Science and Technology Facilities Council

Hartree Centre

CCP-WSI

a Collaborative Computational Project in Wave Structure Interaction

Introduction of OpenFOAM parallel programming

Dr. Xiaohu Guo, Hartree Centre, STFC

Table of Content

Science and Technology Facilities Council

Hartree Centre

Science and Technology Facilities Council

Hartree Centre

OpenFOAM parallel env

Science and Technology Facilities Council

Hartree Centre

- Parallel Environment Settings: etc/config.sh
  - Run-time environment, WM_ARCH, architecture(32/64 bit), WM_COMPILER_ARCH, WM_COMPILER_LIB_ARCH.
  - MPI, different version of MPI settings
  - Third-party domain decomposition libraries,
  - Other third party libraries

- Parallel Communication Wrapper:
  - Pstream
  - Field data updates across processor boundaries
  - Operator and matrix assembly

- Linear Solver parallel support

MPI and Pstream

MPI and it's development history

- MPI-1 was defined (1994) by a broadly based group of parallel computer vendors, computer scientists, and applications developers.
- MPI-2 was released in 1997 – Several additional features including MPI + threads, MPI-I/O, remote memory access functionality and many others.
- MPI-2.1 (2008) and MPI-2.2 (2009) were released with some corrections to the standard and small features.
- MPI-3 (2012) added several new features to MPI
- MPI-3.1 (2015) introduced minor corrections and features
- MPI-4 (June 2021) is the latest version of the standard.
- The Standard itself:
  - at http://www.mpi-forum.org
  - All MPI official releases, in both postscript and HTML
- Other information on Web:
  - at http://www.mcs.anl.gov/mpi
  - pointers to lots of material including tutorials, a FAQ, other MPI pages

- MPI-1: supports the classical message-passing programming model: basic point-topoint communication, collectives, datatypes, etc
- MPI-2: Several additional features including MPI + threads, MPI-I/O, remote memory access functionality and many others.
    - MPI-2.1: some corrections to the standard and small features.
- MPI-3: new features to MPI : Nonblocking collectives, Neighborhood collectives, Improved one-sided communication interface, Tools interface – Fortran 2008 bindings, Matching Probe and Recv for thread-safe probe and receive, Noncollective communicator creation function, “const” correct C bindings, Comm_split_type function, Nonblocking Comm_dup, Type_create_hindexed_block function.
- MPI-3.1 (2015) introduced minor corrections and features
- MPI-4: Persistent Collectives, Partitioned Communication, Sessions, Big Count, Error Handling Improvement , Topology Improvement

C++ stream

- streams provide a convenient and flexible way to perform input and output operations. The C++ Standard Library includes several stream classes that are used for reading from and writing to various sources, such as files, standard input/output (stdin/stdout), strings, and more.
  - std::cout (standard output stream): Used for outputting data to the console or terminal.
  - std::cin (standard input stream): Used for reading data from the console or terminal.
  - std::cerr (standard error stream): Used for error messages or diagnostics.
- Streams in C++ are based on the concept of "insertion" (<<) and "extraction" (>>) operators, which are overloaded for different types to enable input and output operations. These operators are used to write data to a stream (<<) or read data from a stream (>>).
- The term "stream" in the context of C++ refers to a continuous flow of data. It is called a "stream" because it represents a sequence of bytes or characters that can be read from or written to **sequentially**, just like a stream of water or a stream of data flowing through a pipe.

OpenFOAM Pstream

- Pstream derived from IOstream, no object-level changes are required:
  - Pout (standard parallel output stream): Used for parallel outputting data to the console or terminal.
  - Perr (standard parallel paraerror stream): Used for error messages or diagnostics.
- Pstream communication-dependent part is limited to a few functions: initialise, exit, send data and receive data, AlltoAll, Broadcast, GatherScatter and Reduce.

- Fundamental defects:
  - MPI wrapper implementation is limited, lack of support to the latest MPI standard.
  - Traditional master-slave communications are out of dated and become performance bottleneck for large scale

Few useful Pstream functions

- **Pstream::parRun()**

```
if( !Pstream::parRun() ) {
  Info << "Error: The program should be run under mpirun
      \n";
}
```

- **Pstream::myProcNo()** and **Pstream::masterNo()**

```
Info << "My rank in MPI Communicator is "
     << Pstream::myProcNo() << " and master rank "
     << Pstream::masterNo() << "\n";
```

- **Perr**

  Perr << “Some error occurred in parallel job \n” ;

- **Pout**

  Pout << “Hello from processor” << Pstream::myProcNo() << endl;

