Baumzerlegung

Maren Kaluza

November 2018

HELMHOLTZ
CENTRE FOR
ENVIRONMENTAL
RESEARCH - UFZ

ESM

Advanced
Earth System Modelling
Capacity

UFZ

Inhalt

- Einführung
  - Parallelisierung
  - Graphen, Bäume
- Flussnetzwerke (Hydrologie)
- Baumzerlegung
  - Ziel
  - Tree Data Structure
  - Einen Subbaum Abschneiden
  - Baumzerlegung
  - Subtree Data Structure
- MPI
  - Datenaustausch zwischen Rechenknoten
- OpenMP

Einführung

Parallelisierung

Aufteilung von Rechnungen auf mehrere Recheneinheiten, sodass sie zeitgleich gerechnet werden können.

Parallelisierung

Aufteilung von Rechnungen auf mehrere Recheneinheiten, sodass sie zeitgleich gerechnet werden können.

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

Parallelisierung

Aufteilung von Rechnungen auf mehrere Recheneinheiten, sodass sie zeitgleich gerechnet werden können.

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

Parallelisierung

Aufteilung von Rechnungen auf mehrere Recheneinheiten, sodass sie zeitgleich gerechnet werden können.

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

Parallelisierung

Aufteilung von Rechnungen auf mehrere Recheneinheiten, sodass sie zeitgleich gerechnet werden können.

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

Parallelisierung

Aufteilung von Rechnungen auf mehrere Recheneinheiten, sodass sie zeitgleich gerechnet werden können.

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

Parallelisierung

Aufteilung von Rechnungen auf mehrere Recheneinheiten, sodass sie zeitgleich gerechnet werden können.

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

Parallelisierung

Aufteilung von Rechnungen auf mehrere Recheneinheiten, sodass sie zeitgleich gerechnet werden können.

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

Parallelisierung

Aufteilung von Rechnungen auf mehrere Recheneinheiten, sodass sie zeitgleich gerechnet werden können.

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

Parallelisierung

Aufteilung von Rechnungen auf mehrere Recheneinheiten, sodass sie zeitgleich gerechnet werden können.

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

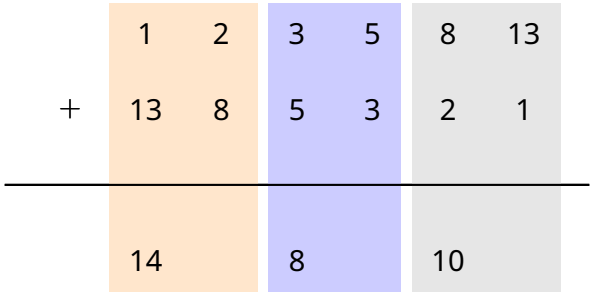
Parallelisierung

Aufteilung von Rechnungen auf mehrere Recheneinheiten, sodass sie zeitgleich gerechnet werden können.

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

Parallelisierung

Aufteilung von Rechnungen auf mehrere Recheneinheiten, sodass sie zeitgleich gerechnet werden können.



Gewinn: Die neue Rechenzeit ergibt sich im Optimalfall aus der seriellen geteilt durch die Anzahl der Recheneinheiten

UFZ

Parallelisierung

Ein Käsekuchen benötigt etwa 18 Stunden, um in einem Backofen zu garen. Wieviel Zeit benötigt der Käsekuchen zum Garen, wenn 6 Backöfen zur Verfügung stehen?

Parallelisierung

1 2 3 5 8 13

Parallelisierung

Fibonacci-Folge (um 1 verschoben)

1 2 3 5 8 13

Parallelisierung

Fibonacci-Folge (um 1 verschoben)

1 2 3 5 8 13 21

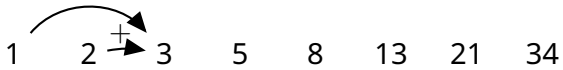
Parallelisierung

Fibonacci-Folge (um 1 verschoben)

1 2 3 5 8 13 21 34

Parallelisierung

Fibonacci-Folge (um 1 verschoben)

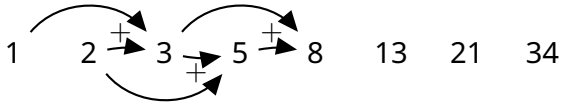


Parallelisierung

Fibonacci-Folge (um 1 verschoben)

Parallelisierung

Fibonacci-Folge (um 1 verschoben)

