MPI and OpenMP in Scientific Software Development

Day 1

Maxim Masterov
SURF, Amsterdam, The Netherlands
27-29.05.2024

Contents

Overview

Day 1

- **@16:00 Dr. Matthias Möller (TU Delft)**
- MPI (recap)
- Connecting to Snellius supercomputer
- Jacobi solver
- Performance analysis & debugging tools
- Hands-on

Day 2

- **@12:00 Dr. Nicola Spallanzani (MaX CoE)**
- MPI communications
- Domain decomposition
- MPI topologies
- Hands-on

Day 3

- **@12:00 Dr. Remco Havenith (RUG)**
- MPI IO
- Advanced OpenMP
- Hybrid programming
- Hands-on

Contents

Overview

Day 1

- **@16:00 Dr. Matthias Möller (TU Delft)**
- MPI (recap)
- Connecting to Snellius supercomputer
- Jacobi solver
- Performance analysis & debugging tools
- Hands-on

Day 2

- **@12:00 Dr. Nicola Spallanzani (MaX CoE)**
- MPI communications
- Domain decomposition
- MPI topologies
- Hands-on

Day 3

- **@12:00 Dr. Remco Havenith (RUG)**
- MPI IO
- Advanced OpenMP
- Hybrid programming
- Hands-on

Disclaimer

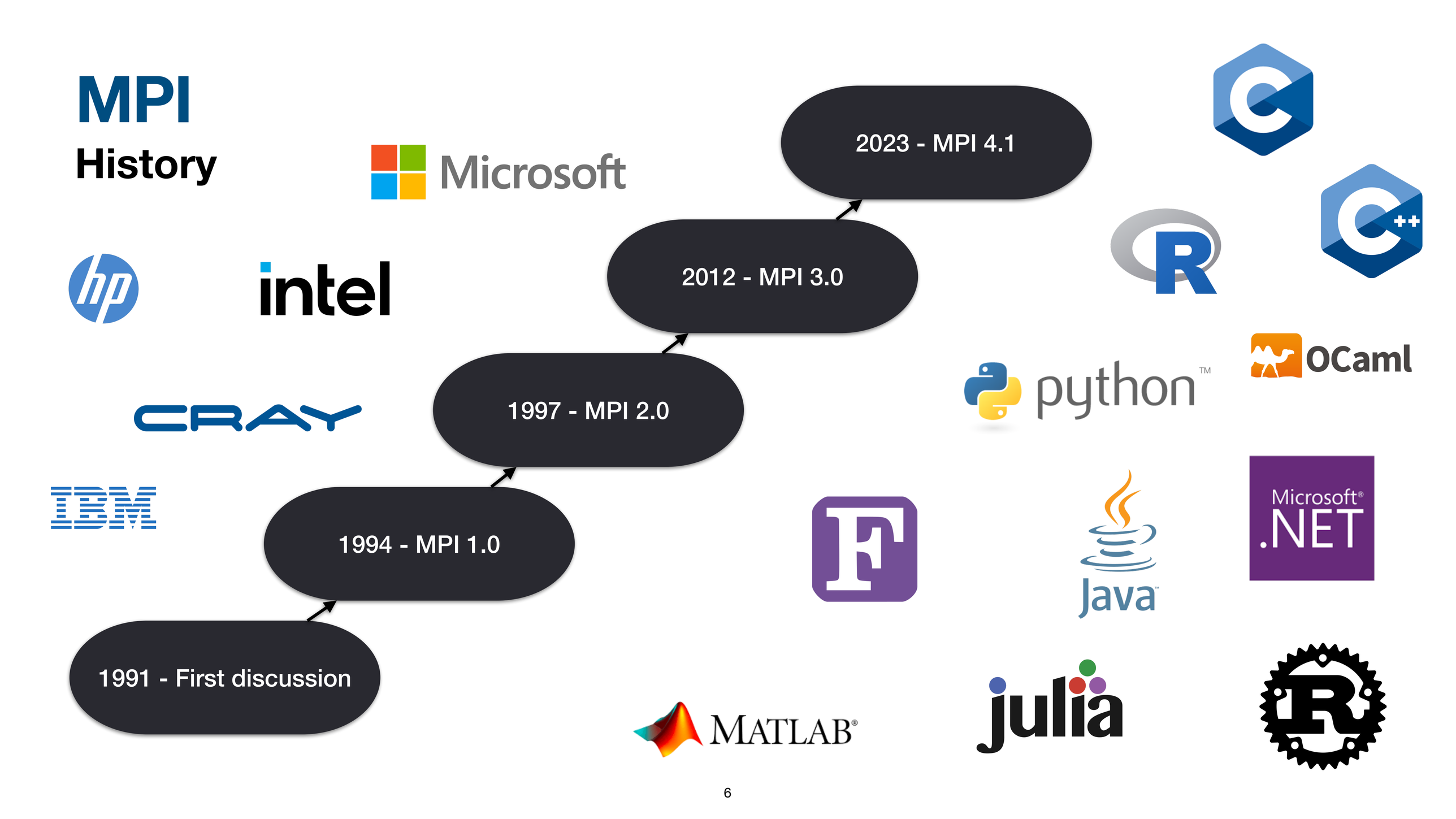
Just a disclaimer

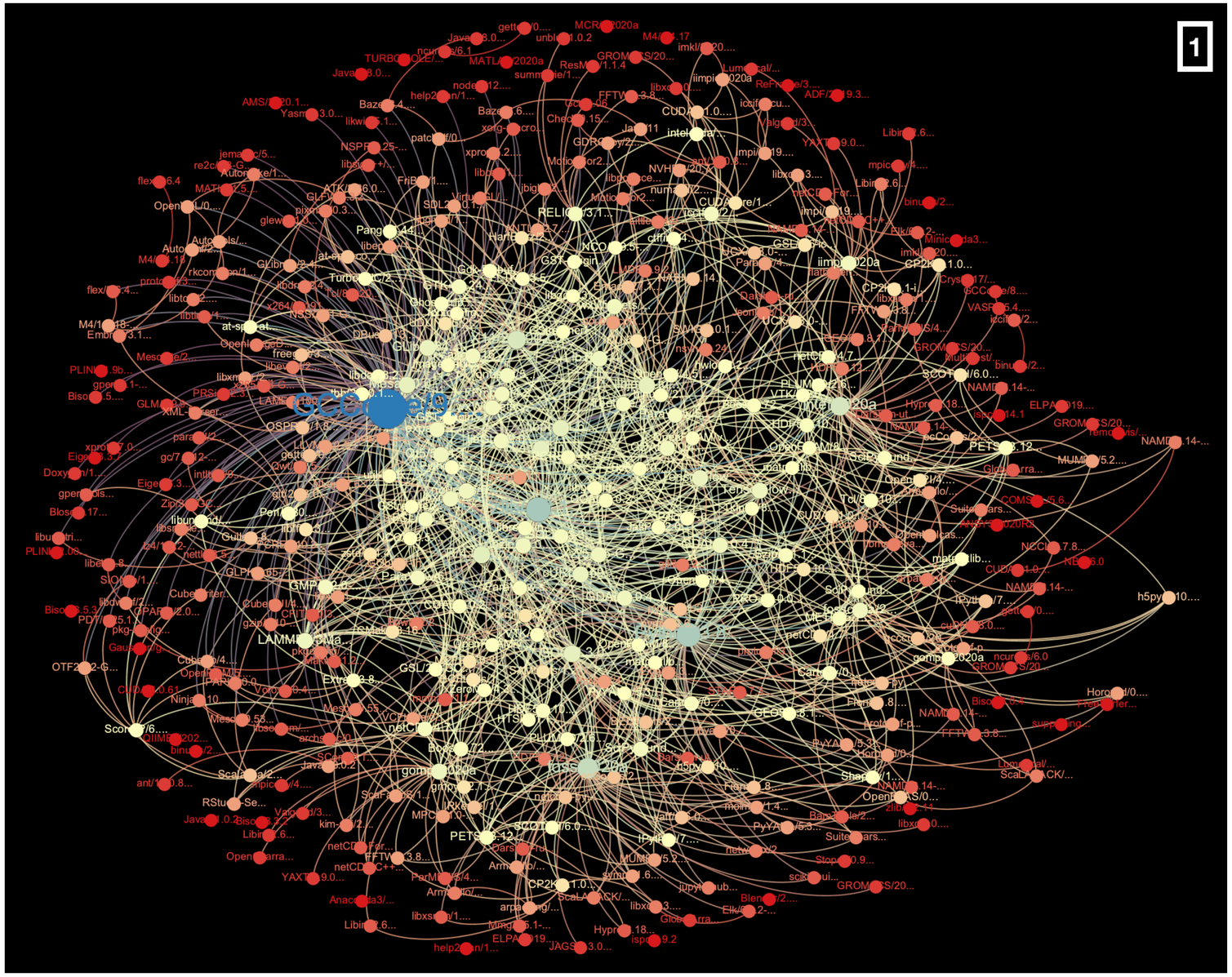
- This is **not the 2nd part** of the course for the “beginners”.
- The whole course is about **hands-on experience**. Please, pay attention, you will write a lot of code.
- All **codes are written in C++**. However, it’s sufficient to know C to do assignments.
- If you are only familiar with **FORTRAN**, please, **check out** some cheat sheets on the Internet.
- All **materials will be available** after the course under MIT license. You can use them or distribute, just keep the references :)
- This is the **3rd time** this course is been **offered**. We might have missed something, made typos etc. Your feedback is most appreciated.

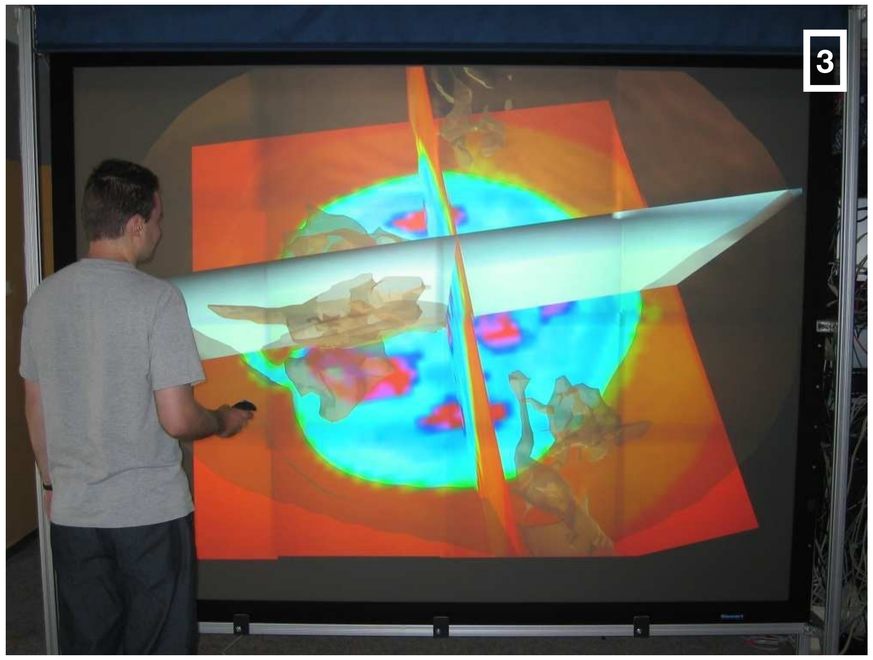
Message Passing Interface

MPI

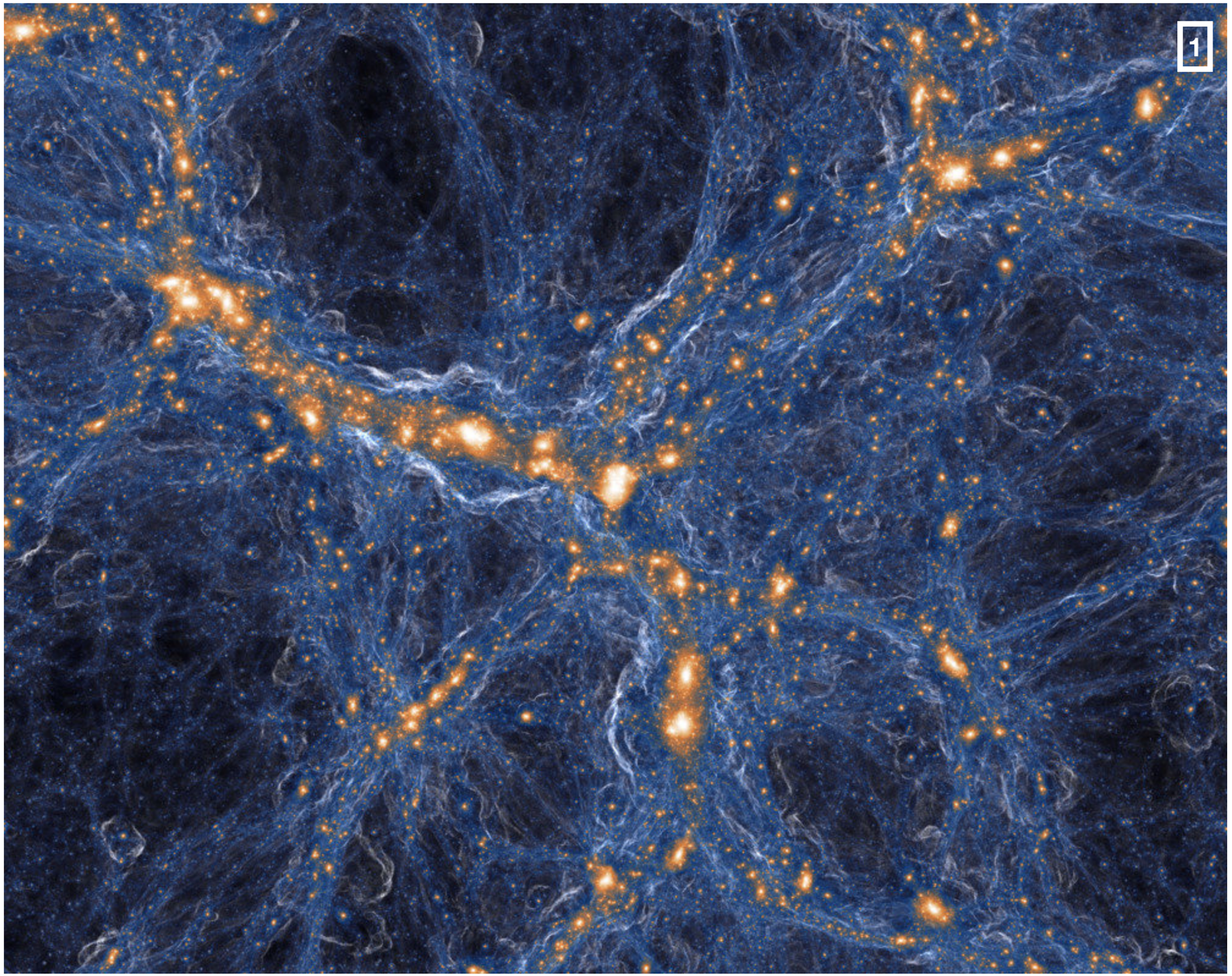
History

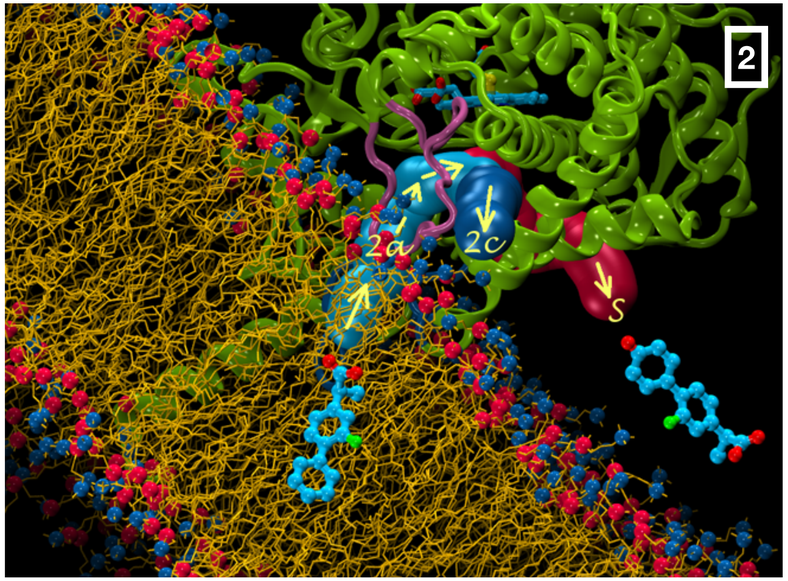

2023 - MPI 4.1
2012 - MPI 3.0
1997 - MPI 2.0
1994 - MPI 1.0
1991 - First discussion
Microsoft
intel
CRAY
IBM
python
OCaml
Java
Microsoft .NET
MATLAB
julia

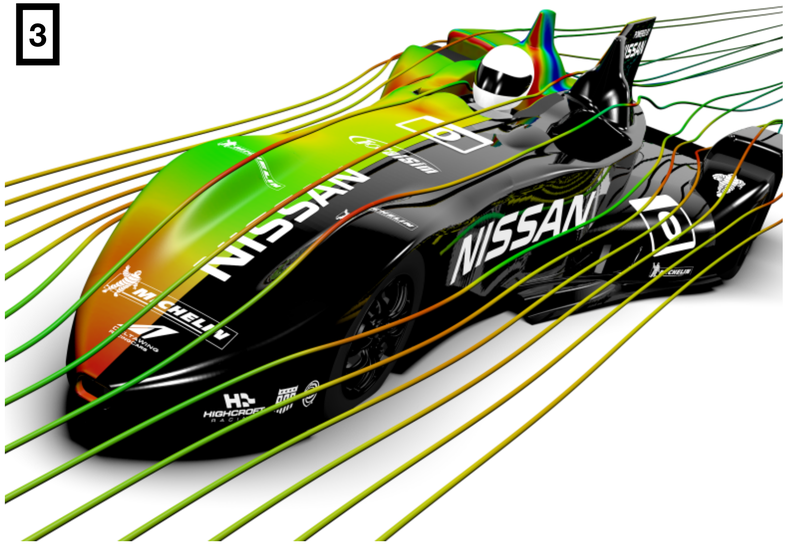




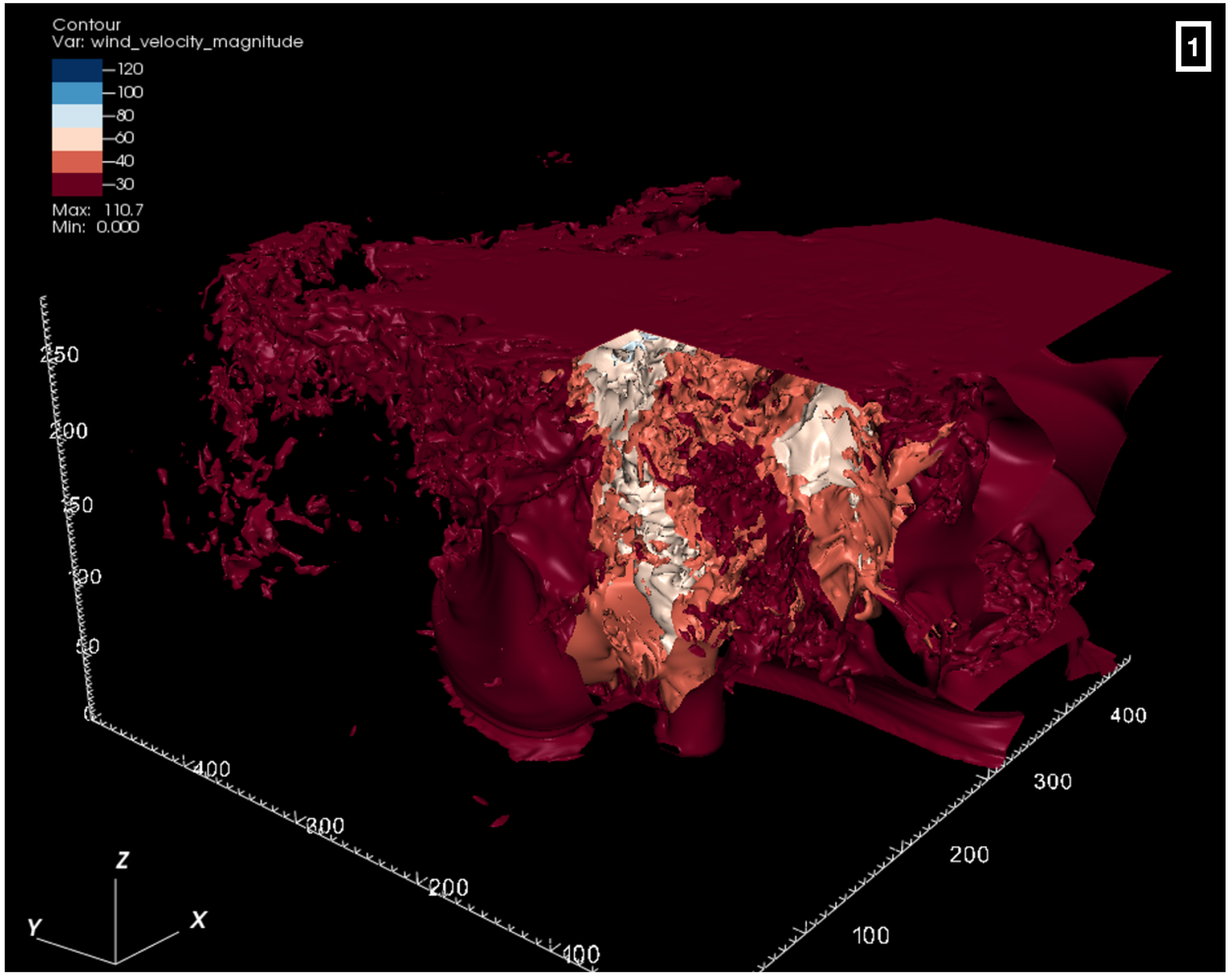
1. Graph of modules installed on Cartesius. Visualised with Gephi.
2. An example of a mobile game. Doolan, Daniel & Tabirca, Sabin. (2007). Bluetooth Gaming with the Mobile Message Passing Interface (MMPI).
3. FlowVR. Visualisation software. Jérémie Allard, Bruno Raffin (2005), A Shader-Based Parallel Rendering Framework, IEEE Visualization 2005 conference proceedings

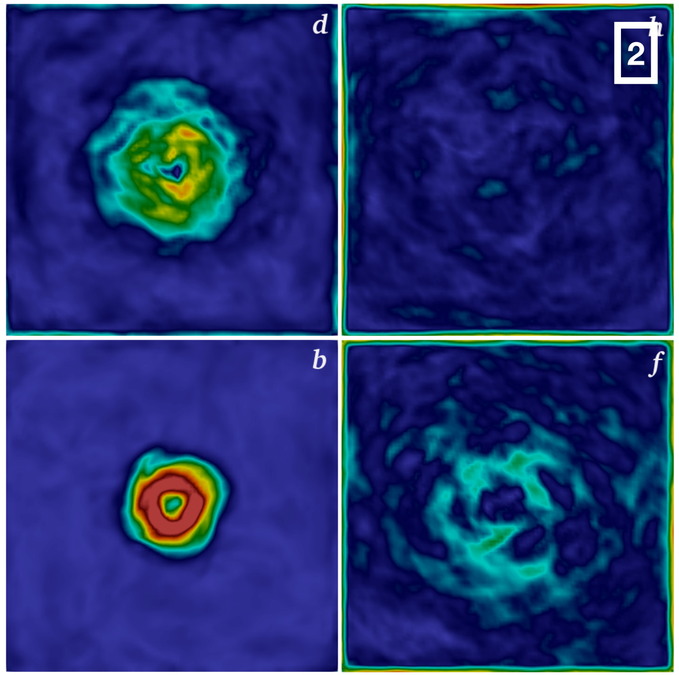


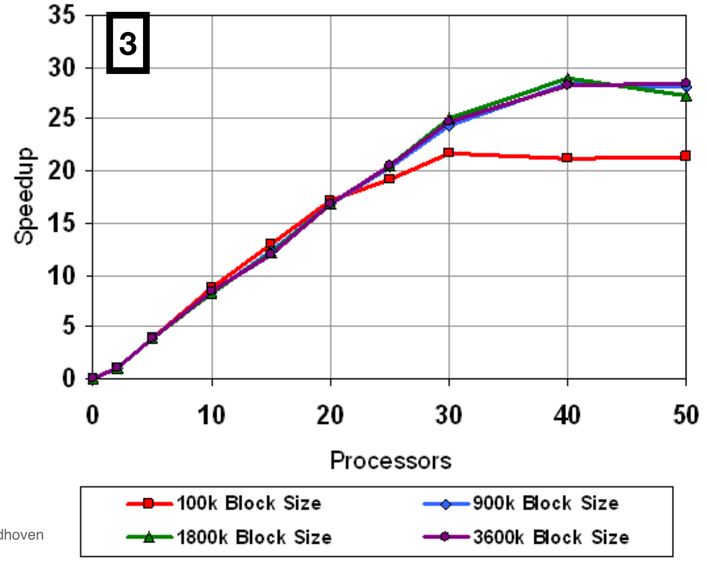




1. IllustrisTNG. Astrophysics model. https://phys.org/news/2018-02-astrophysicists-illustristng-advanced-universe-kind.html
2. VMD. Visualisation software. (2011) *PLoS Computational Biology* Issue Image I Vol. 7(8) August 2011. PLoS Comput Biol 7(8)
3. TotalSim. CFD software. https://www.totalsimulation.co.uk/digital-race-car/