- **OPstream** and I**Pstream**

```
vector data(0, 1, 2);
OPstream toMaster(Pstream::scheduled, Pstream::masterNo
    ());
toMaster << data;
...
IPstream fromMaster(Pstream::scheduled, Pstream::
    masterNo());
fromMaster >> data;
/* Types of schedules      **
**  Pstream::scheduled     **
**  Pstream::blocking      **
**  Pstream::nonBlocking   */
```

// Spreading a value across all processors

- **Pstream::scatter**

```
Pstream::scatter(meshVolume);

Pstream::scatterList(nInternalFaces);
```

//Add the values from all processes together

- **reduce()**

```
reduce(meshVolume, sumOp<scalar>());
```

- **Pstream::myProcNo()** and **Pstream::master()**

```
Pout<< "Hello from process " << Pstream::myProcNo() << endl;

if (Pstream::master())
{
```

- **Collective communication binary operators on labelList**
  - `plusOp<labelList>()`
  - `minusOp<labelList>()`
  - `divideOp<labelList>()`
  - `minOp<labelList>()`
  - `maxOp<labelList>()`

Hands on with Pstream

- Check if parallel environment is defined using Pstream
- If it is parallel run, print the processor rank
- calculate the totlal mesh volume
- Add the values from all processes together
- Spreading a value across all processors with scatter operator
- Gather and Scatter a List

Science and Technology Facilities Council

Hartree Centre

Field data parallel exchange

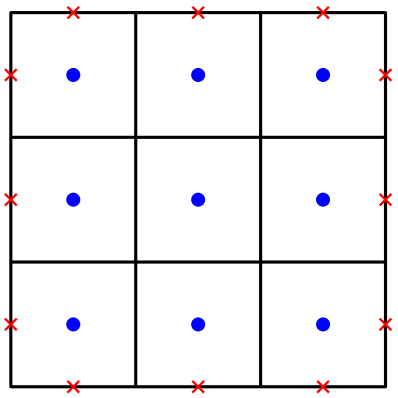
Science and Technology Facilities Council

Hartree Centre

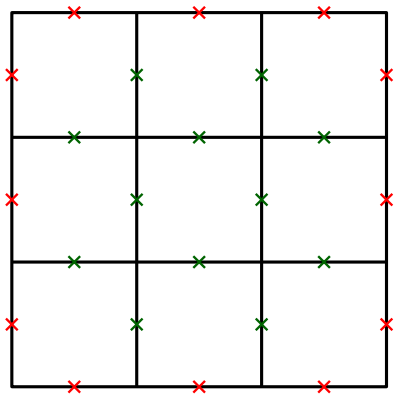
OpenFOAM Field variables

The **geometricField< Type >** is renamed used typedef declarations to indicate where field variables are stored:

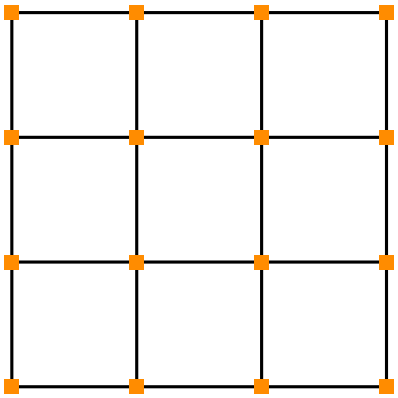
1. **volField<Type>** −→ Field defined at cell centres.
2. **surfaceField<Type>** −→ Field defined at cell faces.
3. **pointField<Type>** −→ Field defined at cell vertices.



