NSSC 2 Exam prep

Contents

Notes

1 Exam 15. April 2021

1.1 MPI

Consider the implementation of an MPI-parallelized numerical integrator in one dimension. Show the pseudocode of a possible implementation and focus on a proper distribution of workload among the processes. Assume variables as needed; ignore implementation specifics of the integration function. Explicitly highlight and discuss the specific, necessary MPI statements, communication strategy and parallel efficiency (specific values for parameters are not important)

Algorithm 1: Pseudo code integration

```
1  //initialize
2  MPI_Init()
3  // create MPI Communicator
4  MPI_Comm comm;
5  comm = MPI_COMM_WORLD;
6  // get my rank and total number of processes
7  MPI_Comm_rank(comm, &my_rank);
8  MPI_Comm_size(comm, &nproc);
9  // set lower/upper bound
10 a=0;
11 b=10;
12 //set number of increments for each process
13 n=100;
14 //set length of increment
15 h=(b-a)/n/nproc
16 //set lower limit for my rank
17 ai=a+myrank*n*h
18 //perform integration
19 my_inegration=integrate(ai,h,n)
20 //use MPI_Gather to gather results on master process (rank=0)
21 MPI_Gather(&my_integration,1,MPI_FLOAT,rec_buf,1,MPI_FLOAT,0,comm)
22 //loop over all results and add them up
23 if (my_rank==0){
24     integral_sum=0.0
25     for (i=0;i++;i<nproc){
26     integral_sum+=rec_buf(i)
27     }
28 }
29 //end MPI program
30 MPI_Finalize()
```

Algorithm 2: Variante by Hiti. Beschreibt 2 Möglichkeiten

```
def integrate_mpi(function, a, b)
    int comm_size, my_rank
    double start, end, length, local_result, global_result


    MPI_Init()
    MPI_Comm_size(MPI_COMM_WORLD, &comm_size)
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank)

    length = (b-a)
    start = my_rank * length / (double) comm_size
    end = start + length / (double) comm_size

    double local_result = integrate(function, a, b)

    // Begin Version 1
    MPI_Allreduce(
        sendbuf = local_result,
        resvbuf = global_result,
        count = 1,
        dtype = MPI_DOUBLE,
        op = MPI_SUM,
        MPI_COMM_WORLD
    )
    // End Version 1

    // Begin Version 2
    vector<double> all_local_results(comm_size)
    if(my_ranke == 0)
    {
        recvcount=1
        recvbuf=&all_local_results
    } else {
        recvcount=0
        recvbuf=nullptr
    }

    MPI_Gather(
        sendbuf=&local_result,
        sendcout=1,
        sendtype=MPI_DOUBLE,

        recvbuf=recvbuf,
        recvcnt=recvcount,
        recvtype=MPI_DOUBLE,

        root=0
        MPI_COMM_WORLD
    )
    if(my_rank == 0)
    {
        global_result = sum(results)
    }
    // End Version 2

    MPI_Finalize() // Nicht vergessen!
    return global_result
```

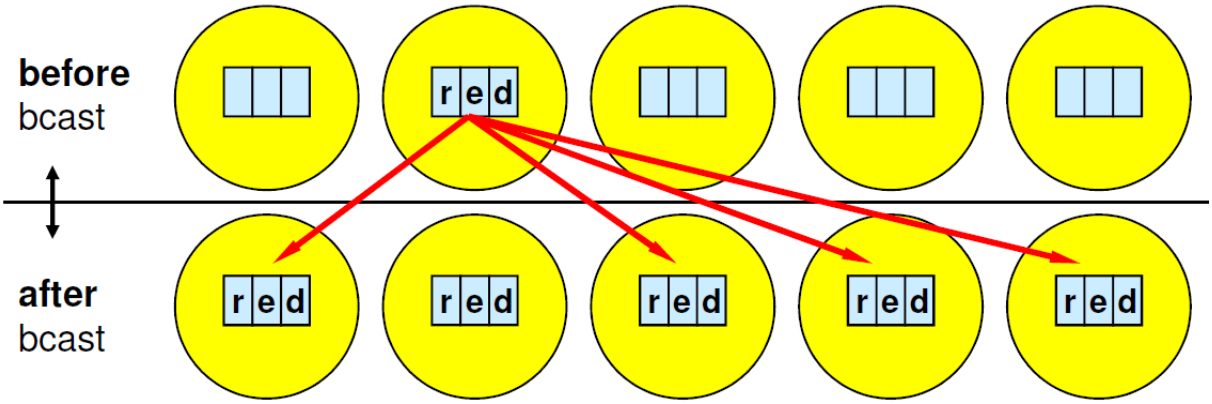
1.2 MPI

Explain via drawings and additional explanations the following collective operations: MPI_Bcast() and MPI_Reduce().

