NHR4CES NHR for Computational Engineering Science

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallelization in OpenFOAM for HPC Deployment

Mohammed Elwardi Fadeli[1,2], Holger Marschall[1] and Christian Hasse[2]

April, 2024

[1] Mathematical Modeling and Analysis (MMA)
[2] Simulation of Reactive Thermo Fluid Systems (STFS)
Energy Conversion Group, NHR4CES - TU Darmstadt

Motivation

- Solving some problems requires more computing power than what typical machines provide.
- Solution $\rightarrow$ Different ways of parallel optimization

Motivation

- Solving some problems requires more computing power than what typical machines provide.
- Solution $\rightarrow$ Different ways of parallel optimization
  - Single core: SIMD

Motivation

- Solving some problems requires more computing power than what typical machines provide.
- Solution $\rightarrow$ Different ways of parallel optimization
  - Single core: SIMD
    - $ax^2 + bx \rightarrow x(ax + b)$

Motivation

- Solving some problems requires more computing power than what typical machines provide.
- Solution $\rightarrow$ Different ways of parallel optimization
  - Single core: SIMD
    - $ax^2 + bx \rightarrow x(ax + b)$
    - 3 multiplications $\rightarrow$ 2 multiplications

Motivation

- Solving some problems requires more computing power than what typical machines provide.
- Solution $\rightarrow$ Different ways of parallel optimization
  - Single core: SIMD
    - $ax^2 + bx \rightarrow x(ax + b)$
    - 3 multiplications $\rightarrow$ 2 multiplications
    - $\begin{pmatrix} x_0 \\ x_1 \end{pmatrix} \circ \begin{pmatrix} ax_0 + b \\ ax_1 + b \end{pmatrix} \rightarrow$ can be computed concurrently.

Motivation

- Solving some problems requires more computing power than what typical machines provide.
- Solution $\rightarrow$ Different ways of parallel optimization
  - Single core: SIMD
    - $ax^2 + bx \rightarrow x(ax + b)$
    - 3 multiplications $\rightarrow$ 2 multiplications
    - $\begin{pmatrix} x_0 \\ x_1 \end{pmatrix} \circ \begin{pmatrix} ax_0 + b \\ ax_1 + b \end{pmatrix} \rightarrow$ can be computed concurrently.
  - Multi-core (multi-threading): OpenMP, `std::thread`, library based, ..., etc

Motivation

- Solving some problems requires more computing power than what typical machines provide.
- Solution $\rightarrow$ Different ways of parallel optimization
  - Single core: SIMD
    - $ax^2 + bx \rightarrow x(ax + b)$
    - 3 multiplications $\rightarrow$ 2 multiplications
    - $\begin{pmatrix} x_0 \\ x_1 \end{pmatrix} \circ \begin{pmatrix} ax_0 + b \\ ax_1 + b \end{pmatrix} \rightarrow$ can be computed concurrently.
  - Multi-core (multi-threading): OpenMP, `std::thread`, library based, ..., etc
  - **Machine clusters: MPI** $\leftarrow$ our workshop's scope

Motivation

- Solving some problems requires more computing power than what typical machines provide.
- Solution $\rightarrow$ Different ways of parallel optimization
  - Single core: SIMD
    - $ax^2 + bx \rightarrow x(ax + b)$
    - 3 multiplications $\rightarrow$ 2 multiplications
    - $\begin{pmatrix} x_0 \\ x_1 \end{pmatrix} \circ \begin{pmatrix} ax_0 + b \\ ax_1 + b \end{pmatrix} \rightarrow$ can be computed concurrently.
  - Multi-core (multi-threading): OpenMP, `std::thread`, library based, ..., etc
  - **Machine clusters: MPI** $\leftarrow$ our workshop's scope
  - Custom accelerator hardware (GPUs, FPGAs, ..., etc)

And more **team members here** - working on

Computationally efficient HPC-ready, (reactive)CFD software and methods.

NHR4CES NHR for Computational Engineering Science

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallelization in OpenFOAM for HPC Deployment

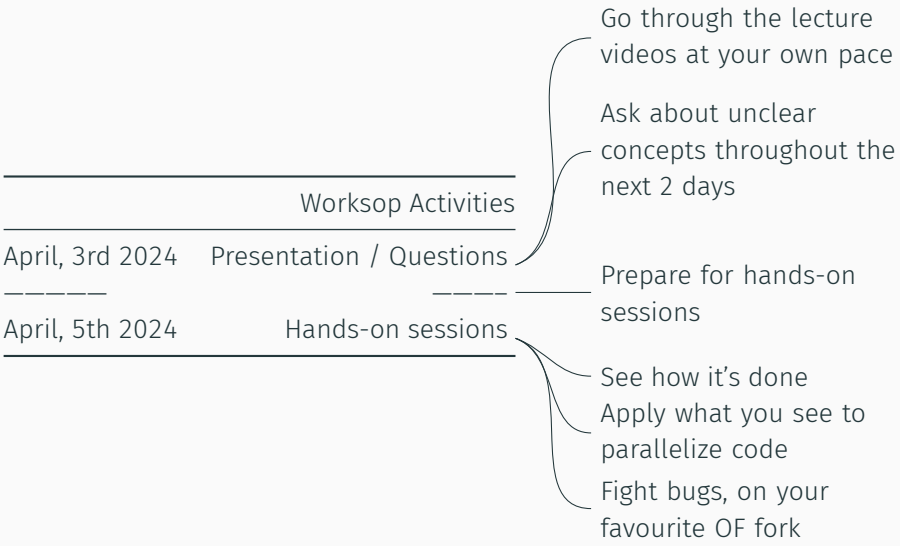
Agenda

Mohammed Elwardi Fadeli[1,2], Holger Marschall[1] and Christian Hasse[2]

April, 2024

[1] Mathematical Modeling and Analysis (MMA)
[2] Simulation of Reactive Thermo Fluid Systems (STFS)
Energy Conversion Group, NHR4CES - TU Darmstadt

Lecture section: Agenda



Lecture section: Table of contents

NHR4CES NHR for Computational Engineering Science

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallelization in OpenFOAM for HPC Deployment

General Introduction

Mohammed Elwardi Fadeli[1,2], Holger Marschall[1] and Christian Hasse[2]

April, 2024

[1] Mathematical Modeling and Analysis (MMA)
[2] Simulation of Reactive Thermo Fluid Systems (STFS)
Energy Conversion Group, NHR4CES - TU Darmstadt

The power of parallel workers

Figure 1: Parallel work during F1 Pit stops;

Types of Parallelism

Data Parallelism

Work units execute the same operations on a (distributed) set of data: **domain decomposition**.