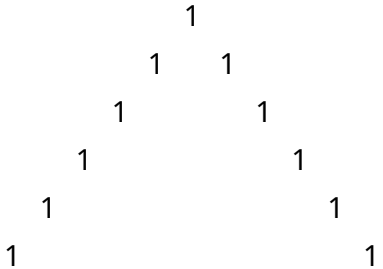


Parallelisierung

Pascalsches Dreieck

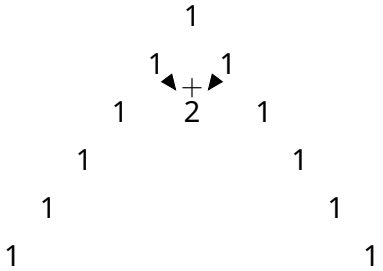
Parallelisierung

Pascalsches Dreieck



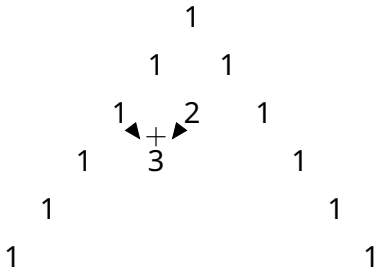
Parallelisierung

Pascalsches Dreieck



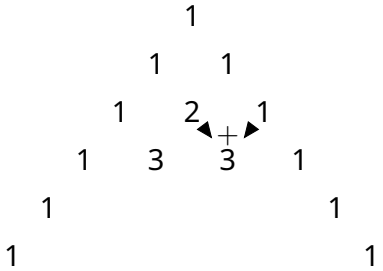
Parallelisierung

Pascalsches Dreieck



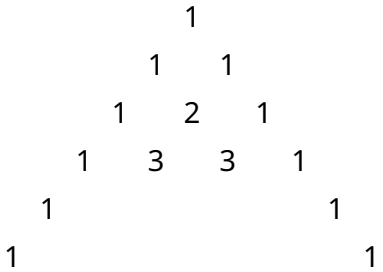
Parallelisierung

Pascalsches Dreieck



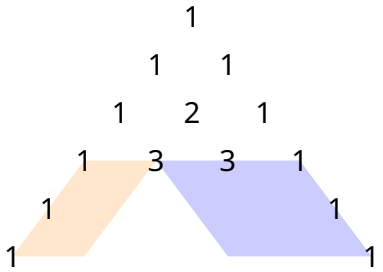
Parallelisierung

Pascalsches Dreieck



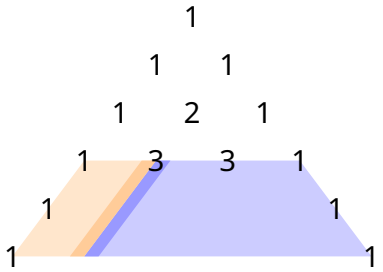
Parallelisierung

Pascalsches Dreieck



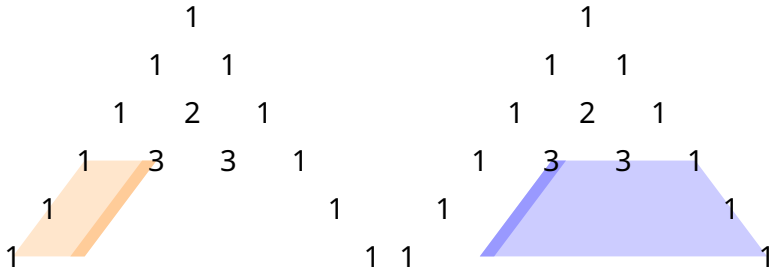
Parallelisierung

Pascalsches Dreieck



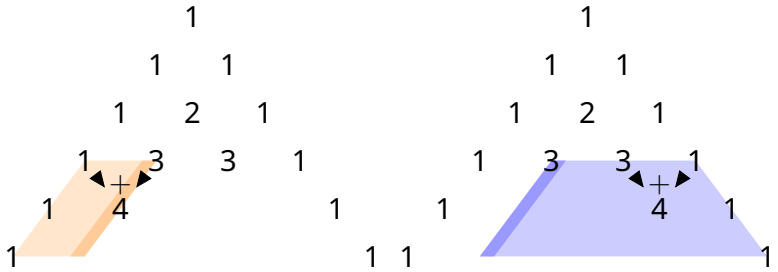
Parallelisierung

Pascalsches Dreieck



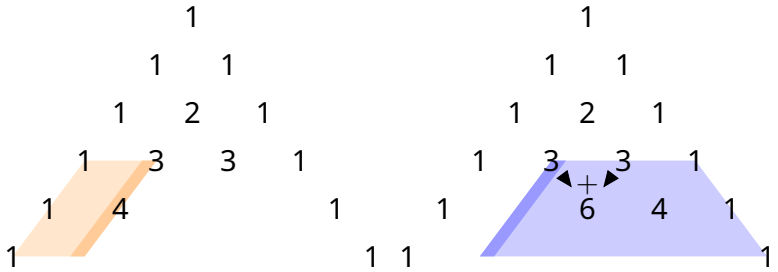
Parallelisierung

Pascalsches Dreieck



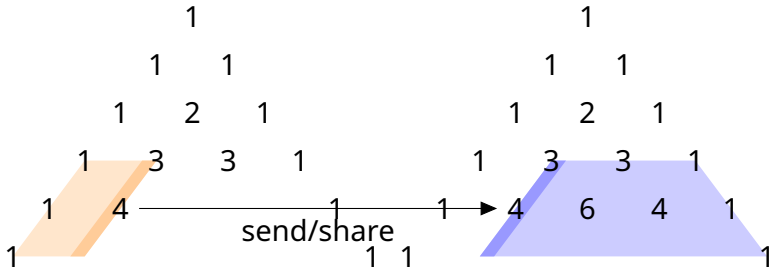
Parallelisierung

Pascalsches Dreieck



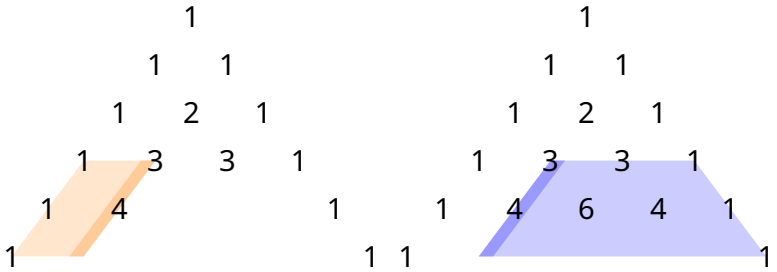
Parallelisierung

Pascalsches Dreieck



Parallelisierung

Pascalsches Dreieck

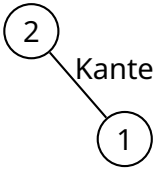


Graphen

1 Knoten

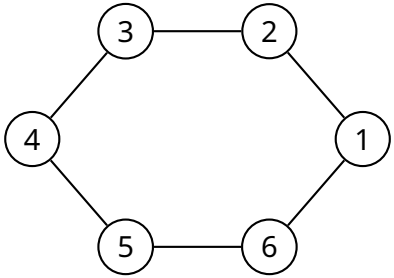
Graph

Graphen



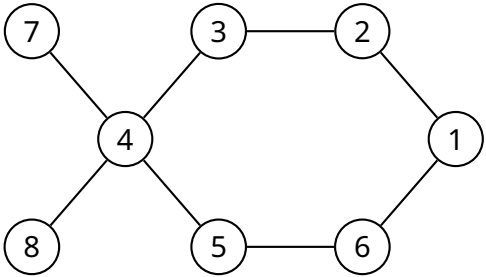
Graph

Graphen



Kreis (insbesondere ein Graph)