- buf = starting address of buffer
- count = number of entries in buffer
- datatype = data type of buffer
- root = rank of broadcast root
- comm = communicator

Algorithm 3: MPI_Bcast

```
1 int MPI_Bcast(void* buf,
2 int count,
3 MPI_Datatype datatype,
4 int root,
5 MPI_Comm comm)
```

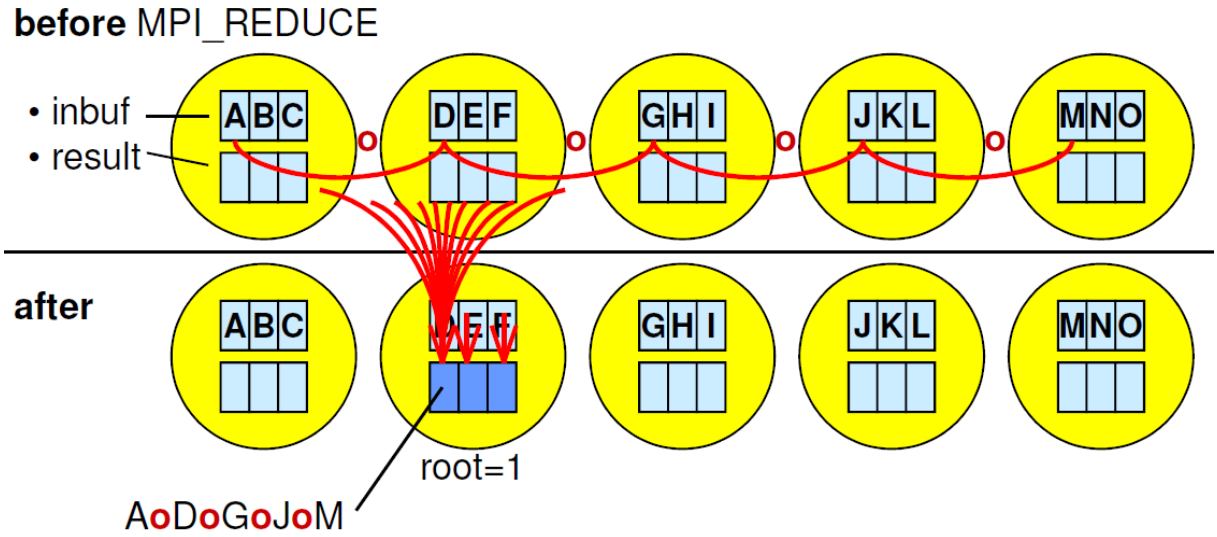


A Broadcast (MPI_Bcast) is used to send data from a single process with rank root to all other processes of the group.
MPI_Reduce can be used to perform different kind of reduction operations like:

- maximum
- minimum
- sum
- product

Algorithm 4: Example for MPI_Reduce with summation

```
1 MPI_Reduce(&inbuf,&resultbuf,1,MPI_INT,MPI_SUM,root,MPI_COMM_WORLD);
```



1.3 ODE

Explain the Euler Method and discuss the implicit and explicit approximations

1.3.1 Explicit Euler

Given

$$y' = f(t, y(t)) \tag{1}$$

Equidistant discretization in t domain vom t_a to t_b

$$h = \frac{t_b - t_a}{N} \tag{2}$$

$$t_j = t_a + jh \tag{3}$$

with j=0,1,2,.....,N
Forward Difference (first order approximation)

$$y'(t_{j-1}) = \frac{y(t_j) - y(t_{j-1})}{h} + O(h^2) \tag{4}$$

$$y(t_j) = y(t_{j-1}) + hy'(t_{j-1}) + O(h^2) \tag{5}$$

$$y(t_j) = y(t_{j-1}) + hf(t_{j-1}, y(t_{j-1})) + O(h^2) \tag{6}$$

Explicit scheme:

$$u_0 = y_0 \tag{7}$$

$$u_{j+1} = u_j + hf(t_j, u_j) \tag{8}$$

1.3.2 Implicit Euler

Given

$$y' = f(t, y(t)) \tag{9}$$

Equidistant discretization in t domain vom t_a to t_b

$$h = \frac{t_b - t_a}{N} \tag{10}$$

$$t_j = t_a + jh \tag{11}$$

with j=0,1,2,.....,N
Backward Difference (first order approximation)

$$y'(t_j) = \frac{y(t_j) - y(t_{j-1})}{h} + O(h^2) \tag{12}$$

$$y(t_j) = y(t_{j-1}) + hy'(t_j) + O(h^2) \tag{13}$$

$$y(t_j) = y(t_{j-1}) + hf(t_j, y(t_j)) + O(h^2) \tag{14}$$

Implicit scheme:

$$u_0 = y_0 \tag{15}$$

$$u_{j+1} = u_j + hf(t_{j+1}, u_{j+1}) \tag{16}$$

1.4 ODE

Explain the key difference between implicit and explicit methods for the numerical solution of ODEs and comment on the consequences with respect to computational demand

1.4.1 Differences explicit/implicit

Explicit:

- Forward Difference
- Demands small step size

Implicit

- Backwards Difference
- Stable for all step sizes

Both Methods:

- single step methods
- Produce local truncation error $O(h^2)$
- Produce global error of $O(h)$

consequence respect computational demand ????

1.5 Consider Poissons's requation $\nabla^2 u = \nabla \cdot \nabla u = f$ and the integral form $\int \nabla \nabla u = \oint \nabla u \, dA = \int f \, dV$.