Task Parallelism

Work units execute on different control paths, possibly on different data sets: **multi-threading**.

Pipeline Parallelism

Work gets split between producer and consumer units that are directly connected. Each unit executes a single phase of a given task and hands over control to the next one.

Types of Parallelism

Data Parallelism

Work units execute the same operations on a (distributed) set of data: **domain decomposition**.

Task Parallelism

Work units execute on different control paths, possibly on different data sets: **multi-threading**.

Pipeline Parallelism

Work gets split between producer and consumer units that are directly connected. Each unit executes a single phase of a given task and hands over control to the next one.

Domain decomposition in OpenFOAM

simple
: Simple geometric decomposition, in which the domain is split into pieces by direction

hierarchical
: Same as simple, but the order in which the directional split is done can be specified

metis & scotch
: Require no geometric input from the user and attempts to minimize the number of processor boundaries. Weighting for the decomposition between processors can be specified

manual
: Allocation of each cell to a particular processor is specified directly.

Domain decomposition in OpenFOAM: Processor boundaries

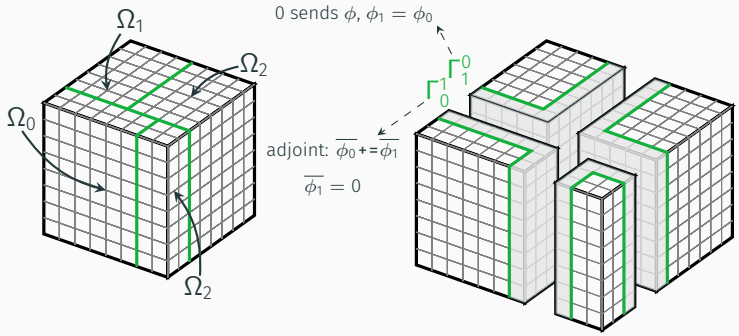


Figure 2: Classical halo approach for inter-processor communication

- Use of a layer of ghost cells to handle comms with neighboring processes → MPI calls not self-adjoint
- Artificial increase in number of computations per process (and does not scale well)

Domain decomposition in OpenFOAM: Processor boundaries

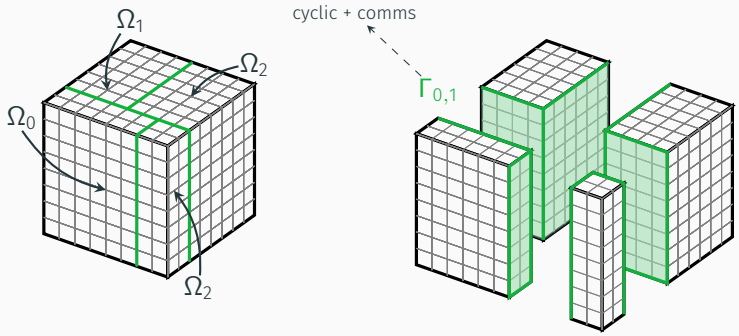


Figure 3: Zero-halo approach for inter-processor communication in OpenFOAM

- Communications across process boundaries handled as a BC
- MPI calls are self-adjoint; all processes perform the same work at the boundaries

Domain decomposition in OpenFOAM: Processor boundaries

Communicate efficiently through processor boundaries

```
1  include $(GENERAL_RULES)/mpi-rules
2  EXE_INC = $(PFLAGS) $(PINC) ...
3  LIB_LIBS = $(PLIBS) ...
```

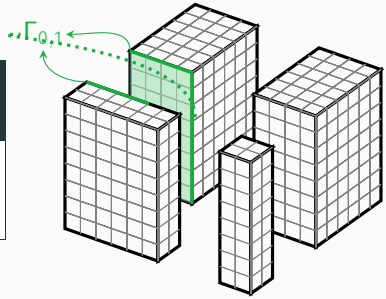


Figure 4: Swapping boundary fields across processor boundaries

- Standard API for common processor boundary fields operations
  - → Generalized for coupled patches, little to no micro management!
- Same local operations carried out on both processors (No checking for processor ranks … etc)

Modes of Parallelism

Distributed Memory

Message Passing Interface (**MPI**): Execute on multiple machines.

Shared Memory

Multi-threading capabilities of programming languages, OpenMP:
One machine, many CPU cores.

Data Streaming

CUDA and OpenCL. Applications are organized into streams (of same-type elements) and kernels (which act on elements of streams) which is suitable for accelerator hardware (GPUs, FPGAs, ..., etc).

Does 'echo' work work with MPI?

```
include $(GENERAL_RULES)/mpi-rules
EXE_INC = $(PFLAGS) $(PINC) ...
LIB_LIBS = $(PLIBS) ...
```

```
Hello World!
Hello World!
Hello World!
```

What about a solver binary?

```
include $(GENERAL_RULES)/mpi-rules
EXE_INC = $(PFLAGS) $(PINC) ...
LIB_LIBS = $(PLIBS) ...
```

This runs on "undecomposed" cases!

Does 'echo' work work with MPI?

```
include $(GENERAL_RULES)/mpi-rules
EXE_INC = $(PFLAGS) $(PINC) ...
LIB_LIBS = $(PLIBS) ...
```

Hello World!
Hello World!
Hello World!

What about a solver binary?

```
include $(GENERAL_RULES)/mpi-rules
EXE_INC = $(PFLAGS) $(PINC) ...
LIB_LIBS = $(PLIBS) ...
```

This runs on "undecomposed" cases!

But the solver is linked to libmpi!

```
include $(GENERAL_RULES)/mpi-rules
EXE_INC = $(PFLAGS) $(PINC) ...
LIB_LIBS = $(PLIBS) ...
```

... libmpi.so ...

