Tree Decomposition

Maren Kaluza

November 2018

HELMHOLTZ
CENTRE FOR
ENVIRONMENTAL
RESEARCH - UFZ

Universität Potsdam

ESM
Advanced
Earth System Modelling
Capacity

UFZ

Table Of Contents

Introduction

Parallelization

Distribution of calculations onto multiple computational units, so parts of the calculations can be done simultaneously.

Parallelization

Distribution of calculations onto multiple computational units, so parts of the calculations can be done simultaneously.

	1	2	3	5	8	13
+	13	8	5	3	2	1

UFZ

Parallelization

Distribution of calculations onto multiple computational units, so parts of the calculations can be done simultaneously.

	1	2	3	5	8	13
+	13	8	5	3	2	1
						14

Parallelization

Distribution of calculations onto multiple computational units, so parts of the calculations can be done simultaneously.

	1	2	3	5	8	13
+	13	8	5	3	2	1
					10	14

Parallelization

Distribution of calculations onto multiple computational units, so parts of the calculations can be done simultaneously.

	1	2	3	5	8	13
+	13	8	5	3	2	1
				8	10	14

Parallelization

Distribution of calculations onto multiple computational units, so parts of the calculations can be done simultaneously.

	1	2	3	5	8	13
+	13	8	5	3	2	1
			8	8	10	14

Parallelization

Distribution of calculations onto multiple computational units, so parts of the calculations can be done simultaneously.

	1	2	3	5	8	13
+	13	8	5	3	2	1
		10	8	8	10	14

Parallelization

Distribution of calculations onto multiple computational units, so parts of the calculations can be done simultaneously.

	1	2	3	5	8	13
+	13	8	5	3	2	1
	14	10	8	8	10	14

Parallelization

Distribution of calculations onto multiple computational units, so parts of the calculations can be done simultaneously.

	1	2	3	5	8	13
+	13	8	5	3	2	1

Parallelization

Distribution of calculations onto multiple computational units, so parts of the calculations can be done simultaneously.

	1	2	3	5	8	13
+	13	8	5	3	2	1
		10		8		14

Parallelization

Distribution of calculations onto multiple computational units, so parts of the calculations can be done simultaneously.

	1	2	3	5	8	13
+	13	8	5	3	2	1
	14	10	8	8	10	14

Parallelization

Distribution of calculations onto multiple computational units, so parts of the calculations can be done simultaneously.

	1	2	3	5	8	13
+	13	8	5	3	2	1
	14		8		10	

Profit: in the optimal case the new calculation time is the serial time by the number of computational units.

Parallelization

A cheese cake needs about 18 hours for baking. How much time is needed to bake the cheese cake with the same result if there where 6 ovens available?

Parallelization

1 2 3 5 8 13

Parallelization

Fibonacci sequene (shifted by 1)

1 2 3 5 8 13

Parallelization

Fibonacci sequene (shifted by 1)

1 2 3 5 8 13 21

Parallelization

Fibonacci sequene (shifted by 1)

1 2 3 5 8 13 21 34

Parallelization

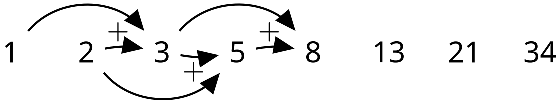
Fibonacci sequene (shifted by 1)

Parallelization

Fibonacci sequene (shifted by 1)

Parallelization

Fibonacci sequene (shifted by 1)