- a) What theorem was used in the integral form above ?
- b) Discuss the step (and graphically show the discrete gradients!) required for a discretization of the integral form of Poissons's equation using the finite volume method
- c) Why are finite volume schemes considered conservative

a) Divergence theorem

b)

- Decomposing the domain into descrete cells (triangles, boxes)
- Construct a corresponding dual representation (control volumes)
- setup of neighbour information between control volumes sharing surface regions
- Approximate surface integrals on shared surfaces
- Define surface integrals on domain boundary according to boundary conditions
- Assemble linear system of equations

c) Because the flux entering a given volume is identical to that leaving the adjacent volume, FVM schemes are considered conservative

1.6 Give an example for a second order linear elliptic operator

As an example we consider $A(x) = I$, $b = 0$ and $c = 0$, where $I \in R^{dxd}$ denotes the identity matrix. This yields the Poisson equation

$$-\Delta u = f \tag{17}$$

In order for (17) to have a unique solution we have to impose boundary conditions. For instance

- u = 0 on $\partial\Omega$ (homogeneous Dirichlet boundary conditions)
- $\Delta u \cdot n = 0$ on $\partial\Omega$ (homogeneous Neumann boundary conditions)

where n denotes the outward pointing normal vector field along ∂
Thus Δ is a second order linear elliptic operator

1.7 What is a second order linear partial differential operator in divergence form ? Give an example

Under a second order linear partial differential operator in divergence form we understand:

$$Lu(x) := -div(A(x)\nabla u(x)) + b(x) \cdot \nabla u(x) + c(x)u(x) \; for x \in \Omega \tag{18}$$

where $A : \bar{\Omega} \to R^{dxd}$ is a matrix-valued function, $b : \bar{\Omega} \to R^d$ and $c : \bar{\Omega} \to R$ are given functions. For given $f : \bar{\Omega} \to R$ one obtains the associated differential equation:

$$Lu(x) = f(x) \; for x \in \Omega \tag{19}$$

1.8 List the five steps of an entire, practical molecular dynamics simulation

- Obtain the electronic ground state energy $E_0(x_L)$
- Use E_0 an effective potential energy for the nuclei
- Compute its derivatives (forces)
- Integrate equations of motion
- Update coordinates and velocities

1.9 Describe (briefly and in words, no implementation needed) the process to detect cluster formation in the post-processing step of a molecular dynamics simulation

- define a notion of connectivity (distance, angels, etc.)
- build a graph (every molecule is a node)

Apply a search algorithm (breadth-first search, depth-first search) to find connected components

1.10 Describe, schematically, how a neural-network potential works. Use a flow diagram where the inputs are atomic coordinates and the output is the potential energy (ignore forces!)

kein plan wie das geht

1.11 Consider the 1D dimensionless Diffusion equation in cartesian coordinates

$$\frac{\partial C}{\partial t} = D\frac{\partial^2 C}{\partial x^2} \tag{20}$$

where D is the diffusion coefficient, C is the concentration field, x is the spatial coordinate, and t is the time. Derive a finite difference discretization using a 1^{st} order explicit scheme in time and a 2^{nd} order central scheme in space
We start with the 1D dimensionless Diffusion equation:

$$\frac{\partial C}{\partial t} = D\frac{\partial^2 C}{\partial x^2} \tag{21}$$

We now need to discretize the derivatives, we therefor use the taylor series of the backward and forward approximation.
forward approximation:

$$\left[\frac{\partial f}{\partial x}\right]_i^n \simeq \frac{f_{i+1}^n - f_i^n}{\Delta x} \tag{22}$$

backward approximation:

$$\left[\frac{\partial f}{\partial x}\right]_i^n \simeq \frac{f_i^n - f_{i-1}^n}{\Delta x} \tag{23}$$

Applying taylor series and do a lot of cumbersome rearranging (which I am way too lazy to write down in latex) we obtain:

$$\left[\frac{\partial f}{\partial x}\right]_i^n = \frac{f_{i+1}^n - f_{i-1}^n}{2\Delta x} \tag{24}$$

$$\left[\frac{\partial^2 f}{\partial x^2}\right]_i^n = \frac{f_{i-1}^n - 2f_i^n + f_{i+1}^n}{\Delta x^2} \tag{25}$$

Now get back to the diffusion equation and apply our obtained discretizations.
discretization in space:

$$\left[\frac{\partial^2 C}{\partial x^2}\right]_i^n = \frac{C_{i-1}^n - 2C_i^n + C_{i+1}^n}{\Delta x^2} \tag{26}$$

discretization in time:

$$\left[\frac{\partial C}{\partial t}\right]_i^n = \frac{C_i^{n+1} - C_i^n}{\Delta t} \tag{27}$$

Finally, we combine both discretizations and with $S := \frac{\Delta t}{\Delta x^2}$ and obtain:

$$C_i^{n+1} = C_i^n DS\Big(C_{i-1}^n - 2C_i^n + C_{i+1}^n\Big) \tag{28}$$

yippie!

1.12 Write the forward and backward approximation for the first derivative of a given function $f(x)$, evaluated at point x_i at a given step n. Show that these two approximations are 1^{st} order accurate in space. Help yourself with the Taylor series expansions.

Basically the first couple of lines of the task before, writing that shit down in latex is killing me.