(a) volField



(b) surfaceField



(c) pointField

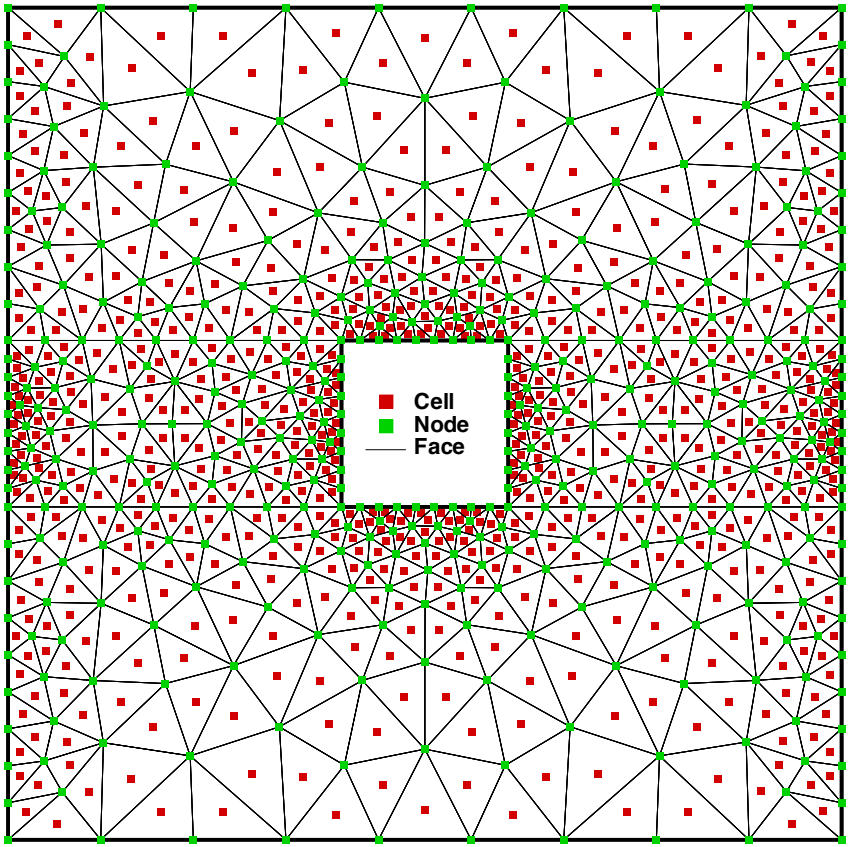
Recap …

- **mesh.C()** - volVectorField storing cell centroids
- **mesh.points()** - pointField storing mesh nodes
- **mesh.V()** - volScalarField storing cell volumes
- **mesh.Sf()** - surfaceVectorField storing face area vector
- **mesh.magSf()** - surfaceScalarField storing face area magnitude
- **mesh.Cf()** - surfaceVectorField storing face centroid

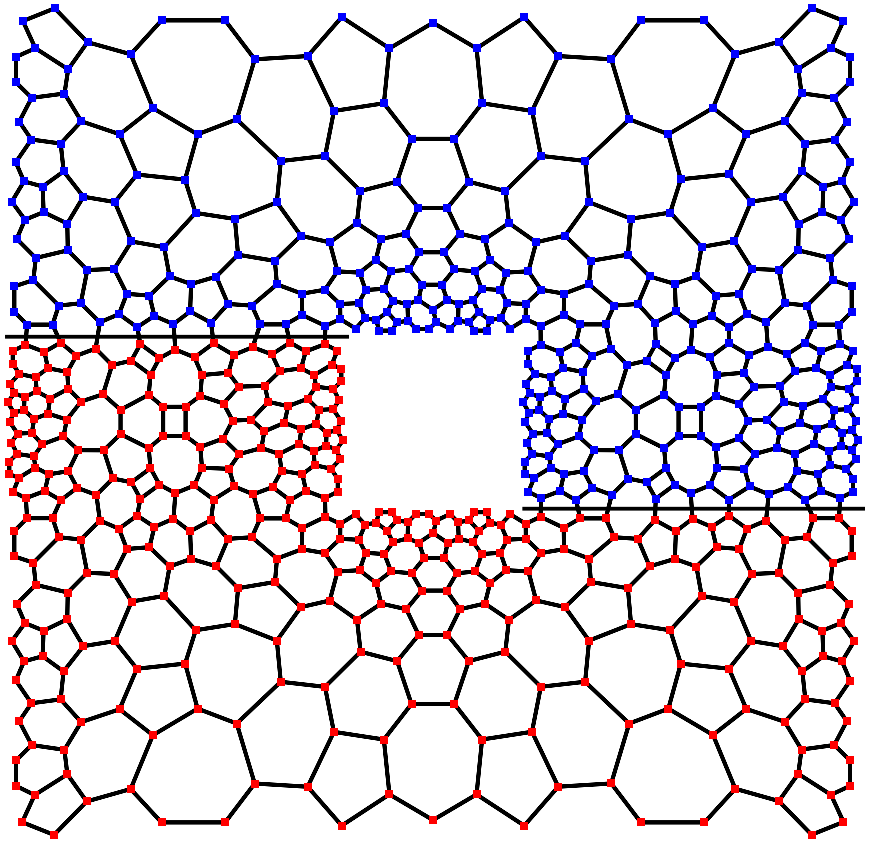
Recap …

- Parallel communications are wrapped in **Pstream** library to hide communication details from users
- Discretization uses the domain decomposition with zero halo layer approach
- Parallel updates are a special case of coupled discretization and linear algebra functionality
  - **processorFvPatch** class for coupled discretization updates
  - **processorLduInterface** and field for linear algebra updates