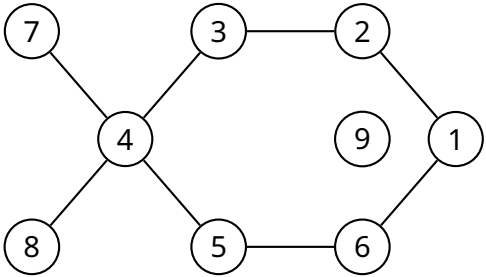
Graphen



Kein Kreis, aber ein zusammenhängender Graph

UFZ

Graphen



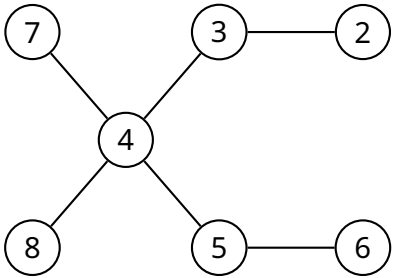
Ein unzusammenhängender Graph

Bäume

Ein Baum ist ein kreisfeier, zusammenhängender Graph

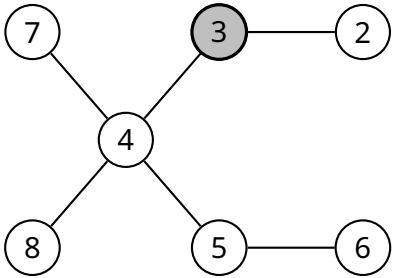
Bäume

Ein Baum ist ein kreisfeier, zusammenhängender Graph



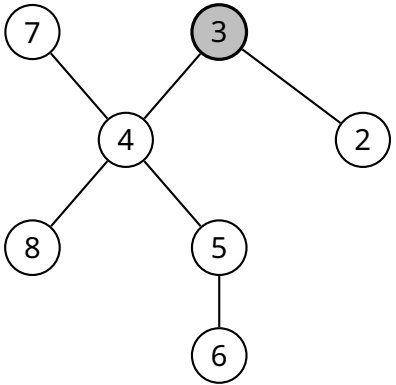
Bäume

Ein Baum ist ein kreisfeier, zusammenhängender Graph



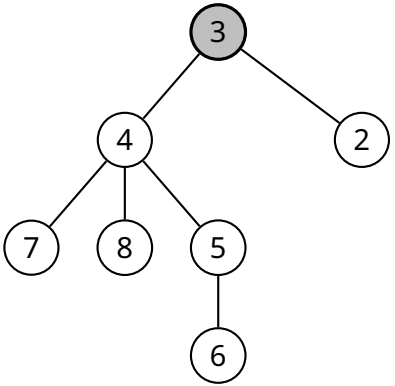
Bäume

Ein Baum ist ein kreisfeier, zusammenhängender Graph



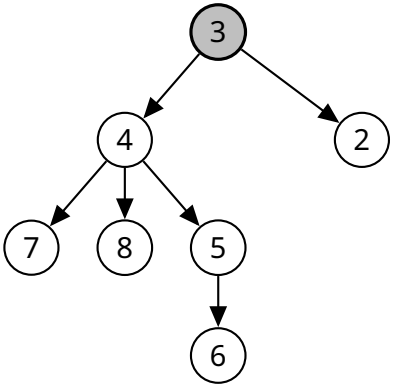
Bäume

Ein Baum ist ein kreisfeier, zusammenhängender Graph



Bäume

Ein Baum ist ein kreisfeier, zusammenhängender Graph

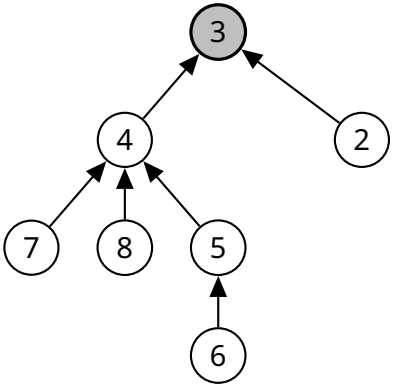


gewurzelter Baum, Out-tree

UFZ

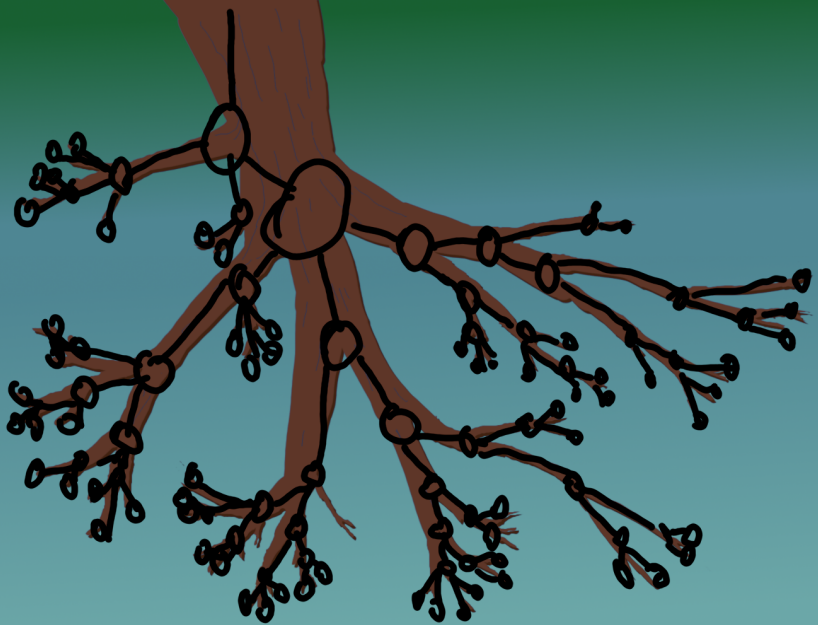
Bäume

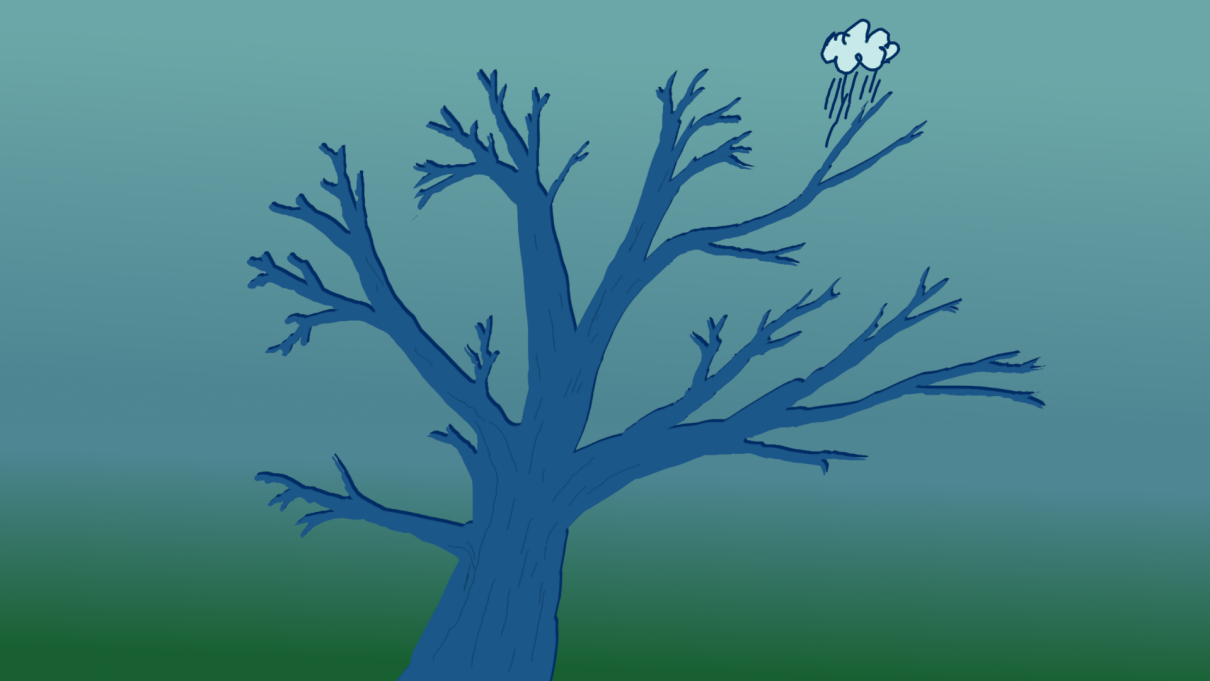
Ein Baum ist ein kreisfeier, zusammenhängender Graph

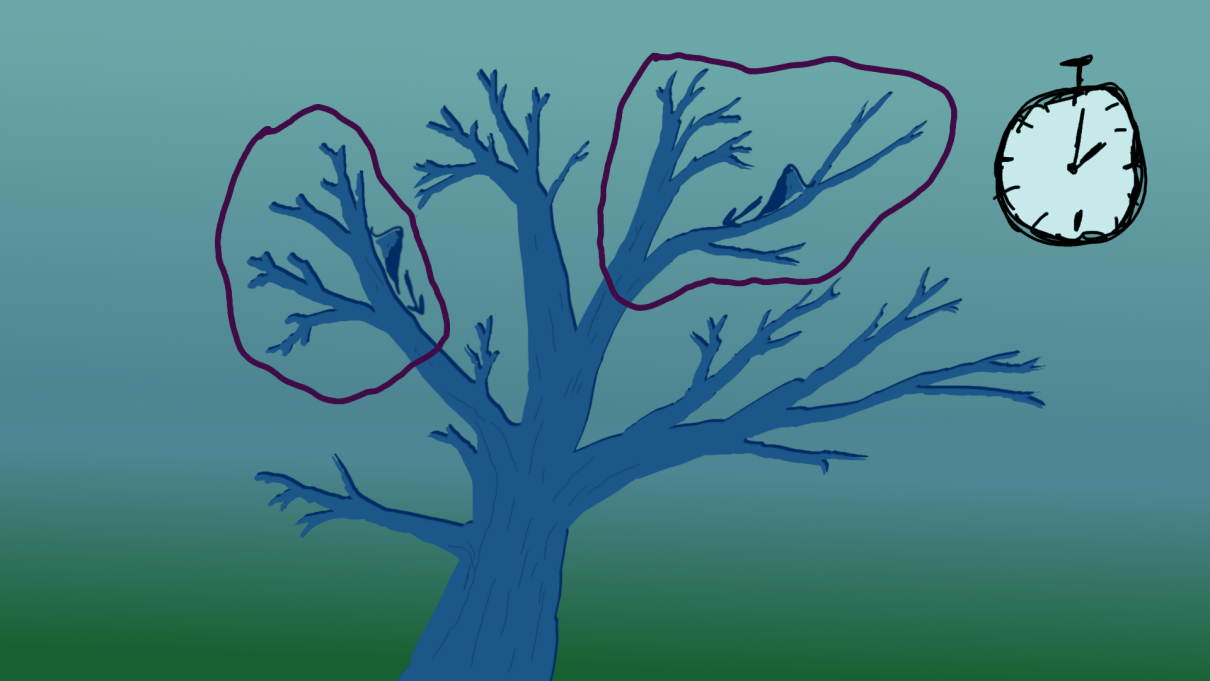


gewurzelter Baum, In-tree

Flussnetzwerke (Hydrologie)







Baumzerlegung

Baumzerlegung

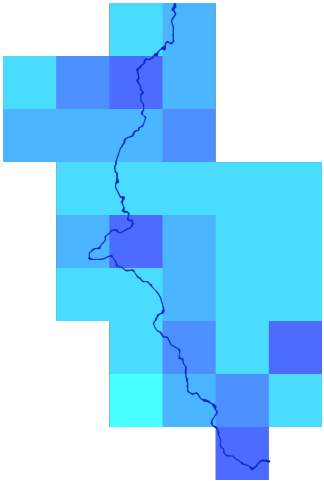
Die Idee ist, den Baum in Subbäume zu zerlegen und diese auf die Recheneinheiten zu verteilen.