Does 'echo' work work with MPI?

```
include $(GENERAL_RULES)/mpi-rules
EXE_INC = $(PFLAGS) $(PINC) ...
LIB_LIBS = $(PLIBS) ...
```

Hello World!
Hello World!
Hello World!

What about a solver binary?

```
include $(GENERAL_RULES)/mpi-rules
EXE_INC = $(PFLAGS) $(PINC) ...
LIB_LIBS = $(PLIBS) ...
```

This runs on "undecomposed" cases!

But the solver is linked to libmpi!

```
include $(GENERAL_RULES)/mpi-rules
EXE_INC = $(PFLAGS) $(PINC) ...
LIB_LIBS = $(PLIBS) ...
```

... libmpi.so ...

Alright we get it now

```
include $(GENERAL_RULES)/mpi-rules
EXE_INC = $(PFLAGS) $(PINC) ...
```

Needs a decomposed case

MPI with OpenFOAM: Parallel mode

Anatomy of MPI programs

```
1 include $(GENERAL_RULES)/mpi-rules
2 EXE_INC = $(PFLAGS) $(PINC) ...
3 LIB_LIBS = $(PLIBS) ...
```

MPI with OpenFOAM: Parallel mode

Anatomy of MPI programs

```
include $(GENERAL_RULES)/mpi-rules
EXE_INC = $(PFLAGS) $(PINC) ...
LIB_LIBS = $(PLIBS) ...
```

How solver programs look

```
include $(GENERAL_RULES)/mpi-rules
EXE_INC = $(PFLAGS) $(PINC) ...
LIB_LIBS = $(PLIBS) ...
```

You don't have to know MPI API to parallelise OpenFOAM code! But you need the concepts.

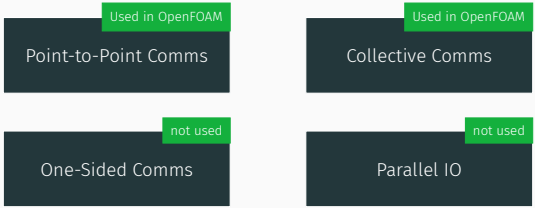
Objectives

1. Have a basic understanding of Parallel programming with MPI in OpenFOAM Code.
2. Be able to send basic custom object types around using MPI.
3. Be aware of some of the common issues around MPI comms.
4. Acquire enough knowledge to learn more on your own
   - Directly from OpenFOAM's code
   - MPI in general

By the end of the Workshop

→ Be able to parallelize basic serial OpenFOAM code.

Communication types in MPI



We'll be focusing on the communications OpenFOAM wraps!

NHR4CES NHR for Computational Engineering Science

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallelization in OpenFOAM for HPC Deployment

Point-to-Point Communications / General Introduction

Mohammed Elwardi Fadeli[1,2], Holger Marschall[1] and Christian Hasse[2]

April, 2024

[1] Mathematical Modeling and Analysis (MMA)
[2] Simulation of Reactive Thermo Fluid Systems (STFS)
Energy Conversion Group, NHR4CES - TU Darmstadt

Communicators and ranks

There may be many processes talking!

MPI Communicators

Objects defining which processes can communicate; Processes are referred to by their **ranks**

- `MPI_COMM_FOAM` in the Foundation version and Foam Extend 5
- `MPI_COMM_WORLD` (All processes) elsewhere
- Size: `Pstream::nProcs()`

MPI rank

Process Identifier (an integer).

- `Pstream::myProcNo()` returns the active process's ID.

Serialization and De-serialization: Basics

Premise: resurrect objects from streams of (binary or readable-text) data.

Standard C++ way

Manually Override `operator<<` and `operator>>` to interact with stream objects.

- This is used extensively in OpenFOAM
- Newer languages provide automatic serialization at language-level 🧐

Third-party libs

Try to handle automatic serialization

- Boost's serialization library if you're into Boost
- `cereal` as header-only library

Serialization for MPI comms

- MPI defines its own **Data Types** so it can be ”cross-platform”
- OpenFOAM gets around it using **parallel streams** as means of serialization
  - OpenFOAM hands over a stream-representation of your data to MPI calls
  - MPI passes the information in those streams around

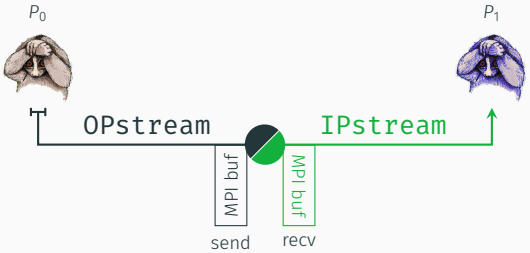


Figure 5: Communication between two processes in OpenFOAM

P2P comms: A first example

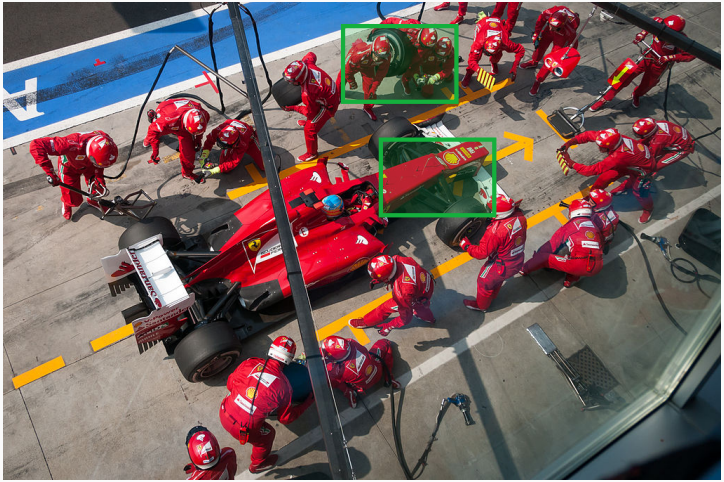


Figure 1: Parallel work during F1 Pit stops;

P2P comms: A first example

- `Pstream` class provides the interface needed for communication
- Each ”send” must be matched with a ”receive”

Slaves talk to master in a P2P fashion

```
include $(GENERAL_RULES)/mpi-rules
EXE_INC = $(PFLAGS) $(PINC) ...
LIB_LIBS = $(PLIBS) ...
```