Parallelization

Pascal triangle

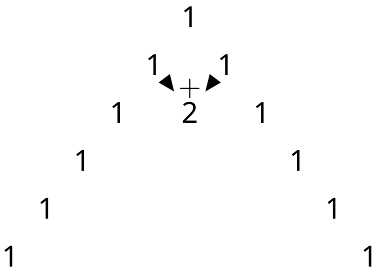
Parallelization

Pascal triangle

					1					
				1		1				
			1				1			
		1						1		
	1								1	
1										1

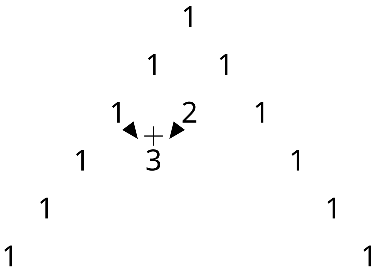
Parallelization

Pascal triangle



Parallelization

Pascal triangle



Parallelization

Pascal triangle

1

1 1

1 2 1

\+

1 3 3 1

1 1

1 1

Parallelization

Pascal triangle

1

1 1

1 2 1

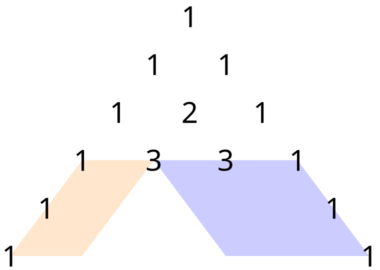
1 3 3 1

1 1

1 1

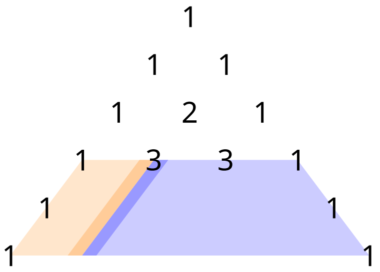
Parallelization

Pascal triangle



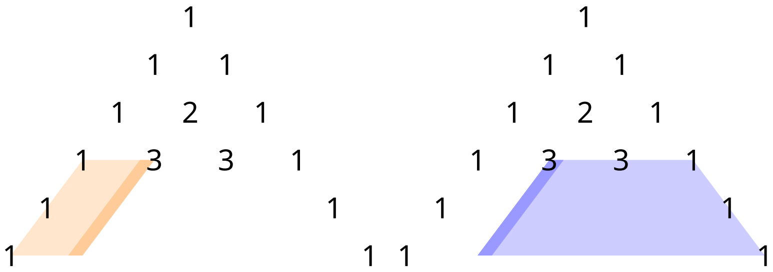
Parallelization

Pascal triangle



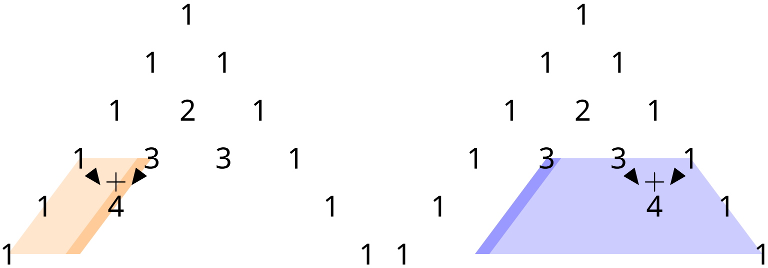
Parallelization

Pascal triangle



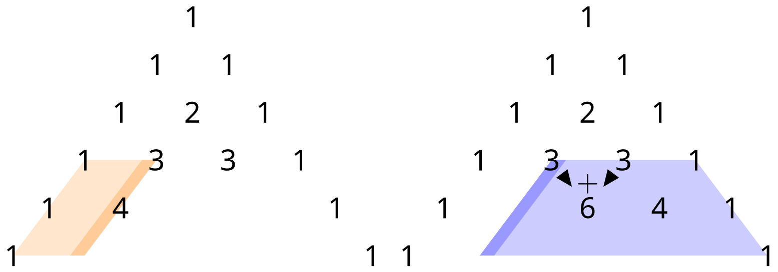
Parallelization

Pascal triangle



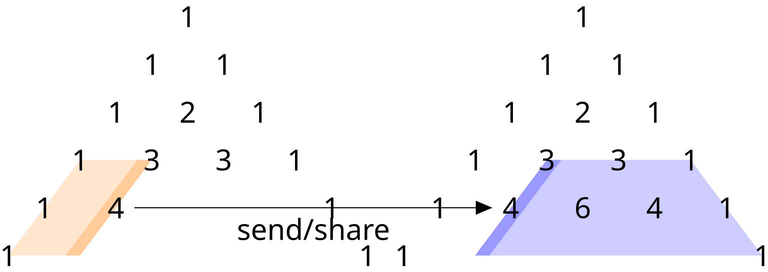
Parallelization

Pascal triangle



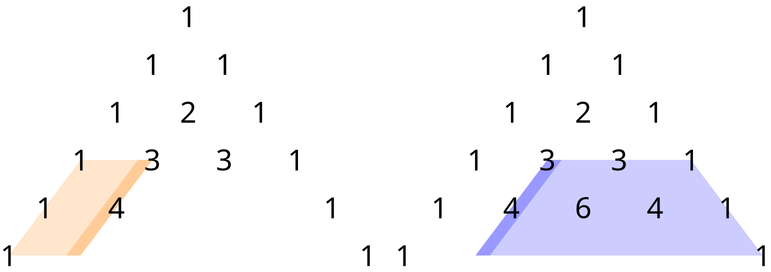
Parallelization

Pascal triangle



Parallelization

Pascal triangle

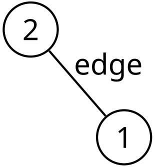


Graphs

vertex

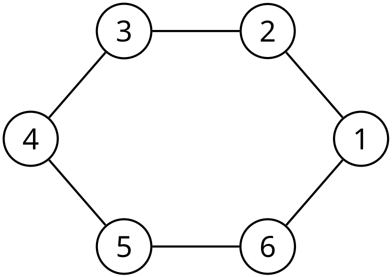
graph

Graphs



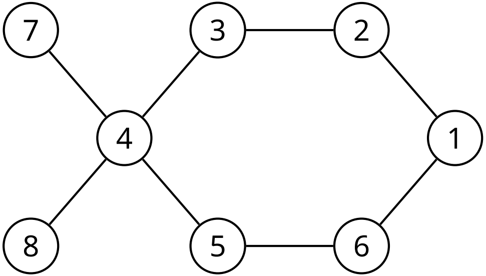
Graph

Graphs



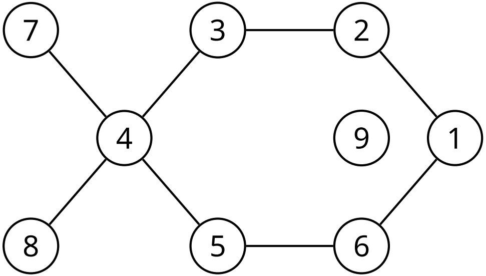
Cycle (nevertheless a graph)

