Carl von Ossietzky Universität Oldenburg

Betriebseinheit für technisch-wissenschaftliche Infrastruktur

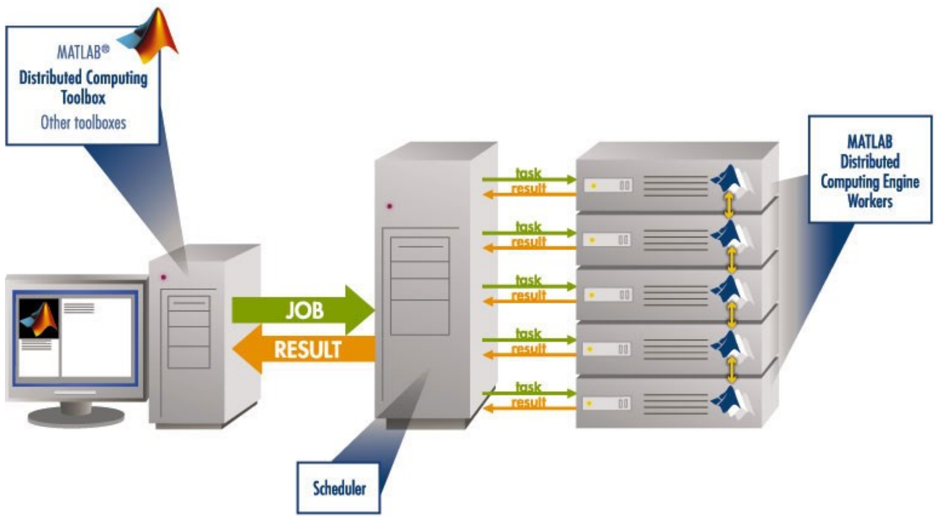
Introduction to High-Performance Computing

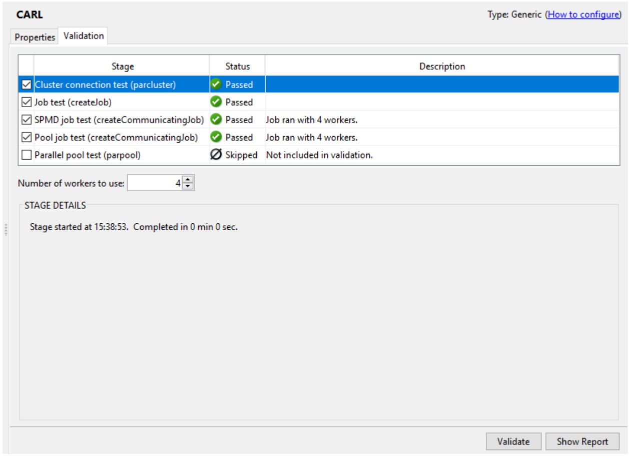
Session 09

Matlab Distributed Compute Server (MDCS)

Previous Session

- introduction to MDCS
  - MDCS can be used to off-load Matlab computations to the HPC cluster in a simple workflow
  - allows parallelization across multiple compute nodes
- configuration of MDCS
  - prepare your local computer for MDCS
- usage of MDCS
  - basic example for submitting a job and retrieving results