NHR4CES NHR for Computational Engineering Science

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallelization in OpenFOAM for HPC Deployment

Point-to-Point Communications / Blocking comms

Mohammed Elwardi Fadeli[1,2], Holger Marschall[1] and Christian Hasse[2]

April, 2024

[1] Mathematical Modeling and Analysis (MMA)
[2] Simulation of Reactive Thermo Fluid Systems (STFS)
Energy Conversion Group, NHR4CES - TU Darmstadt

P2P comms: A first example

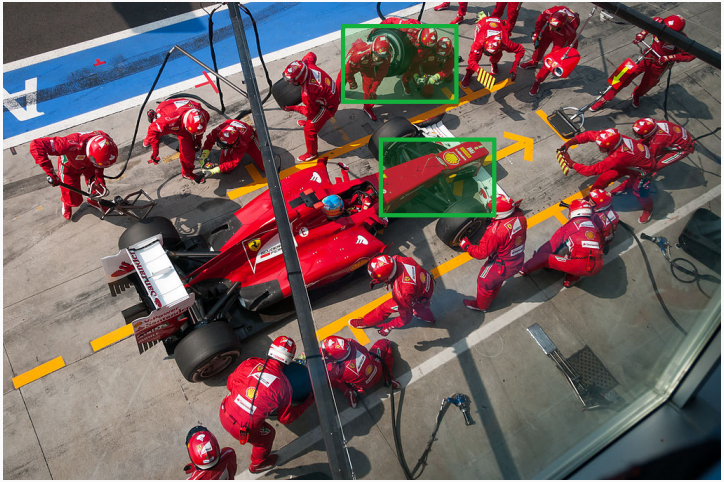


Figure 1: Parallel work during F1 Pit stops;

P2P Blocking comms

`Pstream::commsTypes::blocking` (or just `Pstream::blocking` in Foam Extend) defines properties for the MPI call which is executed by the constructed stream.

Sending is an asynchronous buffered blocking send

- Block until a copy of the passed buffer is made.
- MPI will send it at a later point, we just can't know when.
- All we know is that the buffer is ready to be used after it returns.

Receiving is a blocking receive

- Block until incoming message is copied into passed buffer.

P2P Blocking comms

`Pstream::commsTypes::scheduled` (or just `Pstream::scheduled` in Foam Extend) lets MPI pick the best course of action (in terms of performance and memory). This may also depend on the MPI implementation.

- Does a ”standard send”, Either:
  1. Do a buffered send (like blocking) if the buffer has enough free space to accommodate sent data.
  2. Fall back to a synchronous send otherwise.
- Both blocking and scheduled comms have a chance of causing deadlocks

A Deadlock happens when a process is waiting for a message that never reaches it.

P2P Blocking comms: Deadlocks

- Either a matching send or a recieve is missing (Definitely a deadlock).
- A send-recieve cycle (Incorrect usage or order of send/recieve calls).

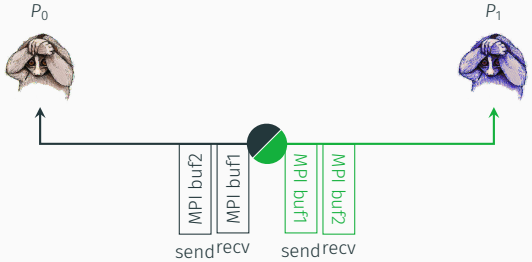


Figure 6: Deadlock possibility due to a 2-processes send-recieve cycle (Kind of depends on MPI implementation used!).

P2P Blocking comms: Stats

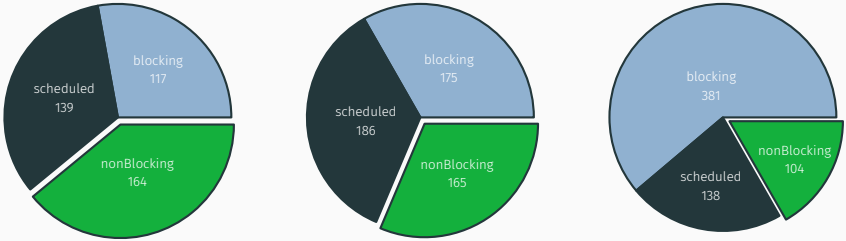


Figure 7: Frequency of usage for each type of OpenFOAM comms in OpenFOAM 10 (left), OpenFOAM v2012 (middle) and Foam-Extend 5 (right)

DISCLAIMER: Data generated pre-maturely; not suitable to compare forks irt. parallel performance -> Better compare history of the same fork instead.

```
include $(GENERAL_RULES)/mpi-rules
EXE_INC = $(PFLAGS) $(PINC) ...
LIB_LIBS = $(PLIBS) ...
```

NHR4CES NHR for Computational Engineering Science

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallelization in OpenFOAM for HPC Deployment

Point-to-Point Communications / Non-Blocking comms

Mohammed Elwardi Fadeli[1,2], Holger Marschall[1] and Christian Hasse[2]

April, 2024

[1] Mathematical Modeling and Analysis (MMA)
[2] Simulation of Reactive Thermo Fluid Systems (STFS)
Energy Conversion Group, NHR4CES - TU Darmstadt

P2P Non-Blocking comms

`Pstream::commsTypes::nonBlocking` (or just `Pstream::nonBlocking` in Foam Extend) does not wait until buffers are safe to re-use.

- Returns immediately (similar to ”async” in multi-threading context).
- The program must wait for the operation to complete (`Pstream::waitRequests`).
- It’s a form of piepline parallelism; i.e. Overlaps computation and communication.