1.13 Apply the Thomas algorithm (for tri-diagonal systems) to solve the linear system $[A][V] = [R]$, where the matrix A and the vectors V and R are

$$\begin{pmatrix} a_1 & c_1 & 0 & 0 \\ b_2 & a_2 & c_2 & 0 \\ 0 & b_3 & a_3 & c_3 \\ 0 & 0 & b_4 & a_4 \end{pmatrix} \cdot \begin{pmatrix} V_1 \\ V_2 \\ V_3 \\ V_4 \end{pmatrix} = \begin{pmatrix} R_1 \\ R_2 \\ R_3 \\ R_4 \end{pmatrix} \tag{29}$$

I'll leave this task to our world renowned Thomas algorithm expert Markus R

1.14 What is approximated by the interpolation functions in isoparametric elements in continuum mechanics ?

Interpolation functions of isoparametric elements approximate the displacement field in natural coordinates (r,s). (wär jetzt meine vermutung. bin mir dabei aber noch sehr unsicher -Hiti)

1.15 What are the advantages to formulate a Ritz Ansatz for small subdomains (i.e. Finite Elemets)

Abgeschreiben aus dem Skript S. 13:

- simpler interpolation functions
- easier to satisfy boundary conditions
- elements are topologically equivalent (i.e. all triangles)
- can model complex shapes
- can be automated

1.16 Why is the Jacobi matrix needed when using isoparametric elements and in which form ? Show the derivation. What restrictions apply to the jacobi matrix ?

Jacobian is needed for

- transformation between local and global coordinates
- derivation of the stiffness matrix

Restrictions:

$$det(J) \neq 0 (Jacobian must not be singular) \tag{30}$$

Keine Ahnung was er mit derivation meint..

2 Exam 25. Juni 2020

2.1 Consider the implementation of a stencil-based Jacobi solver for a two-dimensional elliptic PDE on a square two-dimensional domain.

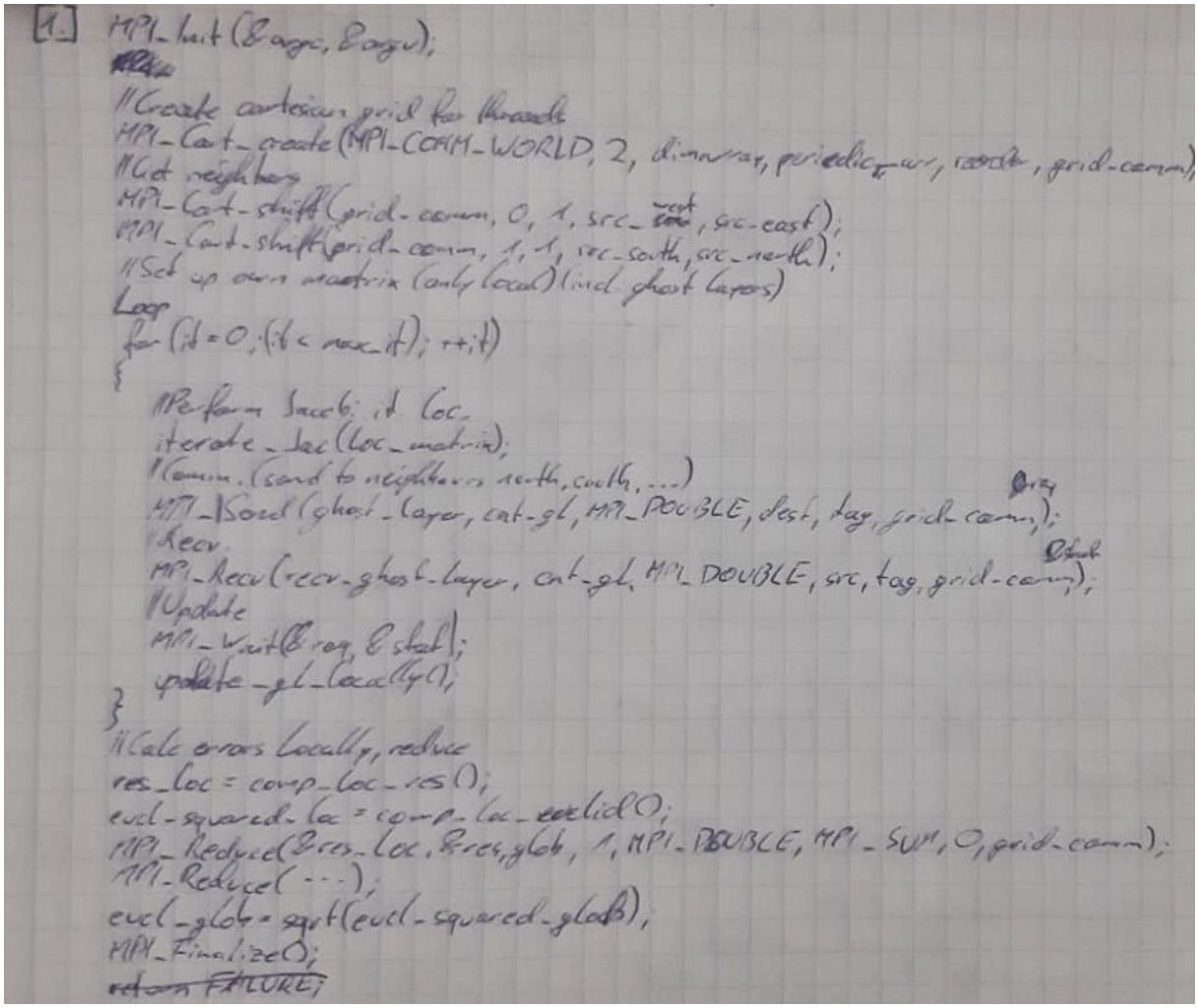
a) Write a pseudocode which shows the key adaption steps required to parallelize the solver using a ghost-layer-based two-dimensional domain decomposition using MPI