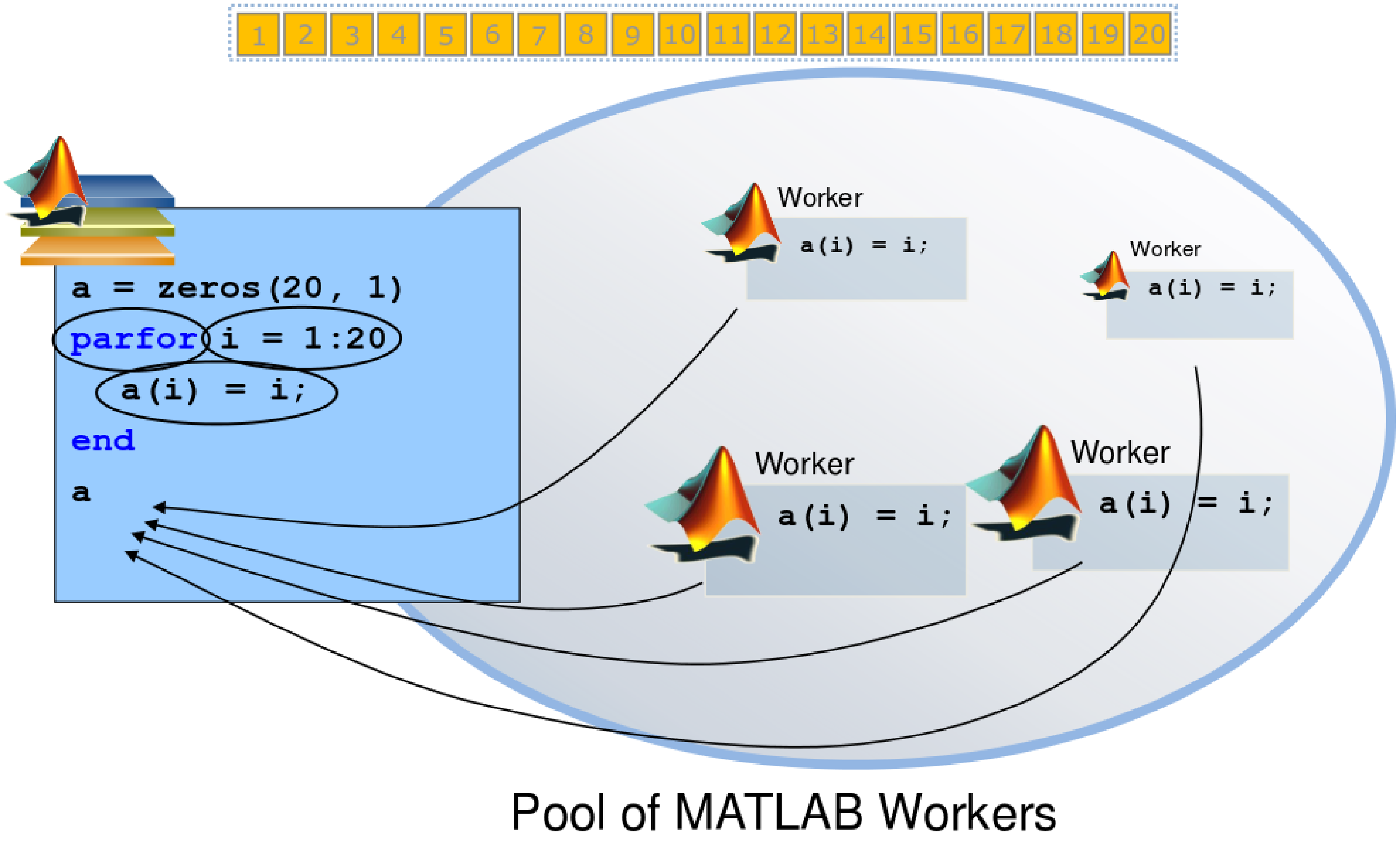





```
>> sched = parcluster('CARL');
>> job = batch(sched, 'paramSweep_batch', 'Pool', 7, ...
               'AttachedFiles', {'odesystem.m'});
>> job.State
>> jobData = load(job);
```

Parallelization with `parfor`

Mechanics of `parfor` Loops



Converting `for` to `parfor`

- requirements for **`parfor`** loops
  - task independent
  - order independent
  - loop index must be consecutive increasing integers

- constraints on the loop body
  - cannot introduce variables (e.g. **`eval`**, **`load`**, **`global`**)
  - cannot contain **`break`** or **`return`** statements
  - cannot contain another **`parfor`** loop
  - …

https://de.mathworks.com/help/parallel-computing/troubleshoot-variables-in-parfor-loops.html

Variable Classification

- all variables referenced at the top level of the `parfor` must be resolved and classified

Classification	Description
loop	serves as a loop index for arrays
sliced	an array whose segments are operated on by different iterations
broadcast	a variable defined before the loop whose value is used inside the loop, but never assigned in the loop
reduction	accumulates a value across iterations of the loop, regardless of iteration order
temporary	variable created inside the loop but unlike sliced or reduction variables, not available outside the loop