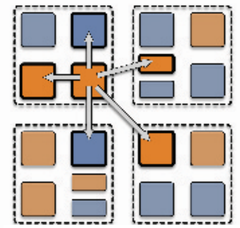
1. VisIt. Visualisation software. https://www.technologynetworks.com/informatics/articles/visit-application-speeds-visualization-workloads-empowering-global-research-305352
2. FoxBerry. Multiphase CFD software. Visualized with ParaView. Masterov, M. V. (2019). Towards industrial-scale bubble columns: the development and application of the high performance computing framework. Technische Universiteit Eindhoven
3. MPIBZIP2. Parallel compressing library. http://compression.ca/mpibzip2/

MPI

Alternatives

Regent

- The goal:
  - Raise the level of abstraction
  - Create domain-specific framework
- Issues, compared to MPI+X:
  - Less known
  - Less mature
- Look at PAW-ATM: https://sourceryinstitute.github.io/PAW/

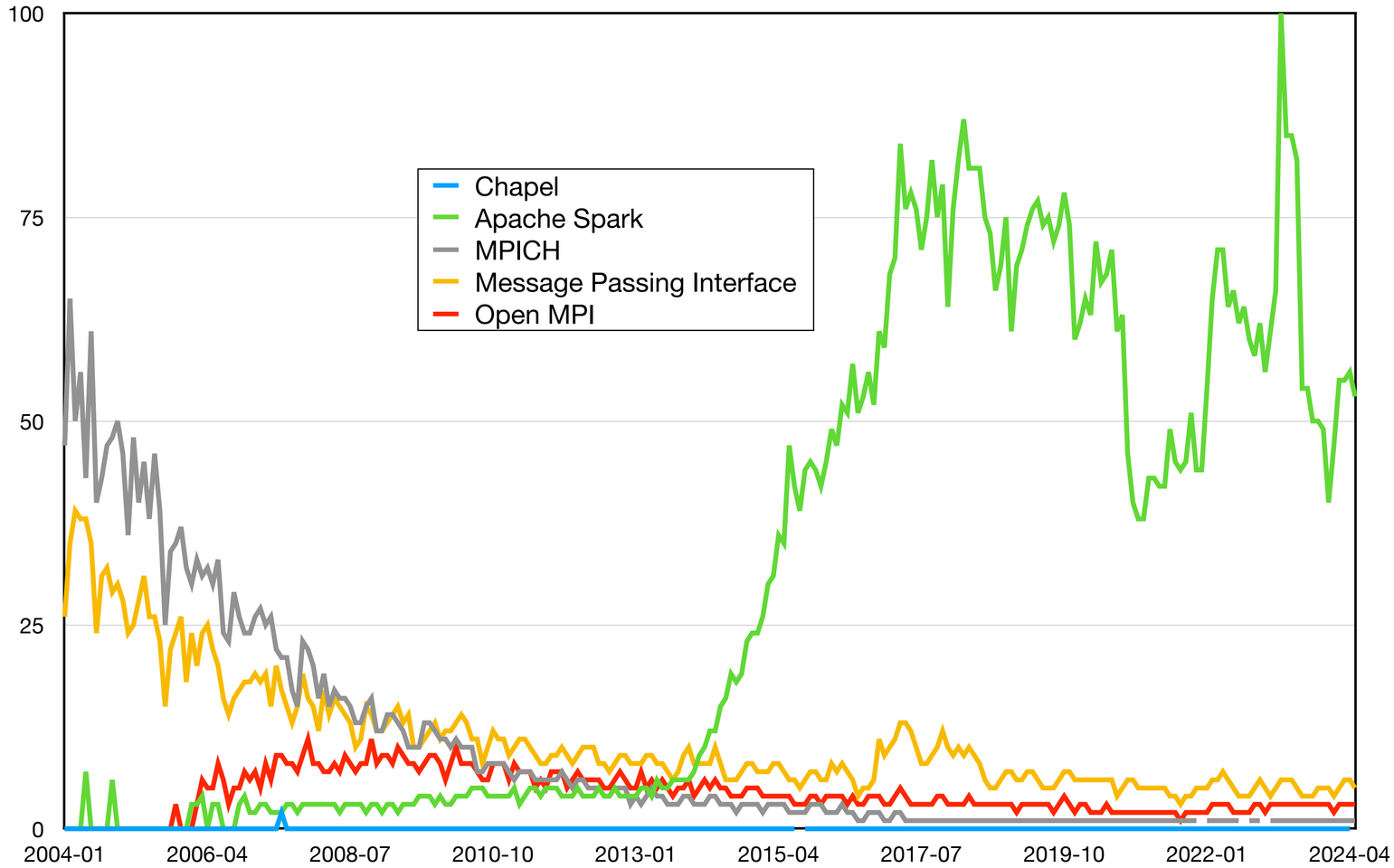


Charm++

MPI

Alternatives

Google search. Interest over time.[1]



1. https://trends.google.com/trends/explore?cat=5&date=all&q=%2Fm%2F0gnckq,%2Fm%2F0ndhxqz,%2Fm%2F03z8q2,%2Fm%2F01gb80,%2Fm%2F07yb2g

MPI

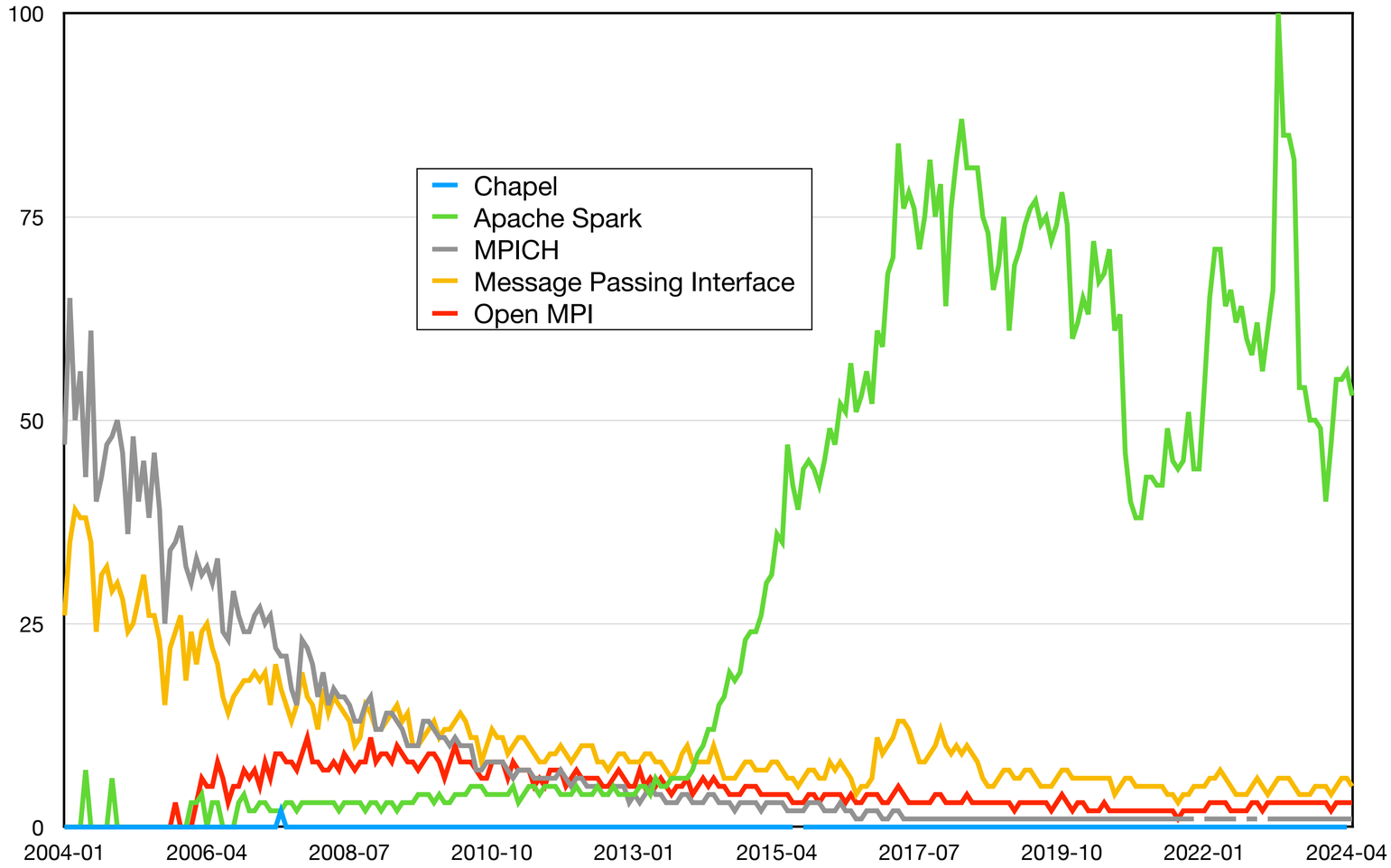
Alternatives

- New hardware —> new software requirements —> new standards
- MPI remains the #1 standard for scientific applications, but there are some competitors

Example, 1D diffusion[2]

Implementation	LoC
MPI+Python	65
Spark+Python	34
Chapel	26

Google search. Interest over time.[1]



1. https://trends.google.com/trends/explore?cat=5&date=all&q=%2Fm%2F0gnckq,%2Fm%2F0ndhxqz,%2Fm%2F03z8q2,%2Fm%2F01gb80,%2Fm%2F07yb2g
2. https://www.dursi.ca/post/hpc-is-dying-and-mpi-is-killing-it.html#mpi

MPI

Alternatives

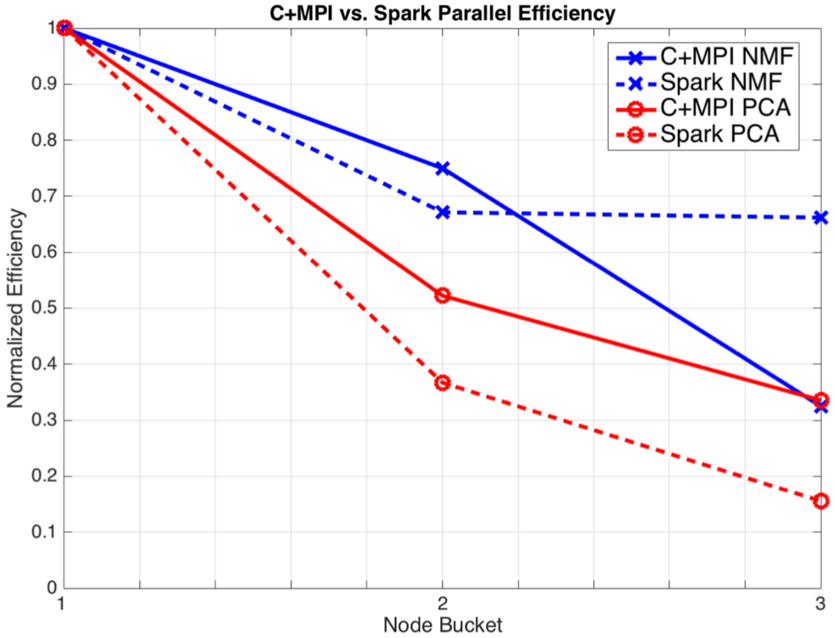


Figure 4: Comparison of parallel efficiency for C+MPI and Spark. The x-axis label "Node Bucket" refers to the node counts. For NMF these are 50, 100, and 300 nodes (left to right) and 100, 300, and 500 nodes for PCA. For both algorithms, efficiency is measured relatively to the performance at the smallest node count.

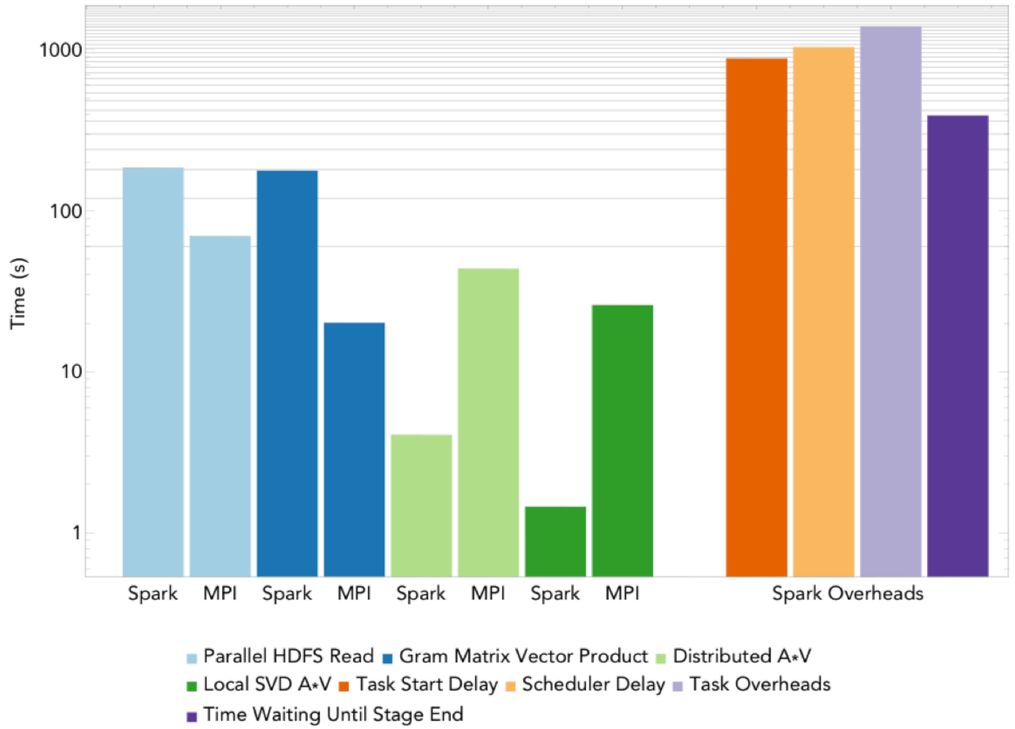
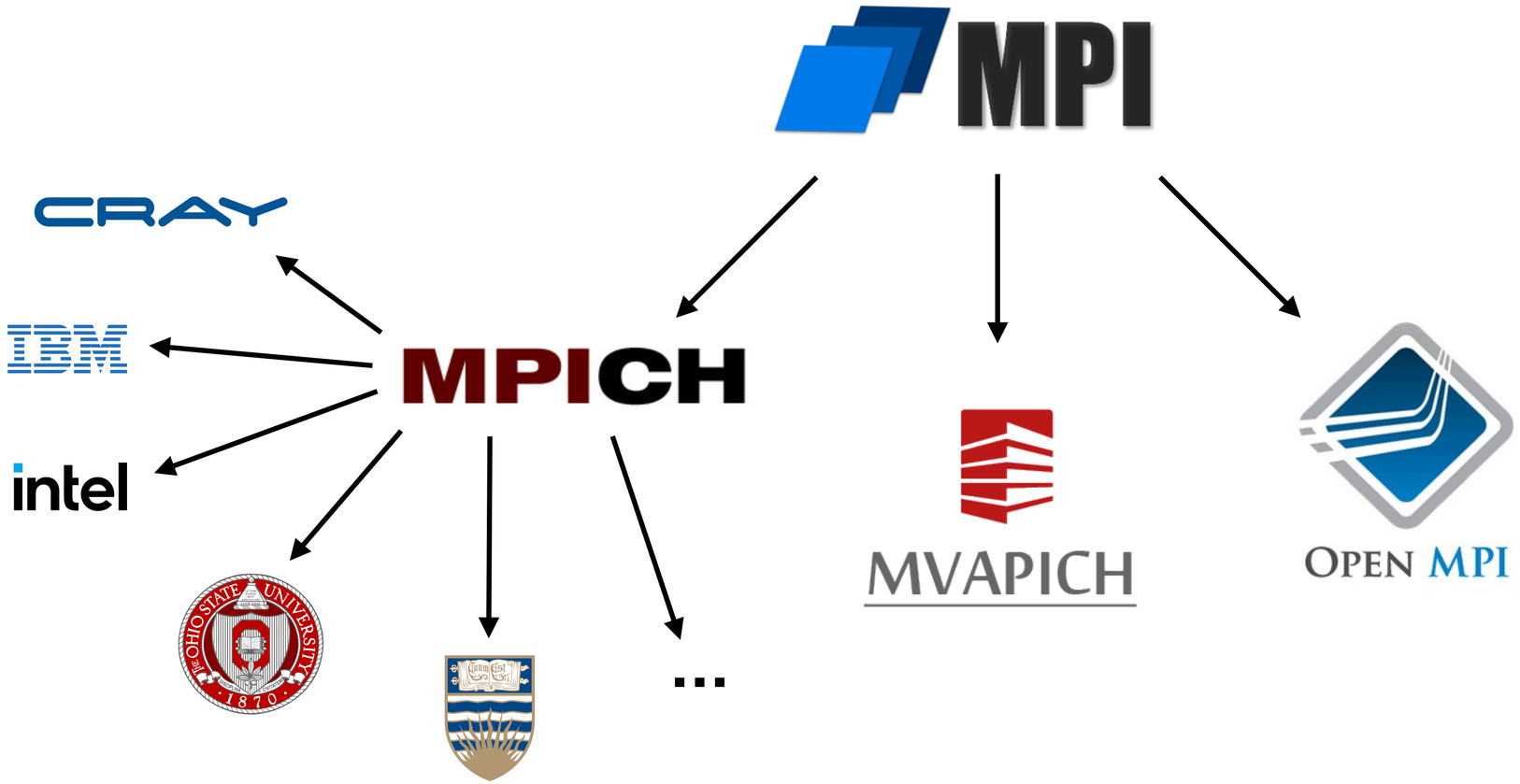


Figure 6: Running time comparison of the Spark and MPI implementations of PCA on the 16TB Atmosphere matrix. Each bin depicts the sum, over all stage, of the time spent in that bin by the average task within a stage.

1. A. Gittens *et al.*, "Matrix factorizations at scale: A comparison of scientific data analytics in spark and C+MPI using three case studies," *2016 IEEE International Conference on Big Data (Big Data)*, 2016, pp. 204-213

MPI

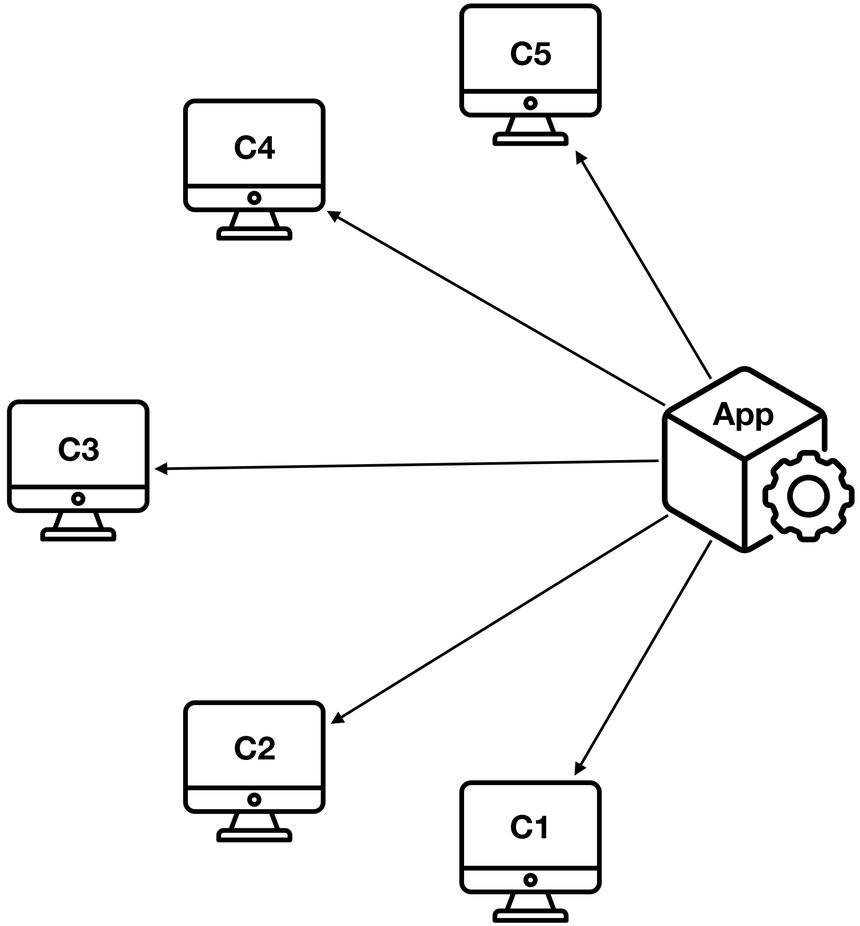
Versions



MPI

Idea

- Distribution of the workload across multiple processes

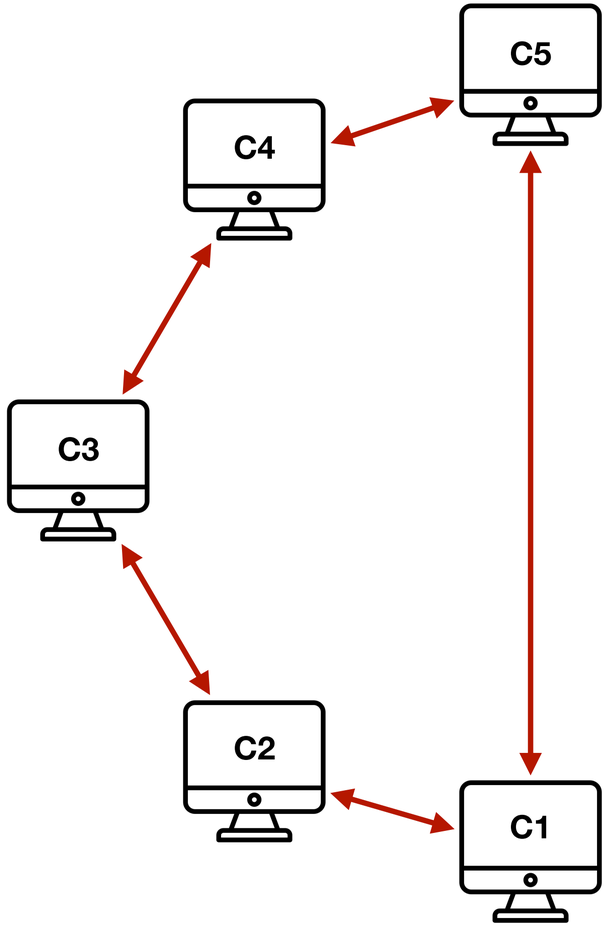


1. https://www.flaticon.com/free-icon/computer_2344269
2. https://www.flaticon.com/free-icon/3d-modeling_4229105

MPI

Idea

- Distribution of the workload across multiple processes
- Main influence from:
  - the network

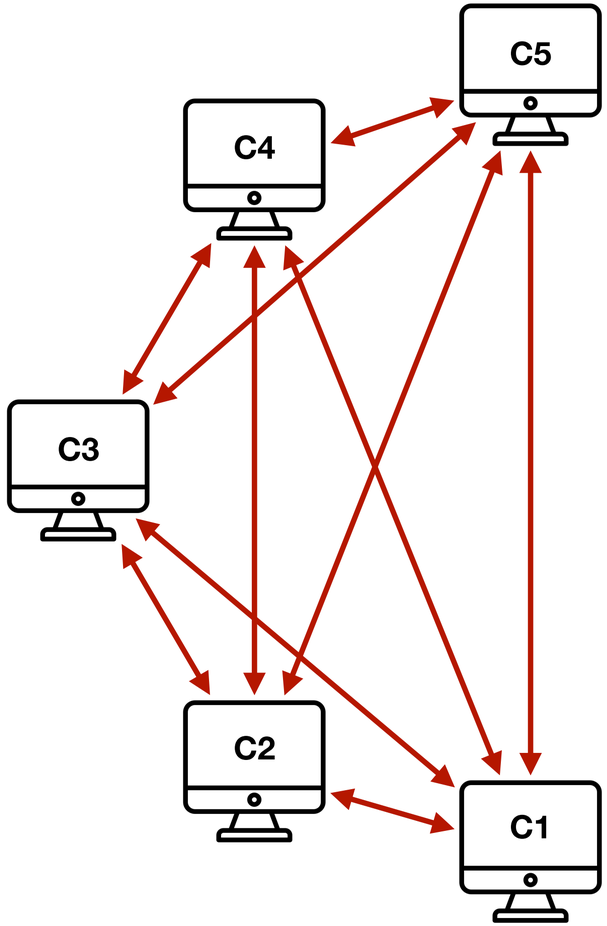


1. https://www.flaticon.com/free-icon/computer_2344269

MPI

Idea

- Distribution of the workload across multiple processes
- Main influence from:
  - the network

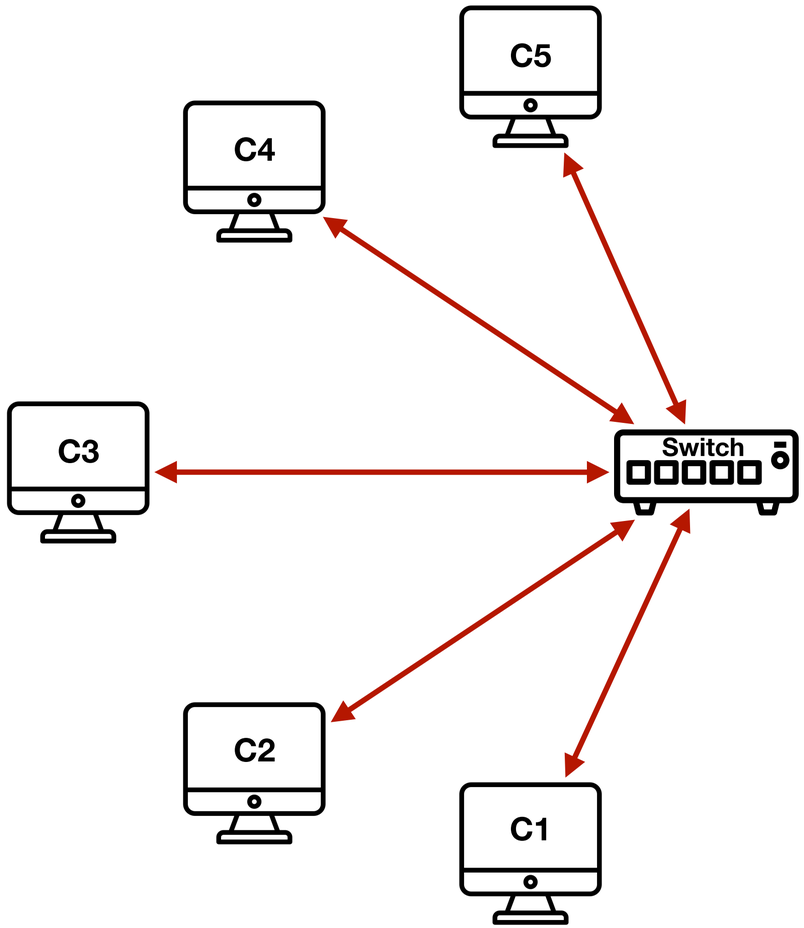


1. https://www.flaticon.com/free-icon/computer_2344269

MPI

Idea

- Distribution of the workload across multiple processes
- Main influence from:
  - the network