Recap …



Unstructured Mesh

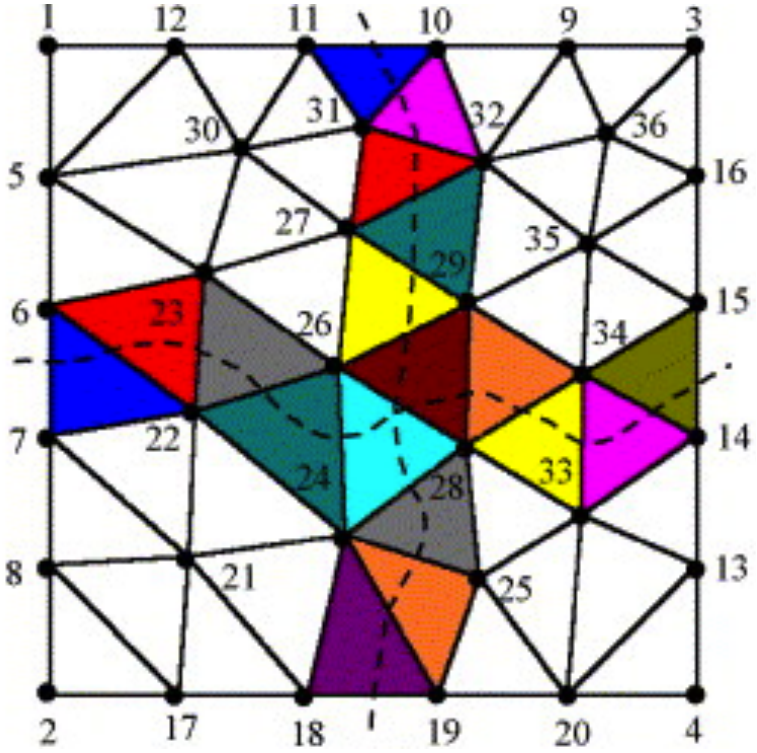


Dual Graph

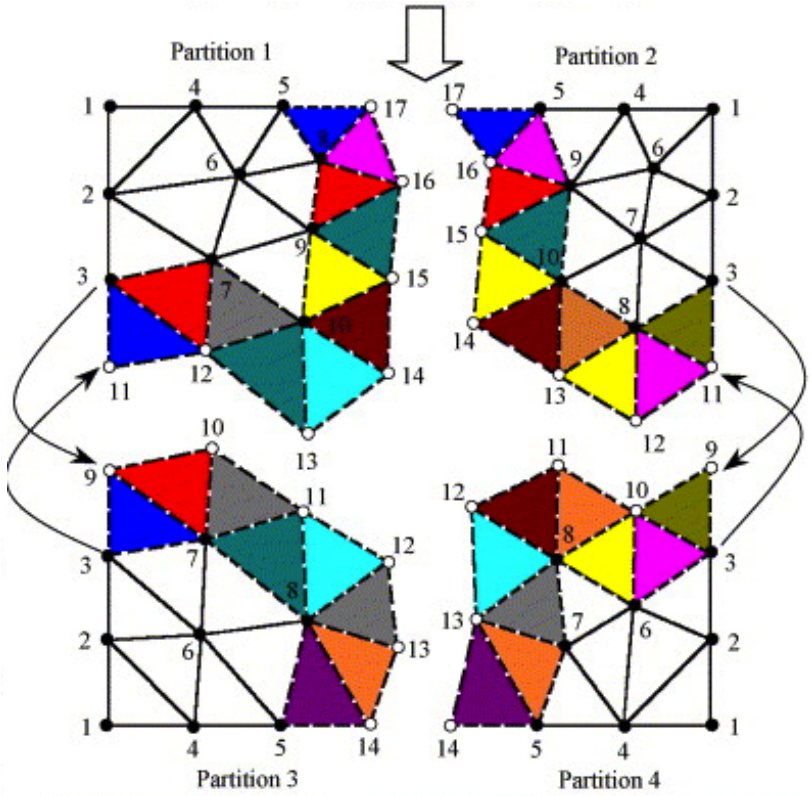
Science and Technology Facilities Council
Hartree Centre