- Der dynamische Fall, in dem die Subbäume zur Rechenzeit abgeschnitten und verteilt werden, wurde untersucht [Li et al., 2011].
- Wir beschäftigen uns mit dem statischen Fall.

UFZ

Baumzerlegung

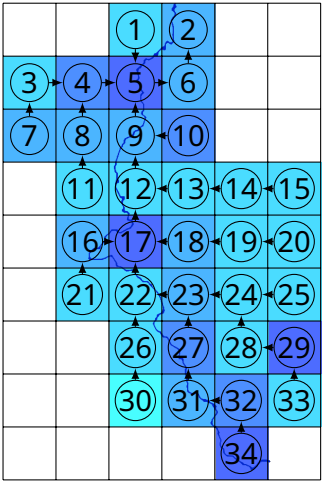
Beispiel: Testbasin



Als Testbasin nehmen wir die Mosel mit 34 Gitterzellen.

Baumzerlegung

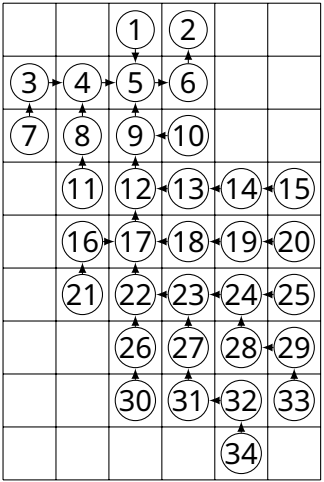
Beispiel: Testbasin



Als Testbasin nehmen wir die Mosel mit 34 Gitterzellen.

Baumzerlegung

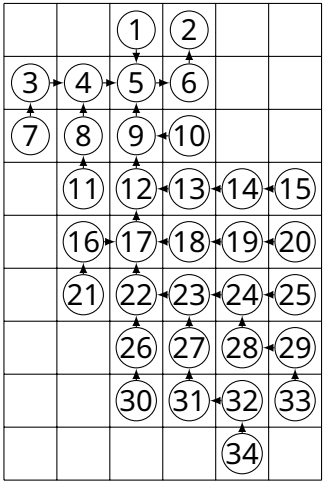
Beispiel: Testbasin



Als Testbasin nehmen wir die Mosel mit 34 Gitterzellen.

Baumzerlegung

Ziel



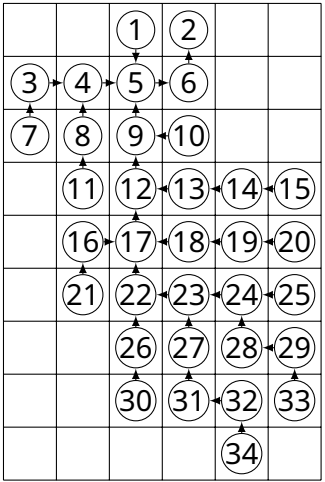
Wir schneiden Subbäume passender Größe rekursiv ab und verteilen sie auf die Recheneinheiten:

1. wähle eine Größenordnung der Subbäume (`lowBound`)
2. suche nach dem kleinsten Subbaum größer `lowBound` im Baum
3. schneide den Subbaum ab, speichere ihn irgendwo
4. wiederhole ab Schritt 2
5. verteile unabhängige Subbäume auf verschiedene Recheneinheiten
6. stelle notwendige Kommunikation zwischen den Recheneinheiten her