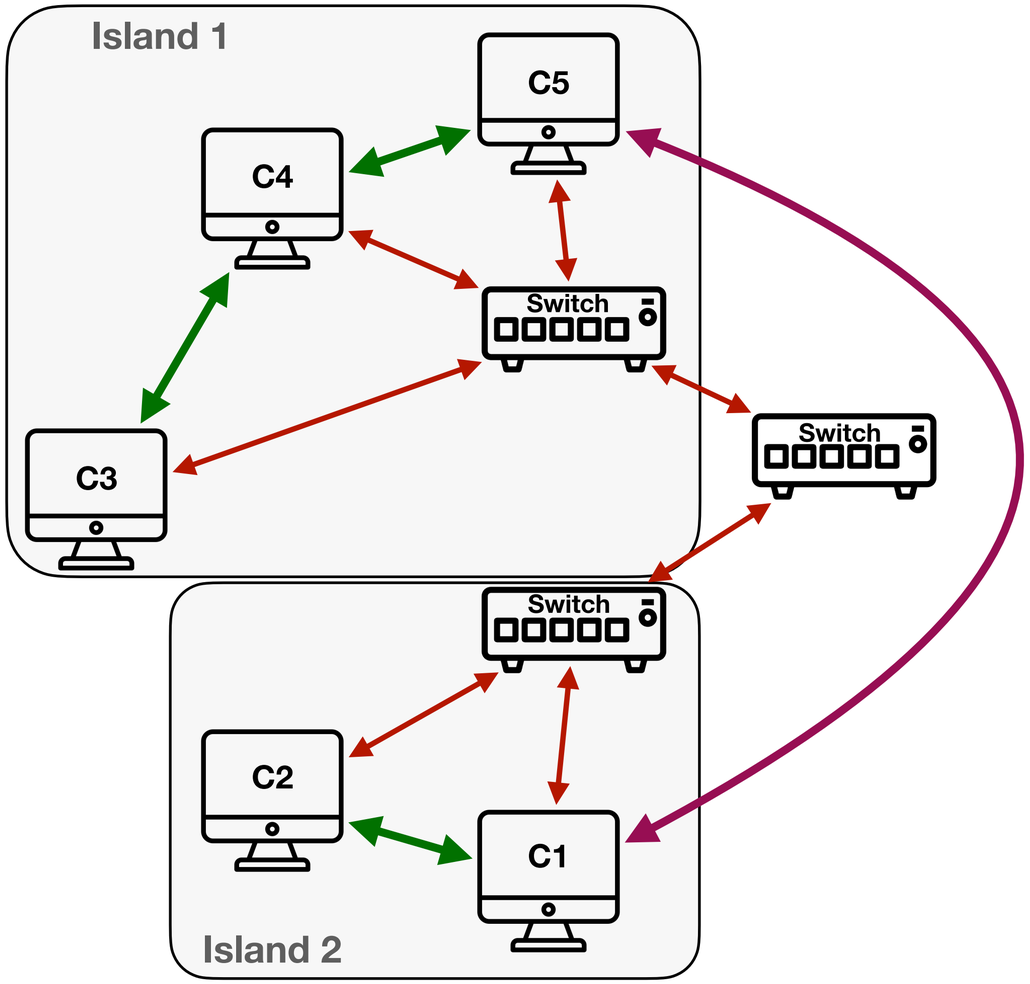
1. https://www.flaticon.com/free-icon/computer_2344269
2. https://www.flaticon.com/free-icon/switch_1089006

MPI

Idea

- Distribution of the workload across multiple processes
- Main influence from:
  - the network
  - the communication frequency
  - the message size
  - the file system

Will come back to it later

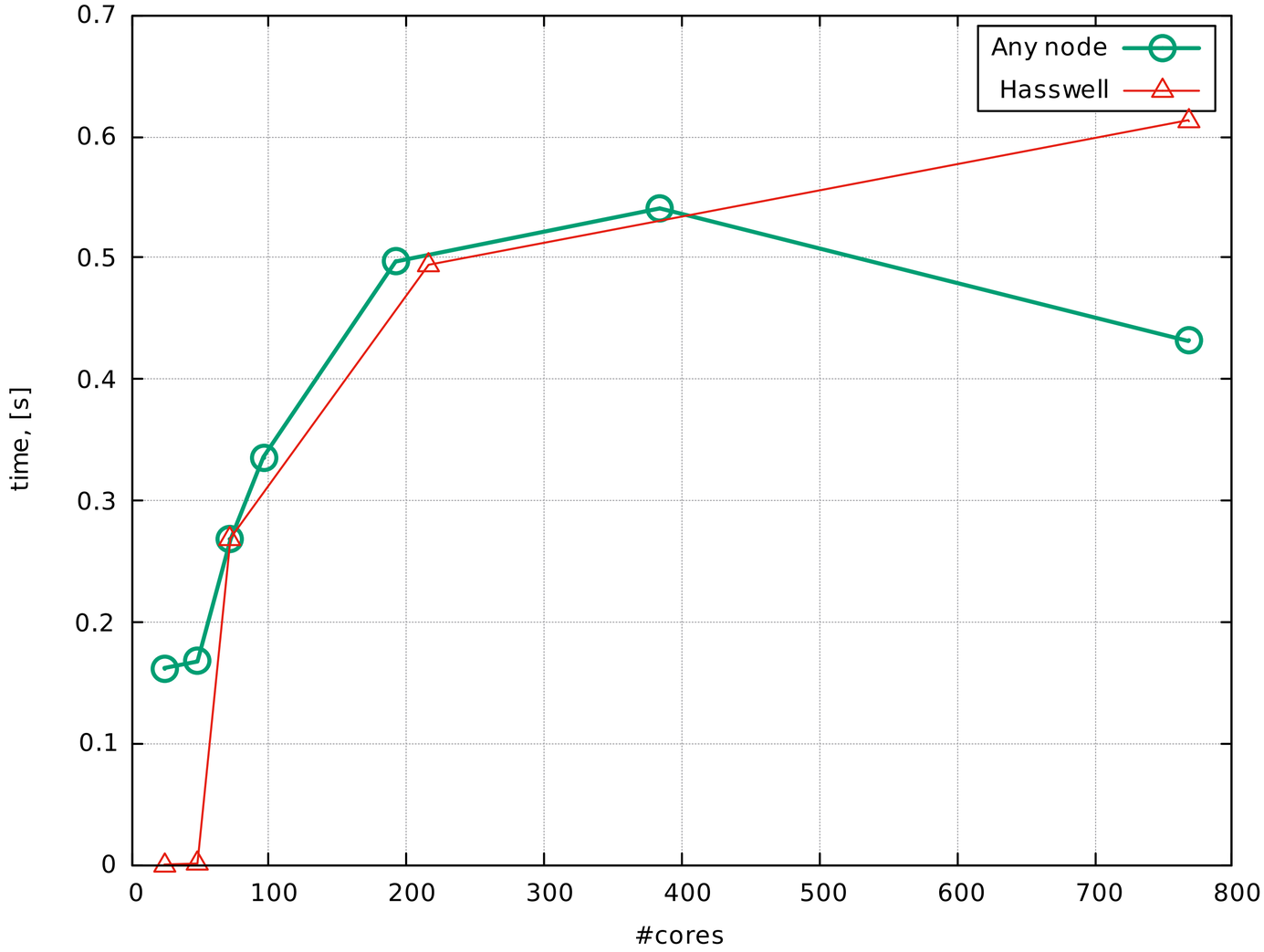


1. https://www.flaticon.com/free-icon/computer_2344269
2. https://www.flaticon.com/free-icon/switch_1089006

MPI

Message size

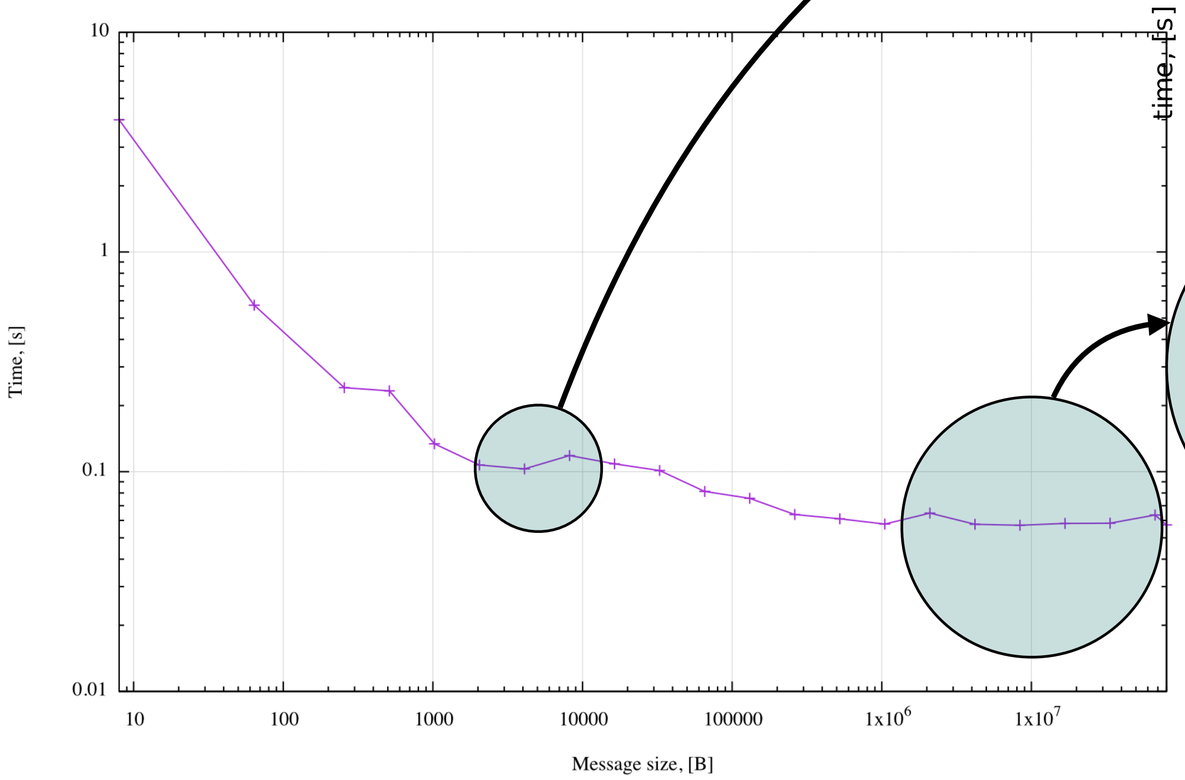
- Communication time (Cartesius)
- CFD code, FoxBerry
- Structured grid with 27M DoFs
- Double precision FP
- Non-blocking communication
- Buffered messaged
- The message size at 700 cores is ~3KB

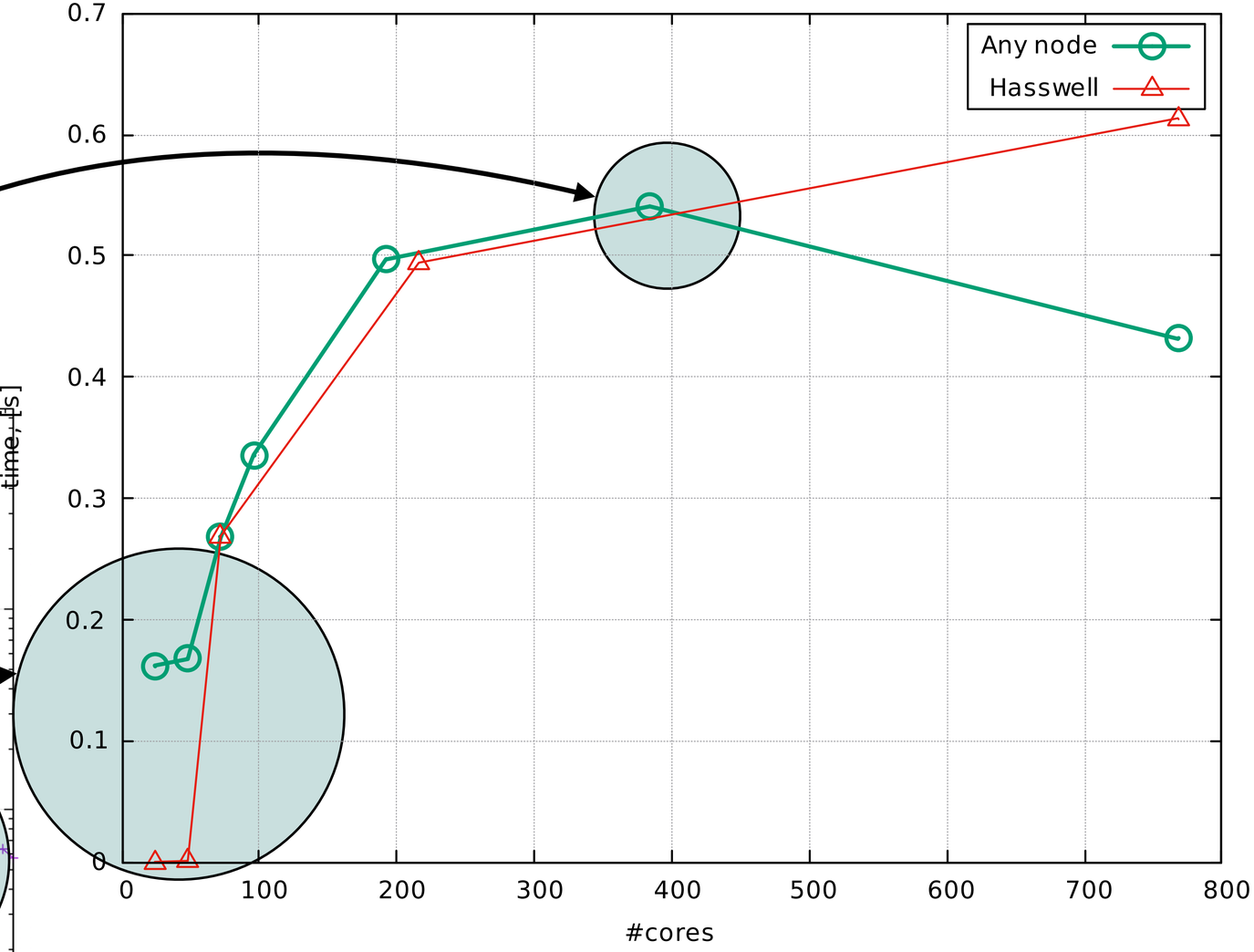


MPI

Message size

- Communication on Cartesius
- CFD code, FoxBerry

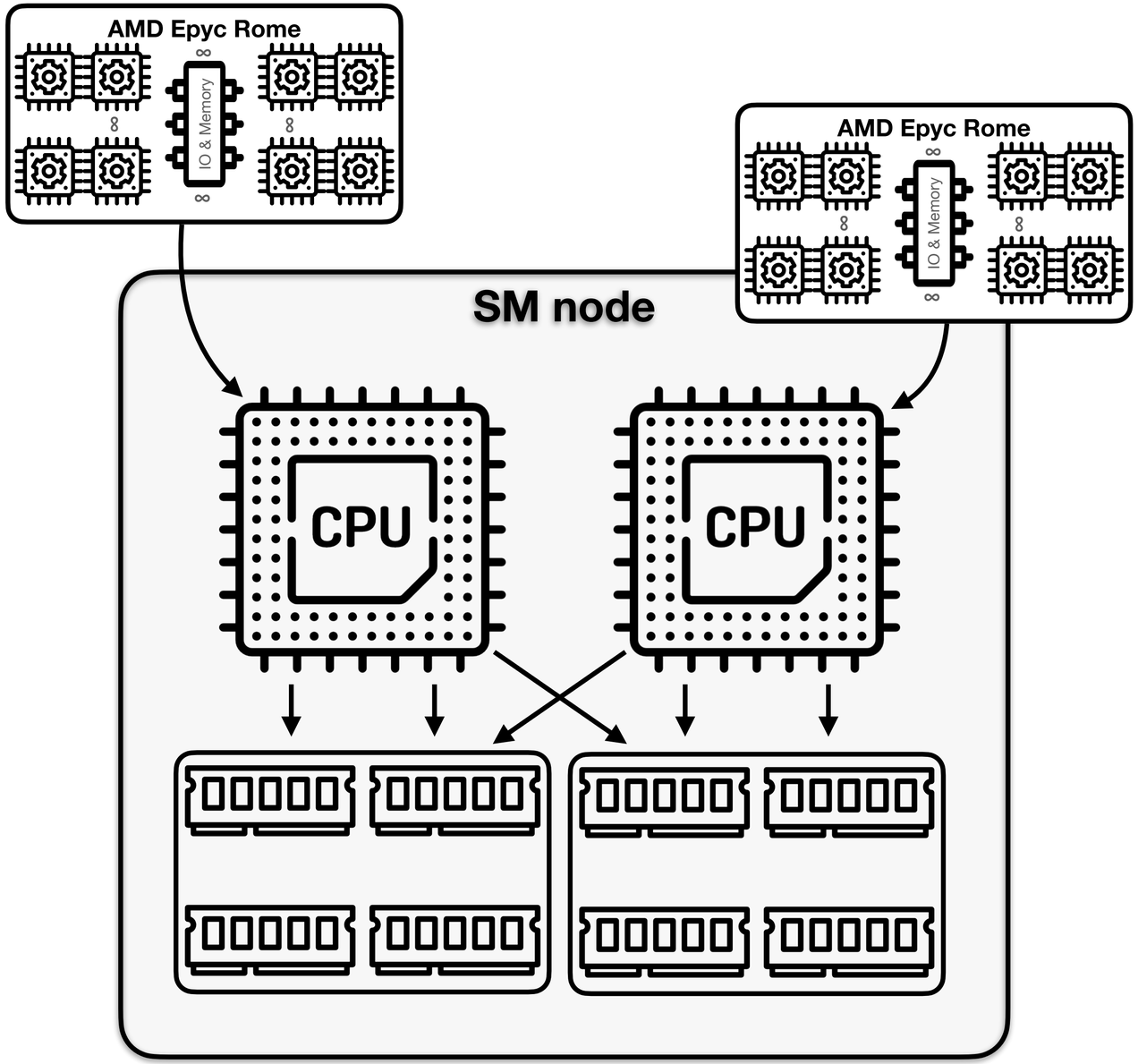




MPI

Shared Memory Nodes

- Behaviour on SM nodes is implementation-specific and is not described by the standard
- Some MPI libraries avoid “standard” communications on SM nodes thus, improving the performance (see BTL in OpenMPI)
- Some implementations provide run-time tuning parameters for SM execution
- There is no such problem as memory affinity (as with threads)



1. https://www.flaticon.com/free-icon/ram_900330
2. https://www.flaticon.com/free-icon/cpu_689379
3. https://www.flaticon.com/free-icon/cpu_4272570
4. https://www.flaticon.com/free-icon/ram_3076285

MPI

Rules of thumb

- Follow the **KISS** principle - keep it simple stupid
- Make it **maintainable**
- Make it **efficient**
- Unless it is completely necessary:
  - Avoid complex data structures
  - Avoid complex logic
- Always do **refactoring**

Typical MPI example

```cpp
#include <cmath>
#include <mpi.h>
#include <stdio.h>

int main(int argc, char** argv) {
    // Initialize the MPI environment
    MPI_Init(&argc, &argv);

    // Get the number of processes
    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);

    // Get the rank of the process
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

    // Get the name of the processor
    char processor_name[MPI_MAX_PROCESSOR_NAME];
    int name_len;
    MPI_Get_processor_name(processor_name, &name_len);

    // Print out a hello world message
    printf("Hello world from processor %s, "
           "rank %d out of %d processors\n",
           processor_name, world_rank, world_size);

    // Finalize the MPI environment.
    MPI_Finalize();
}
```

MPI

Rules of thumb

- Follow the **KISS** principle - keep it simple stupid
- Make it **maintainable**
- Make it **efficient**
- Unless it is completely necessary:
  - Avoid complex data structures
  - Avoid complex logic
- Always do **refactoring**

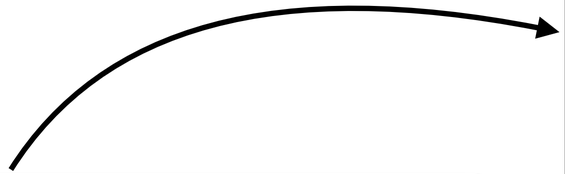
Example of a real code

```
…
if(greater1 && lesser1)
{
    nozzle_1_send_to_root_proc = true;
    for(size_t i = 0; i < (size_t)(max_graph_neighbours); ++i)
    {
        ++comm_look_up_table(my_rank, source[i]);
        MPI_Isend(&nozzle_1_send_to_root_proc,
                  1,
                  MPI_CXX_BOOL,
                  source[i],
                  nozzle1_tag1,
                  _grid->GetLocMPIComm(),
                  &nozzle1_snd_rqst[i]);
    }
}

MPI_Bcast(comm_look_up_table.data(),
          comm_look_up_table.size(),
          MPI_INT,
          rank_with_nozzle1_center,
          internal_MPI_Comm);

MPI_Waitall(max_graph_neighbours, nozzle1_snd_rqst, nozzle1_status);

MPI_Request rqst;
int count_to_root_proc = 0;
if(comm_look_up_table(rank_with_nozzle1_center, my_rank))
{
    int dummy = 0, weighted = 0;
    MPI_Recv(&nozzle_1_send_to_root_proc,
             1,
             MPI_CXX_BOOL,
             rank_with_nozzle1_center,
             nozzle1_tag1,
             _grid->GetLocMPIComm(),
             &nozzle1_status_point1);

    MPI_Dist_graph_neighbors_count(_grid->GetLocMPIComm(),
                                   &max_graph_neighbours, &dummy, &weighted);
    source.resize(max_graph_neighbours);
    source_weights.resize(max_graph_neighbours);
    dest.resize(max_graph_neighbours);
    dest_weights.resize(max_graph_neighbours);
    MPI_Dist_graph_neighbors(_grid->GetLocMPIComm(),
                             max_graph_neighbours,
                             source.data(),
                             source_weights.data(),
                             max_graph_neighbours,
                             dest.data(),
                             dest_weights.data());
…
```

MPI

MPI code

```
int MPI_Send(const void *buf,
             int count,
             MPI_Datatype datatype,
             int dest,
             int tag, MPI_Comm comm);


int MPI_Recv(void *buf,
             int count,
             MPI_Datatype datatype,
             int source,
             int tag, MPI_Comm comm,
             MPI_Status *status);
```



```
class MPIManager{
public:

    inline int send(const int* buff, int size, int pid,
int tag)
    {
        MPI_Send(
                 &buff,
                 size,
                 MPI_INT,
                 pid,
                 tag,
                 mpi_comm);
        // handle errors
    }

    inline int recv(int* buff, int size, int pid, int
tag)
    {
        MPI_Status status;
        MPI_Recv(
                 &buff,
                 size,
                 MPI_INT,
                 pid,
                 tag,
                 mpi_comm,
                 &status);
        // handle errors
    }

private:
    MPI_Comm mpi_comm;
};
```

MPI

MPI code

```
int main(int argc, char** argv) {
    // Initialize the MPI environment
    MPI_Init(&argc, &argv);

    // Get the number of processes
    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);

    // Get the rank of the process
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

    // Get the name of the processor
    char processor_name[MPI_MAX_PROCESSOR_NAME];
    int name_len;
    MPI_Get_processor_name(processor_name, &name_len);

    // Print off a hello world message
    printf("Hello world from processor %s, "
           "rank %d out of %d processors\n",
           processor_name, world_rank, world_size);

    // Finalize the MPI environment.
    MPI_Finalize();
}
```

```
#include <cmath>
#include <mpi.h>
#include <stdio.h>

class MPIManager {
    MPI_Comm comm;
public:
    MPIManager1() : comm(MPI_COMM_WORLD) { }
    void initialize(int argc, char** argv) { … }
    void finalize() { … }
    char *getProcName() { … }
    int getMyRank() { … }
    int getNumProcs() { … }
};

int main(int argc, char** argv) {

    MPIManager mpi_manager;

    // Initialize the MPI environment
    mpi_manager.initialize(argc, argv);

    // Print off a hello world message
    printf("Hello world from processor %s, "
           "rank %d out of %d processes\n",
           mpi_manager.getProcName(),
           mpi_manager.getMyRank(),
           mpi_manager.getNumProcs());

    // Finalize the MPI environment.
    mpi_manager.finalize();
}
```