Figure courtesy of Dr. Pavanakumar Mohanamuraly

processorBoundary



Node based partition



Graph based partition

Science and Technology Facilities Council
Hartree Centre

Figure courtesy of Dr. Pavanakumar Mohanamuraly

processorBoundary – halo exchange mechanism

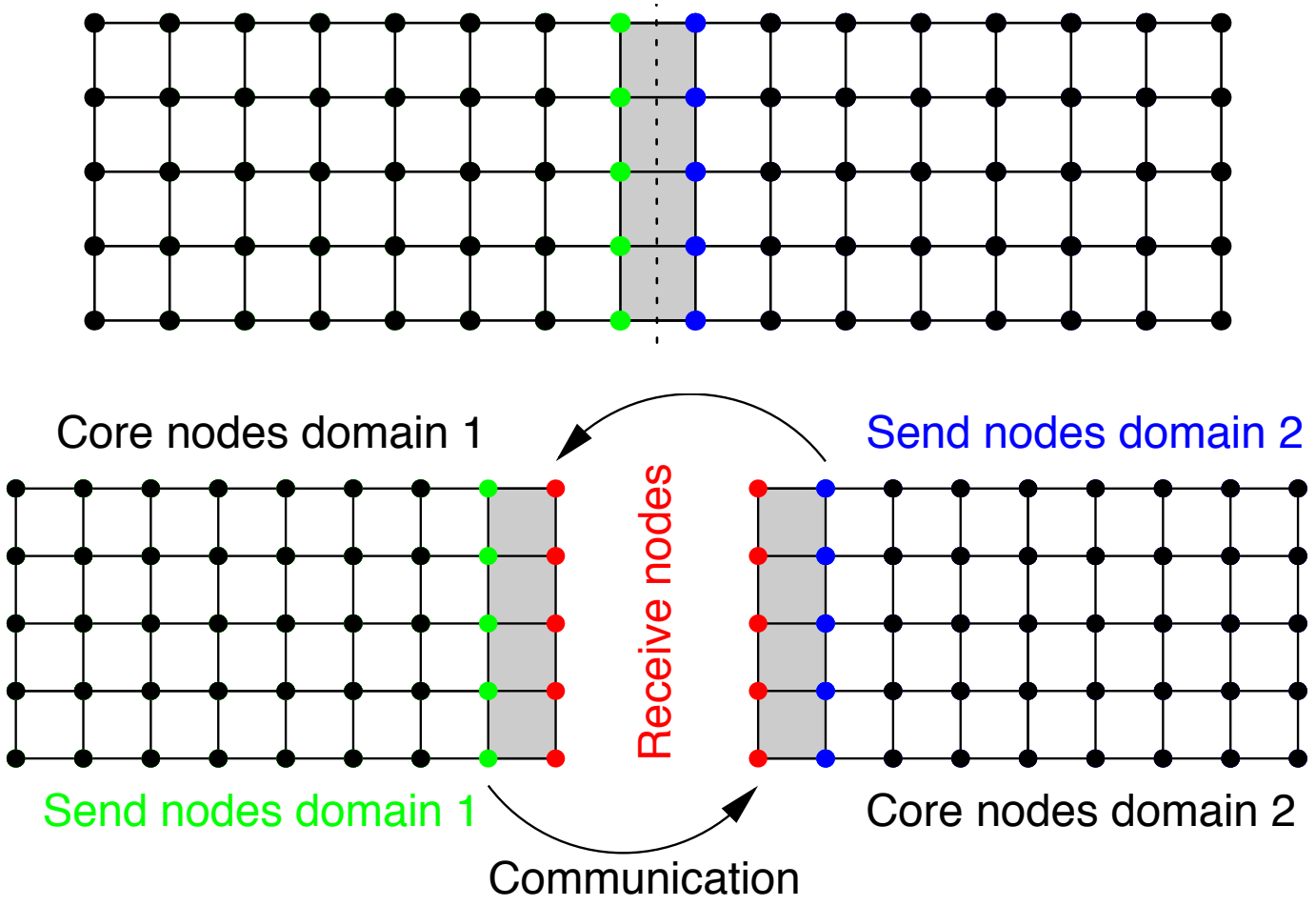


Figure courtesy of Dr. Pavanakumar Mohanamuraly

- OpenFOAM does not create the extra cell padding (Halo/Ghost)
- Reference to immediate cell quantities to “processor” boundary faces stored
- Quantities/Fields are imported during computation at “processor” boundary

The key data-structure for prime field variables

- Communication- blocking wait on the send/receive: **correctBoundaryConditions()**

```
rho.correctBoundaryConditions();
U.correctBoundaryConditions();
p.correctBoundaryConditions();
```

- Access Ghost/Halo Data: **patchNeighbourField()**

```
scalarField rhoGhost = rho.boundaryField()[ipatch].
    patchNeighbourField();
vectorField UGhost   = U.boundaryField()[ipatch].
    patchNeighbourField();
scalarField pGhost   = p.boundaryField()[ipatch].
    patchNeighbourField();
```

OpenFOAM non-blocking communication

```
/// Time step loop
while( runTime.loop() ) {
   ...
   ...
   runTime.write();
}
```