Algorithm 5: Pseudo code integration

```
1  MPI_Init()
2  cart = MPI_cart_create(dims, ndims, MPI_Comm_World)
3  my_rank_coords = MPI_Cart_coords(my_rank)
4  local_grid = Gcreate_grid(my_rank_coords)   // creates an aditional data layer
       for every nerighbour
5
6  for(n_iter < max_iter)
7      new_solution = iterate_over_local_grid()
8      MPI_Isend(ghostlayers)
9      MPI_recv(ghostlayers)
10     MPI_Wait() // because of non blocking send, otherwise swap could happen
           before send is complete
11     swap(solution, new_solution)
12 MPI_Barrier()
13 MPI_Gather(global_grid, local_grid) //
14
15
16 MPI_Finalize()
17 def residual(solution):
18     return solution - analytical_solution
19
20  def errorL2(residual):
21     // pointwise L2Norm(residual)
```

b) include the caluclation of global residual and error norms.

ALTERNATIVE:



2.2 Show via pseudocode and additional explanations the difference between blocking and non-blocking point-to-point communication

The following pseudo code example shows blocking point-to-point communication. This means that the processes are not continuing until the send/receive is finished.

Algorithm 6: Pseudo code integration

```
1  // MPI program with 2 processes, one is sending the other one is receiving:
2  MPI_Init()
3  MPI_Comm_Rank(&rank)
4  if rank==0:
5      MPI_Send(send some stuff)
6      do some further caluclations.....
7  else:
8      MPI_Recv(receive some stuff)
9      do some further calculations.....
10 MPI_Finalize()
```

The following code shows non blocking point-to-point communication. In this case the further calculations are performed while the send/receive is still in progress. It is the responsibility of the programmer to take care that no data is used that is not yet sended/received.

Algorithm 7: Pseudo code integration

```
// MPI program with 2 processes, one is sending the other one is receiving:
MPI_Init()
MPI_Comm_Rank(&rank)
if rank==0:
    MPI_Isend(send some stuff)
    do some further caluclations.....
    MPI_Wait()
if rank==1:
    MPI_Irecv(receive some stuff)
    do some further calculations.....
    MPI_Wait()
MPI_Finalize()
```

A blocking send/receive followed by a MPI_WAIT() is equivalent to a non blocking send/receive. The MPI_Barrier() ensures that MPI_Finalize() is not called before send/receives are completed.

2.3 Name two numerical methods for ODEs (can be explicit or implicit) together with the respective update rules (including the truncation error) assuming a constant step size h

- explicit euler

$u_{j+1} = u_j + hf(t_j, u_j)$
truncation error: $O(h^2)$

- implicit trapezodial

$u_{j+1} = u_j + \frac{h}{2}\{f(t_j, u_j) + f(t_{j+1}, u_{j+1})\} + O(h^3)$

2.4 Single-step and multi-step methods

a. Explain the key difference between both methods for the numerical solution of ODEs.
b. Explain why multi-step methods (compared to single-step methods) require special attention regarding the start-up and when considering a dynamic step size
a.
single step:

- Use only the previous evaluation point
- Intermediate values (to construct higher order methods) are withdrawn after a step is performed

multi-step:

- Previous evaluation points are reused to construct higher order methods (to gain efficiency)
- Linear multistep methods use a linear combination of previous evaluation points

b.
The designer of the method chooses the coefficients, balancing the need to get a good approximation to the true solution against the desire to get a method that is easy to apply. Often, many coefficients are zero to simplify the method.
A dynamic step size can be use if various degrees of precision is needed in calculations. E.g. spaceship orbiting eart and moon (three body problem). If the spaceship is far away, a bigger step size can be used as if the spaceship is near earth/moon.

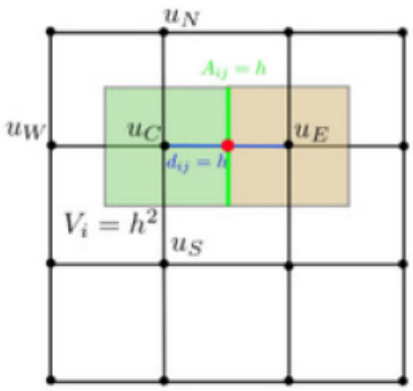
2.5 Consider the discretization of the two dimensional Poisson equation

$$u_{xx} + u_{yy} = f$$

and the integral form

$$\oint \nabla u dA = \int f dV$$

on a regular equidistant grid spacing h and box control volumes (see figure below). Show that the resulting discretization for an interior point u_C is identical for the FVM (using a central difference for ∇u) and the FDM (using second-order central differences).