- Avoids Deadlocks
- Minimizes idle time for MPI processes
- Helps skip unnecessary synchronisation

P2P Non-Blocking comms: An example

Communicate with a neighboring processor

```
1 include $(GENERAL_RULES)/mpi-rules
2 EXE_INC = $(PFLAGS) $(PINC) ...
3 LIB_LIBS = $(PLIBS) ...
```

Overlapping communication and computation

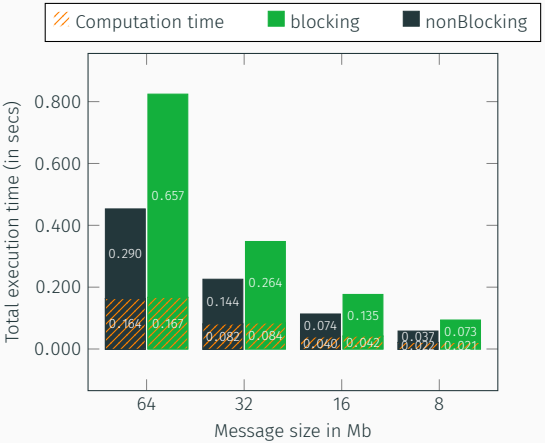
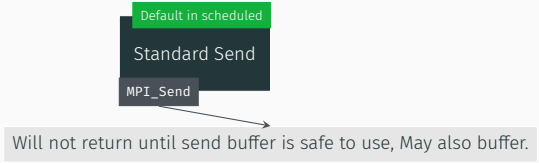
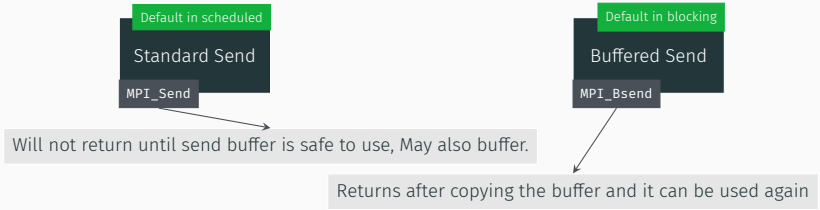


Figure 8: Effect of message size on overlapping communication and computation (4 processors, OpenMPI 4, OpenFOAM 8); Benchmark inspired from [2]

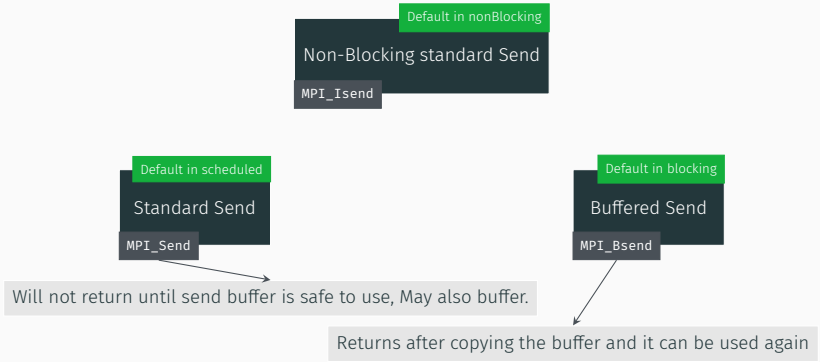
MPI send modes used in OpenFOAM code



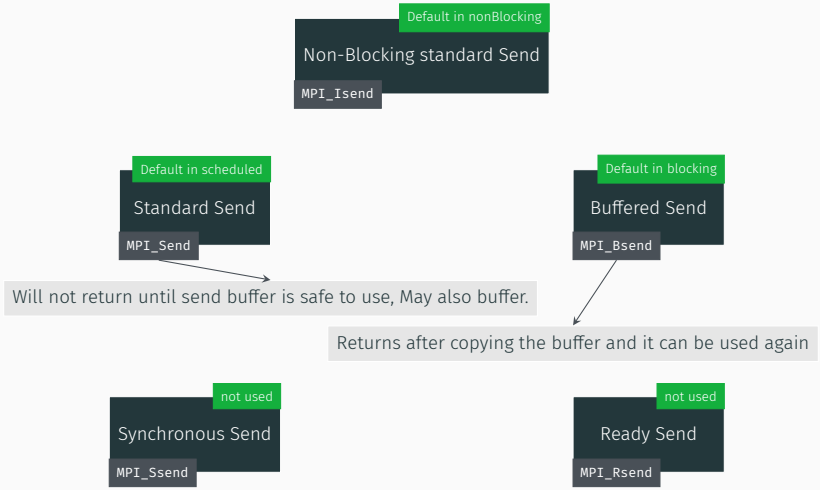
MPI send modes used in OpenFOAM code



MPI send modes used in OpenFOAM code



MPI send modes used in OpenFOAM code



NHR4CES NHR for Computational Engineering Science

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallelization in OpenFOAM for HPC Deployment

Collective Communications / General Introduction

Mohammed Elwardi Fadeli[1,2], Holger Marschall[1] and Christian Hasse[2]

April, 2024

[1] Mathematical Modeling and Analysis (MMA)
[2] Simulation of Reactive Thermo Fluid Systems (STFS)
Energy Conversion Group, NHR4CES - TU Darmstadt

Collective comms

When Two or more processes talk to each other.

- **All processes** call the same function with the same set of arguments.
- Although MPI-2 has non-blocking collective communications, OpenFOAM uses only the blocking variants.
- NOT a simple wrapper around P2P comms.
- Most collective algorithms are $log(nProcs)$

Collective comms

- OpenFOAM puts their interface in **static public methods** of `Pstream` class.
  - Major differences in the API across forks: (ESI and Foundation version) vs Foam Extend.
- Gather (all-to-one), Scatter (one-to-all), All-to-All variants of all-to-one ones.
- What OpenFOAM calls a ”reduce” is Gather+Scatter. This significantly differs from MPI’s concept of a reduce which is an all-to-one operation.
- MPI has also a ”Broadcast” and ”Barrier” but these are not used in OpenFOAM.