Graphs



no cycle, but a connected graph

Graphs



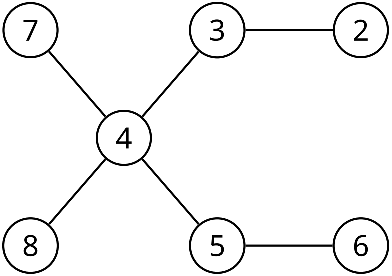
an unconnected graph

Trees

A tree is acyclic connected graph

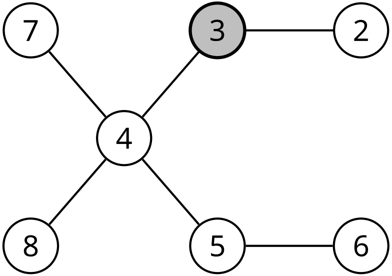
Trees

A tree is acyclic connected graph



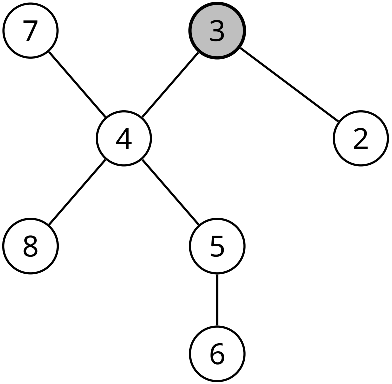
Trees

A tree is acyclic connected graph



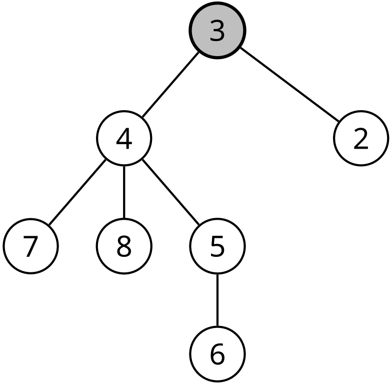
Trees

A tree is acyclic connected graph



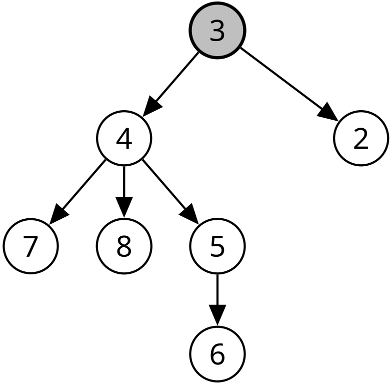
Trees

A tree is acyclic connected graph



Trees

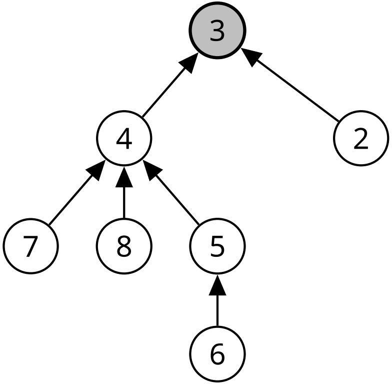
A tree is acyclic connected graph



gewurzelter Baum, Out-tree

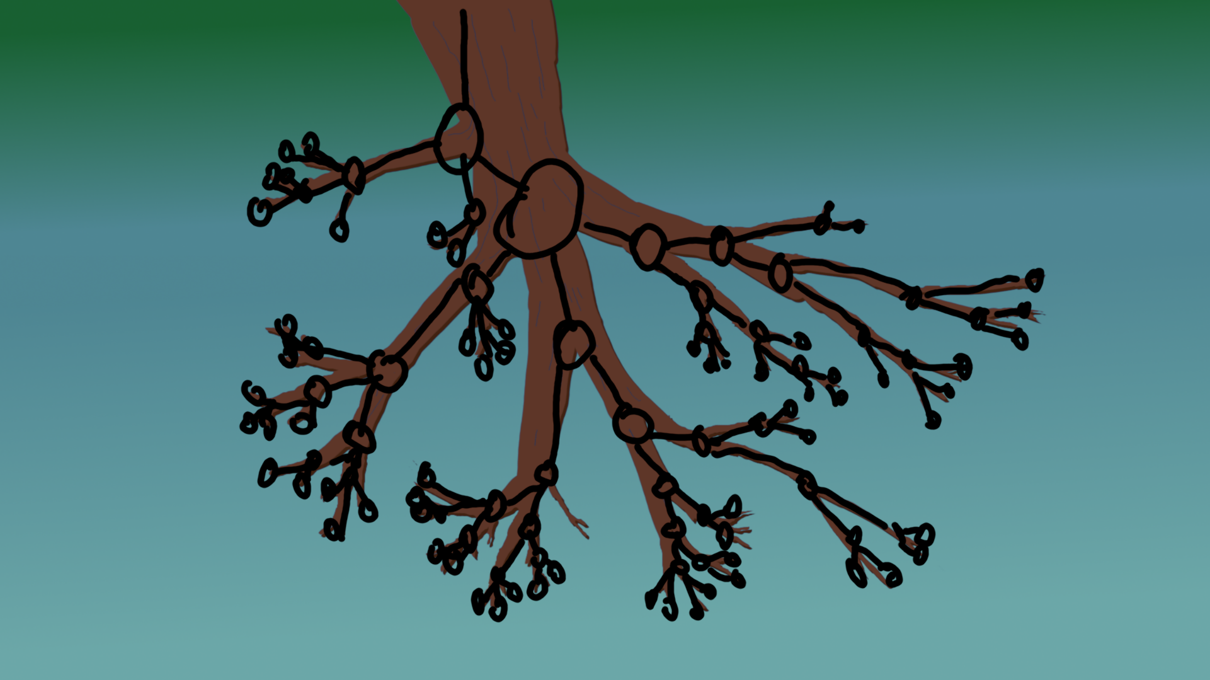
Trees

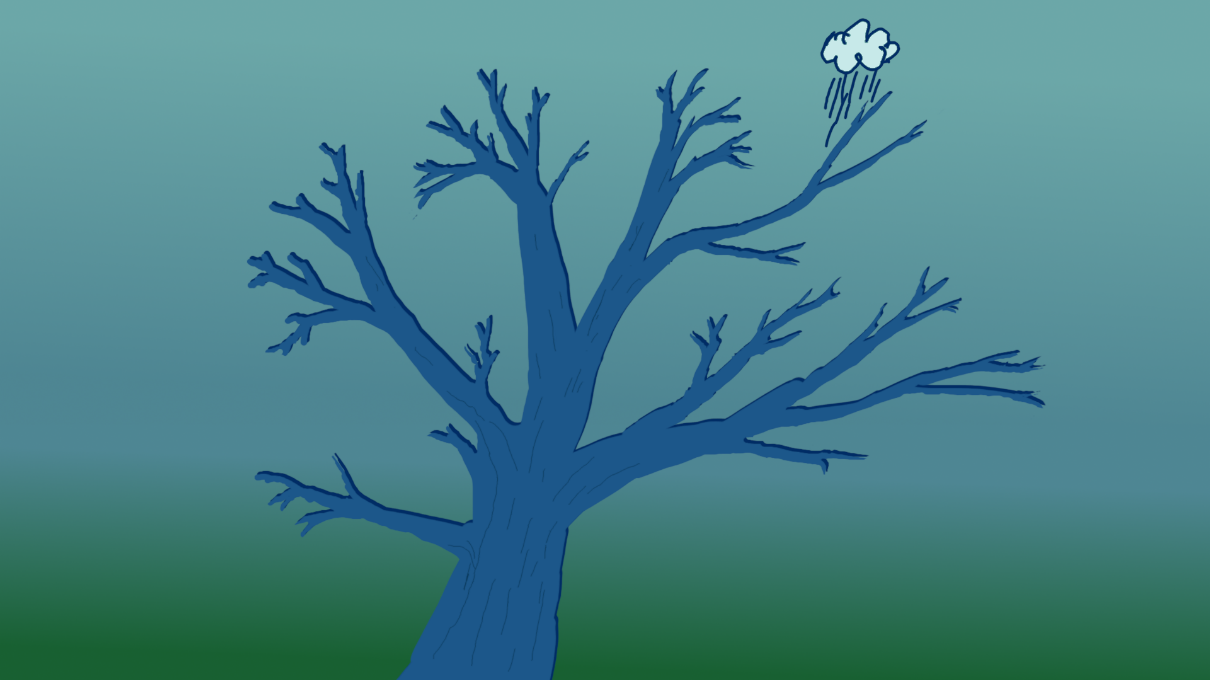
A tree is acyclic connected graph

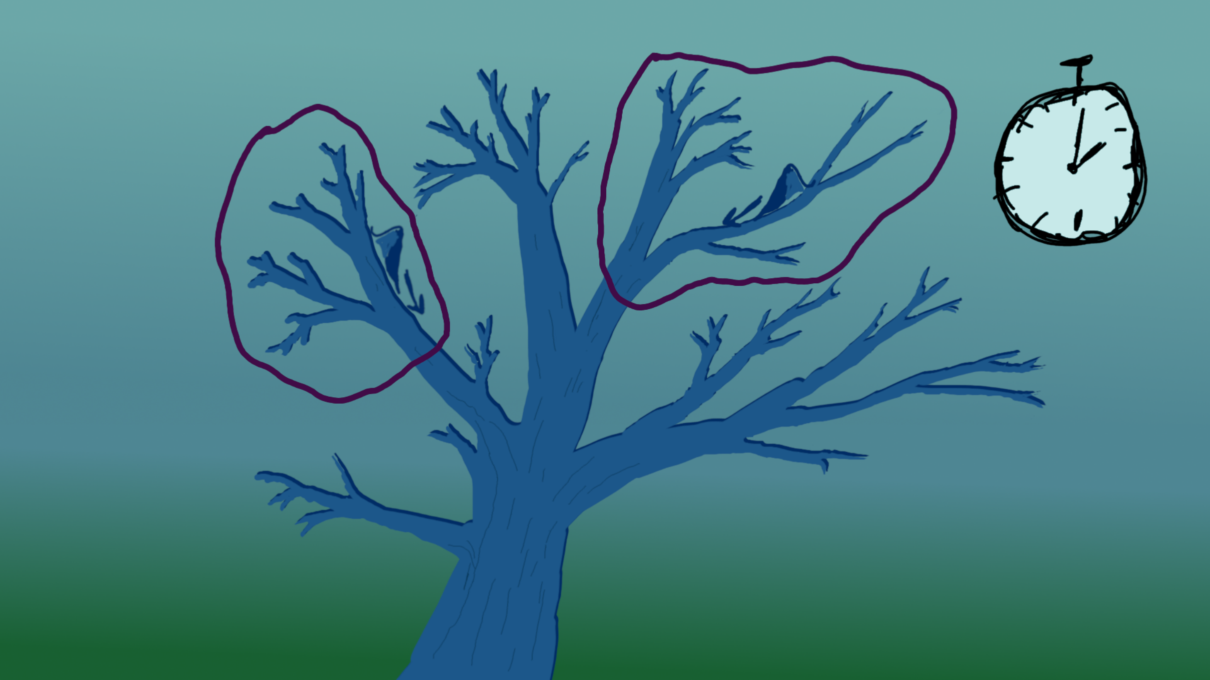


gewurzelter Baum, In-tree

rivernetwork (Hydrology)







Tree Decomposition

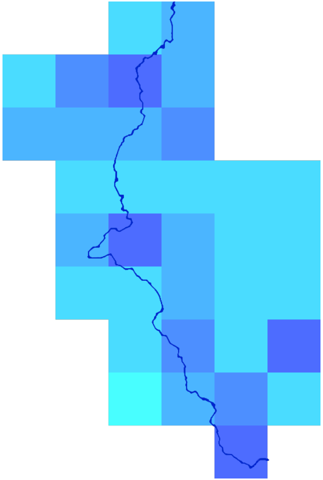
Tree Decomposition

The idea is to decompose the tree into subtrees and distribute these onto computing nodes.

- the case of dynamic distribution of subtrees onto the nodes has been studied [Li et al., 2011]
- we discuss a static distribution

Tree Decomposition

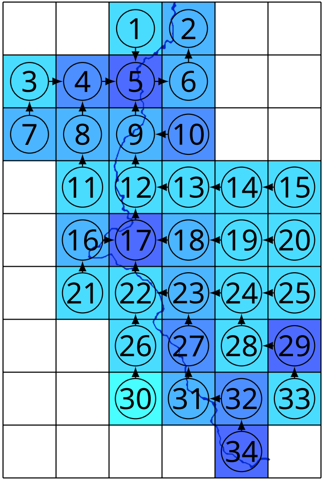
Example: Testbasin



As test basin we use the Moselle with 34 cells.