Carl von Ossietzky Universität Oldenburg

Variable Classification Example

- matrix-vector multiplication

```
N=2048;                    % N is broadcast
b=rand(N,1);               % b is broadcast
A=rand(N,N);               % A is slices input

parfor i=1:N               % i is loop index
  c(i)=A(i,:)*b(:);        % c is sliced output
end
c                          % using c outside the loop
```

`parfor` Examples

- this example cannot be parallized in `parfor`

```
j=zeros(100);          %pre-allocate vector
j(1)=5;
for i=2:100;
    j(i)=j(i-1)+5;
end;
```

  – order of iterations is important

Carl von Ossietzky Universität Oldenburg

`parfor` Examples

- functions with multiple output may confuse Matlab

```
for i=1:10
   [x{i}(:,1), x{i}(:,2)]=functionName(z,w);
end;
```

  - use this instead

```
for i=1:10
    [x1, x2]=functionName(z,w);
    x{i}=[x1 x2];
end;
```

parfor Examples

- be careful not to broadcast unnecessary data (false sharing)

```
data.raw = ...
data.processed = ...

% Inefficient variant:
parfor idx = 1 : N
    % do something with data.processed
end

% This is better:
processedData = data.processed;
parfor idx = 1 : N
    % do something with processedData
end
```

https://undocumentedmatlab.com/blog/a-few-parfor-tips

`parfor` Considerations

- **`parfor`** often only involves minimal code changes
- if a for loop cannot be converted to **`parfor`**, consider wrapping a subset of loop body in a function
  - e.g. load works not in **`parfor`**, however it does work in function that is called inside a **`parfor`** loop
- more information http://blogs.mathworks.com/loren/2009/10/02/using-parfor-loops-getting-up-and-running/
- there is a Code-Analyzer to diagnose **`parfor`** issues

Parallelization with `spmd`

(single program multiple data)

Parallelization with `spmd`

```
                     Client        Worker 1     Worker 2
                     a  b  e  |  c  d  f  |  c  d  f
                     ----------------------------------
a = 3;               3  -  -  |  -  -  -  |  -  -  -
b = 4;               3  4  -  |  -  -  -  |  -  -  -
spmd                          |           |
  c = labindex();    3  4  -  |  1  -  -  |  2  -  -
  d = c + a;         3  4  -  |  1  4  -  |  2  5  -
end                           |           |
e = a + d{1};        3  4  7  |  1  4  -  |  2  5  -
c{2} = 5;            3  4  7  |  1  4  -  |  5  6  -
spmd                          |           |
  f = c * b;         3  4  7  |  1  4  4  |  5  6 20
end
```

Parallelization with `spmd`

- when a **`spmd`** block ends the workspace is saved, the worker is paused
- data is preserved from one block to the next
- does not apply to **`spmd`** block in a function after the function is completed (as regular variables local to a function)

SPMD Example

```
% read image file
x = imread('uol.jpeg');

% add noise and store noisy image
y = imnoise(x, 'salt & pepper', 0.30);

% distribute image (last column is color chanel)
yd = distributed(y);

% remove noise with filter in parallel
spmd
    yl    = getLocalPart(yd);
    yl    = medfilt2(yl, [3,3]);
end

% put together filtered image and store it
xdim = size(x);
z(1:xdim(1),1:xdim(2),1) = yl{1};
z(1:xdim(1),1:xdim(2),2) = yl{2};
z(1:xdim(1),1:xdim(2),3) = yl{3};
```

- read image
- add noise to image
- distribute data
- parallel working on image data (filter)
- on master process put together filtered image



Distributed Data

- Matlab provides different functions to manage distributed data
  - with **`distributed(X)`** you can distribute data among workers
  - with **`distributed.METHOD`** you can create data distributed among workers
  - workers can create codistributed data structures which become distributed data outside of the **`spmd`** block
  - a datastore can be distributed to read manage large data files with multiple workers
  - see ‘**`help distributed`**’ for more information

Distributed Data

distributing data
from client

```
p = parpool('local', 4);      % create a local pool of workers
A = zeros(4);                 % create a 4x4 matrix with zeros
A                             % print A on client
B = distributed(A);           % distribute A to the workers
spmd                          % begin parallel spmd region
   B = B + labindex;          % modify distributed data in B
end                           % end parallel spmd region
B                             % print B on client
delete(p);
```

vs.