NHR4CES NHR for Computational Engineering Science

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallelization in OpenFOAM for HPC Deployment

Collective Communications / Common API

Mohammed Elwardi Fadeli[1,2], Holger Marschall[1] and Christian Hasse[2]

April, 2024

[1] Mathematical Modeling and Analysis (MMA)
[2] Simulation of Reactive Thermo Fluid Systems (STFS)
Energy Conversion Group, NHR4CES - TU Darmstadt

Collective comms: Gather (All-to-one)

Check how something is distributed over processors

```
1 include $(GENERAL_RULES)/mpi-rules
2 EXE_INC = $(PFLAGS) $(PINC) ...
3 LIB_LIBS = $(PLIBS) ...
```

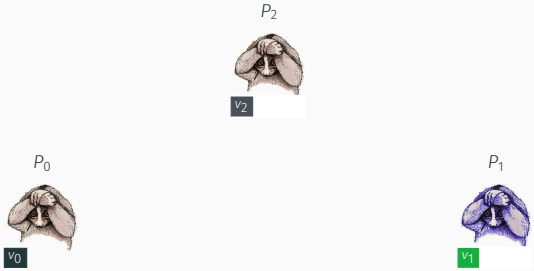


Figure 9: An example OpenFOAM gather operation (More like a MPI-reduce)

Collective comms: Gather (All-to-one)

Check how something is distributed over processors

```
1 include $(GENERAL_RULES)/mpi-rules
2 EXE_INC = $(PFLAGS) $(PINC) ...
3 LIB_LIBS = $(PLIBS) ...
```

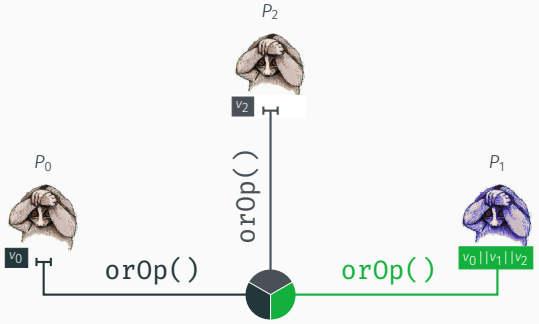


Figure 9: An example OpenFOAM gather operation (More like a MPI-reduce)

Collective comms: Gather (All-to-one)

Check how something is distributed over processors (List-like)

```
1 include $(GENERAL_RULES)/mpi-rules
2 EXE_INC = $(PFLAGS) $(PINC) ...
3 LIB_LIBS = $(PLIBS) ...
```

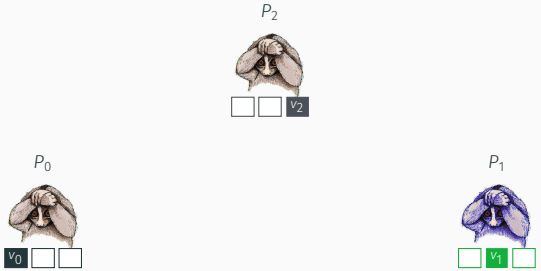


Figure 10: Example OpenFOAM gather operation on list items

Collective comms: Gather (All-to-one)

Check how something is distributed over processors (List-like)

```
1 include $(GENERAL_RULES)/mpi-rules
2 EXE_INC = $(PFLAGS) $(PINC) ...
3 LIB_LIBS = $(PLIBS) ...
```

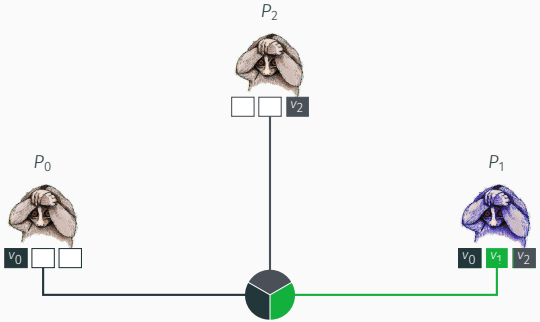


Figure 10: Example OpenFOAM gather operation on list items

Collective comms: Scatter (One-to-all)

Make processes know about something

```
1 include $(GENERAL_RULES)/mpi-rules
2 EXE_INC = $(PFLAGS) $(PINC) ...
3 LIB_LIBS = $(PLIBS) ...
```

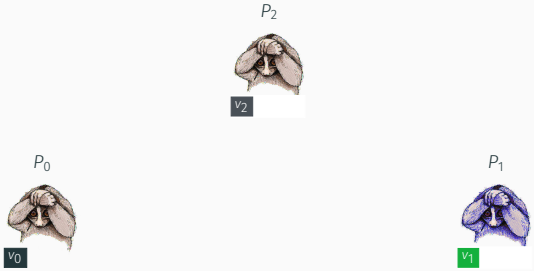


Figure 11: An example OpenFOAM scatter operation (More like a MPI-Bcast)

Collective comms: Scatter (One-to-all)

Make processes know about something

```
1 include $(GENERAL_RULES)/mpi-rules
2 EXE_INC = $(PFLAGS) $(PINC) ...
3 LIB_LIBS = $(PLIBS) ...
```

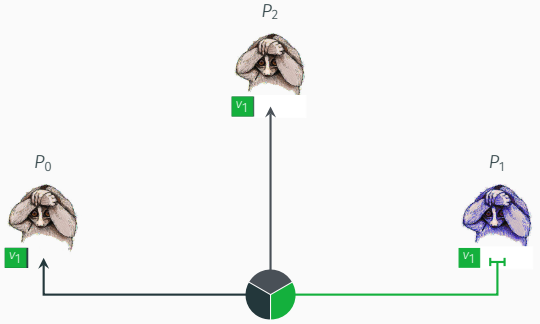


Figure 11: An example OpenFOAM scatter operation (More like a MPI-Bcast)

Collective comms: Scatter (One-to-all)

Make processes know about something (List-like)

```
1 include $(GENERAL_RULES)/mpi-rules
2 EXE_INC = $(PFLAGS) $(PINC) ...
3 LIB_LIBS = $(PLIBS) ...
```

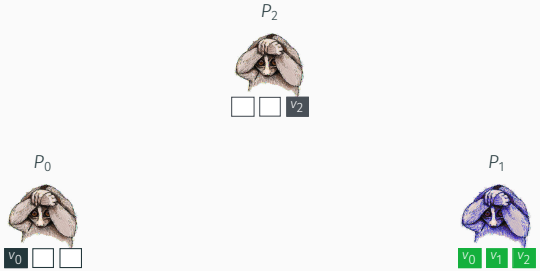


Figure 12: Example OpenFOAM scatter operation on list items

Collective comms: Scatter (One-to-all)

Make processes know about something (List-like)

```
1 include $(GENERAL_RULES)/mpi-rules
2 EXE_INC = $(PFLAGS) $(PINC) ...
3 LIB_LIBS = $(PLIBS) ...
```