Connecting to Snellius

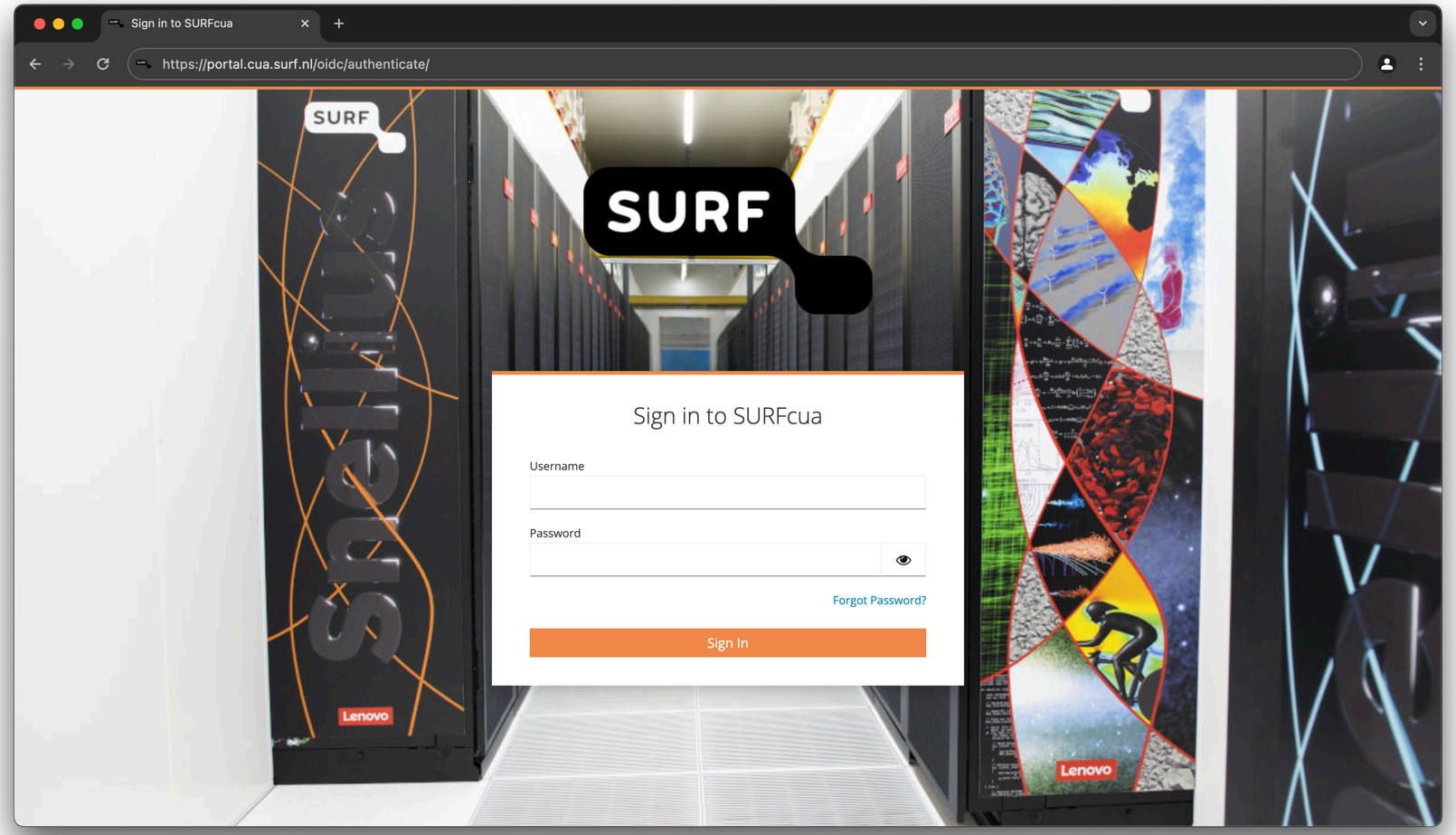
Snellius

Connecting to

- **Windows OS**:
  - MobaXterm: **https://mobaxterm.mobatek.net/**
    (download potable edition)
  - Putty
- **macOS** and **Linux OS**:
  - You are already well equipped
- Domain name: **snellius.surf.nl**
- Use **-X** or **-Y** flag in the **ssh** command to allow **X11 forwarding** (for visualisation)
  - macOS users should install XQuartz: (https://www.xquartz.org/)

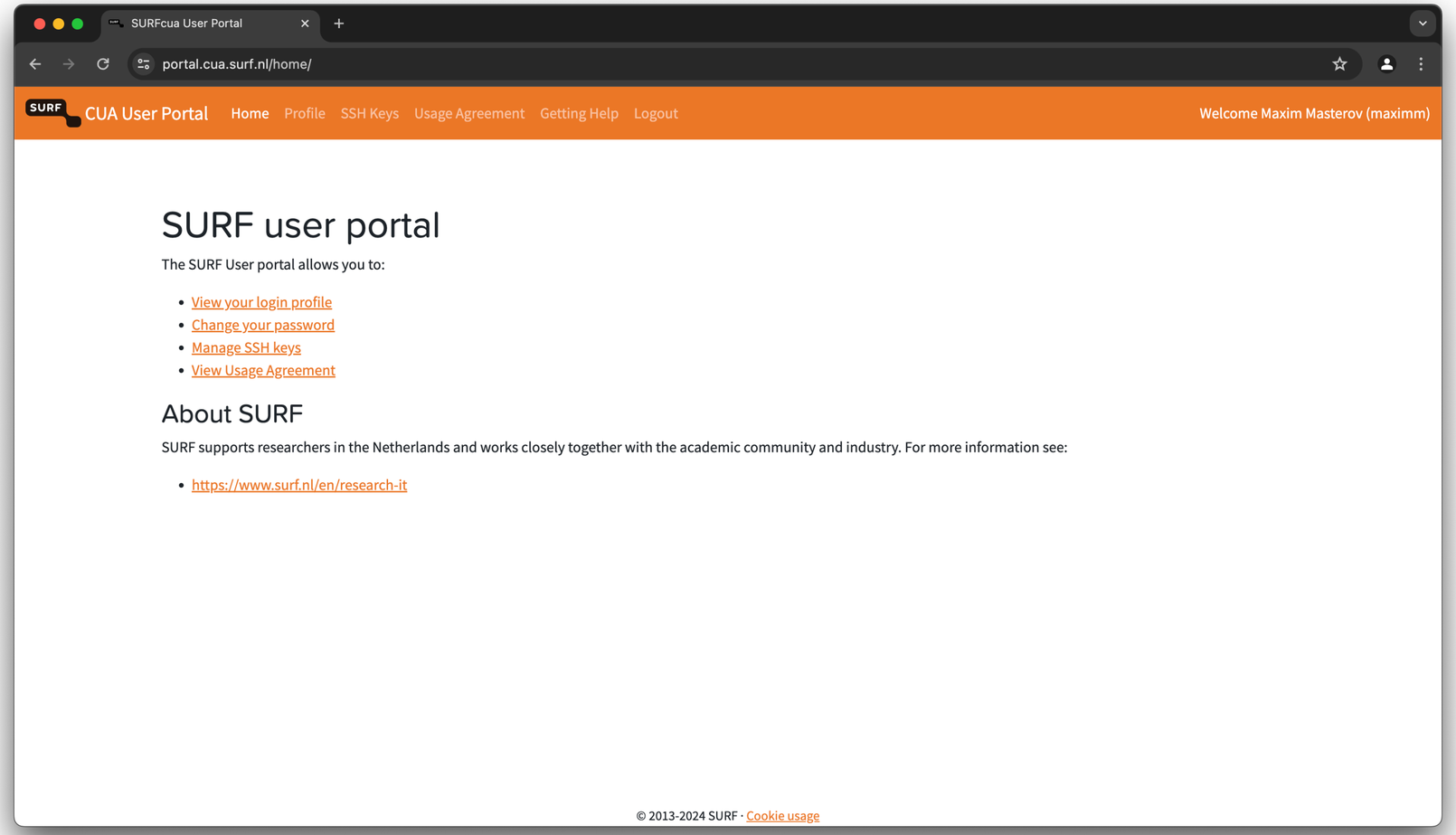
Snellius

Connecting to: https://portal.cua.surf.nl



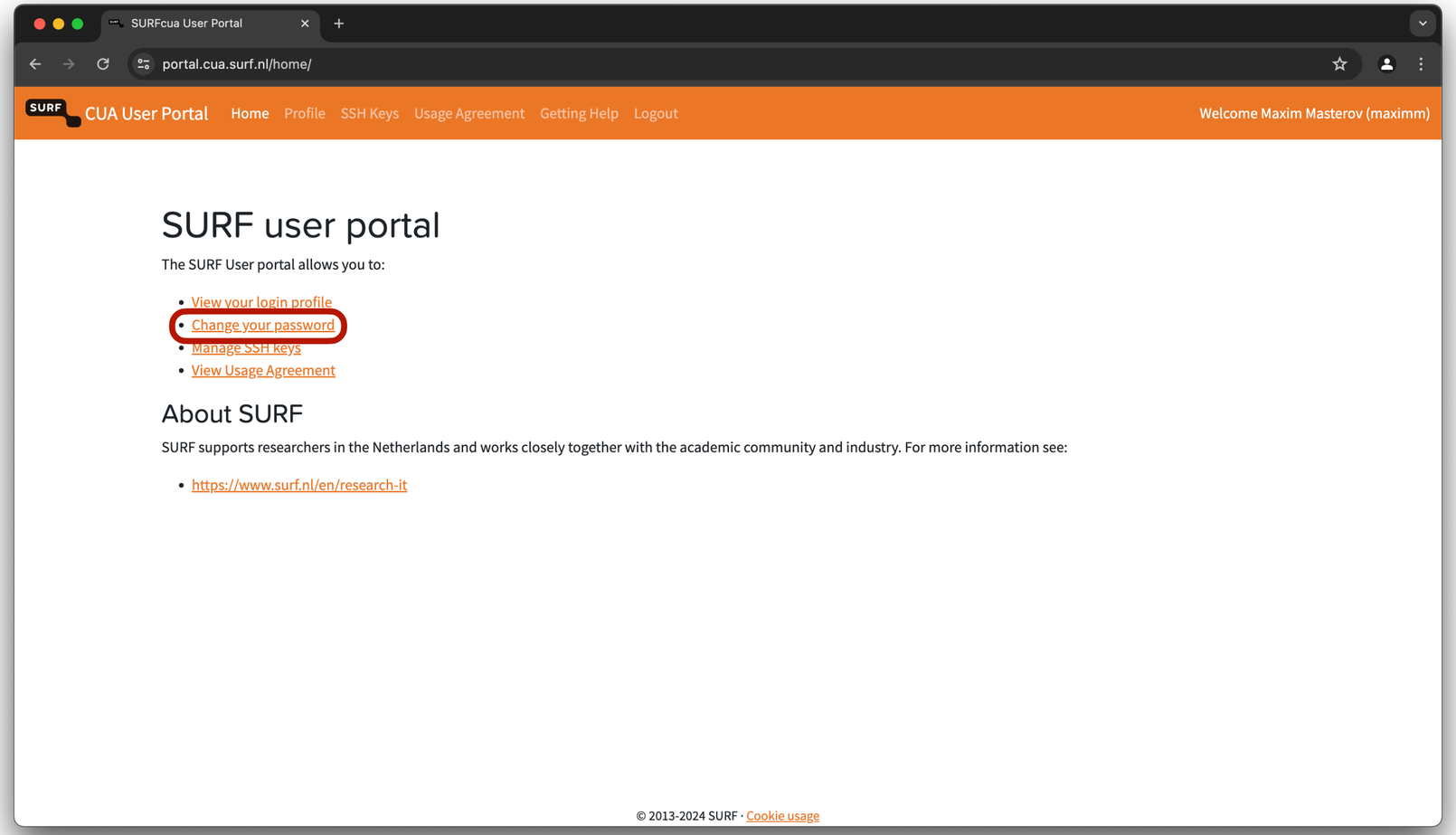
Snellius

Connecting to: https://portal.cua.surf.nl



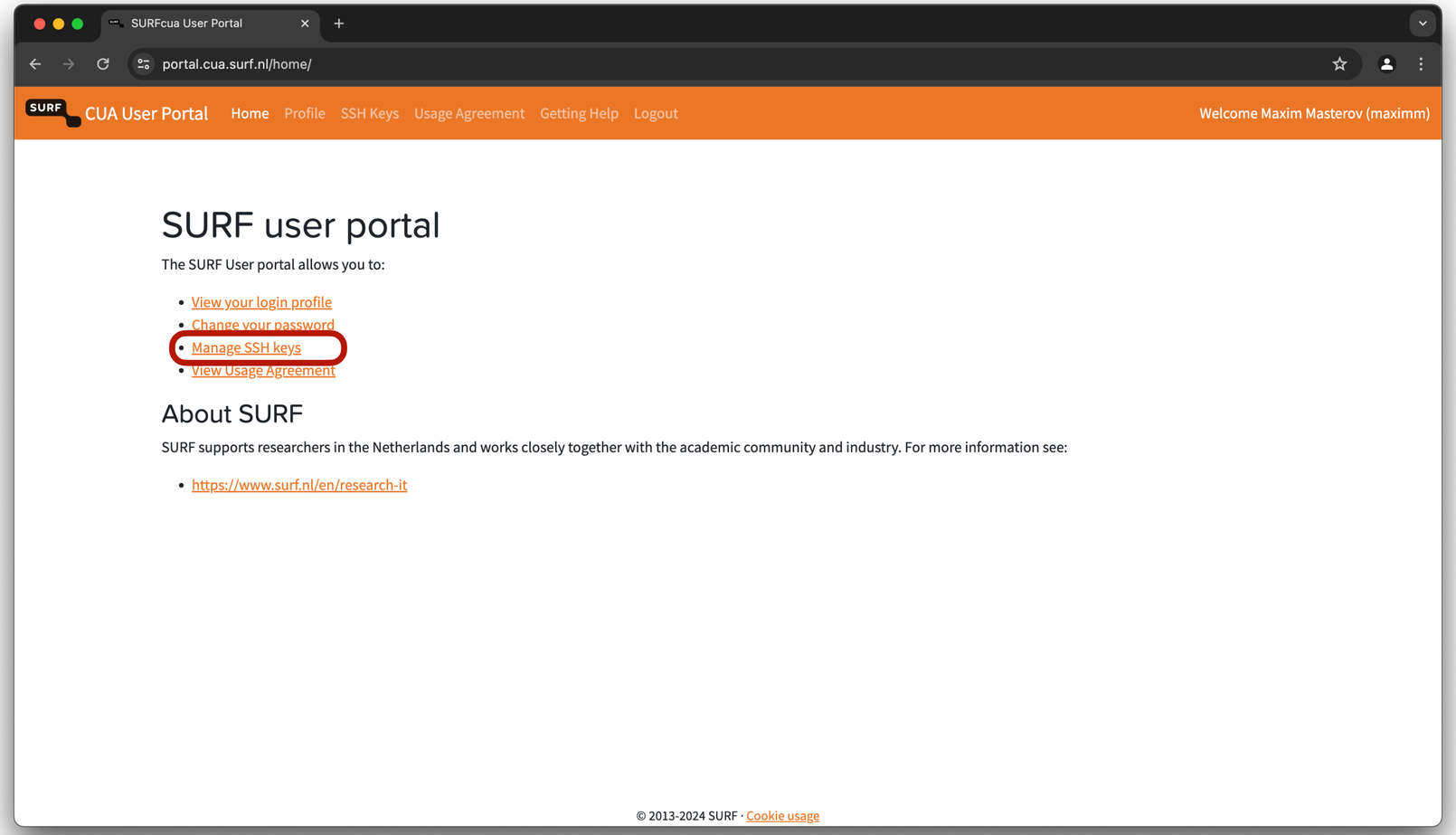
Snellius

Connecting to: https://portal.cua.surf.nl



Snellius

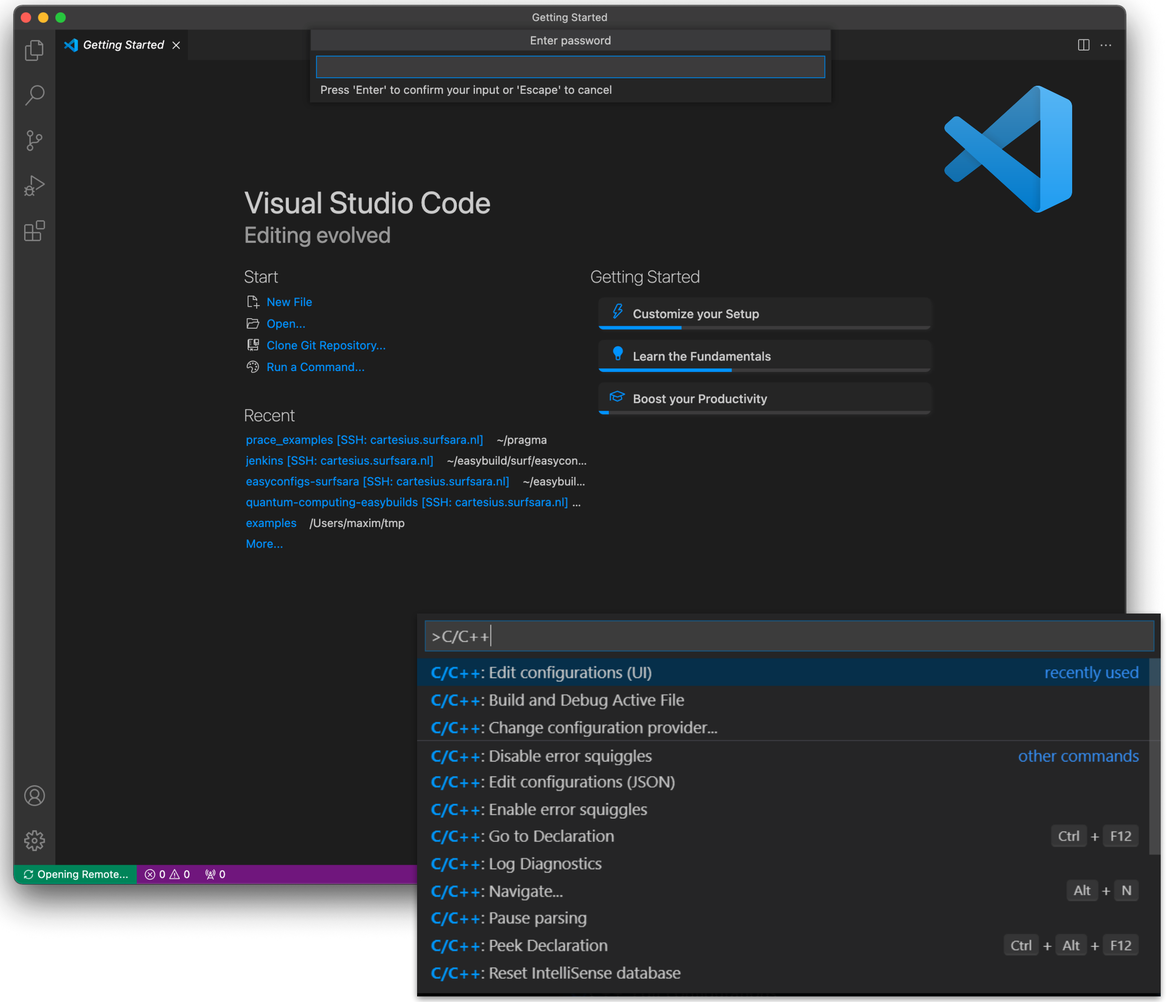
Connecting to: https://portal.cua.surf.nl



Snellius

VS Code (recommendation)

- Download VS Code from
  - **https://code.visualstudio.com**
- Install plugins
  - Locally: ***Remote - SSH***
  - Remotely: **C/C++ Extension Pack**
- Press **F1** and run the **Remote-SSH: Open SSH Host...** command
- Or use any other IDE (or text editor) with support for the remote development (CLion, Eclipse, Atom, …)
- Some help: https://servicedesk.surf.nl/wiki/display/WIKI/Visual+Studio+Code+for+remote+development

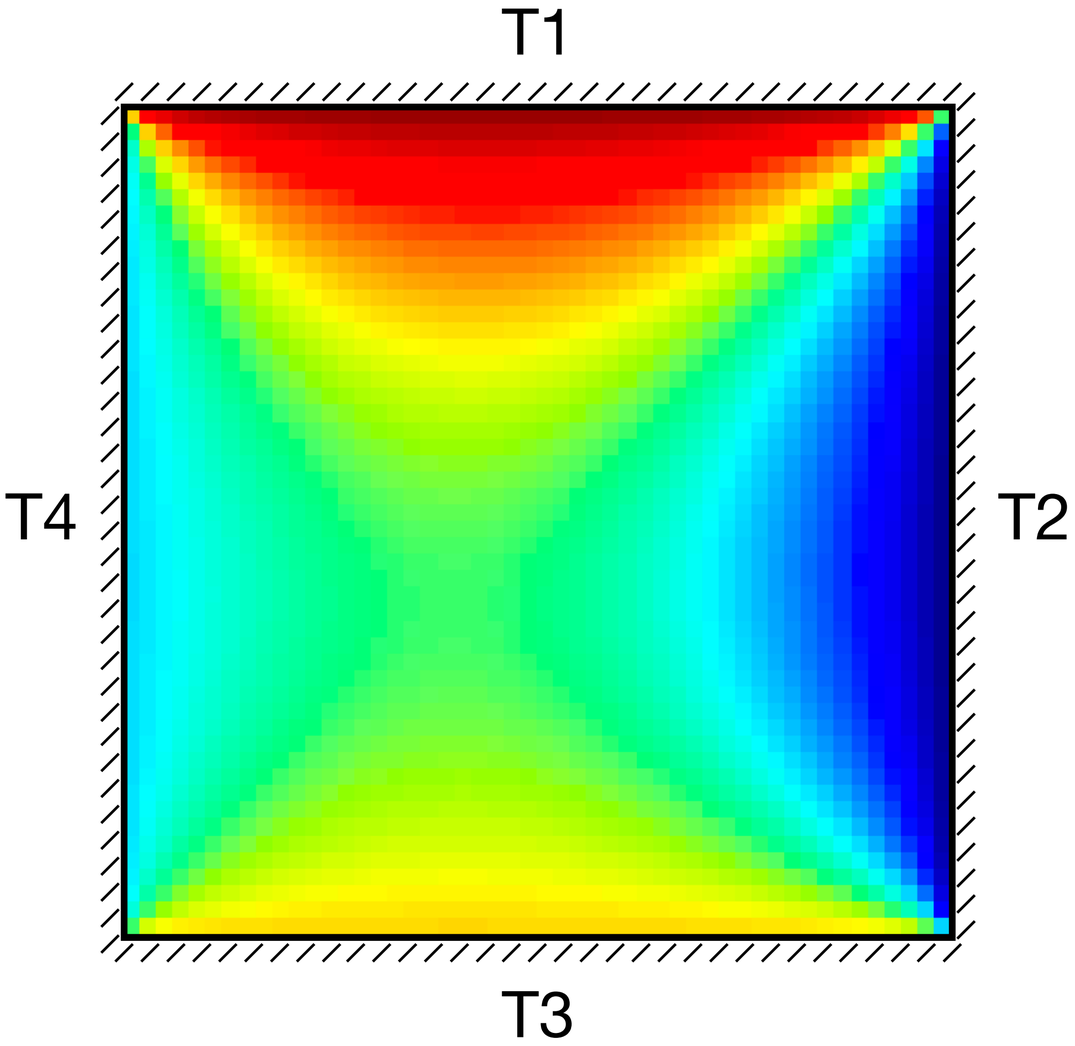


Jacobi Solver

Heat transfer

The problem

- Consider a basic steady state conduction in a closed box
- No sources are present
- All walls have constant temperature



- The governing equation:

$$\Delta T = 0 \qquad \frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0$$

Heat transfer

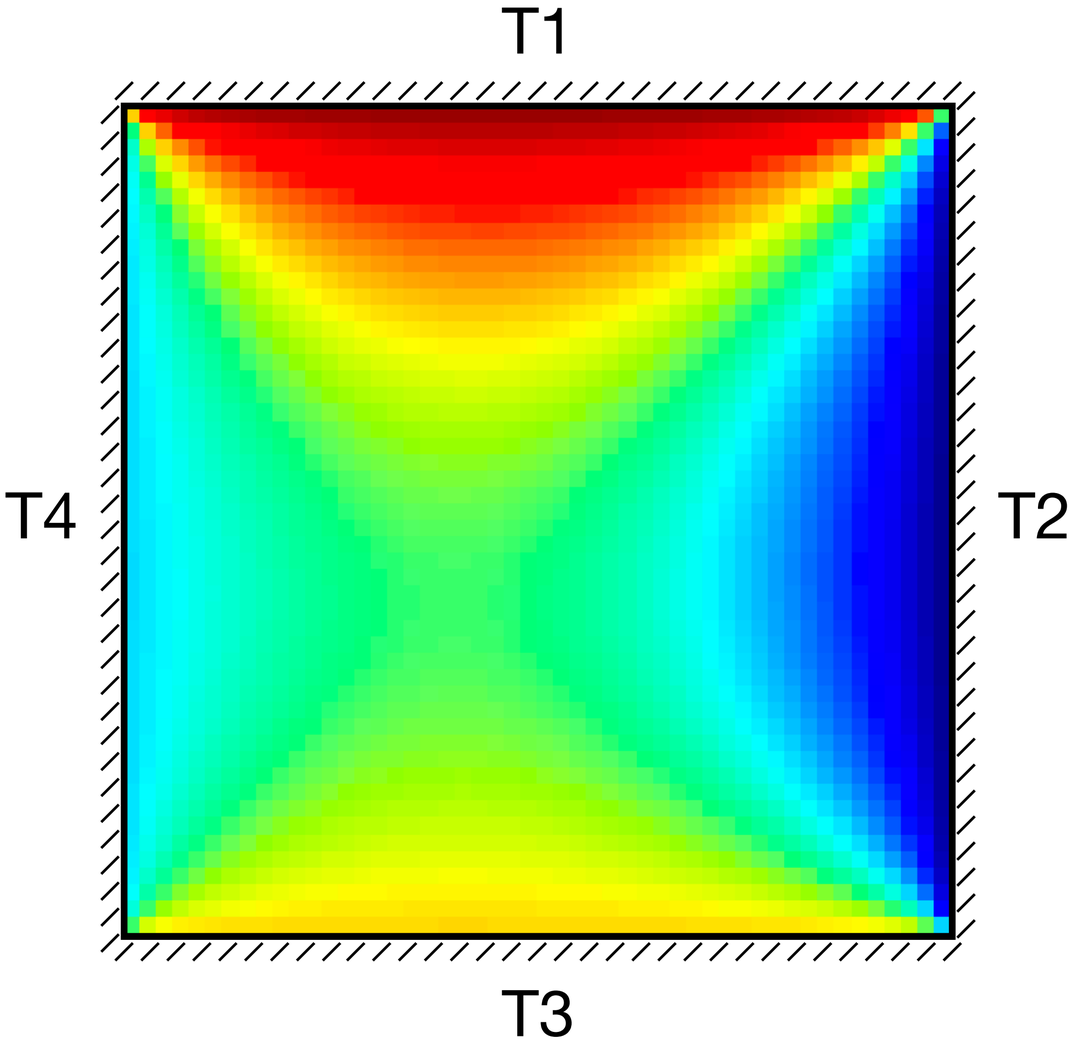
The problem

- The governing equation:

$$\Delta T = 0 \qquad \frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0$$