2.6 Give an example for a second order linear parabolic operator

2.7 What does it mean that a second order linear operator is elliptic ?

2.8 What sets classical molecular dynamics apart from ab-initio molecular dynamics ?

ab-initio:

- The dynamics of electrons and nuclei can be decoupled (conditions apply)
- Fix the nuclei, solve for the electrons, get effective potential energy

limits of ab-initio:

- Hundreds of atoms
- Some picoseconds

classical molecular dynamics: Classical molecular dynamics is based on a parametrization of the potential energy, abstracting away the electrons.
Classical MD simulations with $10^5 - 10^6$ particles are within everyone's reach.

2.9 Describe the minimum image concention

a. When is it applicable?
b. If your simulation box is an axis-aligned straight rectangular prism with side lengths L_x, L_y, L_z, how would you calculate the distance between two particles with positions $\boldsymbol{p}$ and $\boldsymbol{P}$ under this convention?

a.

It is applicable for short range potentials.

- Let $\Delta\boldsymbol{x} = \boldsymbol{x}_L - \boldsymbol{x}_J$ be a vector joining L with any arbitrary image of J
- $\Delta\boldsymbol{x} = (\Delta x, \Delta y, \Delta z)$ is reduced to its value in the minimum image convention simply through:

$$\left.\begin{aligned} \Delta x_{\text{m.i.}} &= \Delta x - L_x \left\lfloor \frac{\Delta x}{L_x} \right\rceil \\ \Delta y_{\text{m.i.}} &= \Delta y - L_y \left\lfloor \frac{\Delta y}{L_y} \right\rceil \\ \Delta z_{\text{m.i.}} &= \Delta z - L_z \left\lfloor \frac{\Delta z}{L_z} \right\rceil \end{aligned}\right\}, \text{where } \lfloor z \rceil \equiv \text{round}(z)$$

b.

2.10 The following code snippet contains a naive Pyhton implementation of the velocity Verlet integrator that uses significantly more memory than strictly necessary. Discuss why and how to fix it.
Note: For simplicity, masses are assumed to have a value of 1.

```
def next_step(pos, vel, dt):
    force = -calc gradient e pot(pos)
    pos = pos + dt * (vel + .5 * force * dt)
    newforce = -calc_gradient_e_pot (pos)
    vel = vel + (force + newforce) * .5 * dt
    return (pos, vel)
```

2.11 Consider the 1D dimensionless Diffusion Equation in cartesian coordinates

$$\frac{\partial C}{\partial t} = D\frac{\partial^2 C}{\partial x^2},$$

where D is the diffusion coefficient, C is the concentration field, x is the spatial coordinate, and t is the time. Derive the discretized counterpart using a 2nd order implicit scheme in time using Crank-Nicolson and a 2nd order central scheme in space.

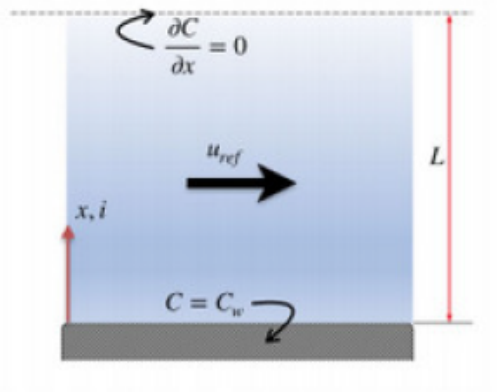
2.12 Consider the 1D Advection-Diffusion Equation (ADE) in cartesian coordinates

$$\frac{\partial C^*}{\partial t^*} + u^* \frac{\partial C^*}{\partial x^*} = D \frac{\partial^2 C^*}{\partial x^{*2}},$$

where the superscript * indicates variables in dimensional form and D the diffusion coefficient. Assume the flow configuration given in the figure below.

a. Derive the dimensionless counterpart of the above equation and identify the Peclet number.

b. How does the dimensionless equation look like for (1) a diffusion-dominated and (2) for an advection-dominated phenomenon?



Let's consider our transport equation (1D ADE)

$$\frac{\partial C^*}{\partial t^*} + u^* \frac{\partial C^*}{\partial x^*} = D \frac{\partial^2 C^*}{\partial x^{*2}}$$

In dimensionless form:

$$\begin{cases} C = \frac{C^*}{C_w} \\ x = \frac{x^*}{L} \\ u = \frac{u^*}{u_{ref}} \\ t = \frac{t^* D}{L^2} \end{cases} \rightarrow \frac{DC_w}{L^2}\frac{\partial C}{\partial t} + \frac{u_{ref}\, u\, C_w}{L}\frac{\partial C}{\partial x} = \frac{DC_w}{L^2}\frac{\partial^2 C}{\partial x^2}$$

$$\frac{\partial C}{\partial t} + \frac{u_{ref} \cdot C_w}{L}\frac{L^2}{DC_w} u \frac{\partial C}{\partial x} = \frac{\partial^2 C}{\partial x^2}$$

$$\frac{\partial C}{\partial t} + \underbrace{\frac{u_{ref} \cdot L}{D}}_{Pe\ (Peclet)} u \frac{\partial C}{\partial x} = \frac{\partial^2 C}{\partial x^2}$$

$$Pe = \frac{\text{diffusion time}}{\text{convection time}} = \frac{L^2/D}{L/u_{ref}} = \frac{L \cdot u_{ref}}{D}$$

$$\frac{\partial C}{\partial t} + Pe \cdot u \frac{\partial C}{\partial x} = \frac{\partial^2 C}{\partial x^2}$$