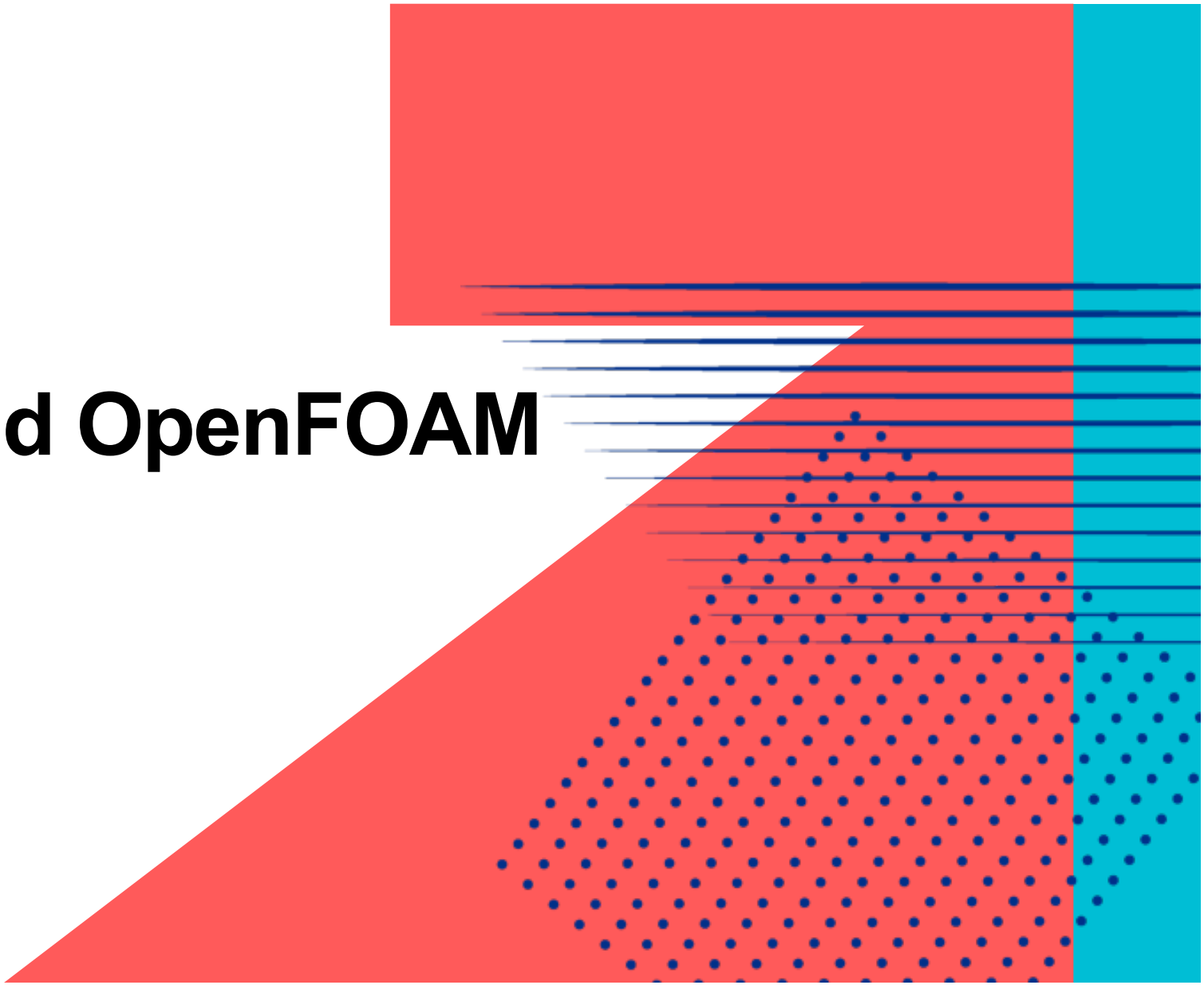
- For every iteration of the **runTime.loop()** non-blocking can be initiated by specify send/recv communicator type
- During application of “processor” BC the export/import is confirmed using blocking call
- This essentially forms the Zero-Halo implementation in OF
- Computation and communication happen simultaneously - latency hiding if non-blocking initiated

- Field data parallel exchange
  - Field variables get data from processor boundary condition or coupledFvPatch
  - Field processor boundary data exchange allow non-block communication
  - Volume field allow faces fluxes exchange in parallel.
- Hands on

```
forAll( rho , icell  )
  rho[icell] = Pstream::myProcNo();

forAll ( mesh.boundaryMesh() , ipatch ) {
  word BCtype = mesh.boundaryMesh().types()[ipatch];
  const UList<label> &bfaceCells =
    mesh.boundaryMesh()[ipatch].faceCells();

if( BCtype == "processor" ) {

rho.correctBoundaryConditions();
```