- All derivatives are approximated with the 2nd-order central difference scheme:

$$\frac{\partial^2 T}{\partial x^2} \approx \frac{T_{i+1,j} - 2T_{i,j} + T_{i-1,j}}{\delta x^2}$$



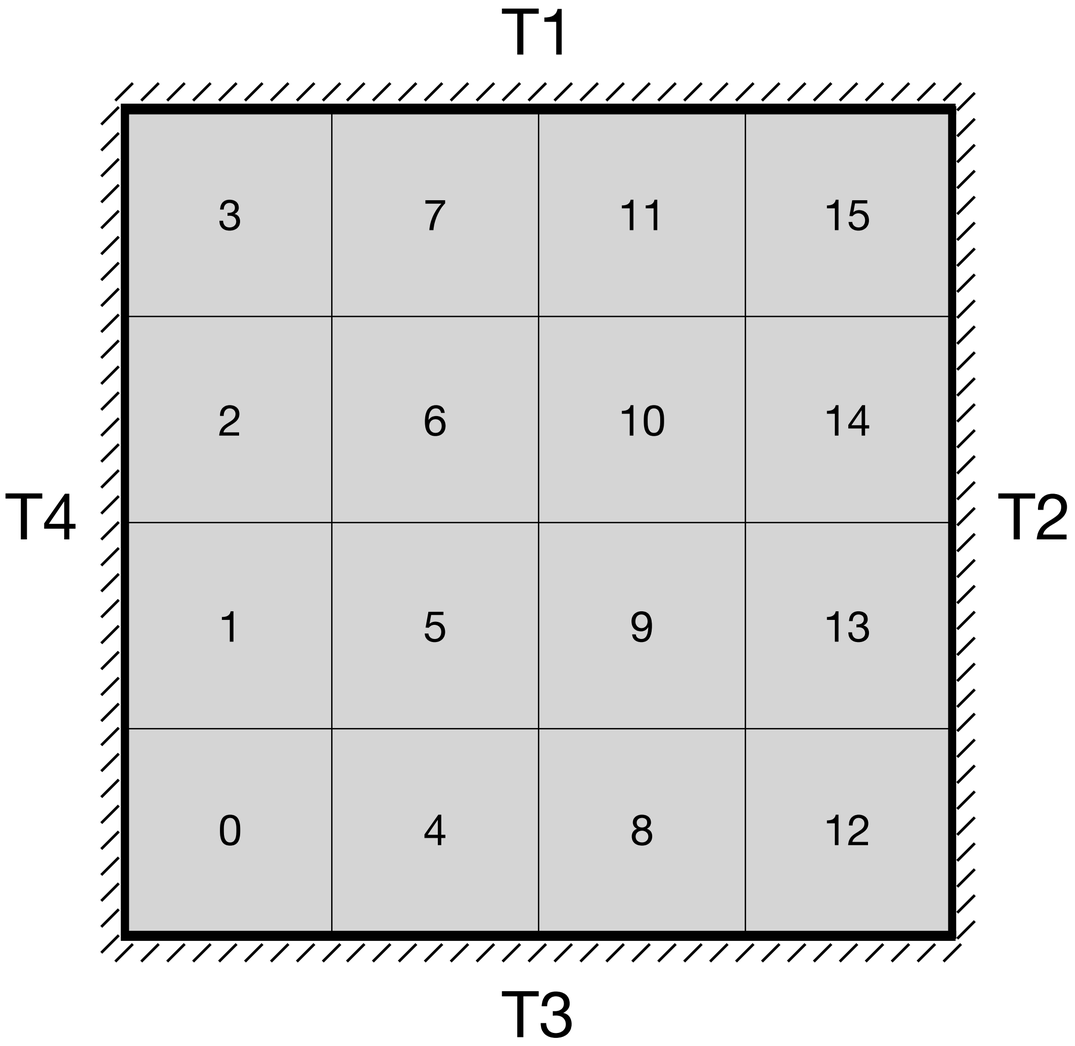
Heat transfer

The problem

- All derivatives are approximated with the 2nd-order central difference scheme:

$$\frac{\partial^2 T}{\partial x^2} \approx \frac{T_{i+1,j} - 2T_{i,j} + T_{i-1,j}}{\delta x^2}$$

$$\frac{\partial^2 T}{\partial y^2} \approx \frac{T_{i,j+1} - 2T_{i,j} + T_{i,j-1}}{\delta y^2}$$



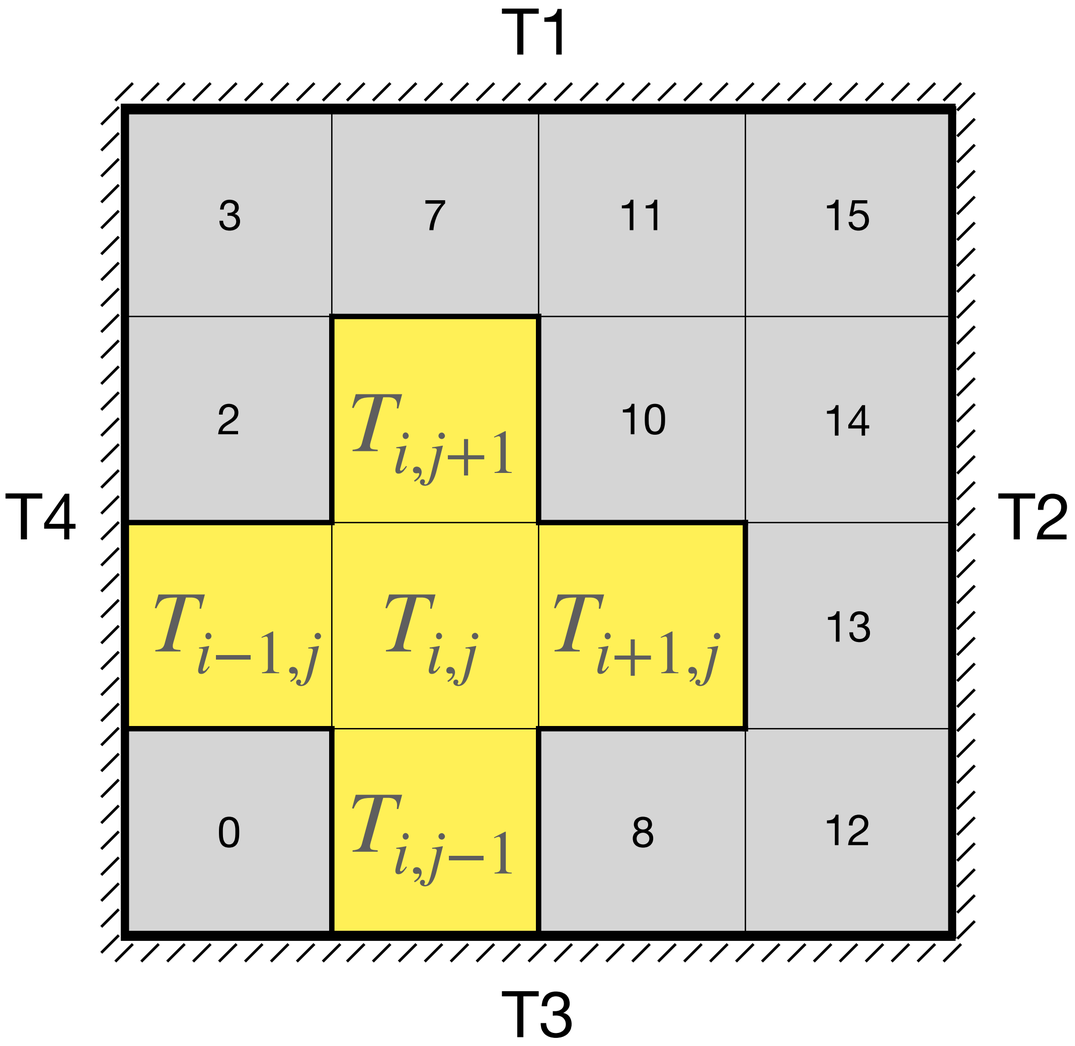
Heat transfer

The problem

- All derivatives are approximated with the 2nd-order central difference scheme:

$$\frac{\partial^2 T}{\partial x^2} \approx \frac{T_{i+1,j} - 2T_{i,j} + T_{i-1,j}}{\delta x^2}$$

$$\frac{\partial^2 T}{\partial y^2} \approx \frac{T_{i,j+1} - 2T_{i,j} + T_{i,j-1}}{\delta y^2}$$

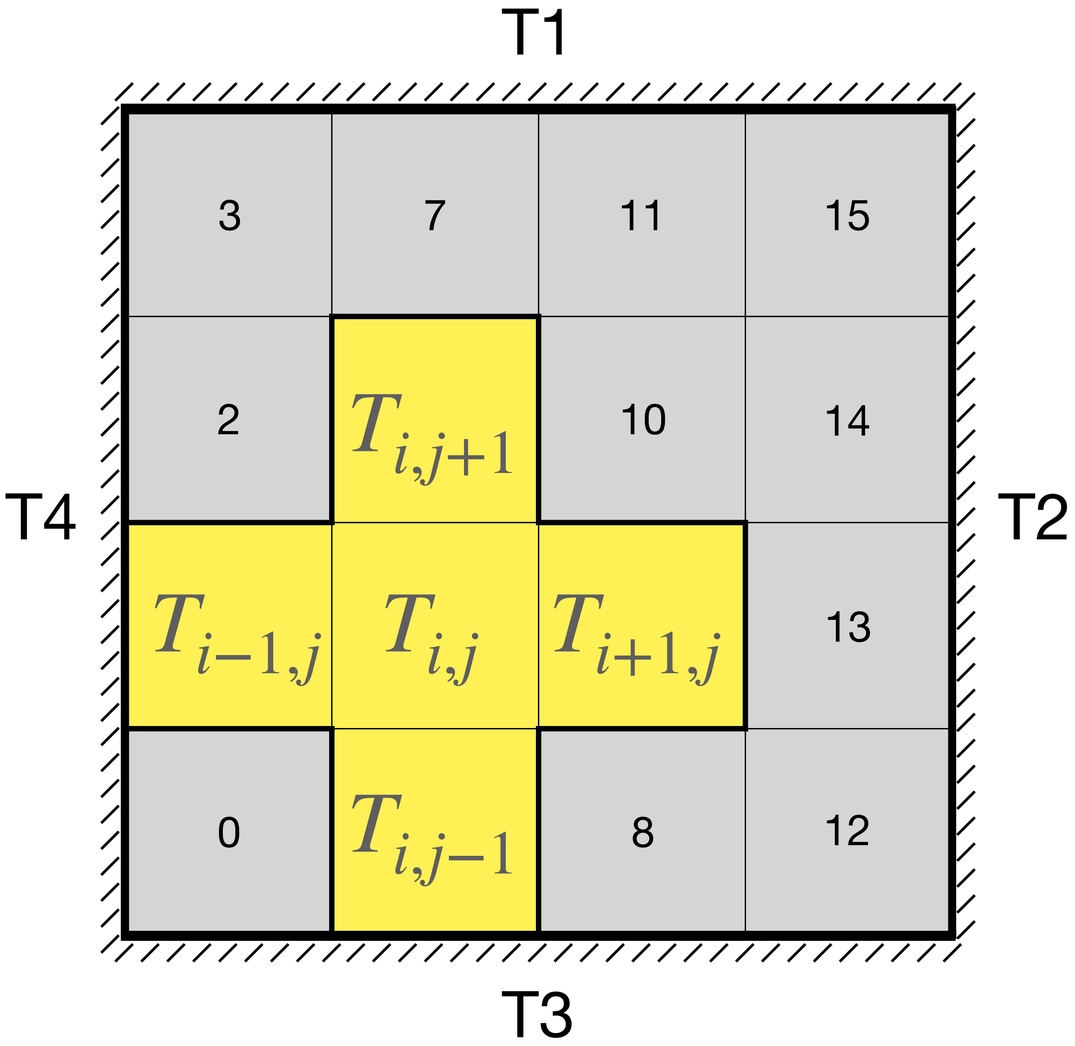


Heat transfer

The problem

- We need to solve the following system of linear equations:

$$\frac{T_{i+1,j} - 2T_{i,j} + T_{i-1,j}}{\delta x^2} + \frac{T_{i,j+1} - 2T_{i,j} + T_{i,j-1}}{\delta y^2} = 0$$



Heat transfer

The problem

- Assume unit grid cells:

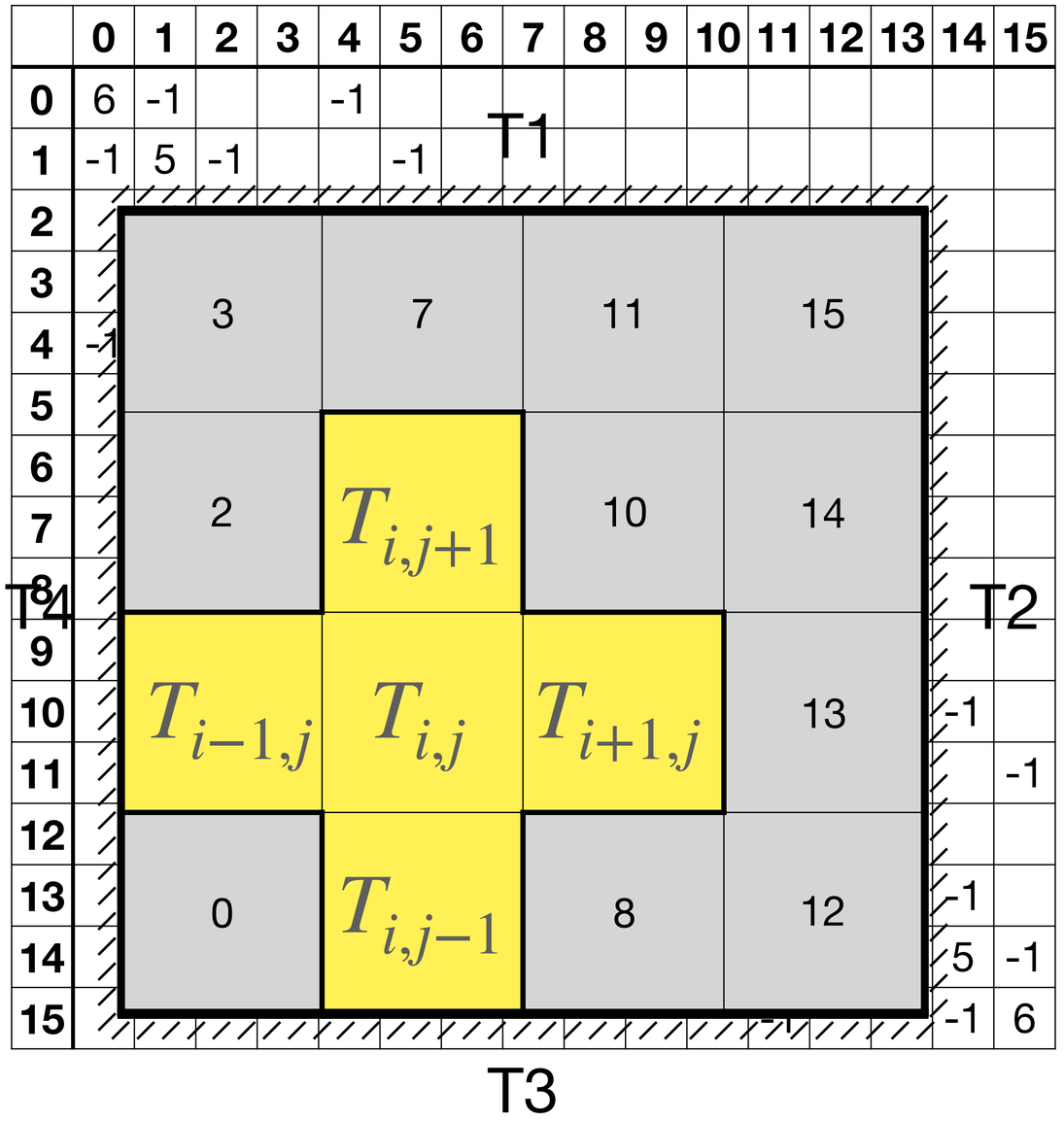
$$4T_{i,j} - T_{i+1,j} - T_{i-1,j}$$

$$-T_{i,j+1} - T_{i,j-1} = 0$$

- Boundary conditions (linear interpolation):

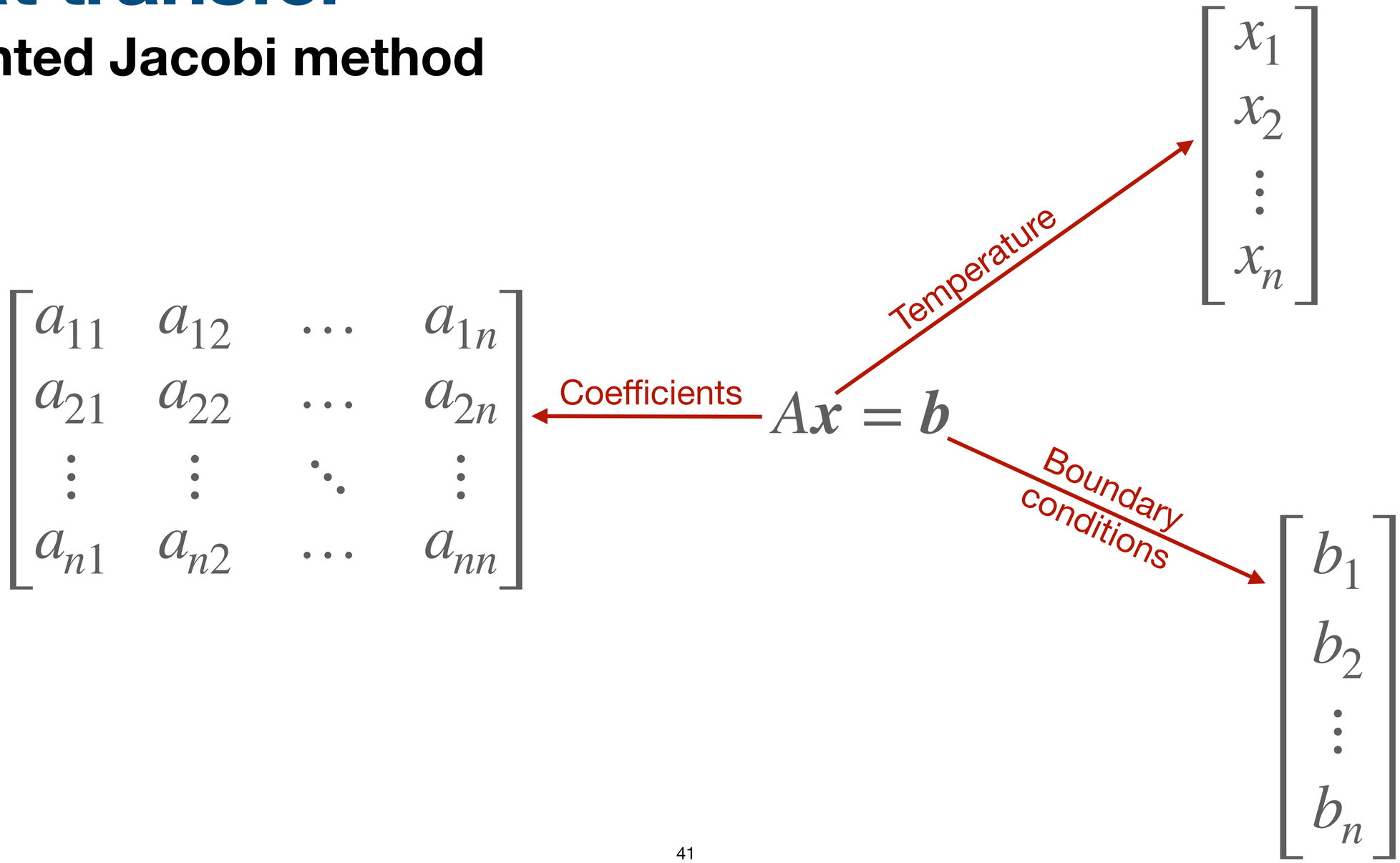
$$T_{i-1,j} = 2T_w - T_{i,j}$$

Matrix representation



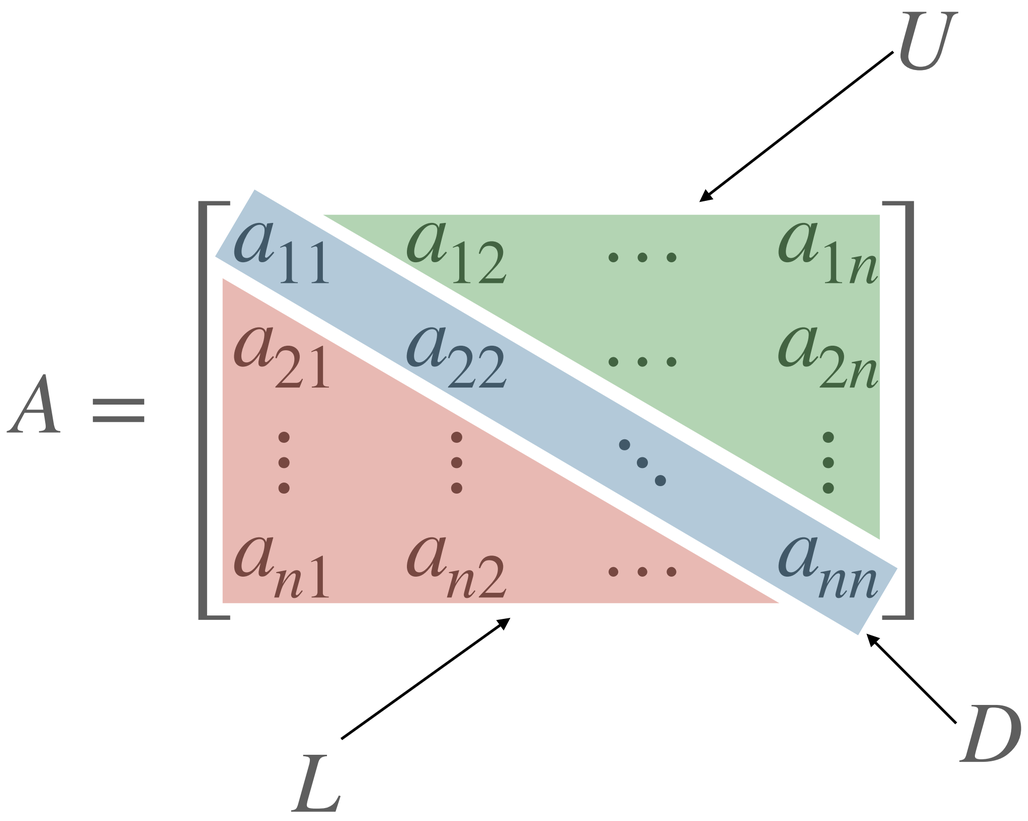
Heat transfer

Weighted Jacobi method



Heat transfer

Weighted Jacobi method



$$A = D + L + U$$

$$\boldsymbol{x}^{(k+1)} = \omega D^{-1}(\boldsymbol{b} - (L + U)\boldsymbol{x}^{(k)}) + (1 - \omega)\boldsymbol{x}^{(k)}$$

$$x_i^{(k+1)} = \omega \frac{1}{a_{ii}}(b_i - \sum_{j \neq i} a_{ij} x_j^{(k)}) + (1 - \omega)\boldsymbol{x}_i^{(k)}$$

Heat transfer

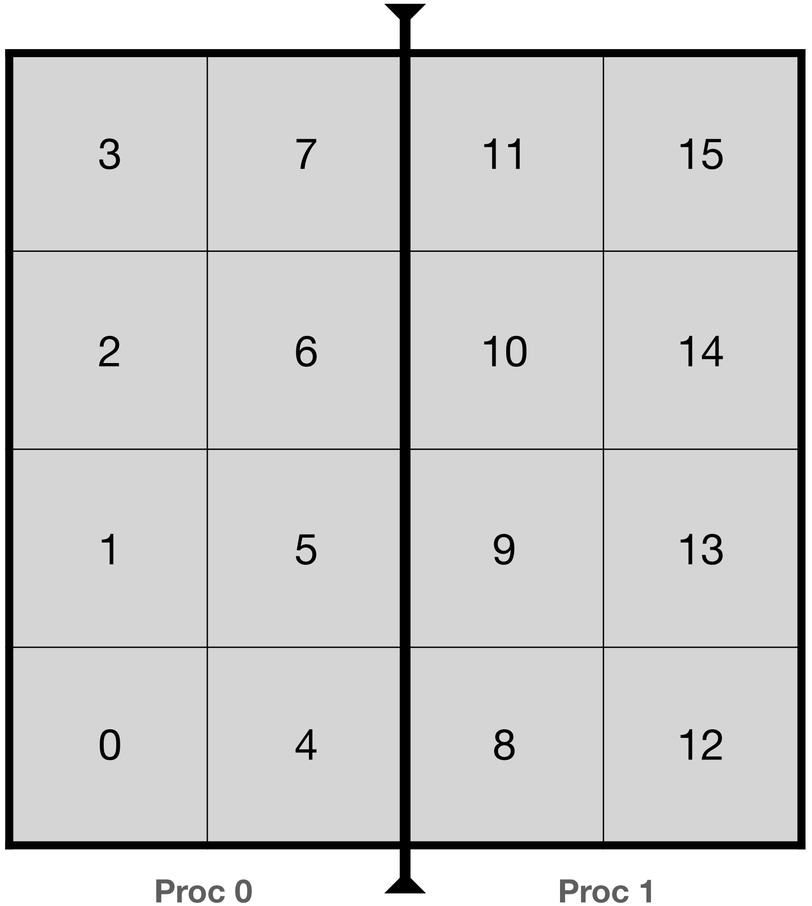
Why weighted Jacobi method?