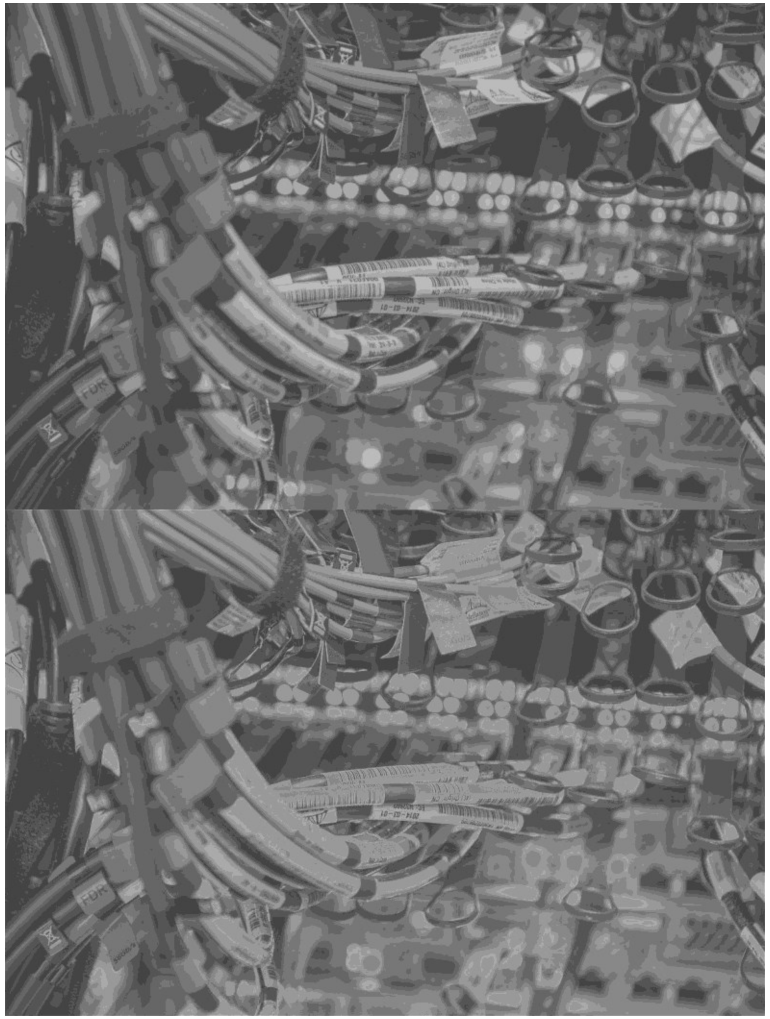
codistributed data
created on workers

```
p = parpool('local', 4);                    % create a pool of workers
spmd                                        % begin parallel spmd region
   codist = codistributor1d(2, [1,1,1,1]);  % define distribution
   B = zeros(4, codist);                    % created codistributed array
   B = B + labindex;                        % modify distributed data in B
end                                         % end parallel spmd region
B                                           % print B on client
delete(p);
```

Example: Image Contrast

- a Matlab script that uses a simple function to change the contrast of an gray-scale image