$Pe \ll 1 \Rightarrow$ Diffusion dominates $\rightarrow \frac{\partial C}{\partial t} = \frac{\partial^2 C}{\partial x^2}$

$Pe \gg 1 \Rightarrow$ Advection dominates $\rightarrow \frac{\partial C}{\partial t} + Pe\, u \frac{\partial C}{\partial x} = 0$

2.13 Consider the 1D pure Advection Equation Equation in cartesian coordinates

$$\frac{\partial C}{\partial t} + U\frac{\partial C}{\partial x} = 0,$$

where U is the (uniform and constant) advection velocity, C is the concentration field, x is the spatial coordinate and t is the time. Show that ...

a. ... when the upwind approximation (1st order in time, 2nd order in space) is used to discretize the equation, a numerical diffusivity is introduced.

b. ... the numerical diffusivity is null when $Co = 1$ (where Co indicates the Courant number).

2.14 What is the interpretation of the determinant of the Jacobi matrix of an isoparametric element ?

lol dafür gibs eine slide aus meiner FEM VO :D

Kurz Zusammengefasst: det(J) gives the area change when transforming to the other coordinate system

Explanation Determinant of Jacobian

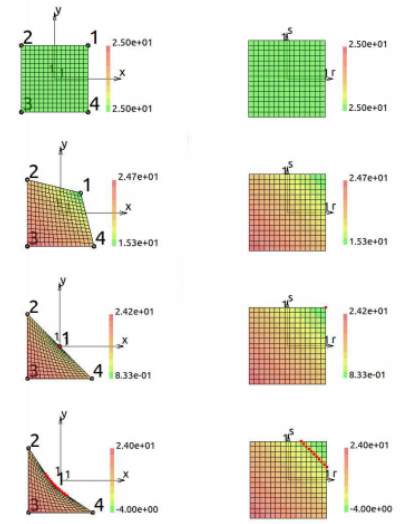


Figure: Element side length 10×10. Left orginal, right natural coordinates. From top to bottom movement of node 1. Red points $\det \underline{\underline{J}} = 0$.

Note: $\det \underline{\underline{J}}$ gives the area change (2D) or volume change (3D).

2.15 The fundamental FEM equation reads:

$[2]K[1]U = [1]F$

a. Explain variables and their dimensions (in terms of lin. Algebra) in continuum solid mechanics.
b. Consider a plane (two-dimensional) mechanical problem with 18 elements and 16 nodes: How many equations has the system?
c. Consider a three-dimensional heat conduction problem with 27 elements and 64 nodes: How many equations has the system?

a)

- K: Global stiffness matrix. K is the sum of all element stiffness matrices. The element stiffness matrix describes how the nodes that make up an element are connected, i.e. how far a node is displaced if a force acts on it.

  dim(K) = NxN
  N = $n_{nodes} \cdot DOFs$ number of nodes x degrees of freedom per node

- u: displacement vector. contains all displacements for each node
  dim(u) = N

- F: vector of reactonary forces
  dim(F) == dim(u)

b)
2D, 18 elements and 16 nodes. 2 x 16 = 32 equations

c)
thermal, 3D, 27 elements, 64 nodes. 64 equations because despite 3D every node has only 1 degree of freedom (temperature)

2.16 Consider a FEM heat conduction problem with the global FEM equation

$$H_{ab}\, T_a = P_b$$

with n degrees of freedom and with the known total stiffness matrix. Dirichlet boundary conditions are given for $a = 1 \ldots m$; Neumann boundary conditions are given for $b = m + 1 \ldots n$.
Write the computation steps by linear algebra manipulations to solve the system, i.e., the unknown T- and P- entries.
Hint: Start by writing down the FEM equations by using sub-matrices and sub-vectors.

3 Lecture Quiz

3.1 MPI

- How many Bytes would be required to store a derived datatype based on a struct of two characters and one double?

  2x char = 2 Byte, but uses 8 Bytes in derived datatype
  1x double = 8 Byte
  total = 16 Byte

- Give an example setup for defining a Cartesian MPI communication topology for a three-dimensional setup if you could use up to 1000 MPI processes (make assumptions for all other properties).

Algorithm 8: Pseudo code integration

```
1 int MPI_Cart_create(MPI_Comm comm_old, int ndims,
2 int *dims, int *periods, int reorder,
3 MPI_Comm *comm_cart)
4 Comm_old = MPI_COMM_WORLD
5 ndims = 3
6 Dims = (10,10,10)
7 Periods = (1, 1,1) (periodic boundary condition for all dims)
8 reorder = 1 (reorder to MPI tasks to optimize cartesian coordinates to
    limit unnecessary/lengthy communications)
```

- How can a point-to-point communication be made non-blocking? What potential advantage is there?
  Using MPI_Isend/MPI_Irecv....
  (potentially) allows the computation and communication to overlap

- Name typical reduction operations?
  MPI_MAX
  MPI_MIN
  MPI_SUM
  MPI_PROD (Product)
  MPI_LAND (Logical AND)
  MPI_BAND (Bitwise AND)
  MPI_LOR (Logical OR)
  MPI_BOR (Bitwise OR)
  MPI_LXOR (Logical exclusive OR)
  MPI_BXOR (Bitwise exclusive OR)
  MPI_MAXLOC (Maximum and location of the maximum)
  MPI_MINLOC (Minimum and location of the minimum)