Tree Decomposition

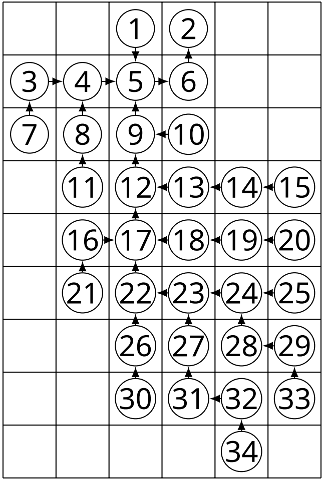
Example: Testbasin



As test basin we use the Moselle with 34 cells.

Tree Decomposition

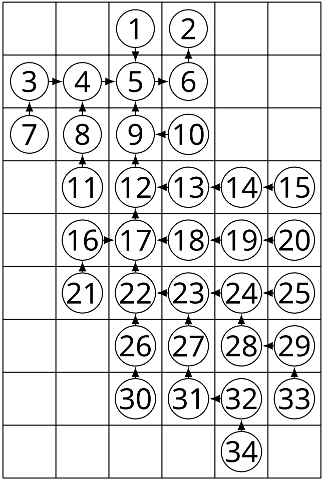
Example: Testbasin



As test basin we use the Moselle with 34 cells.

Tree Decomposition

Goal



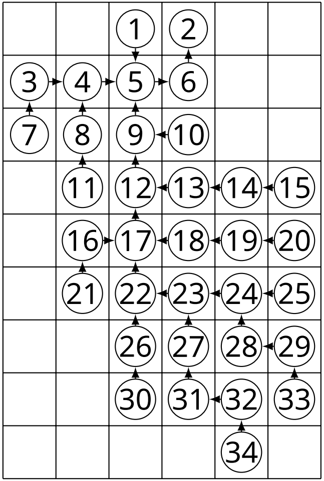
Cut of subtrees with nice sizes recursively and distribute them onto the computing nodes:

1. choose a size range (`lowBound`)
2. search the smallest subtree larger than `lowBound` in the tree
3. cut of that subtree, store it somewhere
4. start from step 2
5. schedule independend subtrees on different nodes
6. establish communication between nodes as far as necessary

UFZ

Tree Decomposition

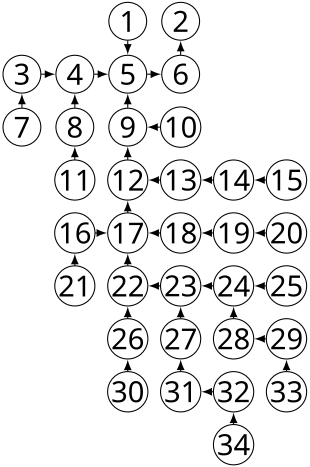
Goal



example: `lowBound`$=3$

Tree Decomposition

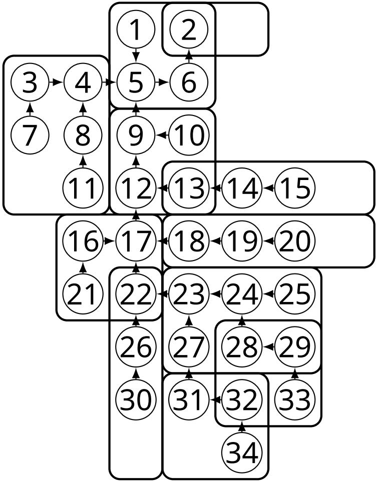
Goal



example: `lowBound`$= 3$

Tree Decomposition

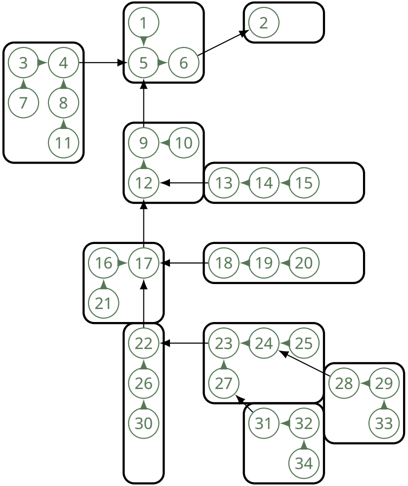
Goal



example: `lowBound`$= 3$

Tree Decomposition

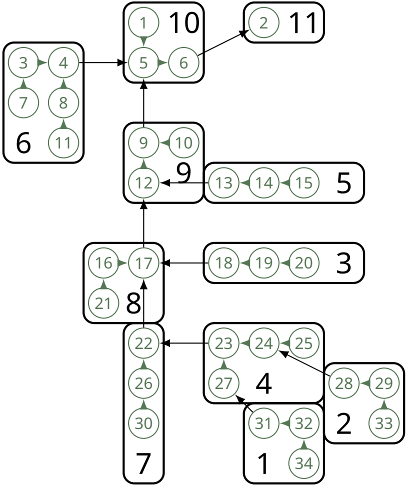
Goal



example: `lowBound`$= 3$

Tree Decomposition

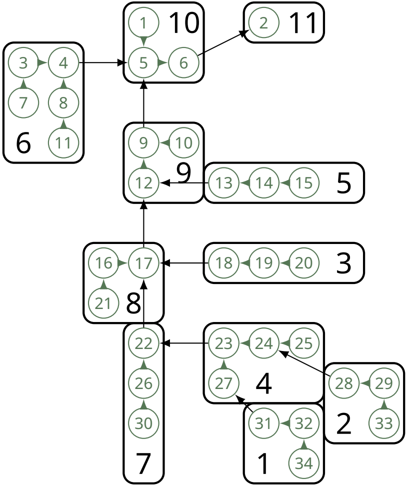
Goal



example: `lowBound`$= 3$

Tree Decomposition

Goal

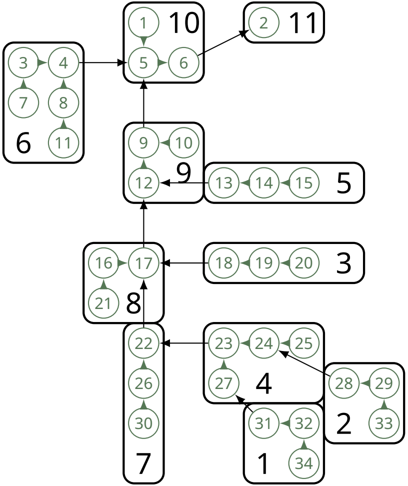


example: $\texttt{lowBound} = 3$

	timeslots						
process 1:	1	4	7	8	9	10	11
process 2:	2	5					
process 3:	3	6					

Tree Decomposition

Goal



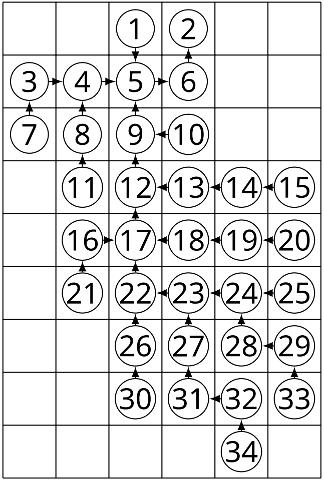
example: `lowBound`$= 3$

	timeslots						
process 1:	1	4	7	8	9	10	11
process 2:	2	5					
process 3:	3	6					

This is the shortest schedule, we can get with this subtrees: The *tree depth* is 7, we can not have a schedule shorter than the tree depth

Tree Decomposition

Tree data structure: basic info



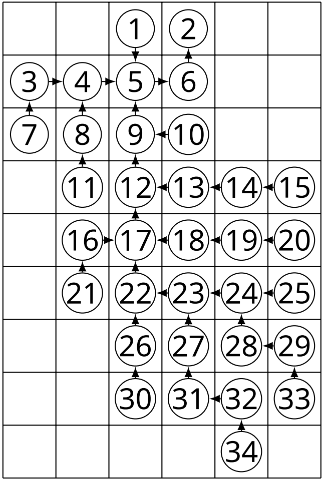
A classical tree data structure contains:

- `post`: a pointer to the parent tree node
- `Nprae`: the number of children
- `prae`: an array of pointers to the children

(note: each node is also a tree)

Tree Decomposition

Tree data structure: specific for tree decomposition



Specific data for the tree decomposition

- `siz`: the size of the tree
- `sizUp`: the size of the smallest subtree larger than `lowBound`
- ST: a pointer to metadata, if the tree node is the root node of a subtree

UFZ

Tree Decomposition

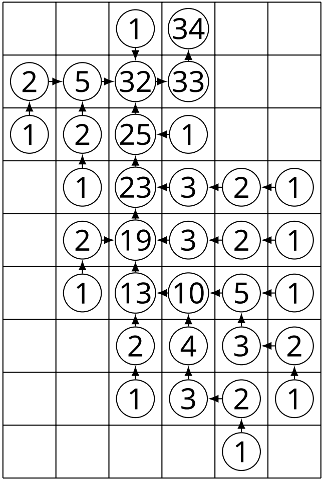
Tree data structure: derived type

```
type ptrTreeNode
  type(treeNode), pointer :: tN
end type ptrTreeNode
type treeNode
  type(ptrTreeNode)                          :: post
  integer(i4)                                :: Nprae
  type(ptrTreeNode),dimension(:),allocatable :: prae
  integer(i4)                                :: siz
  integer(i4)                                :: sizUp
  type(subtreeNode), pointer                 :: ST
end type treeNode
```

UFZ

Tree Decomposition

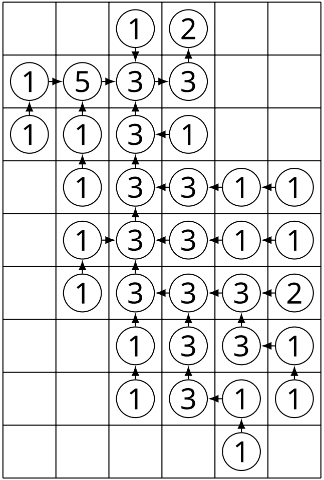
Tree data structure: Set siz for each node



- initialize `siz` with 1 for each tree node
- run though tree in routing order
- for each tree node:
  - for all its children add the value of `siz` of each child to own value of `siz`

Tree Decomposition

Tree data structure: Set sizUp for each node

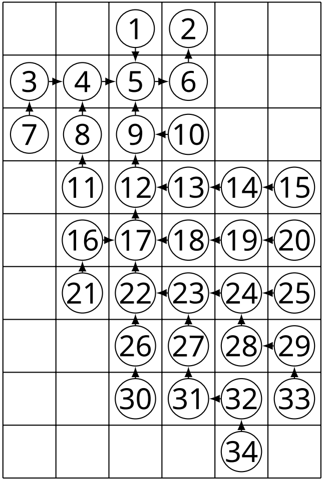


- initialize `sizeUp` with 1 for each tree node
- run though tree in routing order
- for each tree node:
  - for all its children check, if `sizUp` has already been set for at least one child
  - if so, set `sizUp` of current tree node to the smallest of that values of its children
  - if not, check if `siz`≥`sizUp`
  - if so, set `sizUp`=`siz`
  - else `sizUp`=1 (this has to be set, so the subroutine can update the tree after a subtree gets cut of)

UFZ

Tree Decomposition

Cut Of A Subtree

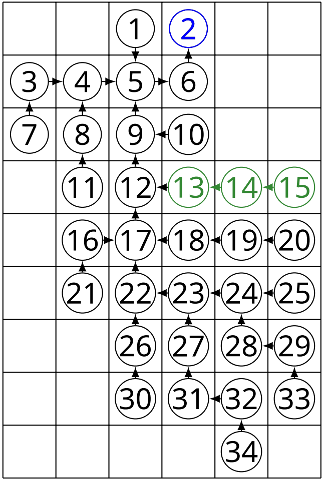


cut of a subtree in sublinear time (depth of tree)
Main idea, follow the branch with the smallest subtree (find_`branch`)
[Thomas H. Cormen, 2009, Harder, 2018]
start as root:

1. if one of the children of the current tree node has `sizUp`> 1
2. then switch to the child with the smallest value for `sizUp`, go to step 1
3. else cut of subtree at that node

Tree Decomposition

Cut Of A Subtree

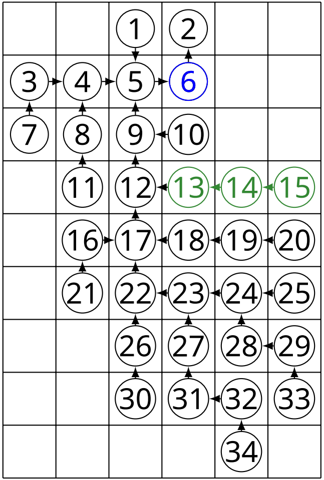


find_branch example:

1. tree node 2 has one child 6
2. 6 has sizUp$>$ 1 therefore we move to 6

Tree Decomposition

Cut Of A Subtree

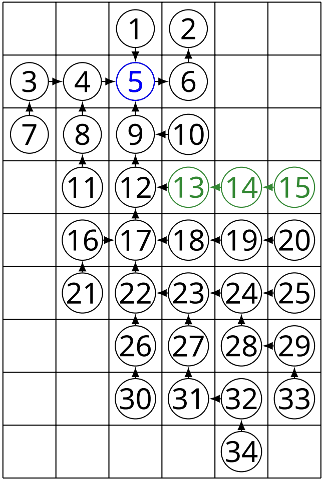


find_branch example:

1. tree node 6 has one child 5
2. 5 has sizUp$>$ 1 therefore we move to 5

Tree Decomposition

Cut Of A Subtree

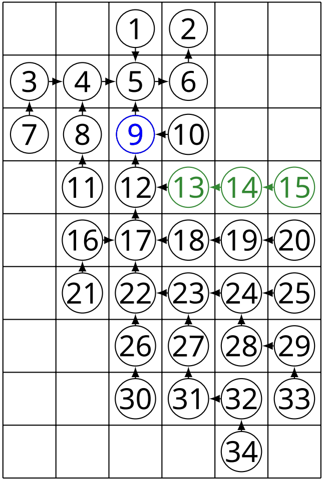


find_branch example:

1. tree node 5 has three childen 1, 4, 9
2. 1 has sizUp=1 (means not set), 4 has sizUp= 5, 9 has sizUp= 3, therefore we move to 9

Tree Decomposition

Cut Of A Subtree

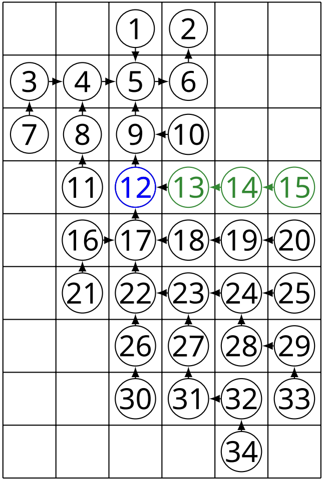


find_branch example:

1. tree node 9 has two childen 10, 12
2. 10 has sizUp=1 (means not set), 12 has sizUp= 3, therefore we move to 12

Tree Decomposition

Cut Of A Subtree

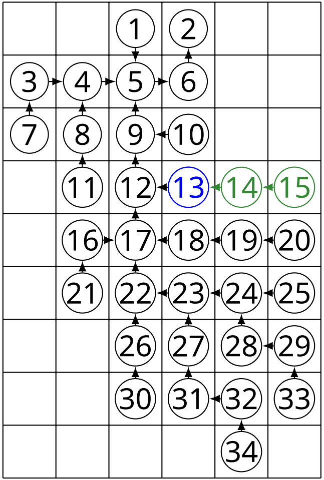


find_branch example:

1. tree node 12 has two childen 13, 17
2. 13 and 17 both have sizUp= 3, therefore we move to 13

Tree Decomposition

Cut Of A Subtree

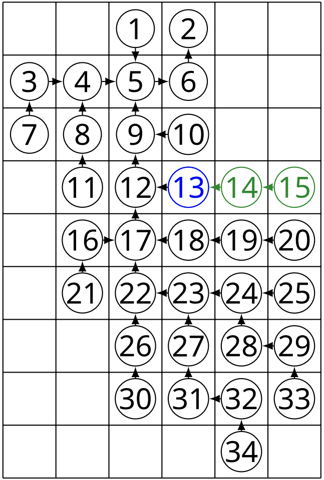


find_branch example:

1. tree node 13 has one child 14
2. sizUp values of all children of 13 are unset, therefore we cut of 13

Tree Decomposition

Cut Of A Subtree

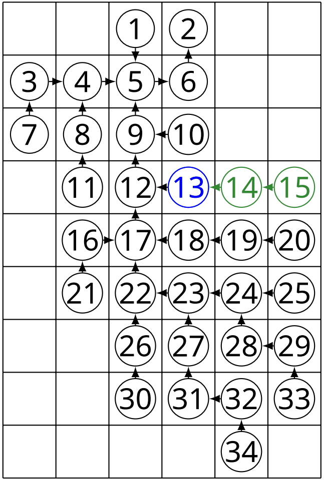


cut of a subtree

- return a pointer to the subtree root (13)
- `update_sizes`
- `initiate_subtreetreenode`
- in the parent node:
  - switch the cut of child with the last child in the `prae` array
  - reduce `Nprae` by one
- `update_tree`

Tree Decomposition

Cut Of A Subtree

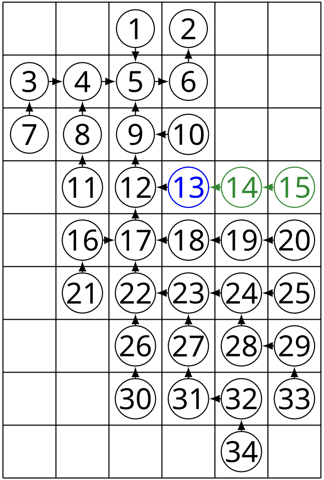


`update_sizes`:
for the cut of subtree (13) with size `redSize` (3)

1. if not root, move to parent
2. reduce `siz` of curret node by `redSiz`, go to 1

Tree Decomposition

Cut Of A Subtree

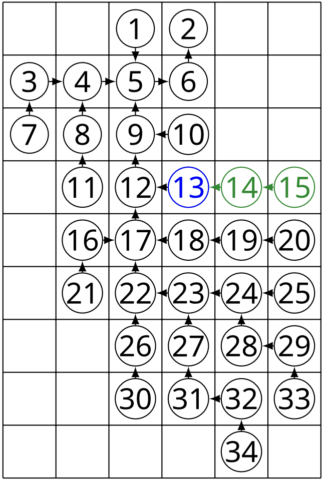


`initiate_subtreetreenode:`

- associate and allocate pointer ST of tree node
- set size of subtreetreenode to current size of tree node (after updating the tree structure, it is the correct size of the subtree)
- initialize other meta data with 0 and nullpointers

Tree Decomposition

Cut Of A Subtree

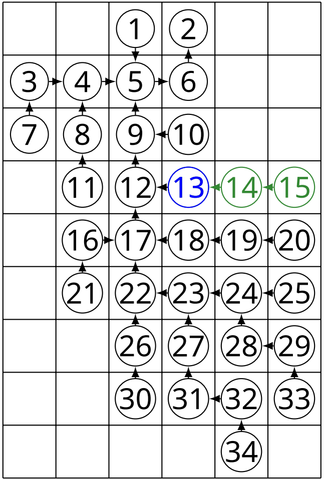


`update_tree`:
start at cut of tree node

1. update `sizUp` (as it was done before)
2. if not root, go to parent, go to step 1

Tree Decomposition

Cut Of A Subtree

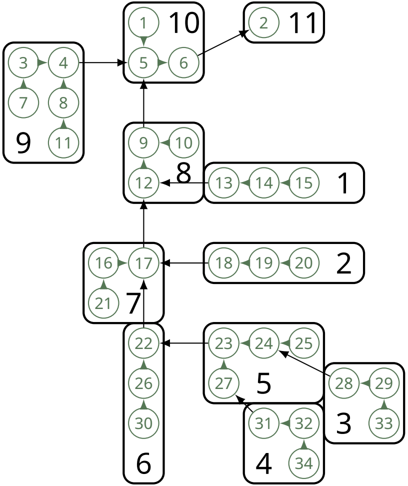


Special cases in the processess of cutting of subtrees:
root has no parent to be updated, so it has to be handled separately

- if one of its children has `sizeUp`>1, follow branch as usual
- else, cut of root

Tree Decomposition

Tree Decomposition

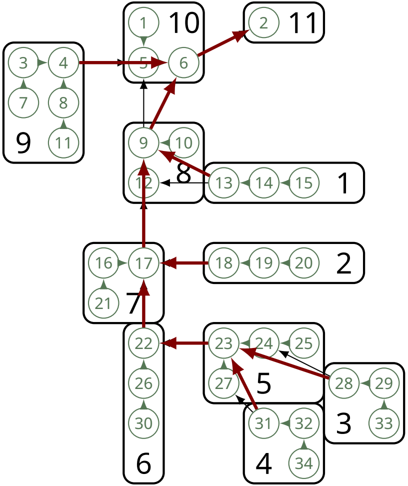


`decompose`

- as long as the last cut of subtree is not root
  - find and cut of subtree, return pointer to subtree
  - write pointer into an array

Tree Decomposition

Tree Decomposition

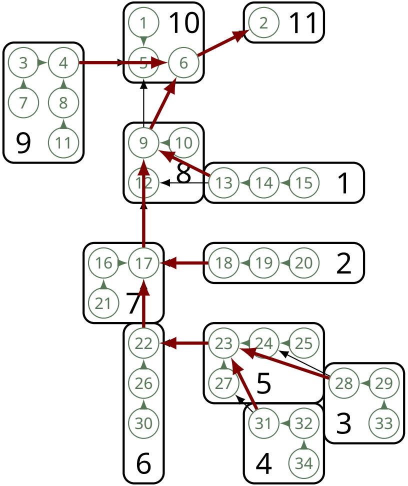


decompose

- as long as the last cut of subtree is not root
  - find and cut of subtree, return pointer to subtree
  - write pointer into an array
- set metadata of subtree nodes appropriately (set pointer to parents and children)

Tree Decomposition

The Subtree Data Structure



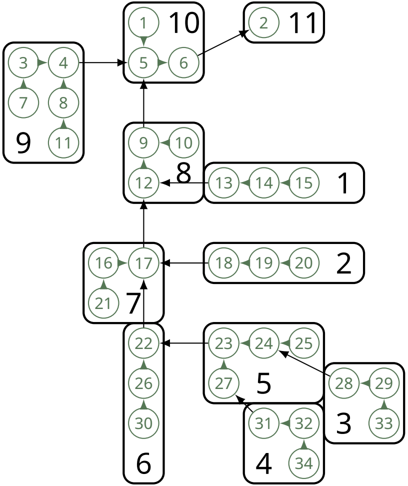
each subtreetree node has an associated pointer to derived type `subtreeNode` with classical tree data structure...