UFZ

Baumzerlegung

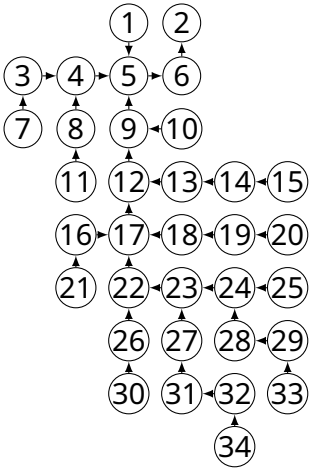
Ziel



example: `lowBound`$= 3$

Baumzerlegung

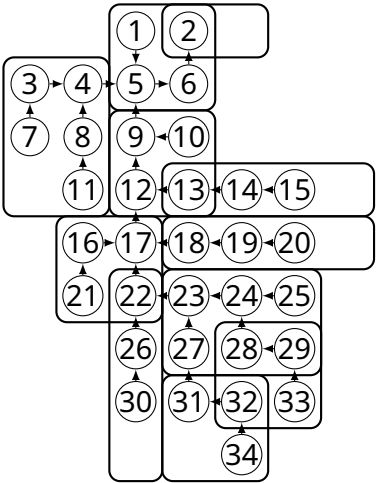
Ziel



example: `lowBound`$= 3$

Baumzerlegung

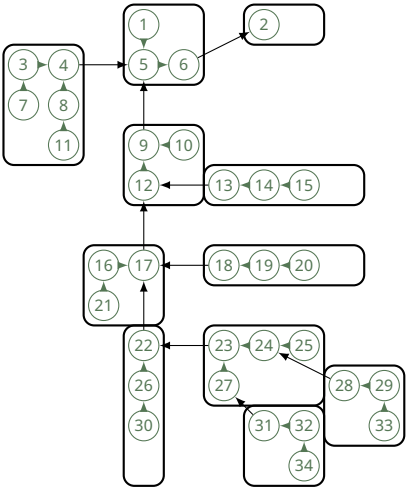
Ziel



example: `lowBound`$= 3$

Baumzerlegung

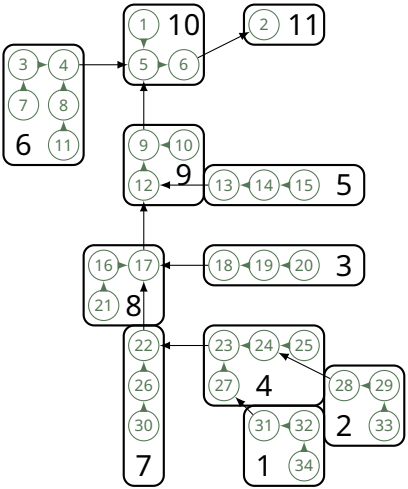
Ziel



Beispiel: `lowBound`$= 3$

Baumzerlegung

Ziel

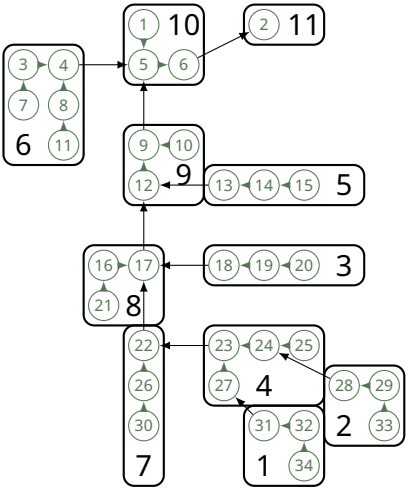


Beispiel: lowBound$= 3$

UFZ

Baumzerlegung

Ziel

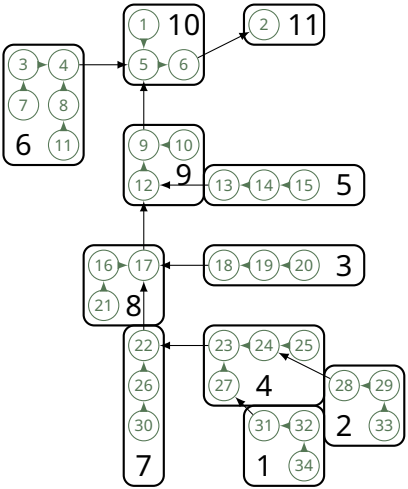


Beispiel: lowBound$= 3$

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

Baumzerlegung

Ziel



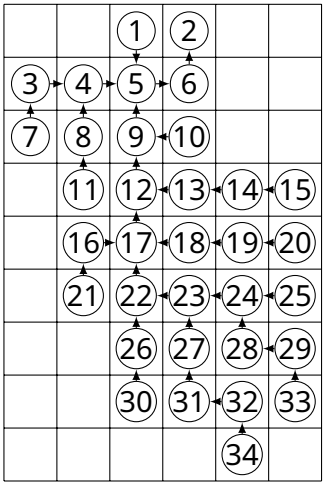
Beispiel: lowBound$= 3$

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

Das ist der kürzest mögliche Schedule, den wir mit dieser Zerlegung erhalten können: Die *Baumtiefe* ist 7 und wir können keinen Schedule erhalten, der diese unterschreitet

Baumzerlegung

Tree Data Structure: Grundinformation



Eine klassische Baumdatenstruktur besteht aus:

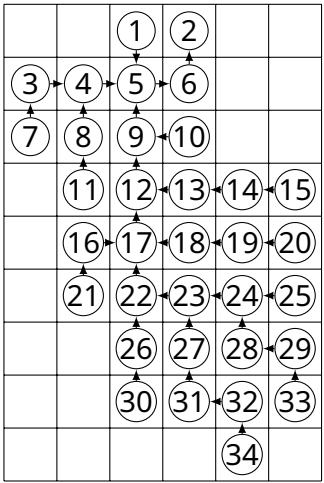
- `post`: Einem Pointer (Zeiger/Pfeil) zum Elternknoten
- `Nprae`: Der Anzahl der Kinder
- `prae`: Einem Array aus Pointern zu den Kindern

(Beachte: Jeder Knoten im Baum ist ebenfalls ein Baum)

UFZ

Baumzerlegung

Tree Data Structure: Spezifisch für unsere Baumzerlegung



Spezifische Metadaten für unsere Baumzerlegung

- `siz`: die Größe des Baums
- `sizUp`: die Größe des kleinsten Subbaums größer als `lowBound`
- ST: ein Pointer auf eine Metadatenstruktur, falls der Knoten auch Wurzel eines Subbaums ist