- Is "MPI_Init" executed by one, several or all MPI processes?
  MPI_Init is executed by all the processes

- What is the first and last routine to be called in a MPI program?
  An MPI program always start with an initialization
  MPI_Init(....)
  and ends with
  MPI_Finalize()

- How is it ensured that a specific message is received by a specific process?
  The tag and the dest parameter ensures that the message is received by a specific process

- pseudocode for matrix vector multiplication

Algorithm 9: Pseudo code Matrix Vector

```
MPI_Init() // start MPI
MPI_Comm_rank(MPI_COMM_WORLD,myrank) // get my rank
MPI_Comm_size(MPI_COMM_WORLD,nranks) // get total number of processes

if myrank==0:
    numsent=0;
    MPI_Bcast(b,cols,MPI_DOUBLE, myrank,MPI_COMM_WORLD) // send vector b to all
        workers. This can be broadcastet since its the same for everybody.
    // now create loop and send the specific row of the matrix to specific
        worker, get busy bitches
    for(i=0; i< nranks-1,rows;i++)
        MPI_Send(&A[i][0],cols,MPI_DOUBLE,i+1,i,MPI_COMM_WORLD)
        numsent++; //we need to keep track how many rows we sent already, our
            workers want to grab a beer asap. lazy fucks

    //now create loop to receive all the results from workers, we use
        MPI_ANY_SOURCE since we are a dirty master process who takes it from
        anybody
    for(i=0; i<rows;i++)
        MPI_Recv(&ans, 1, MPI_DOUBLE, MPI_ANY_SOURCE, MPI_ANY_TAG,MPI_COMM_WORLD
            , \&status);
        sender = status.MPI_SOURCE; // we want to know who sent the new row from
             master in case there is still work to do
        anstype = status.MPI_TAG; // this is the index of the row, we need to
            reassemble the matrix correct ey
        c[anstype] = ans; // I have no idea what this array is doing ? never
            used again as far as I can tell. Oh ok, thats the result, we are
            solving A*b=c, being able to read is an advantage
        // now we check if there are still rows to send. If so, lets do some
            more sending
        if numsent < rows:
            MPI_Send(&A[i][0],cols,MPI_DOUBLE,sender,numsent,MPI_COMM_WORLD)
            numsent++;
        //if there is no more stuff to send, we tell the worker he is free to
            netflix and chill
        else:
            MPI_Send(MPI_BOTTOM, 0, MPI_DOUBLE, sender, DONE, MPI_COMM_WORLD);
    // this is all for the master process
MPI_Bcast(b, cols, MPI_DOUBLE, 0, MPI_COMM_WORLD); // lets get our row, I am
    eager to work
// check if someone spawned more processes than rows in the matrix. If so,
    process just chill
done = myid > rows;
while(!done)
    // receive the row worker should perform the work on, processes receives
        from any source since we could be doing multiple rows if we are fast
        enough/matrix big enough
    MPI_Recv(buffer, cols, MPI_DOUBLE, 0, MPI_ANY_TAG,MPI_COMM_WORLD, &status);
    done = status.MPI_TAG == DONE; // we set done to true in case any worker
        thread receives the message that we are done
    // if we are not done, lets calculate and send to master process
    if(!done):
        row = status.MPI_TAG;
        ans = 0.0;
        for (i = 0; i < cols; i++)
            ans += buffer[i] * b[i];
        MPI_Send(&ans, 1, MPI_DOUBLE, 0, row, MPI_COMM_WORLD);
// now just free the memory and off we goooooooo but who cares, MPI_Finalize() is
    the important line here
MPI_Finalize();
return 0;
```

3.2 ODE

- *What is the order of the forward/backward Euler method?*
  Both are first order, here the example of the explicit Euler method:
  First order ODE:

$$y' = f(t, y(t)) \tag{31}$$

  We discretize using forward FD of first order:

$$y' = \frac{y(t_j) - y(t_{j-1})}{h} + \mathcal{O}(h^2) \tag{32}$$

$$\Rightarrow y(t_j) = y(t_{j-1}) + hf(t_{j-1}, y(t_{j-1})) + \mathcal{O}(h^2) \tag{33}$$

- *Are the Runge-Kutta methods single-step or multistep methods?*
  Singlestep method, these methods work like this:
  1. Use only the previous evaluation point
  2. Intermediate values (to construct higher order methods) are withdrawn after a step is performed

- *How can first order ODEs methods be applied to higher order ODEs?*

  Any explicit (system of) higher order ODEs can be transformed into an equivalent system of first order ODEs

$$\frac{d^2y}{dt^2} = -ky, \qquad \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} y \\ y' \end{bmatrix}, \quad \begin{bmatrix} z_1' \\ z_2' \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -k & 0 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}$$

- *Why are finite volume schemes 'conservative'?*
  Because the flux entering a given volume is identical to that leaving the adjacent volume, these methods are conservative.
  Or: Finite volume schemes are conservative as cell averages change through the edge fluxes. In other words, one cell's loss is another cell's gain!

- *What are advantages of the Finite Volume Method over the Finite Difference Method?*
  In uniform mesh, they are identical in many cases.
  FVM is conservative, FDM not necessarily?????

maybe falsch

3.3 PDE