Science and Technology Facilities Council

Hartree Centre

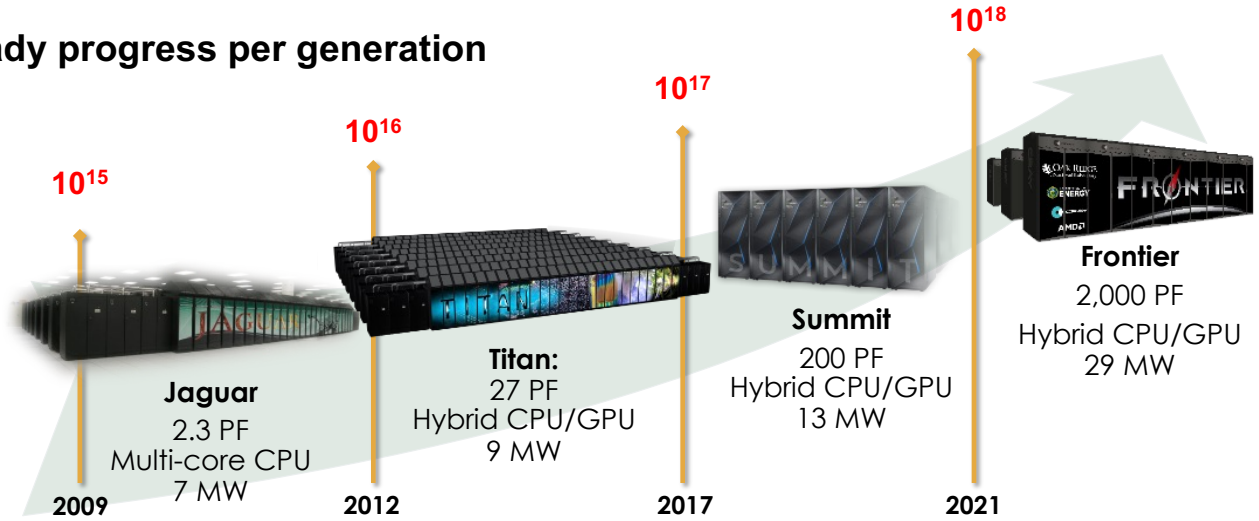


Exascale and OpenFOAM

Science and Technology Facilities Council

Hartree Centre

Steady progress per generation



Science and Technology Facilities Council
Hartree Centre

US ECP and programming models

NERSC Perlmutter
AMD CPU / NVIDIA GPU
CUDA / *OpenMP 5* [(c)]

ORNL Frontier
AMD CPU / AMD GPU
HIP / *OpenMP 5* [(d)]

ANL Aurora
Xeon CPUs / Intel GPUs
DPC++ / *OpenMP 5* [(e)]

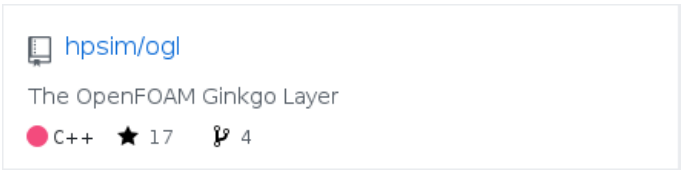
LLNL El Capitan
AMD CPU / AMD GPU
HIP / *OpenMP 5* [(d)]

OpenFoam Ginkgo layer (OGL)