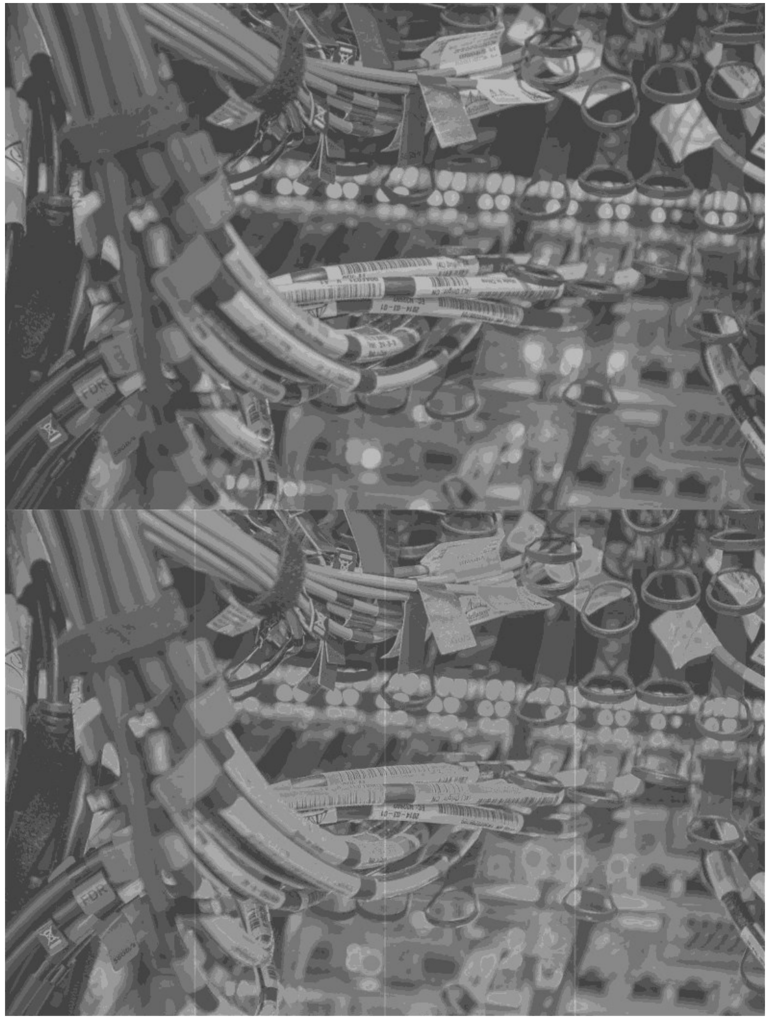
```
% read an image (gray-scale)
y = imread('low_contrast.jpg');

% setup function for contrast manipulation
c = 1.7;
adjustContrast = @(x) c*x(2,2)+(1.0-c)*(mean(x(:)-x(2,2)/9.0));

% apply filter
z = nlfilter(y, [3,3], adjustContrast);

% save image side-by side
imwrite(cat(1,y,z), 'contrast_serial.jpg');
```



Example: Image Contrast

- parallelize with SPMD

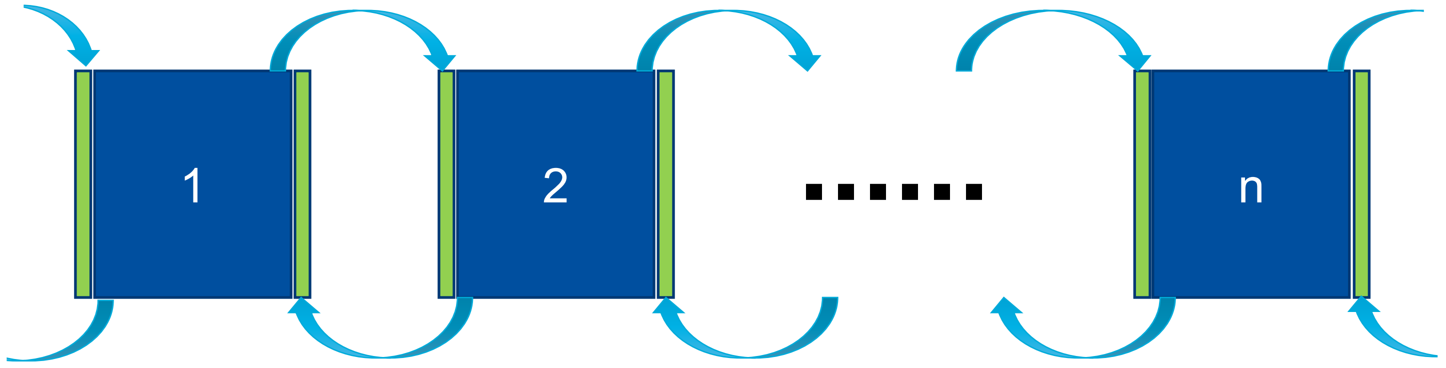
```
% read an image (gray-scale)
y = imread('low_contrast.jpg');

% setup function for contrast manipulation
c = 1.7;
adjustContrast = @(x) c*x(2,2)+(1.0-c)*(mean(x(:)-x(2,2)/9.0));

% distribute image by columns
yd = distributed(y);

% now work in parallel
spmd
    yl = getLocalPart(yd);

    % apply filter
    yl = nlfilter(yl, [3,3], adjustContrast);

end

% combine local images
z = [ yl{:}];

% save image side-by side
imwrite(cat(1,y,z), 'contrast_spmd.jpg');
```