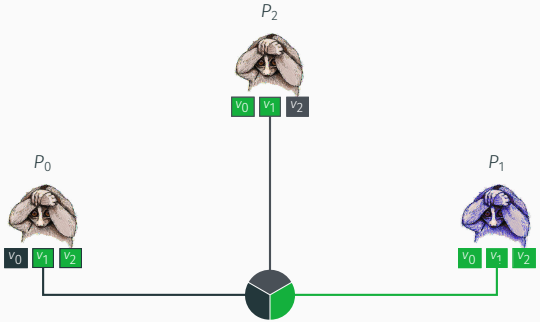


Figure 12: Example OpenFOAM scatter operation on list items

Collective comms: API stats

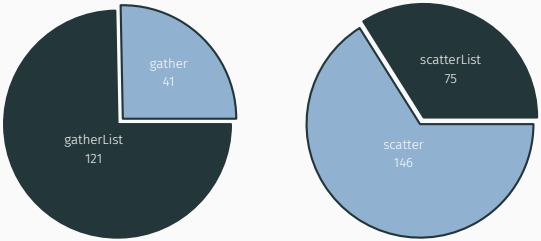


Figure 13: Frequency of usage for each API call of OpenFOAM collective comms (v2012)

- There are also some Fork-specific interface methods we won't discuss (eg. `Pstream::exchange`)

Collective comms: Reduce (All-to-All)

Do something with a var on all processors (eg. sum them up)

```
1 include $(GENERAL_RULES)/mpi-rules
2 EXE_INC = $(PFLAGS) $(PINC) ...
3 LIB_LIBS = $(PLIBS) ...
```

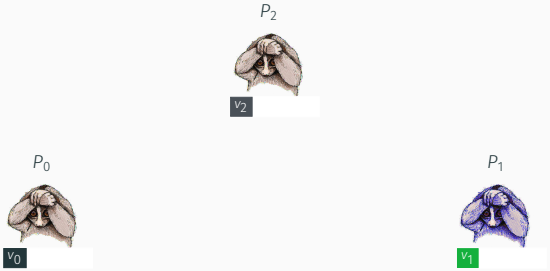


Figure 14: An example OpenFOAM reduce operation (MPI-Allreduce)

Collective comms: Reduce (All-to-All)

Do something with a var on all processors (eg. sum them up)

```
1 include $(GENERAL_RULES)/mpi-rules
2 EXE_INC = $(PFLAGS) $(PINC) ...
3 LIB_LIBS = $(PLIBS) ...
```

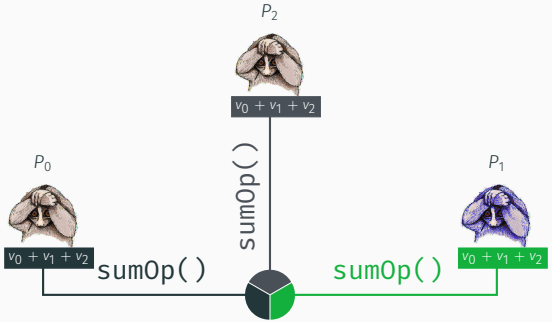


Figure 14: An example OpenFOAM reduce operation (MPI-Allreduce)

Oh, there is a reduce here!

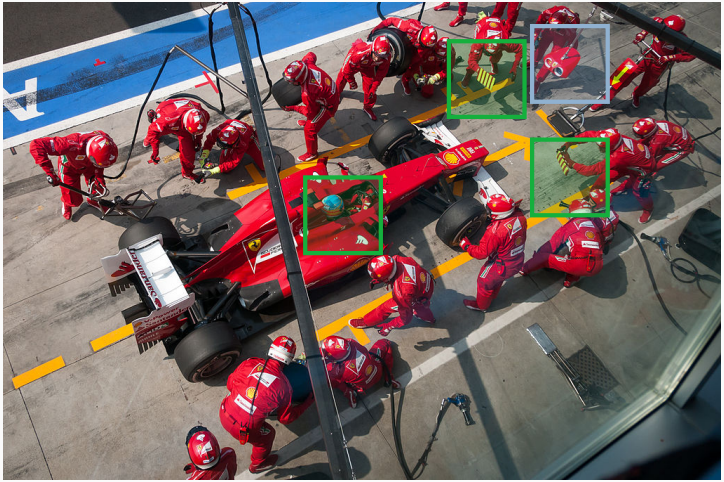


Figure 1: Parallel work during F1 Pit stops; CC BY 2.0, from commons.wikimedia.org

What if one of them just walks away before work is done?

Figure 1: Parallel work during F1 Pit stops;

Collective comms: Issues

You can still fall for endless loops if you're not careful!

Infinite loops due to early returns and collective comms

```
1 include $(GENERAL_RULES)/mpi-rules
2 EXE_INC = $(PFLAGS) $(PINC) ...
3 LIB_LIBS = $(PLIBS) ...
```

NHR4CES NHR for Computational Engineering Science

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallelization in OpenFOAM for HPC Deployment

How do I send my own data?

Mohammed Elwardi Fadeli[1,2], Holger Marschall[1] and Christian Hasse[2]

April, 2024

[1] Mathematical Modeling and Analysis (MMA)
[2] Simulation of Reactive Thermo Fluid Systems (STFS)
Energy Conversion Group, NHR4CES - TU Darmstadt

Sending around a simple struct

Say we have something like this:

Push an edge from master to All

```
1 include $(GENERAL_RULES)/mpi-rules
2 EXE_INC = $(PFLAGS) $(PINC) ...
3 LIB_LIBS = $(PLIBS) ...
```

error: No match for operator<<(OPstream&,Edge&)
error: No match for operator>>(IPstream&,Edge&)

Sending around a simple struct

So, Edges can't be communicated as MPI messages, provide the necessary serialization and de-serialization operators:

Stream Operators for Edge class

```
1 include $(GENERAL_RULES)/mpi-rules
2 EXE_INC = $(PFLAGS) $(PINC) ...
3 LIB_LIBS = $(PLIBS) ...
```

```
Info << ej;
"16 5.2"
```

How about sending a list of custom objects?

