g., Gadi at NCI)

Distributed-memory multiprocessors

- Vary widely, but all present a **set of nodes** with local memory and CPUs each, interconnected through a **high speed network**
- Memory addresses in one CPU are **private** and don't map to other CPUs: no common global shared address space available
- ↑ **High scalability**: memory bandwidth grows linearly (at least) with number of nodes
- ↑ $ **effective**: can be built from commodity hw
- ↓ Programmer **explicitly handles** many details of data communication and synchronization
- ↓ Can be difficult to optimally map existing global data structures to distributed-memory
- ↓ Non-Uniform memory access times



Flat distributed-memory

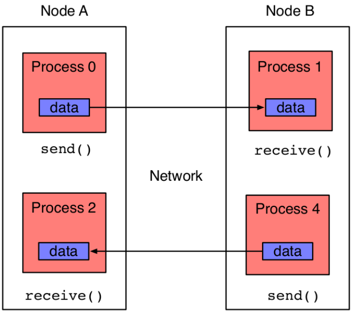


Hybrid shared/distributed-mem

Message-passing programming model at a glance

Message-passing programming model

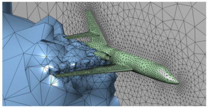
- Parallelism realized by **multiple processes** (aka tasks) each with their **own local memory address space**
- Data is moved from address space of one process to that of another by **sending/receiving messages**
- Processes may run on separate compute nodes, different cores within a node, or even on same processor core
- **Strictly required** if target parallel computer is of **distributed-memory** type. However, applies to shared-mem & hybrid systems as well
- “De facto” standard is MPI
- Julia MPI bindings provided by the `MPI.jl` package



Parallel FEM simulation pipeline steps (common approach)

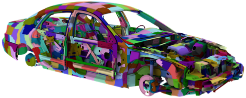
1. Unstructured mesh generation

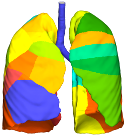
Delaunay triangulations mainstream



2. Mesh partition

Graph-based algorithms mainstream



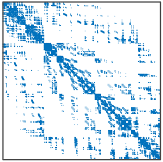


3. Discrete system assembly

Involves numerical integration on elements

Embarrassingly (trivially) parallel process

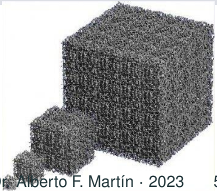
$$AU = F$$



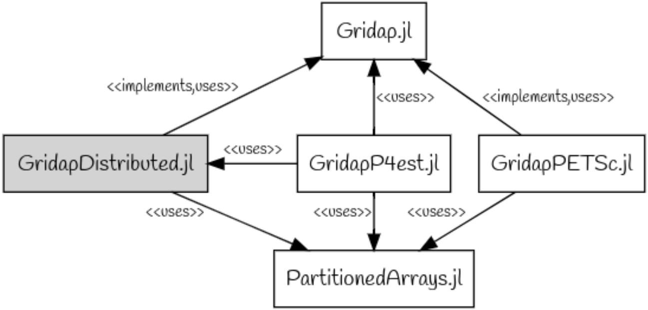
4. Discrete system solvers

Significance of **algorithmically scalable** solvers (FLOPs/mem demands linearly bounded with resolution)

Multilevel methods mainstream for discrete PDEs (Multigrid, Multilevel Domain Decomposition)

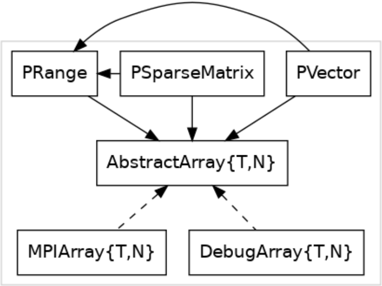


Parallel distributed-memory packages in Gridap at a glance

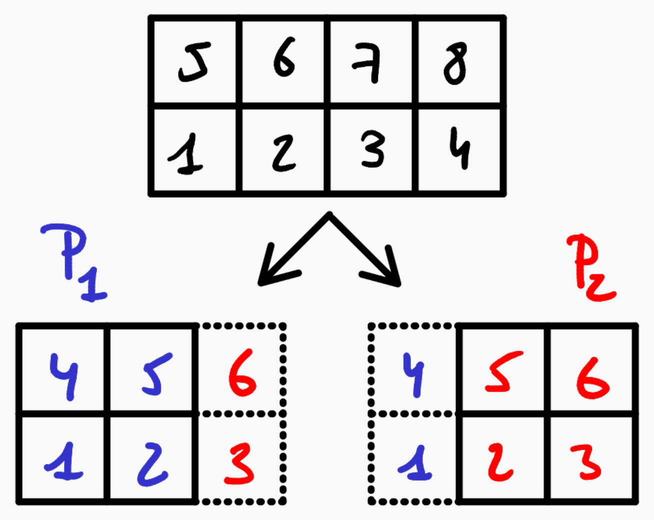


- `P4est` are `PETSc` message-passing (MPI) libraries written in `C`
- `P4est` provides parallel scalable AMR grounded on forest-of-octrees (UT Austin, University of Bonn, . . .)
- `PETSc` comprehensive library of linear and nonlinear parallel scalable solvers for PDEs (Argonne National Labs, US)
- `GridapSolvers.jl` left out as it is still WIP (ask Jordi if interested)
- **WARNING!!**: `GridapP4est.jl` not supported in macOS due to Julia limitations on this platform

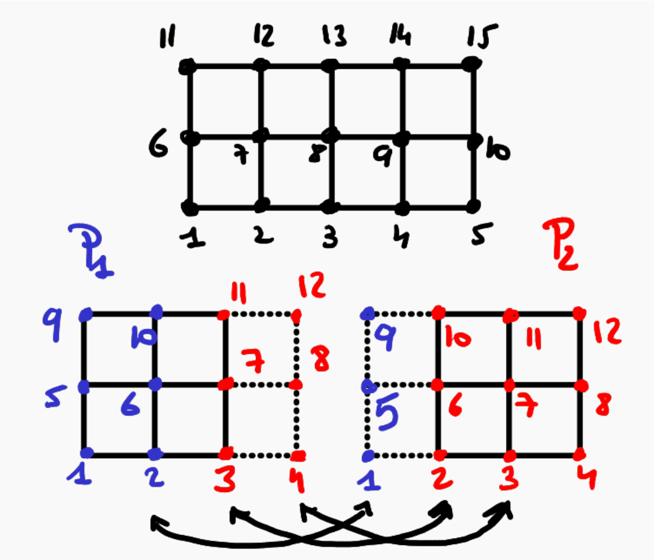
PartitionedArrays.jl (main sw abstractions)



- Distributed-memory linear algebra package grounded on MPI
- Distributed vectors (`PVector`) and sparse matrices (`PSparseMatrix`)
- Supports **Debug** and **MPI** modes
- `PRange` sw abstraction is crucial in distributed-memory computations
- It represents an **index set of global identifiers** partitioned among parallel tasks such that there may be overlapping among them


5 6 7 8
1 2 3 4
P1
P2
4 5 6
1 2 3
4 5 6
1 2 3

PRange example 2: FE Space partition



PRange example 3: sparse linear system