- Simple implementation
- Has simple but frequently used mathematical operations:
  - **L2-norm**
  - **MVP**
  - **Vector update (copy)**
- Simple parallelisation
- Straightforward hybrid parallelisation
- Often used as a supplement to more advanced linear solvers (e.g. as a smoother in AMG)

Heat transfer

Data representation

Grid representation



Matrix representation

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	6	-1			-1											
1	-1	5	-1			-1										
2		-1	5	-1			-1									
3			-1	6				-1								
4	-1				5	-1			-1							
5		-1			-1	4	-1			-1						
6			-1			-1	4	-1			-1					
7				-1			-1	5				-1				
8					-1				5	-1			-1			
9						-1			-1	4	-1			-1		
10							-1			-1	4	-1			-1	
11								-1			-1	5				-1
12									-1				6	-1		
13										-1			-1	5	-1	
14											-1			-1	5	-1
15												-1			-1	6

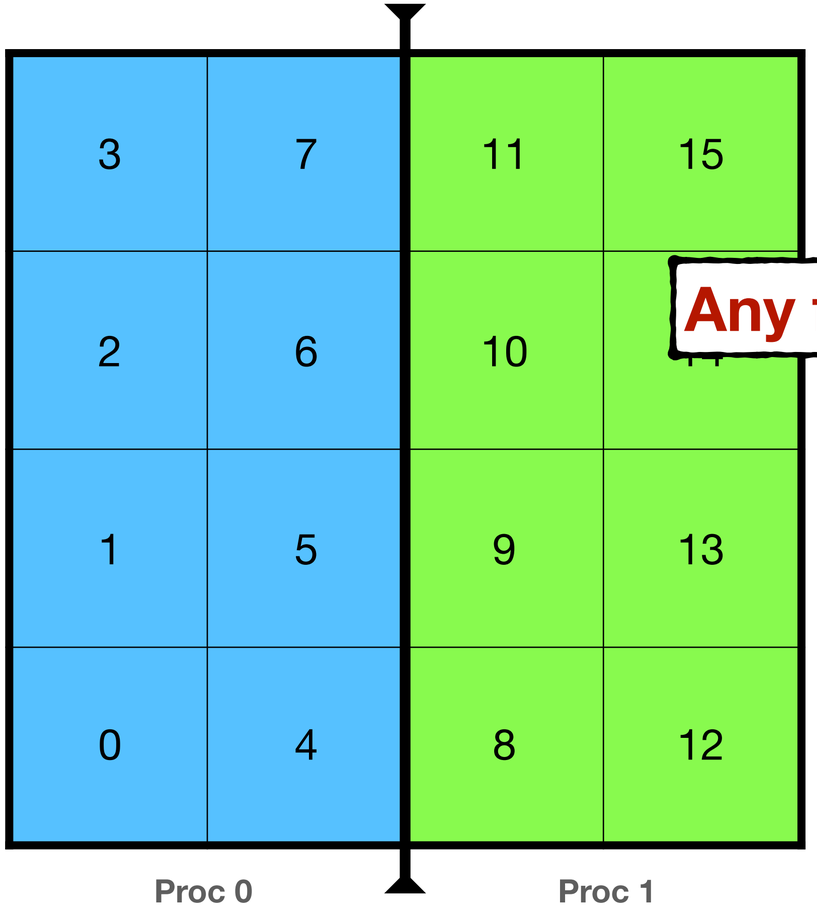
Proc 0

Proc 1

Heat transfer

Data representation

Grid representation



Matrix representation

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	6	-1			-1											
1	-1	5	-1			-1										
2		-1	5	-1			-1									
3			-1	6				-1								
4	-1				5	-1			-1							
5		-1			-1	4	-1			-1						
6			-1			-1	4	-1			-1					
7				-1			-1	5				-1				
8					-1				5	-1			-1			
9						-1			-1	4	-1			-1		
10							-1			-1	4	-1			-1	
11								-1			-1	5				-1
12									-1				6	-1		
13										-1			-1	5	-1	
14											-1			-1	5	-1
15												-1			-1	6

Proc 0

Proc 1

Any foreseen problems?

MVP

Data representation

Proc 0

Proc 1

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	6	-1			-1											
1	-1	5	-1			-1										
2		-1	5	-1			-1									
3			-1	6				-1								
4	-1				5	-1			-1							
5		-1			-1	4	-1			-1						
6			-1			-1	4	-1			-1					
7				-1			-1	5				-1				
8					-1				5	-1			-1			
9						-1			-1	4	-1			-1		
10							-1			-1	4	-1			-1	
11								-1			-1	5				-1
12									-1				6	-1		
13										-1			-1	5	-1	
14											-1			-1	5	-1
15												-1			-1	6

x

x0	
x1	
x2	
x3	
x4	
x5	
x6	
x7	
x8	
x9	
x10	
x11	
x12	
x13	
x14	
x15	

=

y0
y1
y2
y3
y4
y5
y6
y7
y8
y9
y10
y11
y12
y13
y14
y15

$$y_i = \sum_{j=0}^{N} a_{ij} x_j$$

$$y_5 = a_{51}x_1 + a_{54}x_4 + a_{55}x_5 + a_{56}x_6 + a_{59}x_9$$

MVP

Data representation

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	6	-1			-1											
1	-1	5	-1			-1										
2		-1	5	-1			-1									
3			-1	6				-1								
4	-1				5	-1			-1							
5		-1			-1	4	-1			-1						
6			-1			-1	4	-1			-1					
7				-1			-1	5				-1				
8					-1				5	-1			-1			
9						-1			-1	4	-1			-1		
10							-1			-1	4	-1			-1	
11								-1			-1	5				-1
12									-1				6	-1		
13										-1			-1	5	-1	
14											-1			-1	5	-1
15												-1			-1	6

Proc 0

Proc 1

x

x0
x1
x2
x3
x4
x5
x6
x7
x8
x9
x10
x11
x12
x13
x14
x15

=

y0
y1
y2
y3
y4
y5
y6
y7
y8
y9
y10
y11
y12
y13
y14
y15

We need access to the value from the neighbouring process!

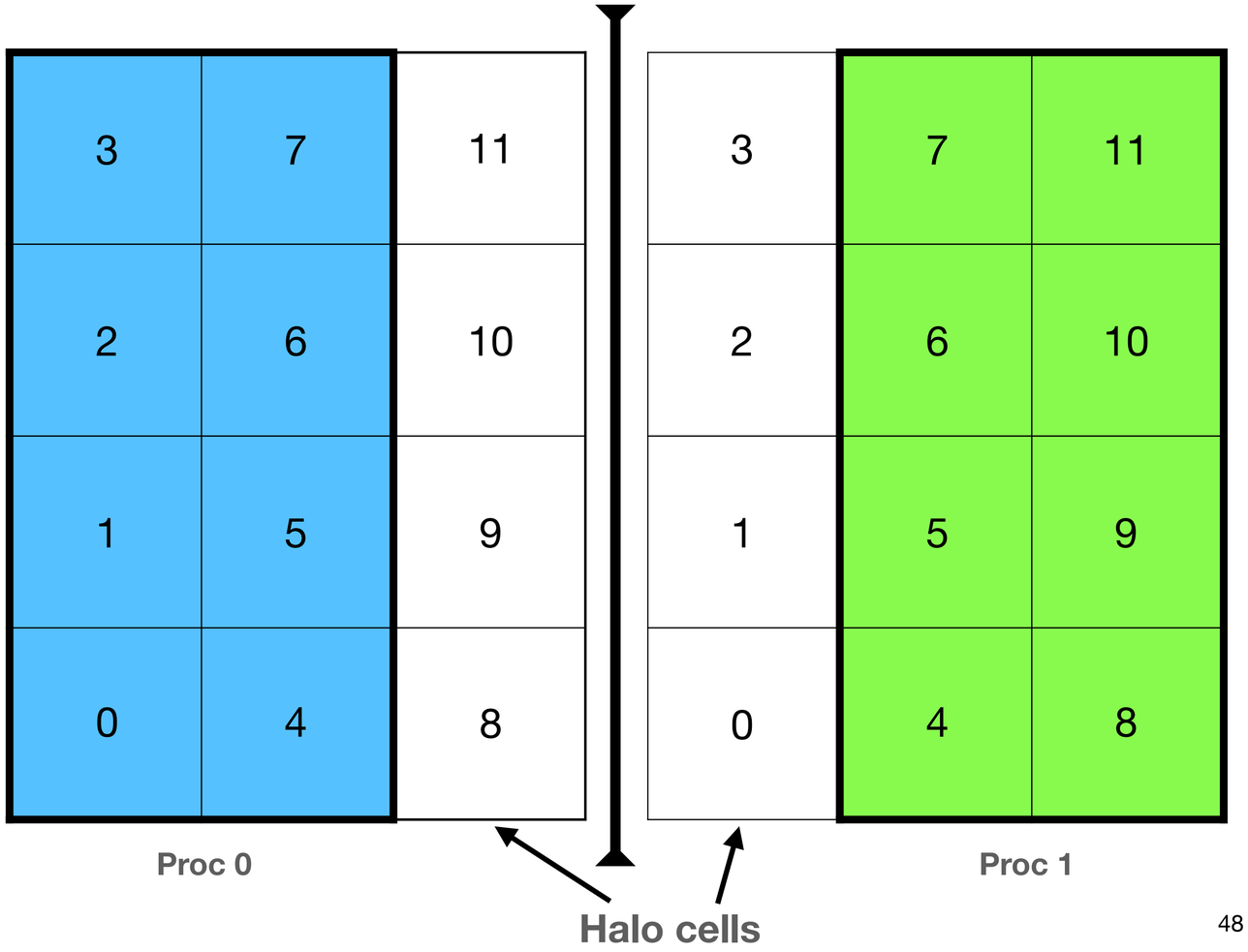
$$y_i = \sum_{j=0}^{N} a_{ij} x_j$$

$$y_5 = a_{51}x_1 + a_{54}x_4 + a_{55}x_5 + a_{56}x_6 + a_{59}x_9$$

Heat transfer

Data representation

Grid representation



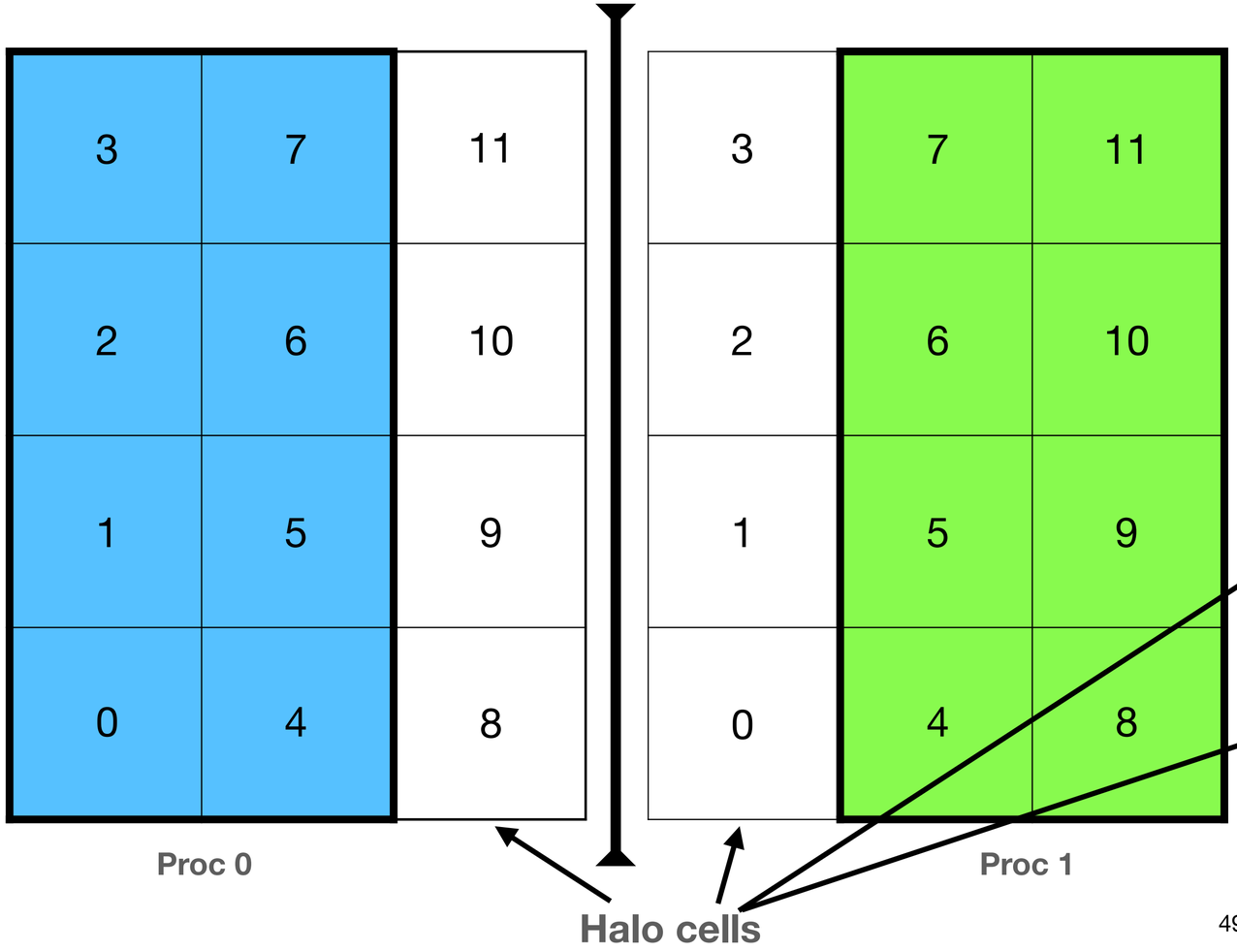
Matrix representation

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Proc 0	0	6	-1			-1											
	1	-1	5	-1			-1										
	2		-1	5	-1			-1									
	3			-1	6				-1								
	4	-1				5	-1			-1							
	5		-1			-1	4	-1			-1						
	6			-1			-1	4	-1			-1					
	7				-1			-1	5				-1				
Proc 1	8					-1				5	-1			-1			
	9						-1			-1	4	-1			-1		
	10							-1			-1	4	-1			-1	
	11								-1			-1	5				-1
	12									-1				6	-1		
	13										-1			-1	5	-1	
	14											-1			-1	5	-1
	15												-1			-1	6

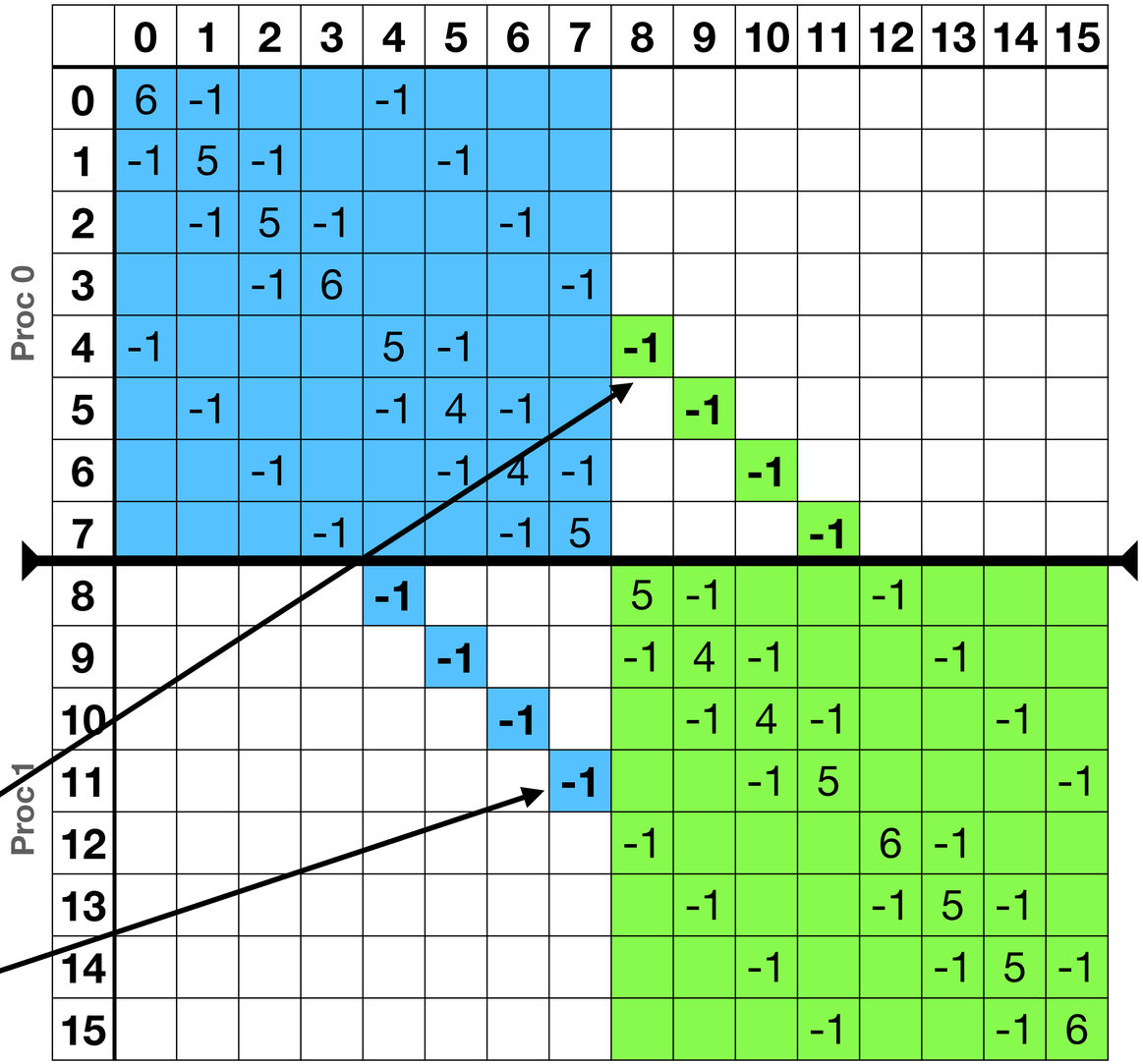
Heat transfer

Data representation

Grid representation



Matrix representation

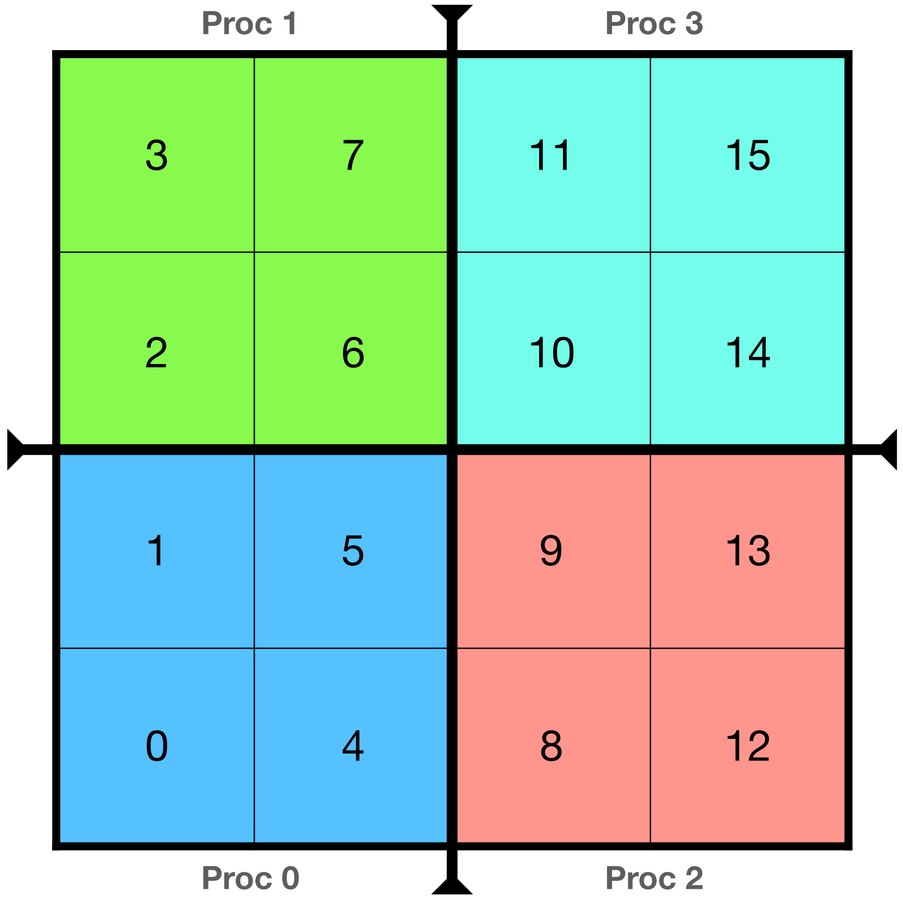


	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	6	-1			-1											
1	-1	5	-1			-1										
2		-1	5	-1			-1									
3			-1	6				-1								
4	-1				5	-1			**-1**							
5		-1			-1	4	-1			**-1**						
6			-1			-1	4	-1			**-1**					
7				-1			-1	5				**-1**				
8					**-1**				5	-1			-1			
9						**-1**			-1	4	-1			-1		
10							**-1**			-1	4	-1			-1	
11								**-1**			-1	5				-1
12									-1				6	-1		
13										-1			-1	5	-1	
14											-1			-1	5	-1
15												-1			-1	6

No problems so far!