UFZ

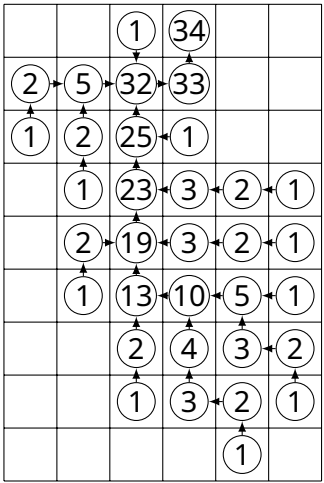
Baumzerlegung

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
```

Baumzerlegung

Tree Data Structure: Bestimmung von siz für jeden Knoten

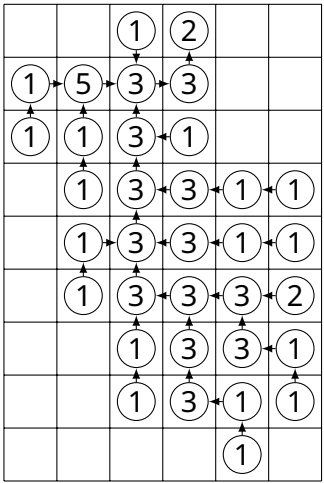


- initialisiere `siz` mit 1 für jeden Knoten im Baum
- gehe in Routing Order die Knoten im Baum ab
- für jeden Knoten:
  - addiere den Wert `siz` jedes seiner Kinder auf den eigenen Wert `siz`

UFZ

Baumzerlegung

Tree Data Structure: Bestimmung von sizUp für jeden Knoten im Baum

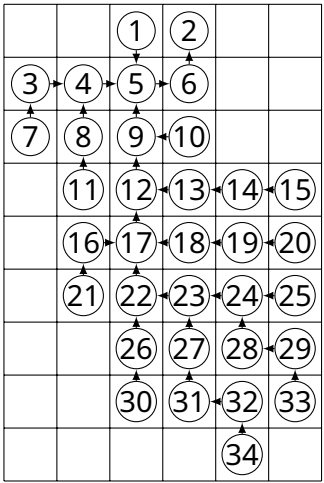


- initialisiere `sizeUp` mit 1 für jeden Knoten im Baum
- gehe in Routing Order die Knoten im Baum ab
- für jeden Knoten:
  - für jedes Kind prüfe, ob `sizUp` bereits für min. ein Kind gesetzt wurde
  - falls ja, setze `sizUp` auf den kleinsten dieser Werte
  - falls nein, prüfe, ob $\texttt{siz} \geq \texttt{sizUp}$
  - falls ja, setze $\texttt{sizUp} = \texttt{siz}$
  - falls nein `sizUp=1`

UFZ

Baumzerlegung

Einen Subbaum abschneiden



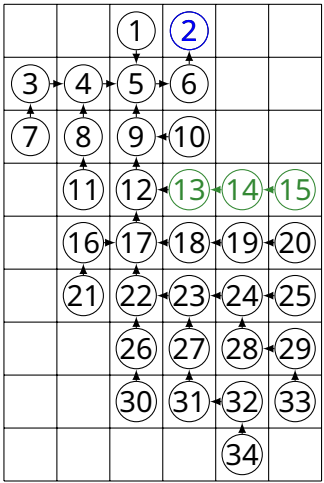
Einen Subbaum in sublinearer Laufzeit abschneiden (Baumtiefe)
Hauptidee, folge den Abzeigungen mit dem kleinsten Subbaum größer (find_branch)
[Thomas H. Cormen, 2009, Harder, 2018]
Starte von der Wurzel:

1. falls ein Kind des aktuellen Baums `sizUp`>1 hat
2. gehe zum Kind mit dem kleinsten Wert für `sizUp`, und starte von Schritt 1
3. andernfalls schneide den Subbaum ab

UFZ

Baumzerlegung

Einen Subbaum abschneiden

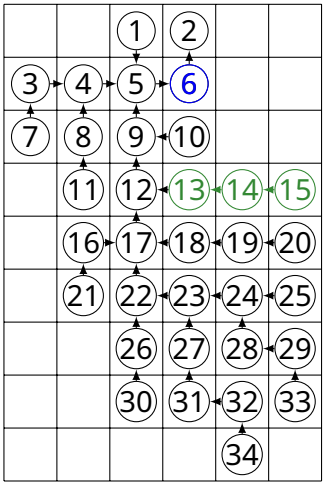


find_branch example:

1. Knoten 2 hat ein Kind 6
2. 6 hat sizUp$>$ 1 also gehen wir zu 6

Baumzerlegung

Einen Subbaum abschneiden

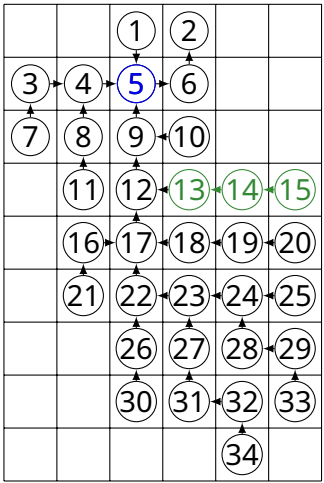


find_branch example:

1. Knoten 6 hat ein Kind 5
2. 5 hat sizUp$>$ 1 also gehen wir zu 5

Baumzerlegung

Einen Subbaum abschneiden

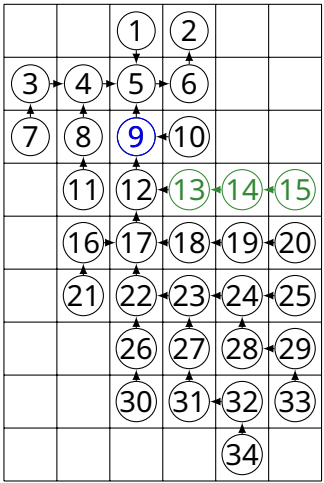


find_branch example:

1. Knoten 5 hat drei Kinder 1, 4, 9
2. 1 hat sizUp=1 (ungesetzt), 4 hat sizUp= 5, 9 hat sizUp= 3, also gehen wir zu 9

Baumzerlegung

Einen Subbaum abschneiden

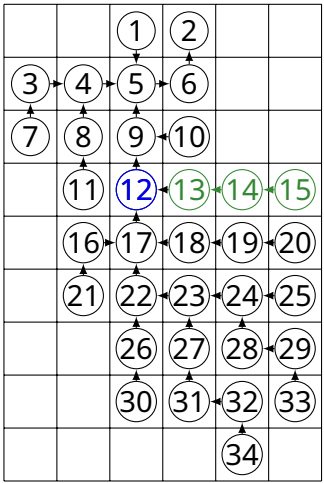


find_branch example:

1. Knoten 9 hat zwei Kinder 10, 12
2. 10 hat sizUp=1 (ungesetzt), 12 hat sizUp= 3, also gehen wir zu 12

Baumzerlegung

Einen Subbaum abschneiden

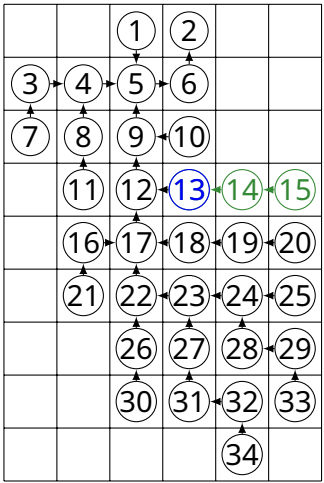


find_branch example:

1. Knoten 12 hat zwei Kinder 13, 17
2. 13 und 17 haben beide sizUp$=$ 3, also gehen wir zu 13

Baumzerlegung

Einen Subbaum abschneiden

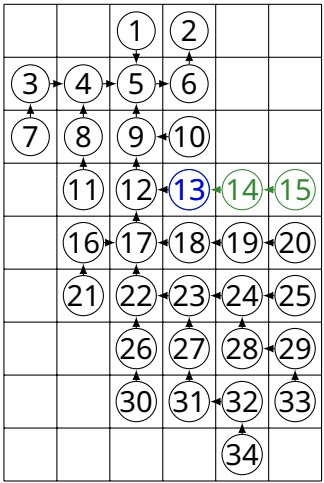


find_branch example:

1. Knoten 13 hat ein Kind 14
2. die Werte von sizUp für alle Kinder von 13 sind ungesetzt, daher schneiden wir 13 ab

Baumzerlegung

Einen Subbaum abschneiden



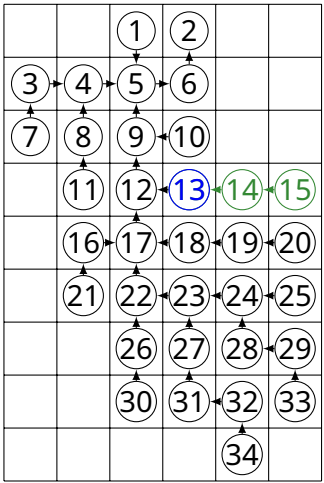
einen Subbaum abschneiden

- gebe den Pointer auf die Subbaumwurzel (13) zurück
- `update_sizes`
- `initiate_subtreetreenode`
- Im Elternknoten:
  - vertausche das abgeschnittene Kind mit dem letzten im Array der Kinder `prae`
  - reduziere `Nprae` um eins
- `update_tree`

UFZ

Baumzerlegung

Einen Subbaum abschneiden



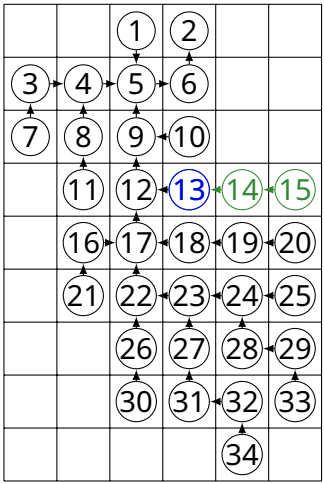
`update_sizes`:
für den abgeschnittenen Baum (13) mit Größe `redSize` (3)

1. falls wir noch nicht bei der Wurzel sind, gehe zum Elternknoten
2. reduziere `siz` des aktuellen Knoten um `redSiz`, und gehe zu Schritt 1

UFZ

Baumzerlegung

Einen Subbaum abschneiden



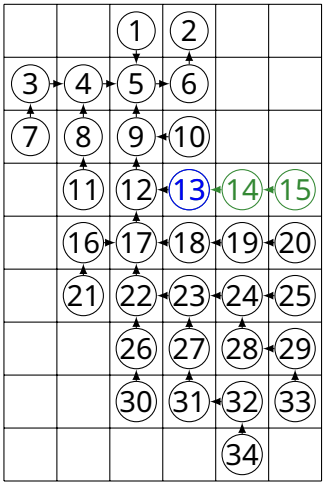
`initiate_subtreetreenode`:

- Initialisiere den Pointer ST des abgeschnittenen Subbaumknotens
- Setze die Größe des Subbaumbaumknotens auf die aktuelle Größe des Knotens (nach dem Updaten des Baums hat er die richtige Größe)
- Initialisiere übrige Metadaten mit 0 und Nullpointern

UFZ

Baumzerlegung

Einen Subbaum abschneiden



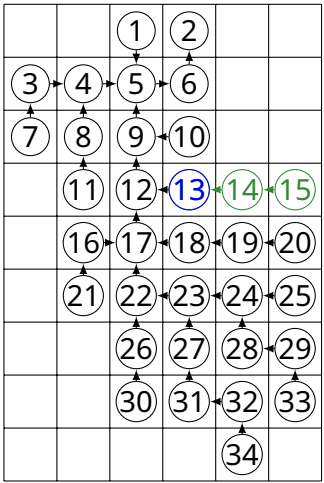
`update_tree`:
Starte beim abgeschnittenen Subbaum

1. update `sizUp` (wie zuvor)
2. falls der aktuelle Knoten nicht die Wurzel ist, gehe zum Elternknoten und starte von Schritt 1

UFZ

Baumzerlegung

Einen Subbaum abschneiden



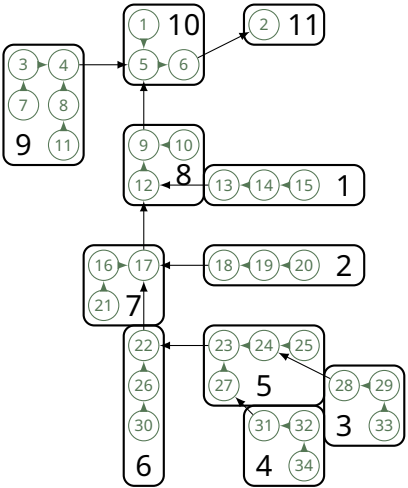
Spezialfälle beim Abschneiden von Subbäumen:
Die Wurzel hat keinen Elternknoten, der geupdatet werden muss und wird daher gesondert behandelt

- Falls eines der Kinder `sizeUp`> 1 hat, folge dem Zweig wie üblich
- andernfalls, schneide die Wurzel ab

UFZ

Baumzerlegung

Baumzerlegung



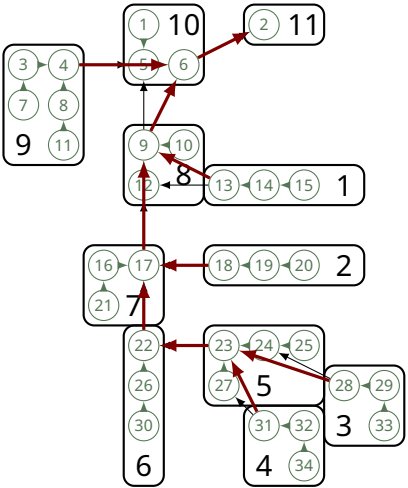
`decompose`

- Solange der letzte abgeschnittene Subbaum nicht die Wurzel ist:
  - Finde einen geeigneten Subbaum, schneide ihn ab und gib den Pointer zurück
  - schreibe den Pointer in ein Array

UFZ

Baumzerlegung

Baumzerlegung



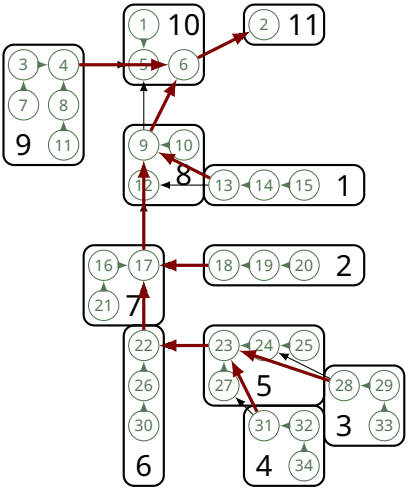
`decompose`

- Solange der letzte abgeschnittene Subbaum nicht die Wurzel ist:
  - Finde einen geeigneten Subbaum, schneide ihn ab und gib den Pointer zurück
  - schreibe den Pointer in ein Array
- Setze die Metadaten der Subbaumknoten passend (setze Pointer zu Elternknoten und Kindern im Baum der Subbäume)

UFZ

Baumzerlegung

Subtree Data Structure



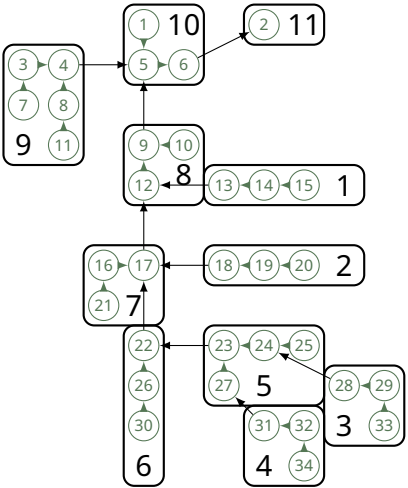
Jeder Subbaum hat einen assoziierten Pointer auf einen derived type `subtreeNode`, der die klassische Baumdatenstruktur umfasst…

- `postST`: ein Pointer auf den Elternknoten (zeigt auf den Wurzelknoten des Elternsubbaums in der Baumdatenstruktur)
- `NpraeST`: Anzahl der Kinder im Subbaumbaum
- `praeST`: Array der Pointer auf die Kinder im Subbaumbaum

UFZ

Baumzerlegung

Subtree Data Structure



…und spezifische Informationen für das Scheduling

- `levelST`: ein Array mit den Entfernungen zum Wurzelknoten und zum weitest entfernten Blatt im Subbaumbaum

Baumzerlegung

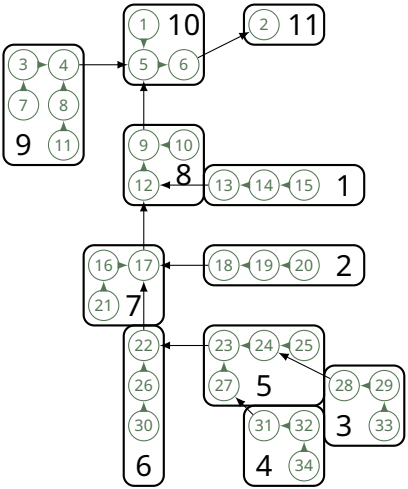
Tree data structure: derived type