- algorithm produces artifacts when parallelized on multiple workers
  - problem is that increasing contrast requires information from neighbouring pixel
  - distributing the data adds additional boundaries

Communication between Matlab Workers

- solution is communication between workers
  - each worker has to sent one boundary left and one right
  - each worker has to receive one boundary from left and one from right
  - extra columns are added before filter is applied, and need to be removed again afterwards

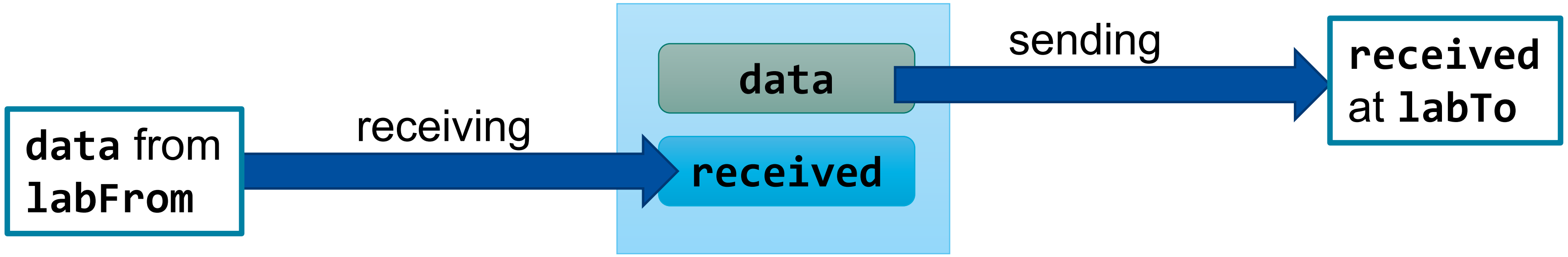


Communication between Matlab Workers

- the function **`labSendReceive`** simultaneously sends and receives data

```
received = labSendReceive(labTo, labFrom, data)
```

  - sends **`data`** to **`labTo`**
  - receives **`data`** from **`labFrom`** and stores it in **`received`**



Communication between Matlab Workers

```
column = labSendReceive ( previous, next, xl(:,1) );

if ( labindex() < numlabs() )
  xl = [ xl, column ];
end

column = labSendReceive ( next, previous, xl(:,end -1));

if ( 1 < labindex() )
  xl = [ column, xl ];
end
```

Exercise

Exercises

- try out the following examples from the lecture

1. MDCS configuration
2. basic example with parfor
3. SPMD example noise reduction
4. SPMD example contrast
5. SPMD example heat

Carl von Ossietzky Universität Oldenburg

Heat Example in Matlab

```
% 2d-heat example in Matlab
% initial setup
NXPROB = 20;        % number of grid rows
NYPROB = 20;        % number of grid columns
STEPS  = 100;       % number of iterations
TIME   = 0;         % initial and current time

uvals  = zeros(2, NXPROB, NYPROB);  % allocate grid
uvals  = inidat(uvals);             % initialize grid

plotdat(uvals, 1, TIME);            % make plot

it = 1;
for TIME=1:STEPS                    % time iteration
    uvals = updateu(uvals, it);     % update thermal energy
    it    = 3 - it;
end

plotdat(uvals, 1, TIME);            % make plot
```

The End

contact me directly of you have questions about the course

contact hpcsupport@uol.de if you have requests or encounter problems regarding the HPC cluster

Betriebseinheit für technisch-wissenschaftliche Infrastruktur