Heat transfer

Data representation



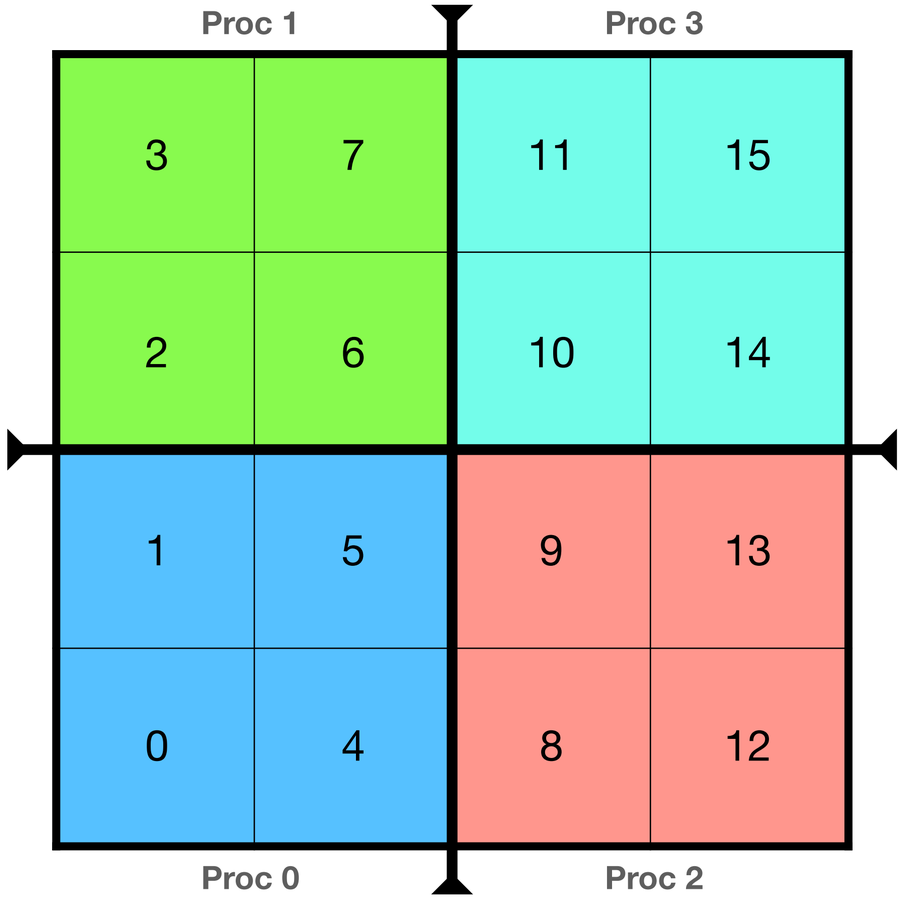
Matrix representation

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	6	-1			-1											
1	-1	5	-1			-1										
2		-1	5	-1			-1									
3			-1	6				-1								
4	-1				5	-1			-1							
5		-1			-1	4	-1			-1						
6			-1			-1	4	-1			-1					
7				-1			-1	5				-1				
8					-1				5	-1			-1			
9						-1			-1	4	-1			-1		
10							-1			-1	4	-1			-1	
11								-1			-1	5				-1
12									-1				6	-1		
13										-1			-1	5	-1	
14											-1			-1	5	-1
15												-1			-1	6

Proc 0 Proc 1 Proc 0 Proc 1 Proc 2 Proc 3 Proc 2 Proc 3

Heat transfer

Data representation



Matrix representation

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	6	-1			-1											
1	-1	5	-1			-1										
2		-1	5	-1			-1									
3			-1	6				-1								
4	-1				5	-1			-1							
5		-1			-1	4	-1			-1						
6			-1			-1	4	-1			-1					
7				-1			-1	5				-1				
8					-1				5	-1			-1			
9						-1			-1	4	-1			-1		
10							-1			-1	4	-1			-1	
11								-1			-1	5				-1
12									-1				6	-1		
13										-1			-1	5	-1	
14											-1			-1	5	-1
15												-1			-1	6

Proc 0 Proc 1 Proc 0 Proc 1 Proc 2 Proc 3 Proc 2 Proc 3

Heat transfer

Data representation

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	6	-1			-1											
1	-1	5	-1			-1										
2		-1	5	-1			-1									
3			-1	6				-1								
4	-1				5	-1			-1							
5		-1			-1	4	-1			-1						
6			-1			-1	4	-1			-1					
7				-1			-1	5				-1				
8					-1				5	-1			-1			
9						-1			-1	4	-1			-1		
10							-1			-1	4	-1			-1	
11								-1			-1	5				-1
12									-1				6	-1		
13										-1			-1	5	-1	
14											-1			-1	5	-1
15												-1			-1	6

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	6	-1			-1											
1	-1	5	-1			-1										
4	-1				5	-1			-1							
5		-1			-1	4	-1			-1						

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2		-1	5	-1			-1									
3			-1	6				-1								
6			-1			-1	4	-1			-1					
7				-1			-1	5				-1				

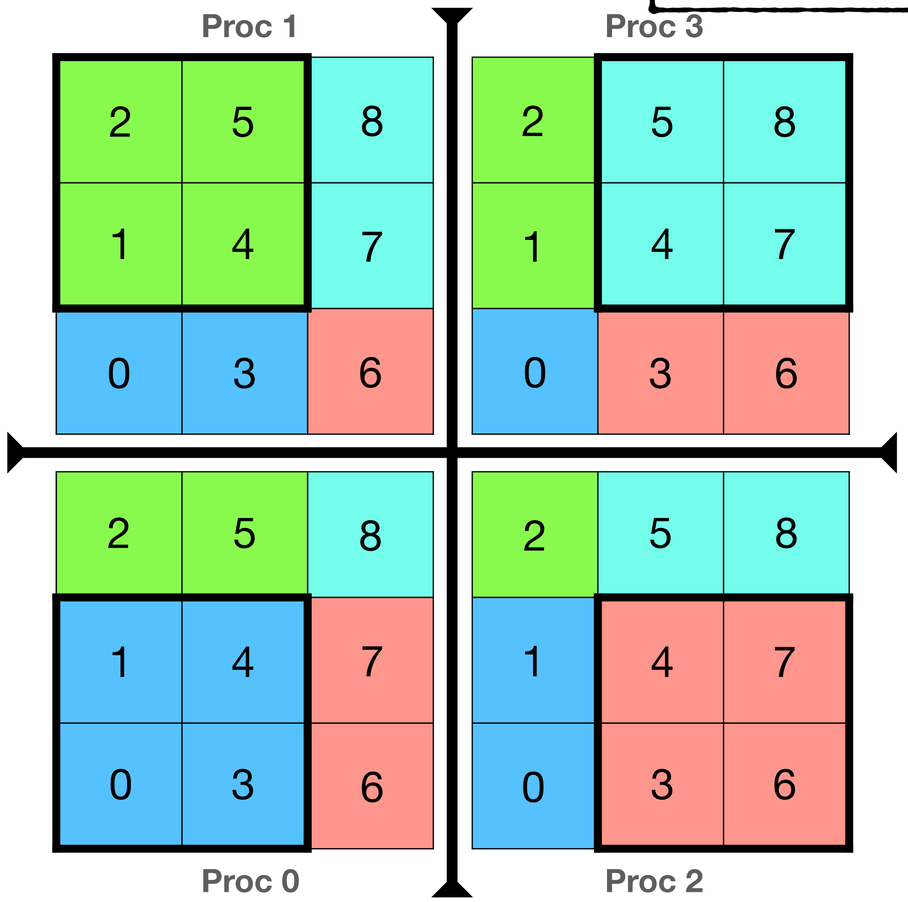
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
8					-1				5	-1			-1			
9						-1			-1	4	-1			-1		
12									-1				6	-1		
13										-1			-1	5	-1	

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
10							-1			-1	4	-1			-1	
11								-1			-1	5				-1
14											-1			-1	5	-1
15												-1			-1	6

Heat transfer

Data representation

Optimised layout!
Use local indices!



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	6	-1		-1												
1	-1	5	-1		-1											
2	-1			5	-1		-1									
3		-1		-1	4	-1		-1								

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	-1	5	-1		-1											
1		-1	6			-1										
2		-1		-1	4	-1		-1								
3			-1		-1	5			-1							

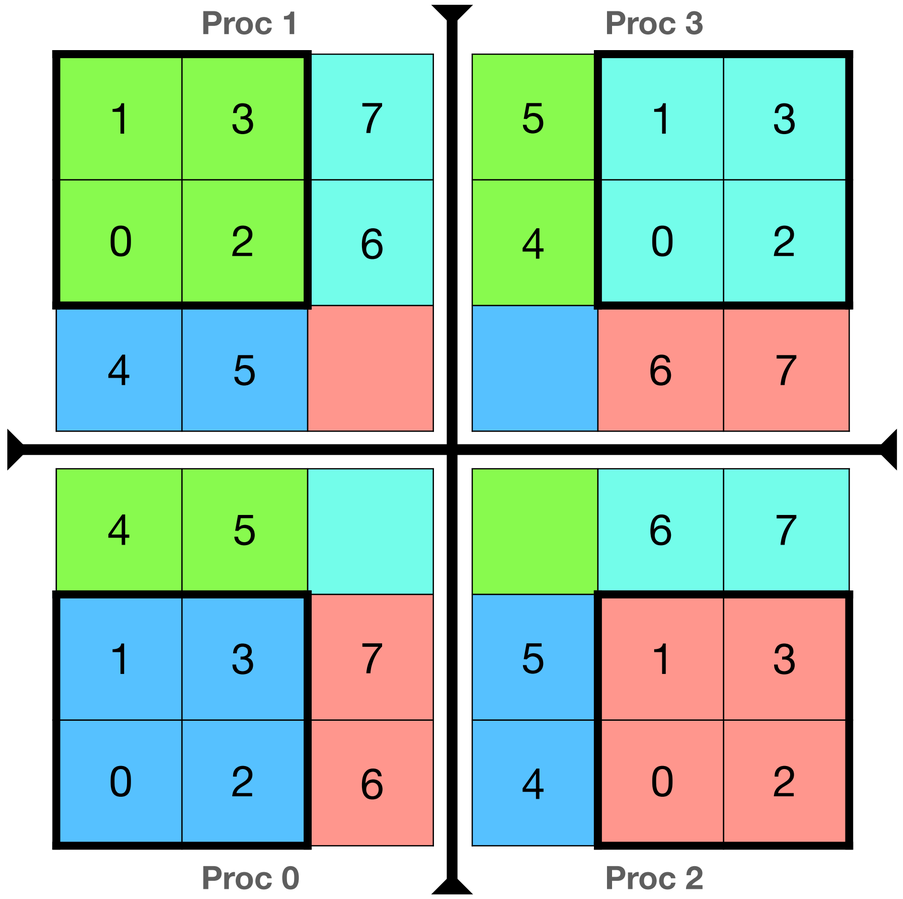
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	-1			5	-1		-1									
1		-1		-1	4	-1		-1								
2				-1			6	-1								
3					-1		-1	5	-1							

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0		-1		-1	4	-1		-1								
1			-1		-1	5			-1							
2					-1		-1	5	-1							
3						-1		-1	6							

Heat transfer

Data representation

Re-enumerateion!



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	6	-1	-1													
1	-1	5		-1	-1											
2	-1		5	-1			-1									
3		-1	-1	4		-1		-1								

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	5	-1	-1		-1											
1	-1	6		-1												
2	-1		4	-1		-1	-1									
3		-1	-1	5				-1								

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	5	-1	-1		-1											
1	-1	4		-1		-1	-1									
2	-1		6	-1												
3		-1	-1	5				-1								

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	4	-1	-1		-1		-1									
1	-1	5		-1		-1										
2	-1		5	-1				-1								
3		-1	-1	6												

Heat transfer

Data representation

Optimised storage!

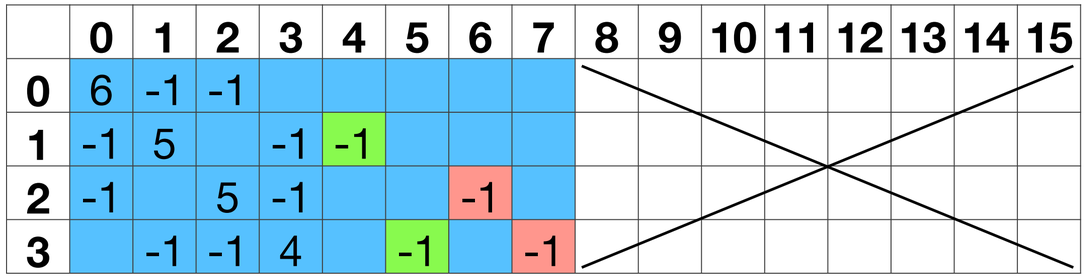
Memory reduction (~2x):

- Regular storage: $N_{loc} \times N_{glob}$
- Modified storage: $N_{loc} \times (N_{loc} + N_{halo})$

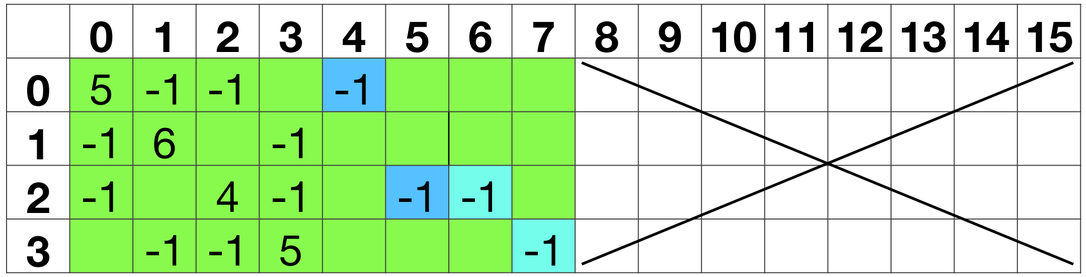
Easier access to halo values

Compact storage of the “local” values

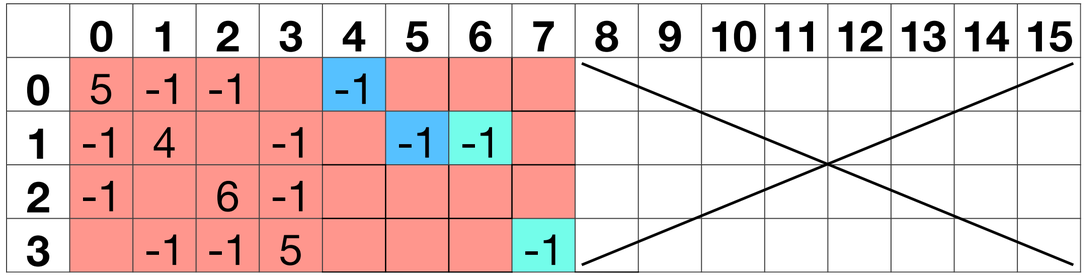
In many numerical codes the halo values are stored in a separate array



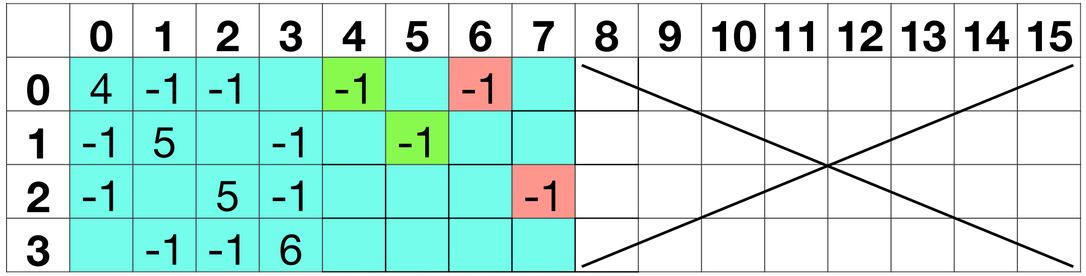
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	6	-1	-1													
1	-1	5		-1	-1											
2	-1		5	-1			-1									
3		-1	-1	4		-1		-1								



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	5	-1	-1		-1											
1	-1	6		-1												
2	-1		4	-1		-1	-1									
3		-1	-1	5				-1								



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	5	-1	-1		-1											
1	-1	4		-1		-1	-1									
2	-1		6	-1												
3		-1	-1	5				-1								



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	4	-1	-1		-1		-1									
1	-1	5		-1		-1										
2	-1		5	-1				-1								
3		-1	-1	6												

Heat transfer

Repository

- SSH to Snellius
- Clone the repository from: **https://github.com/sara-nl/prace_jacobi.git**
- Compile with *./make_all.sh <type>* (run *./make_all.sh* to see all available types)

```
$ git clone https://github.com/sara-nl/prace_jacobi.git
$ cd prace_jacobi
$ ./make_all.sh mpi
```

Heat transfer

Code structure

Matrix, vector and field representation

Auxiliary functions

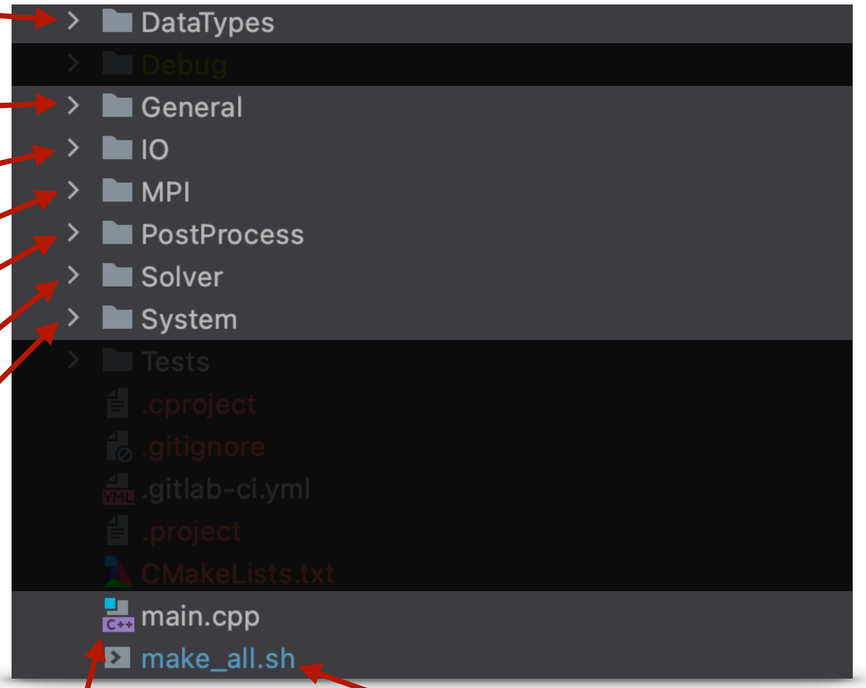
Input/output with the file system

MPI core: decomposition, general functions

gnuplot scripts for visualisation of the final field

Jacobi solver and mathematical operations

Numerical system

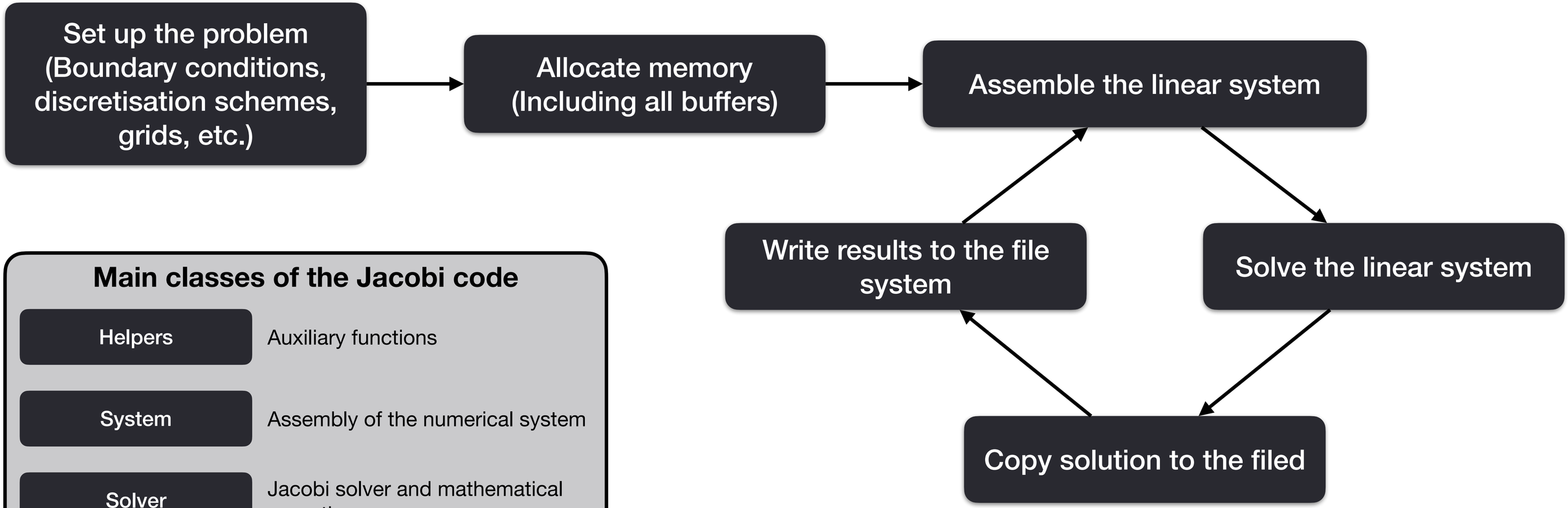


Main function

Make script

Heat transfer

Generic logic



Main classes of the Jacobi code

Helpers	Auxiliary functions
System	Assembly of the numerical system
Solver	Jacobi solver and mathematical operations
IO	Operations with the file system

Hands-on #1.1

Hands-on

Filling the empty space

- In ***Solver/solver.cpp*** implement the following (look for the ***NOT_IMPLEMENTED*** macro):
  - Vector copy: ***Solver::copyVector()***
  - Residual calculation: ***Solver::calculateResidual()***
  - L2-norm calculation: ***Solver::calculateNorm()***
- Test your implementation by compiling with:
  - *`./make_all.sh omp`* (will result in a serial binary)
- Run the executable without any arguments
- Visualise results with ***gnuplot***

```
void Solver::copyVector(Vector &vec_in, Vector &vec_out) {

    /*
     * for every n-th elements in `vec_in`
     *     assign element of vec_in(n) to vec_out(n)
     */
    NOT_IMPLEMENTED
}

void Solver::calculateResidual(Matrix &A, Vector &x,
                               Vector &b, Vector &res) {

    /*
     * assign `b` to `res`
     *
     * for vector `x` and matrix `A`, the residual `res`
     * is calculated as:
     *     res(i) = b(i) - sum( A(i, j) * x(j) )
     */
    NOT_IMPLEMENTED
}

double Solver::calculateNorm(Vector &vec) {

    /*
     * for vector `vec` with n elements
     *   L2-norm = sqrt( sum( vec(n) * vec(n) ) )
     *             |     \_____________________/|
     *             | local for every MPI process|
     *              \___________________________/
     *           one value for all MPI processes
     */
    NOT_IMPLEMENTED

    return 0.0;
}
```

Hands-on

Filling the empty space

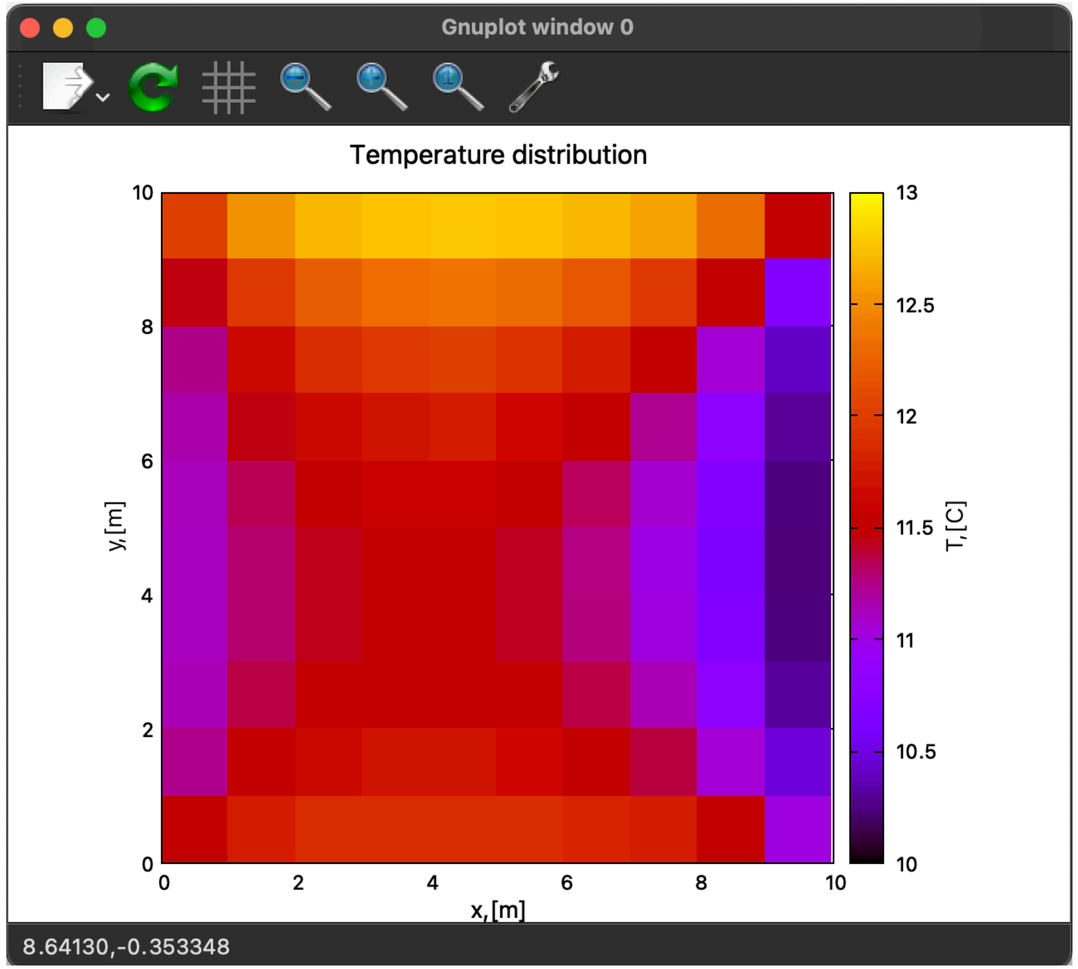
- Class ***Vector*** is inherited from the ***Matrix*** class
- Most important methods:

```
/*
 * Get number of local rows
 * (including halo cells)
 */
Matrix::numRows();

/*
 * Get number of local columns
 * (including halo cells)
 */
Matrix::numCols();

/*
 * Get number of local elements
 * (excluding halo cells)
 */
Matrix::getLocElts();
```

```
void Solver::copyVector(Vector &vec_in, Vector &vec_out) {

    /*
     * for every n-th elements in `vec_in`
     *     assign element of vec_in(n) to vec_out(n)
     */
    NOT_IMPLEMENTED
}

void Solver::calculateResidual(Matrix &A, Vector &x,
                               Vector &b, Vector &res) {

    /*
     * assign `b` to `res`
     *
     * for vector `x` and matrix `A`, the residual `res`
     * is calculated as:
     *     res(i) = b(i) - sum( A(i, j) * x(j) )
     */
    NOT_IMPLEMENTED
}

double Solver::calculateNorm(Vector &vec) {

    /*
     * for vector `vec` with n elements
     *   L2-norm = sqrt( sum( vec(n) * vec(n) ) )
     *             |     \_____________________/|
     *             | local for every MPI process|
     *              \___________________________/
     *           one value for all MPI processes
     */
    NOT_IMPLEMENTED

    return 0.0;
}
```

Hands-on

Building and execution




```
$ module load 2022 foss/2022a
$ ./make_all.sh omp
$
$ cat first_run.sh
#!/bin/bash
#SBATCH --reservation=mossd_cpu_course
#SBATCH -p rome
#SBATCH -n 1
#SBATCH --job-name=first_run
#SBATCH --output=first_run.out
#SBATCH --error=first_run.err
#SBATCH -time=0:05:00

module load 2022 foss/2022a
srun ./a.out
$
$ sbatch first_run.sh
```

Job script

```
$ cat first_run.out
0     0.588603
1     0.392595
…
352   1.02557e-06
353   9.92268e-07
Writing results to file: output.dat
Elapsed time (Jacobi): 0.005982s.
Elapsed time (IO): 0.012528s.
$
$ # if used “ssh -X” or “ssh -Y”
$ module load gnuplot/5.4.4-GCCcore-11.3.0
$ cp PostProcess/plot_omp.plt .
$ gnuplot
gnuplot> load “plot_omp.plt”
```

