Theoretische Elektrotechnik

Parallel Computing inside the Julia Programming Language

Instructors:
Damian Belz (DB), Albert Piwonski (AP), Rodrigo Rezende (RR).

Scientific board:
Mirsad Hadžiefendić, Marcus Christian Lehmann.

Date & Location:
June 24th 2019, 9 AM – 6 PM,
Einsteinufer 17, 10587 Berlin, Room EN-616/617.

Table of contents

1. Introduction to Parallel Computing
2. Parallel Computing inside Julia
3. Parallel Computing inside Our Code
4. Bibliography

1 Introduction to Parallel Computing

2 Parallel Computing inside Julia

3 Parallel Computing inside Our Code

4 Bibliography

Introduction to Parallel Computing

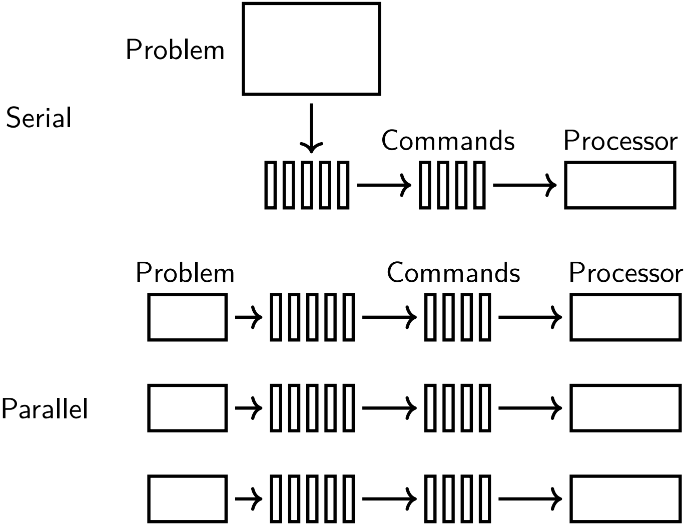
- With Julia a Problem can be solved using the **Parallel Computing**
- Parallel Computing is the *simultaneously* usage of processing elements to solve a problem

'Parallel Computing'

1. A problem is divided in more parallel parts
2. Every part is then translated in serial commands to be solved or executed
3. The commands of each part will then be simultaneously run in the processing elements avaiable (more than one)
4. A general *control* from the process is necessary to process parallel data

Introduction to Parallel Computing

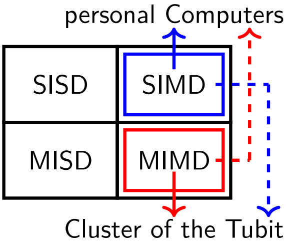
- Graphically this procedure can be understood as follows:



Introduction to Parallel Computing

- Why to use parallel computing?
  - To solve problems which requires a solution with great time consuming tasks (computing-time), for example: 'Grand Challenges Problems' [1]
  - Usage of non-local resources
  - Better usage of available processor elements ("multicore")

Classification of parallel computers:
('Flynn´s taxonomy')



Speedup(Amdahl's Law):

speedup $= 1/(1-P)$

speedup $= 1/((P/N)-S)$

Beispiel: für P $=$ 0.95 und N $=$ 10000
→ speedup $= 19.96$

1 Introduction to Parallel Computing

2 Parallel Computing inside Julia

3 Parallel Computing inside Our Code

4 Bibliography

Parallel Computing inside Julia

- There are different levels of **Parallel Computing** in Julia [2]:
  1. 'Julia Coroutines'('Green Threading')
  2. 'Multi Threading'
  3. 'Multi-Core' or 'Distributed Processing'

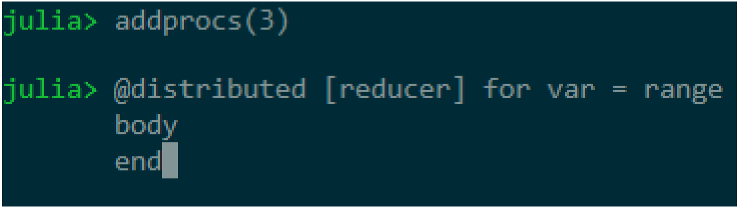
- 1.Complete control over "inter-tasks" and communication between these **"tasks"**
- 2. 'Fork-join approach': parallel **Threads** are covered separately from each other
- 3. This is a interface to enable the division and executions of processes in the available **multicores**

Parallel Computing inside Julia

Some useful commands in Julia

- @everywhere - macro used to load a specific variable on every operating process
- @addprocs - macro to add a work processes
- @distributed - macro to perform the parallelisation of a for loop

Examples:



```
julia> addprocs(3)

julia> @distributed [reducer] for var = range
       body
       end
```

Exercise! :)

- Make a code with two for loops. Both loops will be assigning values from 1:N to a vector of size=(N,1) . One loop must be serial and the the other must be parallel. Measure the time of each loop!
- Hint 1: write the two for loops inside a function
- Hint 2: use the macro @distributed
- Hint 3: use the macro @time

1 Introduction to Parallel Computing

2 Parallel Computing inside Julia

3 Parallel Computing inside Our Code

4 Bibliography

Parallel Computing inside Our Code

- Important thoughts for the use of Parallel Computing in our Code:

  1. Can this problem be solved with parallel computing ?
     - AMDAHL's Law
  2. *Move of data*:
     - The time used to **transfer** all the data to the cores
  3. The **size** of the problem
     - With bigger problems the win-time with the parallel computing might be bigger

- We measured the run time of various parts of our code (for a Mesh with 1365 Nodes): [1] [2]

  1. **Assemlbing of the Stiffness Matrix**: **3.59 s**
  2. Solving the system of equations: 1.95 s
  3. Plotting: 4.99 s

  Overall simulation time: 25.92 s

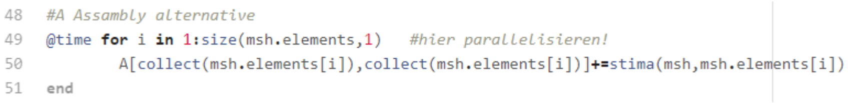
[1] Hardware: Intel Core i7-4702MQ CPU @ 2.2 GHz,16 GB RAM

[2] using the macro `@time` after calling the function for a second time

Parallel Computing inside Our Code

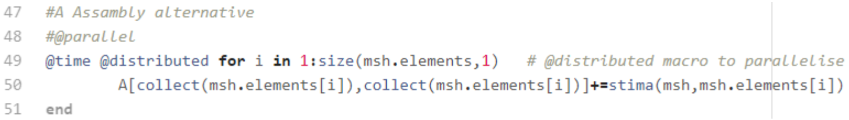
- Use in our code:
  1. Part of the code that can be solved with parallel computing
  2. Big problem
  3. Easy calculation

- Original Code



```
#A Assambly alternative
@time for i in 1:size(msh.elements,1)   #hier parallelisieren!
        A[collect(msh.elements[i]),collect(msh.elements[i])]+=stima(msh,msh.elements[i])
end
```

- Parallel Code



```
#A Assambly alternative
#@parallel
@time @distributed for i in 1:size(msh.elements,1)   # @distributed macro to parallelise
        A[collect(msh.elements[i]),collect(msh.elements[i])]+=stima(msh,msh.elements[i])
end
```

Bibliography I

[1] Grand Challenges Problems, https://en.wikipedia.org/wiki/Grand_Challenges

[2] Parallel Computing, The Julia Language https://docs.julialang.org/en/v1/manual/parallel-computing/#Parallel-Computin