```
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
```

UFZ

Baumzerlegung

Scheduling



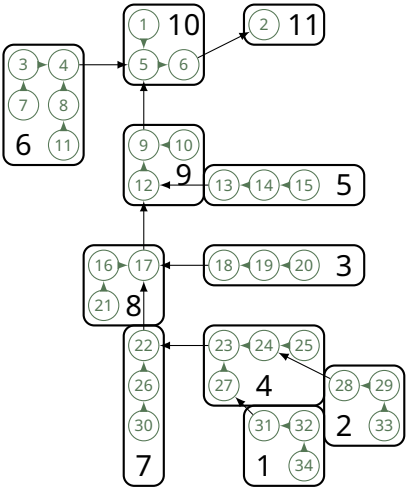
Unterschied zweier Scheduling Methoden:

round robin

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

Baumzerlegung

Scheduling



Unterschied zweier Scheduling Methoden:

Hus Algorithmus
[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

UFZ

MPI

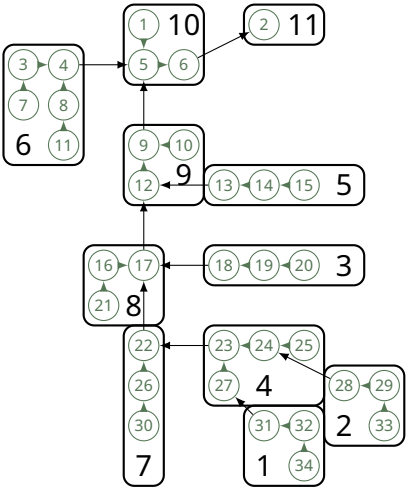
MPI

- n Prozesse führen das Programm aus
- jeder Prozess kennt seinen rank und die Anzahl der Prozesse
- in unserem Fall ist der Prozess mit rank 0 der Masterprozess und koordiniert
- jeder Prozess hat seinen eigenen Speicher, Daten müssen via *message sending interface* (MPI) ausgetauscht werden

UFZ

Baumzerlegung

Datenaustausch zwischen Rechenknoten: Eine weitere Datenstruktur



Baumdatenstrukturen bestehen zum großen Teil aus Pointern, die auf physikalischen Speicher zeigen. Diese über Rechenknoten zu verschicken hilft wenig. Lösung:

- speichere die Indizes der Gitterzellen in einem Array in Routing Order, in dem die Subbäume am Stück liegen
- speichere ein `toNode` Array für die Kanten

UFZ

MPI

Struktur des Programms

Falls Prozess rank 0 hat:

- Zerlege Baum
- Bereite Array der Gitterzellen in Routing Order und toNode Array vor
- Verschicke Subarrays dieser beiden Arrays zu den anderen Prozessen
- Verschicke die Indizes der Blätter, in denen Wurzeln anderer Subbäume enden zu den jeweiligen Prozessen
  - Erhalte Daten der Wurzelknoten der Subbäume und sende sie weiter an die entsprechenden Blattknoten der benachbarten Subbäume
- Zum Schluss: Sammele die Subarrays wieder ein

anderfalls:

- Erhalte Subarrays, toNodes, und Indizes der verbundenen Subbäume
  - Sammele Daten für Blätter
  - Routing
  - Verschicke die berechneten Wurzeldaten zum Masterprozess
- Zum Schluss: Verschicke Subarrays zurück

UFZ

MPI

Funktioniert es?

Nein, nicht so einfach.
Kommunikation benötigt mehr Zeit, als das Routing durch Arrays mit Größen in der Größenordnung von 10000. Ein repräsentatives Basin ist die Donau und hat 26507 Gitterzellen.
Ideen:

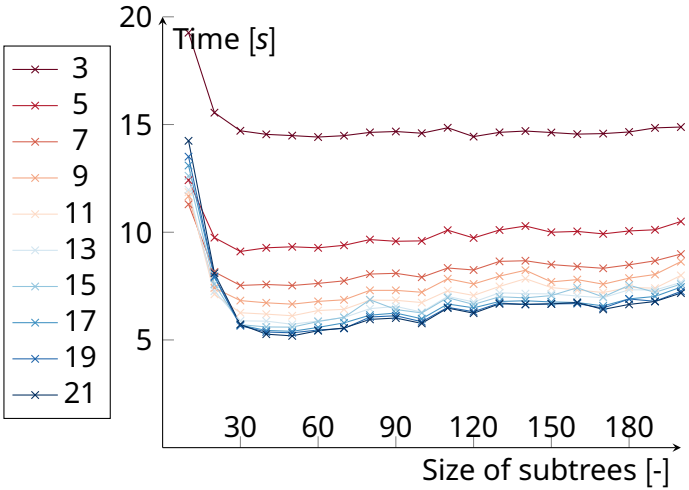
- wiederhole Routing viele Male in jedem Subtree und halte die Daten in einem Array vor
- parallelisiere nur unabhängige Bäume mit MPI

Wir haben Glück: die erste Idee funktioniert

UFZ

MPI

Times

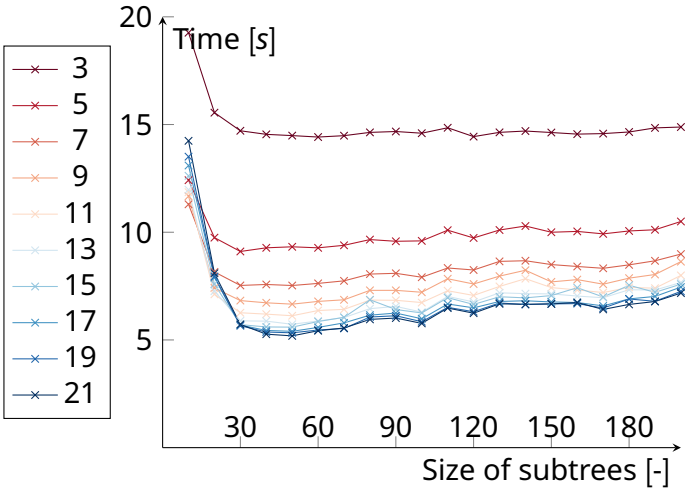


- das Basin ist die Donau mit 26507 Gitterzellen
- sammeln Arrays von Daten mit Länge 1000 bevor wir sie verschicken
- die Zeit wird alle 1000 Schritte gemessen (sodass 1000×1000 mal pro Zeitmessung geroutet wird)

UFZ

MPI

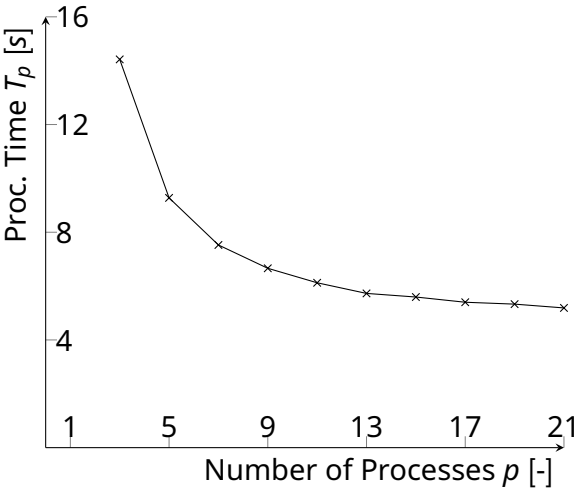
Times



- Zerlegung in Subbäume mit Größen kleiner als 50 bewirkt einen Kommunikationsoverhead, das heißt, die Kommunikation zwischen Knoten braucht mehr Zeit, als das 1000-malige Routen durch einen Subbaum

MPI

Times



- für jede Anzahl an Prozessen ist der y-Wert das Minimum der oberen Kurven
- mehr Prozesse machen es schneller, aber die Kurve geht nicht gegen 0

UFZ

OpenMP

UFZ

OpenMP

Unterschiede zu MPI

- Arrays werden nicht über das Netzwerk gesendet, sondern über shared memory ausgetauscht
  - wir müssen uns mit Race Problemen befassen
- Das Sortieren der Subbäume auf Threads wird nicht durch uns getan, sondern von OpenMP dynamisch:
  - das Routing durch einen Subtree wird einer Aufgabe (Task) zugewiesen
  - Eine wartende CPU bekommt eine solche Aufgabe, wenn verfügbar

UFZ

OpenMP

some code

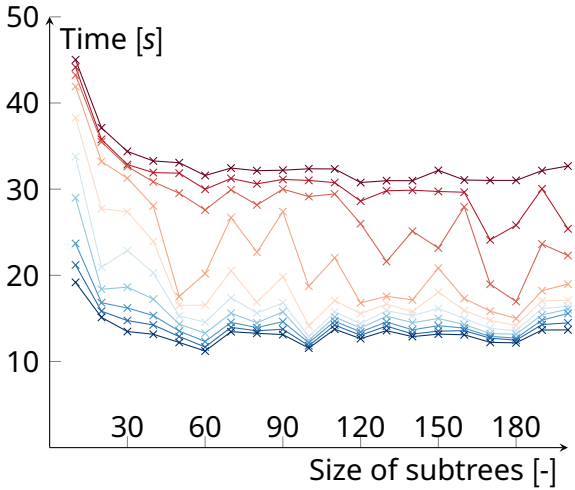
```
!$OMP parallel private(rank) shared(testarray)
!$OMP single
call routing(root,testarray)
!$OMP end single
!$OMP barrier
!$OMP end parallel
```

UFZ