- `postST`: a pointer to the parent tree node (points to tree node)
- `NpraeST`: number of (subtreetree) children
- `praeST`: array of pointers to the (subtreetree) children

UFZ

Tree Decomposition

The Subtree Data Structure



...and specific information for scheduling

- `levelST`: an array saving the distance to the root node and the distance to the farthest leaf in the subtreetree structure for scheduling purposes

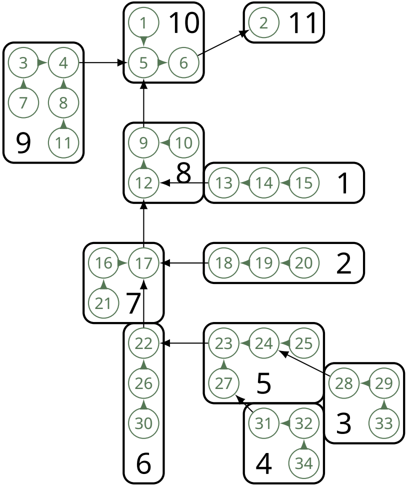
Tree Decomposition

Tree data structure: derived type

```
type treeNode
  ... (s.o.)
  type(subtreeNode), pointer                 :: ST
end type treeNode
type subtreeNode
  type(ptrTreeNode)                          :: postST
  integer(i4)                                :: NpraeST
  type(ptrTreeNode),dimension(:),allocatable :: praeST
  integer(i4)                                :: sizST
  integer(i4), dimension(2)                  :: levelST
end type subtreeNode
```

Tree Decomposition

Tree decomposition: scheduling



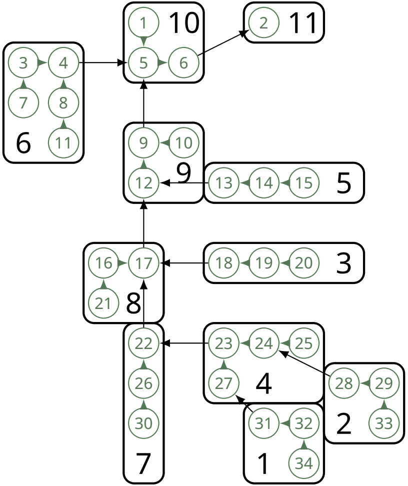
difference between two scheduling methods:

round robin

	timeslots							
process 1:	1	4			7		10	
process 2:	2		5			8		11
process 3:	3			6	9			

Tree Decomposition

Tree decomposition: scheduling



difference between two scheduling methods:

Hu's algorithm[Hu, 1961, Cheng and Sin, 1990]

	timeslots						
process 1:	1	4	7	8	9	10	11
process 2:	2	5					
process 3:	3	6					

MPI

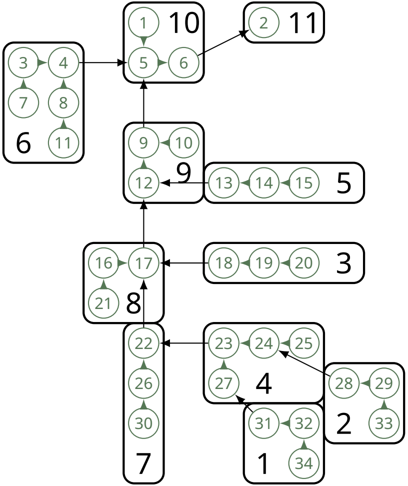
MPI

MPI

- n processes run the program
- each process knows its own rank and the number of processes
- in our case, process with rank 0 is the master process and cooridnates
- each process has its own memory, data has to be exchanged via *message sending interface* (MPI)

Tree Decomposition

Data Exchange Between Computing Nodes: Another Data Structure



tree data structures mainly constist of pointers that refer to physical memory adresses. Sending this data across nodes does not help. Solution:

- save the inices of the grid cells into an array in routing order, where the subtrees lay together
- save a `toNode` array for the links/edges

UFZ

MPI

basic structure

if process has rank 0:

- decompose tree
- prepare array in routing order and toNode array
- send subarrays and corresponding toNodes to the other processes
- send indices of leaves which are connected to subtree roots to processes
  - collect data from root nodes of the subtrees and send it to corresponding leaf nodes of adjacent subtree
- in the end: recollect subarrays

else:

- receive subarray, toNodes, and node indices of corresponding subtrees
  - collect input data for some leaves
  - do routing
  - send root output data to master
- in the end: send subarray back

MPI

Does It Work?

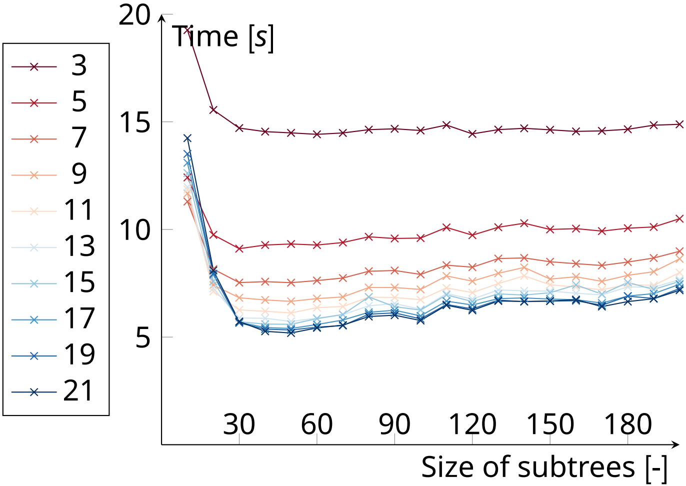
No, not that easily.
Communication needs more time than routing through arrays of size 10000.
A representative basin, Donau, has 26507 cells.
Ideas:

- do routing several times in a subtree, send array of data instead of one value
- or parallelize only independent trees with MPI

We are lucky: the first idea works

MPI

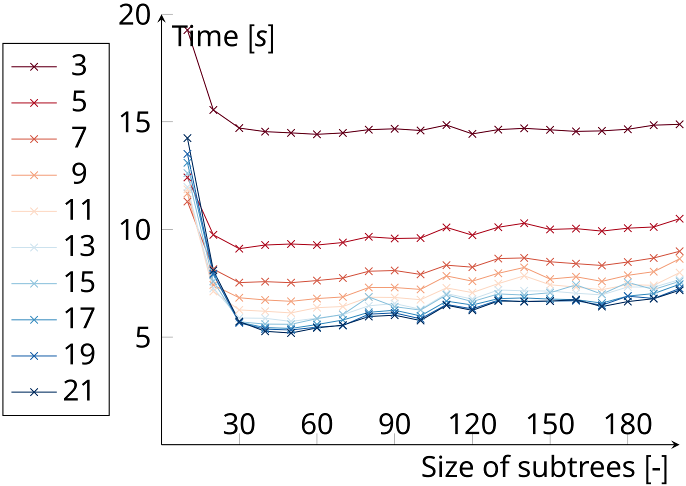
Times



- the basin tested, is the donau with 26507 cells
- collect arrays of 1000 data points before sending
- time is measured every 1000 steps (so 1000×1000 routing per time measurement)

MPI

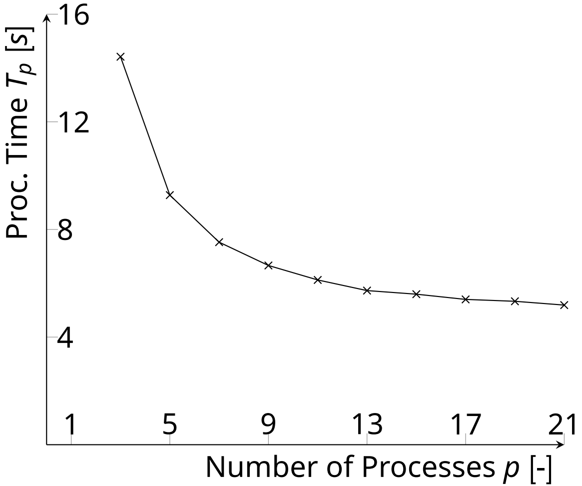
Times



- cutting into subtrees with sizes smaller than 50 results in communication overhead, means communication between the nodes takes more time than routing 1000 times through a subtree