Output from the job

Hands-on #1.2

Hands-on

Filling the empty space

- Add MPI communications:
  - In the iterative loop, the data must be exchanged between the real and the halo elements of the vector
    - Find the right implementation in ***DataTypes/vector.cpp***
    - Insert the call for the function into the right place

```
while ( (iter < max_iter) && (residual_norm > tolerance) ) {

    for(int i = A.numRows() - 1; i >= 0; i--) {
        double diag = 1.;           // Diagonal element
        double sigma = 0.0;         // Just a temporary value

        x(i) = b(i);

        for(int j = 0; j < A.numCols(); ++j) {
            if (j != i)
                sigma = sigma + A(i, j) * x_old(j);
            else
                diag = A(i, j);
        }
        x(i) = (x(i) - sigma) * omega / diag;
    }

    for(int i = 0; i < x.numRows(); ++i) {
        x(i) += (1 - omega) * x_old(i);
        x_old(i) = x(i);
    }

    calculateResidual(A, x, b, res);
    residual_norm = calculateNorm(res) / calculateNorm(b);

    if (my_rank == 0)
        cout << iter << '\t' << residual_norm << endl;

    ++iter;
}
```

Hands-on

Filling the empty space

- Note that the L2-norm calculation should actually fire up an MPI call
  - Which one?
  - Look at ***MPI/common.cpp*** to find the right function
- The ***MPI/common.cpp*** file has a lot of functions marked with ***NOT_IMPLEMENTED***. Replace this macro with correct MPI calls, use MPI_COMM_WORLD as a communicator
- Compile the code with the `*mpi*` flag and execute it

```
double Solver::calculateNorm(Vector &vec) {

    /*
     * for vector `vec` with n elements
     *   L2-norm = sqrt( sum( vec(n) * vec(n) ) )
     *             |     \_____________________/|
     *             | local for every MPI process|
     *              \___________________________/
     *           one value for all MPI processes
     */
    NOT_IMPLEMENTED

    return 0.0;
}
```

```
$ module load 2022 foss/2022a
$ ./make_all.sh mpi
$
$ cat first_run.sh
#!/bin/bash
#SBATCH --reservation=mossd_cpu_course
#SBATCH -p rome
#SBATCH -n 4
#SBATCH --job-name=first_run
#SBATCH --output=first_run.out
#SBATCH --error=first_run.err
#SBATCH -time=0:05:00

module load 2022 foss/2022a
srun ./a.out
$
$ sbatch first_run.sh
```

Job script

Tools

Tools

Overview: HPC tools

Tool name	Costs	Description
ARM DDT	Non-free	Full featured graphical, parallel debugger
HPCToolkit	Free	Integrated suite of tools for parallel program performance analysis
Intel One API	Free under certain conditions	Stack of different performance analysis and debuggingtools (MPI/OpenMP/SIMD)
Valgrind	Free	Memory errors debugging tool
TotalView	Non-free	Full featured graphical, parallel debugger
Vampir	Non-free	Full featured trace visualizer for parallel program OTF trace files
memP	Free	Lightweight memory profiling tool
mpiP	Free	Lightweight MPI profiling tool
MUST	Free	MPI runtime error detection tool
PAPI	Free	A standardized and portable API for accessing performance counter hardware
likwid	Free	A **tool** to measure hardware performance counters
TAU	Free	Full featured parallel program performance analyses toolkit
Extrae	Free	MPI/OpenMP profiler
Scalasca	Free	performance analysis tool for MPI+OpenMP
Darshan	Free	IO profiler
nvprof	Free	Thread profiler (inc. GPU) from NVIDIA
gdb	Free	Standard GNU debugger
ARM MAP	Non-free	performance analysis tool for MPI+OpenMP
uProf	Free	performance analysis tool for MPI+OpenMP
gprof	Free	Standard unix/linux profiling utility

Different support for:

- Hardware
- Parallelisation strategies
- Compilers
- Interface

Other important aspects:

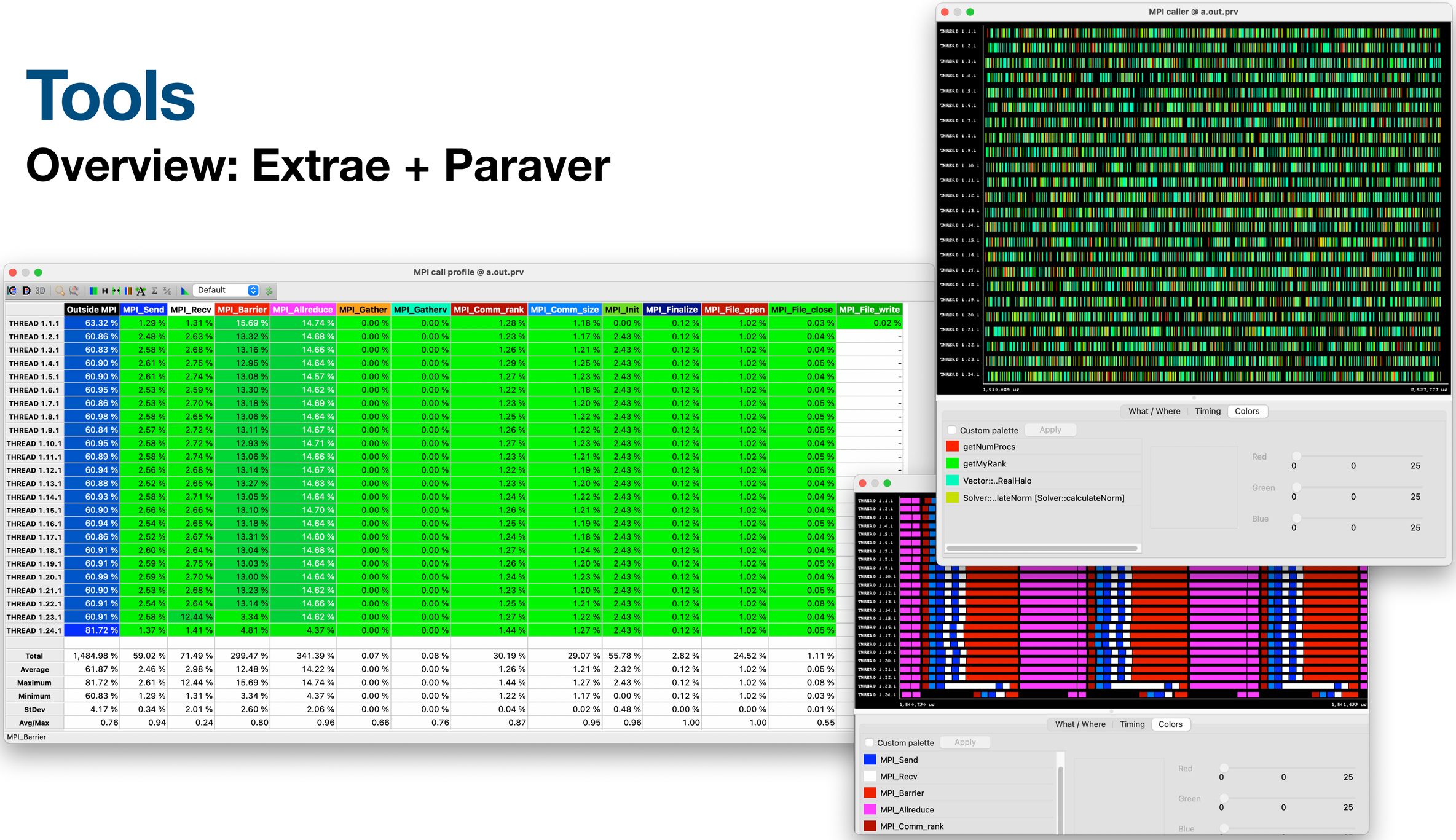
- Learning curve
- Completeness of the reports
- Costs and licenses
- Community support
- Documentation

Good overview:

https://hpc.llnl.gov/software/development-environment-software

Tools

Overview: Extrae + Paraver



	Outside MPI	MPI_Send	MPI_Recv	MPI_Barrier	MPI_Allreduce	MPI_Gather	MPI_Gatherv	MPI_Comm_rank	MPI_Comm_size	MPI_Init	MPI_Finalize	MPI_File_open	MPI_File_close	MPI_File_write
THREAD 1.1.1	63.32 %	1.29 %	1.31 %	15.69 %	14.74 %	0.00 %	0.00 %	1.28 %	1.18 %	0.00 %	0.12 %	1.02 %	0.03 %	0.02 %
THREAD 1.2.1	60.86 %	2.48 %	2.63 %	13.32 %	14.68 %	0.00 %	0.00 %	1.23 %	1.17 %	2.43 %	0.12 %	1.02 %	0.04 %	-
THREAD 1.3.1	60.83 %	2.58 %	2.68 %	13.16 %	14.66 %	0.00 %	0.00 %	1.26 %	1.21 %	2.43 %	0.12 %	1.02 %	0.04 %	-
THREAD 1.4.1	60.90 %	2.61 %	2.75 %	12.95 %	14.64 %	0.00 %	0.00 %	1.29 %	1.25 %	2.43 %	0.12 %	1.02 %	0.05 %	-
THREAD 1.5.1	60.90 %	2.61 %	2.74 %	13.08 %	14.57 %	0.00 %	0.00 %	1.27 %	1.23 %	2.43 %	0.12 %	1.02 %	0.04 %	-
THREAD 1.6.1	60.95 %	2.53 %	2.59 %	13.30 %	14.62 %	0.00 %	0.00 %	1.22 %	1.18 %	2.43 %	0.12 %	1.02 %	0.04 %	-
THREAD 1.7.1	60.86 %	2.53 %	2.70 %	13.18 %	14.69 %	0.00 %	0.00 %	1.23 %	1.20 %	2.43 %	0.12 %	1.02 %	0.05 %	-
THREAD 1.8.1	60.98 %	2.58 %	2.65 %	13.06 %	14.64 %	0.00 %	0.00 %	1.25 %	1.22 %	2.43 %	0.12 %	1.02 %	0.05 %	-
THREAD 1.9.1	60.84 %	2.57 %	2.72 %	13.11 %	14.67 %	0.00 %	0.00 %	1.26 %	1.22 %	2.43 %	0.12 %	1.02 %	0.05 %	-
THREAD 1.10.1	60.95 %	2.58 %	2.72 %	12.93 %	14.71 %	0.00 %	0.00 %	1.27 %	1.23 %	2.43 %	0.12 %	1.02 %	0.04 %	-
THREAD 1.11.1	60.89 %	2.58 %	2.74 %	13.06 %	14.66 %	0.00 %	0.00 %	1.23 %	1.21 %	2.43 %	0.12 %	1.02 %	0.05 %	-
THREAD 1.12.1	60.94 %	2.56 %	2.68 %	13.14 %	14.67 %	0.00 %	0.00 %	1.22 %	1.19 %	2.43 %	0.12 %	1.02 %	0.05 %	-
THREAD 1.13.1	60.88 %	2.52 %	2.65 %	13.27 %	14.63 %	0.00 %	0.00 %	1.23 %	1.20 %	2.43 %	0.12 %	1.02 %	0.04 %	-
THREAD 1.14.1	60.93 %	2.58 %	2.71 %	13.05 %	14.64 %	0.00 %	0.00 %	1.24 %	1.22 %	2.43 %	0.12 %	1.02 %	0.04 %	-
THREAD 1.15.1	60.90 %	2.56 %	2.66 %	13.10 %	14.70 %	0.00 %	0.00 %	1.26 %	1.21 %	2.43 %	0.12 %	1.02 %	0.04 %	-
THREAD 1.16.1	60.94 %	2.54 %	2.65 %	13.18 %	14.64 %	0.00 %	0.00 %	1.25 %	1.19 %	2.43 %	0.12 %	1.02 %	0.05 %	-
THREAD 1.17.1	60.86 %	2.52 %	2.67 %	13.31 %	14.60 %	0.00 %	0.00 %	1.24 %	1.18 %	2.43 %	0.12 %	1.02 %	0.04 %	-
THREAD 1.18.1	60.91 %	2.60 %	2.64 %	13.04 %	14.68 %	0.00 %	0.00 %	1.27 %	1.24 %	2.43 %	0.12 %	1.02 %	0.04 %	-
THREAD 1.19.1	60.91 %	2.59 %	2.75 %	13.03 %	14.64 %	0.00 %	0.00 %	1.26 %	1.20 %	2.43 %	0.12 %	1.02 %	0.05 %	-
THREAD 1.20.1	60.99 %	2.59 %	2.70 %	13.00 %	14.64 %	0.00 %	0.00 %	1.24 %	1.23 %	2.43 %	0.12 %	1.02 %	0.04 %	-
THREAD 1.21.1	60.90 %	2.53 %	2.68 %	13.23 %	14.62 %	0.00 %	0.00 %	1.23 %	1.20 %	2.43 %	0.12 %	1.02 %	0.05 %	-
THREAD 1.22.1	60.91 %	2.54 %	2.64 %	13.14 %	14.66 %	0.00 %	0.00 %	1.25 %	1.21 %	2.43 %	0.12 %	1.02 %	0.08 %	-
THREAD 1.23.1	60.91 %	2.58 %	12.44 %	3.34 %	14.62 %	0.00 %	0.00 %	1.27 %	1.22 %	2.43 %	0.12 %	1.02 %	0.04 %	-
THREAD 1.24.1	81.72 %	1.37 %	1.41 %	4.81 %	4.37 %	0.00 %	0.00 %	1.44 %	1.27 %	2.43 %	0.12 %	1.02 %	0.05 %	-
Total	1,484.98 %	59.02 %	71.49 %	299.47 %	341.39 %	0.07 %	0.08 %	30.19 %	29.07 %	55.78 %	2.82 %	24.52 %	1.11 %	
Average	61.87 %	2.46 %	2.98 %	12.48 %	14.22 %	0.00 %	0.00 %	1.26 %	1.21 %	2.32 %	0.12 %	1.02 %	0.05 %	
Maximum	81.72 %	2.61 %	12.44 %	15.69 %	14.74 %	0.00 %	0.00 %	1.44 %	1.27 %	2.43 %	0.12 %	1.02 %	0.08 %	
Minimum	60.83 %	1.29 %	1.31 %	3.34 %	4.37 %	0.00 %	0.00 %	1.22 %	1.17 %	0.00 %	0.12 %	1.02 %	0.03 %	
StDev	4.17 %	0.34 %	2.01 %	2.60 %	2.06 %	0.00 %	0.00 %	0.04 %	0.02 %	0.48 %	0.00 %	0.00 %	0.01 %	
Avg/Max	0.76	0.94	0.24	0.80	0.96	0.66	0.76	0.87	0.95	0.96	1.00	1.00	0.55	

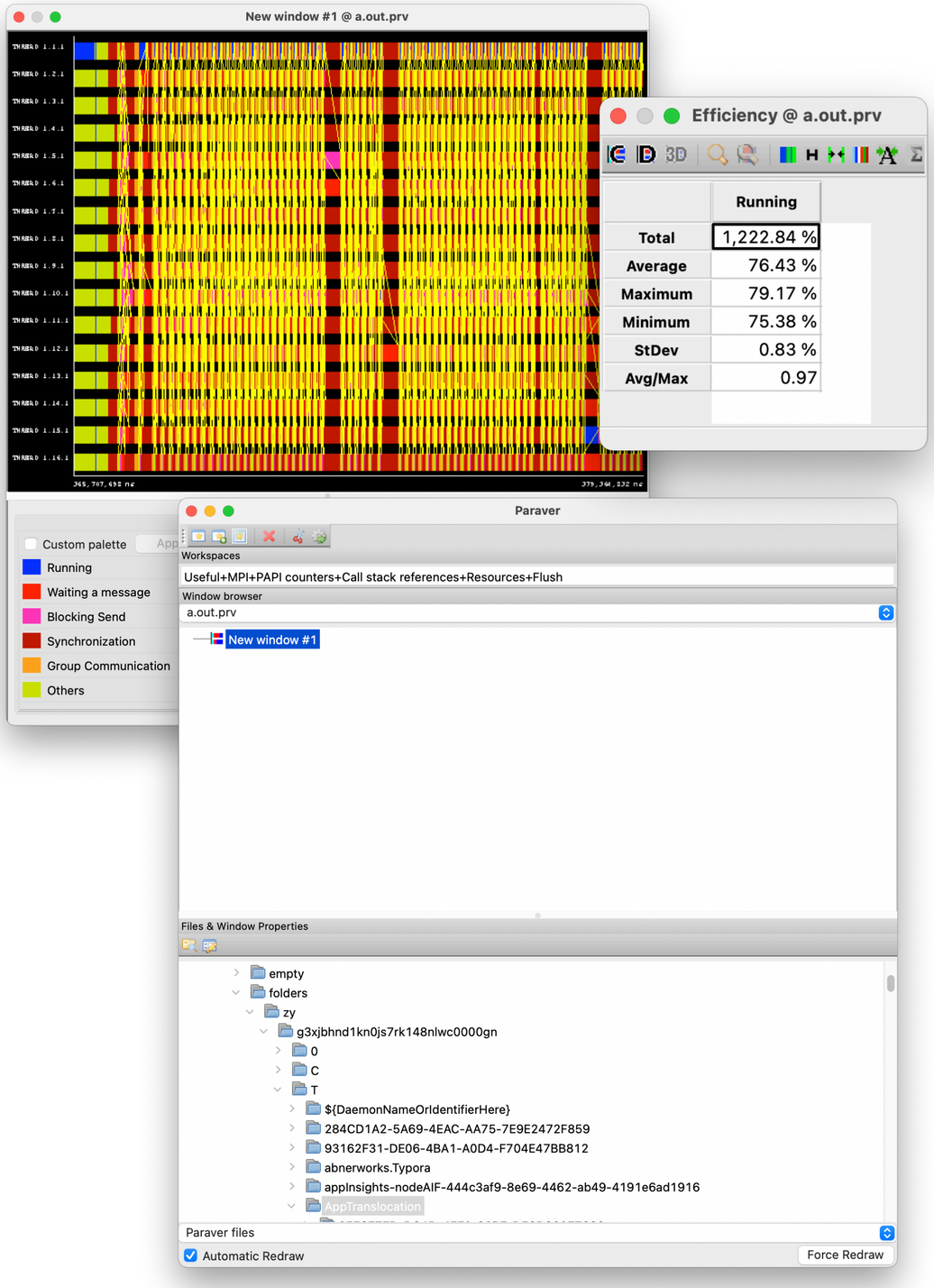
MPI_Barrier

Hands-on #1.3

Hands-on

Profiling

- Profile the code with ***Extrae*** (use *32x32* grid)
- Download ***Paraver*** to your local machine: https://tools.bsc.es/downloads
- Find the bottlenecks using ***Paraver*** (see `Hints` menu)
  - Visualise the ***“MPI profile”***
  - Visualise the ***“Profile caller line”***
  - Visualise the ***“Caller function”***



Hands-on

Profiling

- Comment out the ***io.WriteFile()*** call in ***main.cpp***
- Copy configuration files to the project folder
- Modify the ***trace.sh*** script:
  - Replace “double dots” with a single “dot” in the **EXTRAE_CONFIG_FILE** environment variable
  - Comment out **LD_PRELOAD** for Fortran apps and uncomment it for C apps

```
$ module load 2022
$ module load Extrae/4.0.4-gompi-2022a
$
$ cd prace_jacobi
$
$ cp ${EBROOTEXTRAE}/share/example/MPI/ld-preload/trace.sh .
$ cp ${EBROOTEXTRAE}/share/example/MPI/extrae.xml .
```

trace.sh

```
#!/bin/bash

source /sw/arch/RHEL8/EB_production/2022/software/Extrae/
4.0.4-gompi-2022a/etc/extrae.sh

export EXTRAE_CONFIG_FILE=./extrae.xml
# For C apps
export LD_PRELOAD=${EXTRAE_HOME}/lib/libmpitrace.so
# For Fortran apps
#export LD_PRELOAD=${EXTRAE_HOME}/lib/libmpitracef.so

## Run the desired program
$*
```

Hands-on

Profiling

- Execute a test on 16 cores (Rome partition) with 10x10 grid and 4x4 decomposition
- Merge intermediate trace files into a single Paraver trace file at the end of the job script
- Visualise and analyse results
  - Copy the following files to your local machine: **a.out.pcf**, **a.out.prv**, **a.out.row**
  - Open a.out.prv using Paraver

test_extrae.sh

```
#!/bin/bash
#SBATCH –reservation=mossd_cpu_course
#SBATCH -p rome
#SBATCH -n 16
#SBATCH --job-name=mpi_training
#SBATCH --output=out.p_np_16
#SBATCH --error=err.p_np_16
#SBATCH --time=0:05:00

module load 2022 foss/2022a Extrae/4.0.4-gompi-2022a

### Export trace.sh and executable
export EXE="./a.out -s 10 10 -d 4 4"

### Run the application
srun ./trace.sh $EXE

### Merge trace files
mpi2prv -f TRACE.mpits
```

```
Use the following keys:
  -s – set number of the grid cells in each direction (i j)
  -d – set decomposition for each direction (i j)
       (doesn’t affect the METIS decomposition, but should be
        set anyway!)
```

Hands-on #1.4

Hands-on

Simple profiling

- Measure the elapsed time of the Jacobi solver with the “foss/2022a” toolchain:
  - use *32x32* grid, 16 processes
  - see “Elapsed time (Jacobi):”
- Change the toolchain to “intel/2022a” (i.e. change OpenMPI to IMPI)
  - What happened to the elapsed time with IMPI?

```
$ cat simple_profiling.sh
#!/bin/bash
#SBATCH --reservation=mossd_cpu_course
#SBATCH -p rome
#SBATCH -n 16
#SBATCH --job-name=simple_profiling
#SBATCH --output=simple_profiling.out
#SBATCH --error=simple_profiling.err
#SBATCH --time=0:05:00

echo "Open MPI"
module purge
module load 2022 foss/2022a
./make_all.sh mpi
srun ./a.out -s 32 32 -d 1 16

echo "Intel MPI"
module purge
module load 2022 intel/2022a
./make_all.sh mpi
srun ./a.out -s 32 32 -d 1 16
$
$ sbatch simple_profiling.sh
```

Job script

```
$ cat simple_profiling.out | grep "Elapsed time (Jacobi):"
Elapsed time (Jacobi): 0.153555s.
Elapsed time (Jacobi): 3.860522s.
```

Results

Hands-on

Simple profiling

- **MPI_Wtime()** returns a floating-point number of seconds, representing elapsed wall-clock time since some moment in the past.
- **MPI standard doesn’t guarantee that clocks are synchronised!**
- **MPI_WTIME_IS_GLOBAL** - boolean variable that indicates whether clocks are synchronized. The value returned for **MPI_WTIME_IS_GLOBAL** is 1 if clocks at all processes in **MPI_COMM_WORLD** are synchronized, 0 otherwise.
- Check if clocks are synchronised.

```
void runProblem(int argc, char** argv) {

    // ...

    int flag = 0;
    int *value_ptr;

    MPI_Comm_get_attr(MPI_COMM_WORLD,
                      MPI_WTIME_IS_GLOBAL,
                      &value_ptr,
                      &flag);

    printByRoot("MPI_WTIME_IS_GLOBAL: "
                + std::to_string(*value_ptr));
}
```

Hands-on

Simple profiling

- **OpenMPI**
  - The boolean variable **MPI_WTIME_IS_GLOBAL**, a predefined attribute key that indicates whether clocks are synchronized, **does not have a valid value** in Open MPI, as the clocks are not guaranteed to be synchronized.

Hands-on

Simple profiling

- **Solution #1:**
  - Use other functions that do not rely on an arbitrary start moment in the past, e.g. “gettimeofday()” **(expensive, limited precision)**
- **Solution #2:**
  - report min, max and average elapsed times using **MPI_Wtime() (fine resolution)**

```
double Helpers::tic() {

    double current_time = 0.0;
    struct timeval tv;

    gettimeofday(&tv, NULL);
    current_time = (double)tv.tv_sec
                   + 1.0e-6 * (double) tv.tv_usec;

    return current_time;
}
```

```
void reportElapsedTime(double start, double end,
                       const std::string &message)
{
     double elp_time = end - start;
     double elp_time_min = elp_time;
     double elp_time_max = elp_time;

     findGlobalMin(elp_time_min);
     findGlobalMax(elp_time_max);
     findGlobalSum(elp_time);

     printByRoot("Elapsed time (" + message + "): ");
     printByRoot("  Min: "
           + std::to_string(elp_time_min) + "s.");
     printByRoot("  Max: "
           + std::to_string(elp_time_max) + "s.");
     printByRoot("  Avg: "
           + std::to_string(elp_time / getNumProcs()) +
           "s.");
}
```

Hands-on

Simple profiling

- **Solution #3:**
  - Force custom synchronisation using MPI:
    - Sync all processes
    - Get time from one of the processes with **MPI_Wtime()**
    - Broadcast the measured time to all processes and calculate local offsets
    - Adjust subsequent timings using these offsets

```
double syncTime() {
    double local_time, base_time, offset;

    int my_rank = getMyRank();

    // Synchronize all processes
    MPI_Barrier(MPI_COMM_WORLD);

    // Get the local time
    local_time = MPI_Wtime();

    // Rank 0 gets the reference time
    if (my_rank == 0) base_time = local_time;

    // Broadcast the reference time to all processes
    MPI_Bcast(&base_time, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);

    // Calculate the offset for each process
    offset = base_time - local_time;

    return offset;
}

...

void runProblem(int argc, char** argv) {
    ...

    double offset = syncTime();
    elp_time[0] = helpers.tic() + offset;
    // Some work
    elp_time[1] = helpers.toc() + offset;

    ...
}
```

@16:00
Dr. Matthias Moller, TUD
“Case studies of OpenMP & MPI”