- Plugin for OpenFOAM to offload **linear solver** to GPUs

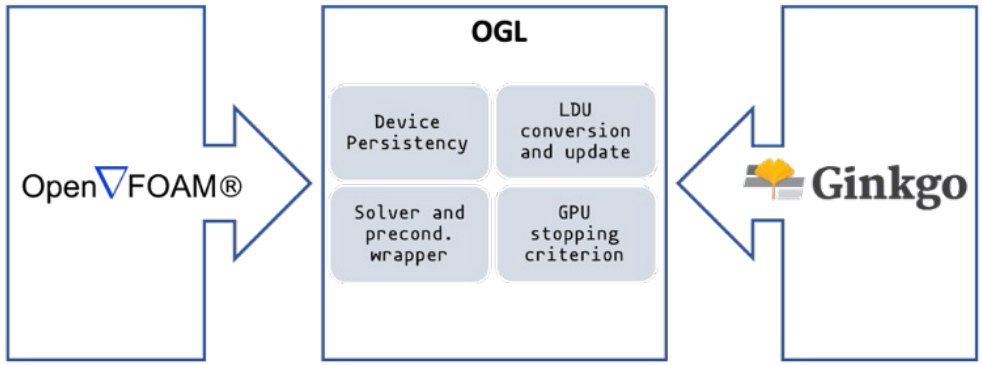
OpenFOAM® Ginkgo

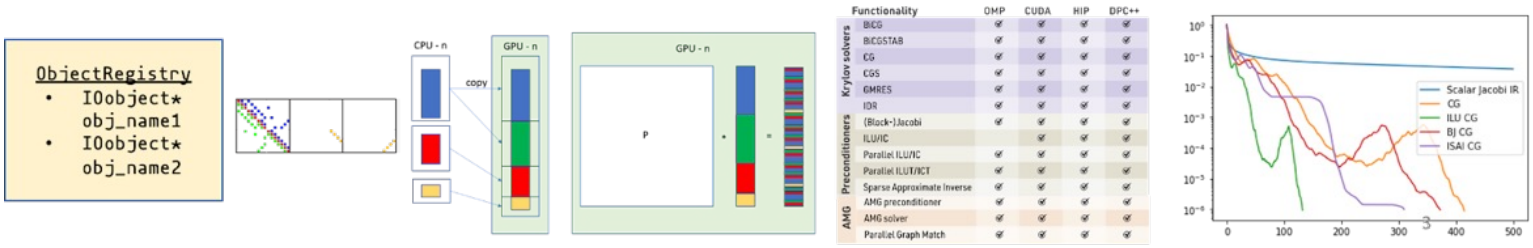
- Simple to use, just add *libOGL.so* to controlDict, no special (yet) application or **recompilation** of OF needed
- Cmake build process of OGL takes care of pulling, building, and installing Ginkgo
- Available at github.com/hpsim/ogl



Science and Technology Facilities Council
Hartree Centre

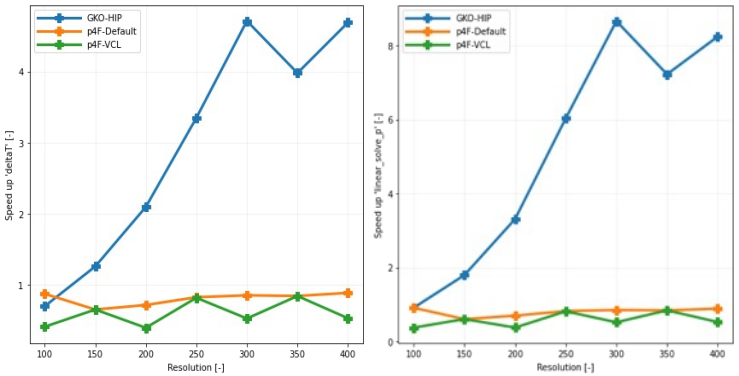
Thanks to Prof. Hartwig Anzt for providing the slides



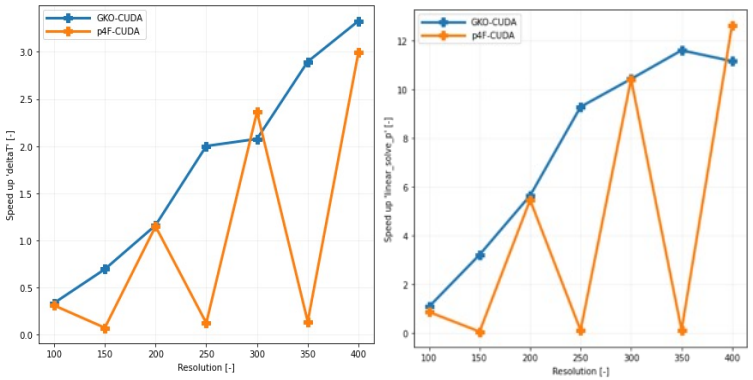


Thanks to Prof. Hartwig Anzt for providing the slides

- On tested machines a speed up of up to approx. 8 MI100 and 12 A100 in the linear solver are observed;
- This corresponds to a speed up of up to approx. 5 NLA and 4 HoreKa for the full time step;
- On machines with AMD GPUs Ginkgo is currently the only available option. On NVIDIA machines Ginkgo performed reliable across all tested mesh sizes.



Speedup of the full timestep (left) and the linear solver (right) of OpenFOAM with Ginkgo (blue), PETSc with ViennaCL (green), and default PETSc (orange) as computational backends wrt. OpenFOAMs default CPU backend. Ginkgo and PETSc based simulations are run on 8 AMD MI100 GPUs and 8 AMD EPYC 7302 cores, simulations using exclusively CPUs are run on 32 AMD EPYC 7302 cores.



Performance issues for computational grids with resolutions of (n*50)^3 for PETSc based simulations. Speedup of the full timestep (left) and the linear solver (right) of OpenFOAM with Ginkgo (blue) and PETSc (orange) as computational backends wrt. OpenFOAMs default CPU backend using the all available 76 cores.

Science and Technology Facilities Council
Hartree Centre

Thanks to Prof. Hartwig Anzt for providing the slides

Hands On Session


UKRI
Science and
Technology
Facilities Council
Hartree Centre
Questions?


UKRI
Science and Technology Facilities Council
Hartree Centre
Thank you
hartree.stfc.ac.uk
@HartreeCentre
STFC Hartree Centre
yiyun.tan@stfc.ac.uk