MPI

Times



- for each number of processes the *y*-value is the minimum of the curves above
- more processors still make it faster but it does not tend to zero

OpenMP

OpenMP

differences to MPI

- arrays are not send over the network, but exchanged via shared memory
  - we have to handle data race problems
- sorting of subtrees to threads is not done by us but dynamically:
  - the routing through a subtree is assigned to a task
  - a waiting CPU gets a task/subtree when available

OpenMP

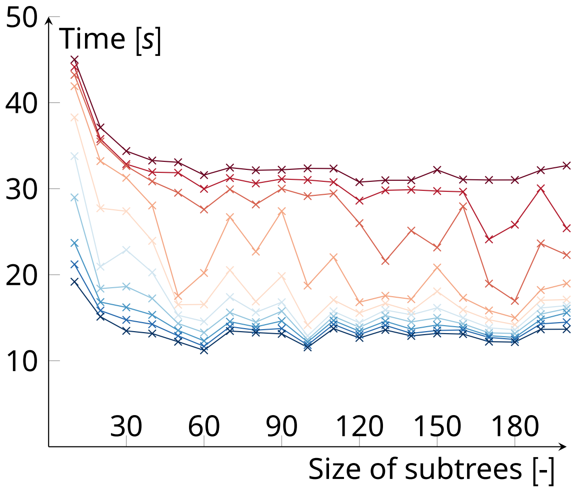
some code

```
!$OMP parallel private(rank) shared(testarray)
!$OMP single
call routing(root,testarray)
!$OMP end single
!$OMP barrier
!$OMP end parallel
```

```
recursive subroutine routing(root,array)
   ...
   do jj=1,root%tN%Nprae
      !$OMP task shared(root,array)
       call routing(root%tN%ST%prae(jj),array)
      !$OMP end task
   end do
   !$OMP taskwait
   if (associated(root%tN%post%tN)) then
        tNode=root%tN%post%tN%ind
        !$OMP critical
        array(tNode)=array(tNode)+array(root%tN%ind)
        !$OMP end critical
     end if
end subroutine routing
```

OpenMP

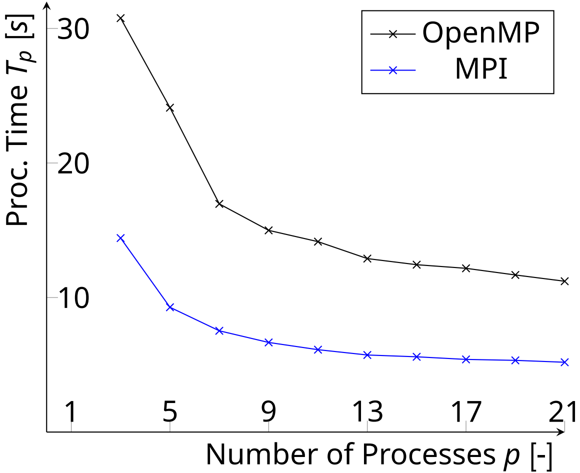
Times



- the basin tested, is again the donau with > 22000 cells
- collect arrays of 1000 data points before writing into shared array
- time is measured every 1000 steps (so 1000×1000 routing per time measurement)

OpenMP

Times



- for each number of processes the y-value is the minimum over `lowBound`

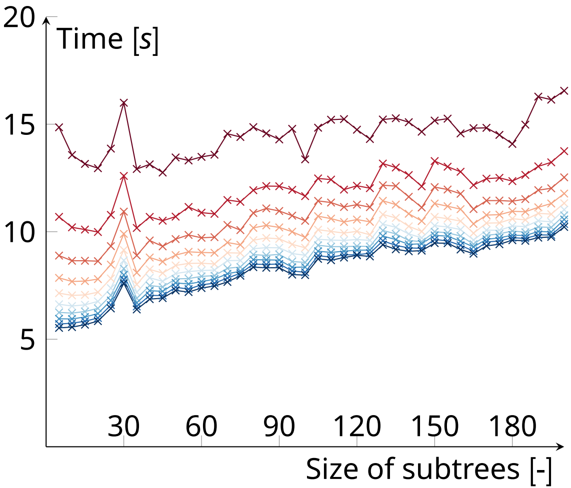
OpenMP

Times, reasons MPI is faster

- code with OpenMP tasks compiles poorly with gnu, intel is better
- Tasks are not sorted by priority. The tree is not a binary tree and it is unbalanced. (Approach: Use newer version of OpenMP, currently reading more literature)

OpenMP

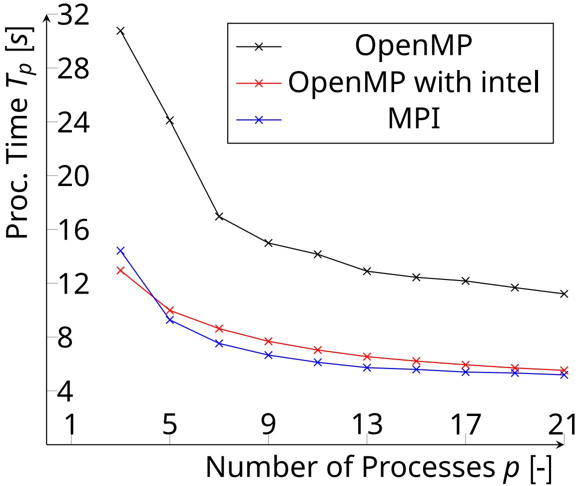
Times, compiled with Intel



- the basin tested, is again the donau with > 22000 cells
- collect arrays of 1000 data points before writing into shared array
- time is measured every 1000 steps (so 1000×1000 routing per time measurement)

OpenMP

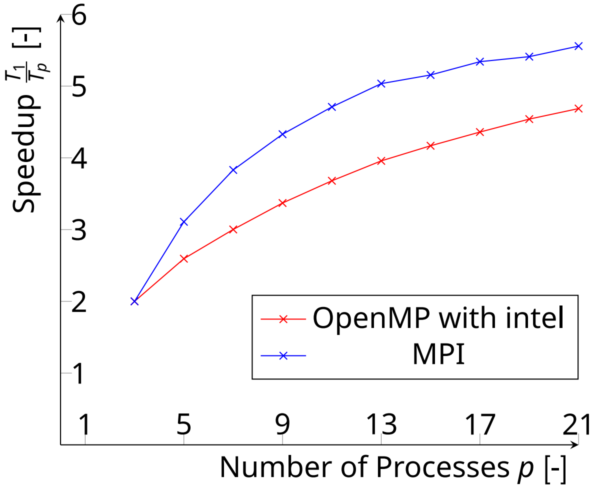
Times



- for each number of processes the y-value is the minimum over `lowBound`

OpenMP

Times



- Speedup is the sequential time devided by the processing time with p processors $\frac{T_1}{T_p}$.
- Best case szenario is $\frac{T_1}{T_p} = p$
- We will never reach this because of the lower bound given by the tree depth

Cheng, T. and Sin, C. (1990).
A state-of-the-art review of parallel-machine scheduling research.
European Journal of Operational Research, 47(3):271–292.

Harder, J. (2018).
Discussions.

Hu, T. C. (1961).
Parallel sequencing and assembly line problems.
Operations research, 9(6):841–848.

Li, T., Wang, G., Chen, J., and Wang, H. (2011).
Dynamic parallelization of hydrological model simulations.
Environmental Modelling & Software, 26(12):1736–1746.

Thomas H. Cormen, Charles E. Leiserson, R. L. R. C. S. (2009).
Introduction to Algorithms.
MIT Press, 3rd edition.

UFZ