Gathering info about Edges on all procs

```
1 include $(GENERAL_RULES)/mpi-rules
2 EXE_INC = $(PFLAGS) $(PINC) ...
3 LIB_LIBS = $(PLIBS) ...
```

Compiles, and works as expected if Edge is modified so it can be put in a List

A Better way: Make Edge a child of one of the OpenFOAM classes

Better ways to define an Edge

```
1 include $(GENERAL_RULES)/mpi-rules
2 EXE_INC = $(PFLAGS) $(PINC) ...
3 LIB_LIBS = $(PLIBS) ...
```

What if the struct has a pointer?

Easiest solution -> Follow all pointers

```
1 include $(GENERAL_RULES)/mpi-rules
2 EXE_INC = $(PFLAGS) $(PINC) ...
3 LIB_LIBS = $(PLIBS) ...
```

Hoping that all members are deep-copyable
* Code for illustration only, do not use raw pointers

What if the struct has a reference to the mesh?

- Unlike pointers, references in C++ need to be initialized when declared (in constructors for members of a reference type).
- You'll most likely have a reference to the mesh → It's good practice.
- The mesh is special because we know that it's partitioned. And we want our objects to use the mesh on the other process when we send them over!

The solution includes:

- Switch to `LinkedList` instead of random-access ones.
  → Why? Because they allow for passing custom constructor arguments.
- The Edge will have to get a new construction function (usually something nested in a sub-class `Edge::iNew::operator()`)
- We will explore this in more detail during the hands-on sessions

NHR4CES NHR for Computational Engineering Science

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallelization in OpenFOAM for HPC Deployment

Application examples & advanced topics

Mohammed Elwardi Fadeli[1,2], Holger Marschall[1] and Christian Hasse[2]

April, 2024

[1] Mathematical Modeling and Analysis (MMA)
[2] Simulation of Reactive Thermo Fluid Systems (STFS)
Energy Conversion Group, NHR4CES - TU Darmstadt

Solving PDEs over decomposed domains (P2P comms)

General Transport Equation for a physical transport property

$$\partial_t \phi + \nabla \cdot (\phi \mathbf{u}) - \nabla \cdot (\Gamma \nabla \phi) = S_\phi(\phi)$$

Discretized form (Finite Volume notation)

$$[\![\partial_t [\phi]]\!] + [\![\nabla \cdot \left(F [\phi]_{f(F,S,\gamma)}\right)]\!] - [\![\nabla \cdot \left(\Gamma_f \nabla [\phi]\right)]\!] = [\![S_l [\phi]]\!].$$

- Receive neighbour values from neighbouring processor.
- Send face cell values from local domain to neighouring processor
- Interpolate to processor patch faces

Adaptive Mesh Refinement on polyhedral meshes

1. Refine each processor's part of the mesh, but we need to keep the global cell count under a certain value:

Reduce nAddCells or nTotalAddCells?

```
1 include $(GENERAL_RULES)/mpi-rules
2 EXE_INC = $(PFLAGS) $(PINC) ...
3 LIB_LIBS = $(PLIBS) ...
```

Adaptive Mesh Refinement on polyhedral meshes

2. To decide on whether to refine cells at processor boundaries, we need cell levels from the other side:

ownLevel holds neiLevel after swapping!

```
1 include $(GENERAL_RULES)/mpi-rules
2 EXE_INC = $(PFLAGS) $(PINC) ...
3 LIB_LIBS = $(PLIBS) ...
```

Advanced topics

The need for Load Balancing in AMR settings

- AMR operations tend to unbalance cell count distribution accross processors
- Using Blocking comms means more idle process time
  - Non-Blocking are not a solution.
  - Spending some time on rebalancing the mesh is.
- Naturally, load balancing itself involves parallel communication!

Licensing

Questions?

Compile and link against MPI implementations

Compiler wrappers are your best friends!

Grab correct compiler/linker flags

```
1 include $(GENERAL_RULES)/mpi-rules
2 EXE_INC = $(PFLAGS) $(PINC) ...
3 LIB_LIBS = $(PLIBS) ...
```

-I/usr/lib/x86_64-linux-gnu/openmpi/include/openmpi ...
-pthread -L/usr/lib/x86_64-linux-gnu/openmpi/lib ...

OpenFOAM environment autmatically figures things out for you:

Typical Make/options file for the ESI fork

```
1 include $(GENERAL_RULES)/mpi-rules
2 EXE_INC = $(PFLAGS) $(PINC) ...
3 LIB_LIBS = $(PLIBS) ...
```

Miscellaneous

- MPI standards: Blocking send can be used with a Non-blocking receive, and vice-versa
- But OpenFOAM wrapping makes it ”non-trivial” to get it to work
- You can still use MPI API directly, eg. if you need one-sided communication.
- Overlapping computation and communication for non-blocking calls is implemented on the MPI side, so, put your computations after the recieve call.

Sources and further reading i

[1] C. Augustine. ***Introduction to Parallel Programming with MPI and OpenMP.*** Source of the great 'pit stops' analogy. Oct. 2018. URL: https://princetonuniversity.github.io/PUbootcamp/sessions/parallel-programming/Intro_PP_bootcamp_2018.pdf.

[2] Fabio Baruffa. ***Improve MPI Performance by Hiding Latency.*** July 2020. URL: https://www.intel.com/content/www/us/en/developer/articles/technical/overlap-computation-communication-hpc-applications.html.

Sources and further reading ii

[3] Pavanakumar Mohanamuraly, Jan Christian Huckelheim, and Jens-Dominik Mueller. **"Hybrid Parallelisation of an Algorithmically Differentiated Adjoint Solver".** In: *Proceedings of the VII European Congress on Computational Methods in Applied Sciences and Engineering (ECCOMAS Congress 2016)*. Institute of Structural Analysis and Antiseismic Research School of Civil Engineering National Technical University of Athens (NTUA) Greece, 2016. DOI: 10.7712/100016.1884.10290. URL: https://doi.org/10.7712/100016.1884.10290.

[4] B. Steinbusch. ***Introduction to Parallel Programming with MPI and OpenMP.*** Mar. 2021. URL: https://www.fz-juelich.de/SharedDocs/Downloads/IAS/JSC/EN/slides/mpi/mpi-openmp-handouts.pdf?__blob=publicationFile.

[5] EuroCC National Competence Center Sweden. ***Intermediate MPI.*** May 2022. URL: https://enccs.github.io/intermediate-mpi/.