```
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

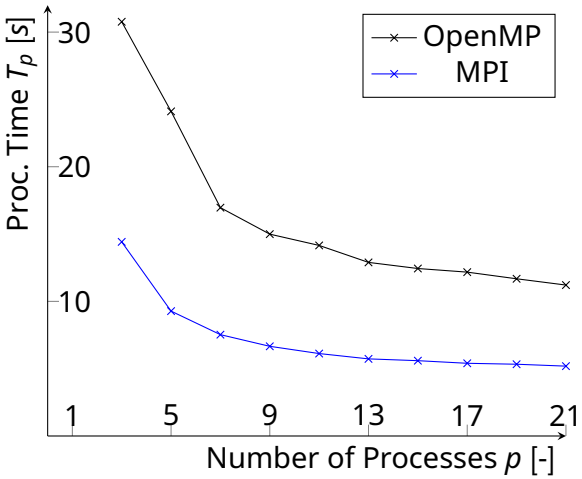


- Das Basin ist wieder die Donau mit $>$ 22000 Gitterzellen
- Wir sammeln Arrays der Größe 1000, bevor wir sie ins shared arrays schreiben
- Die Zeit wird alle 1000 Schritte gemessen (sodass wieder 1000×1000 mal pro Zeitschritt geroutet wird)

UFZ

OpenMP

Times



- für jede Anzahl an Prozessen ist der y-Wert wieder das Minimum über alle Werte zu einem `lowBound`-Wert

OpenMP

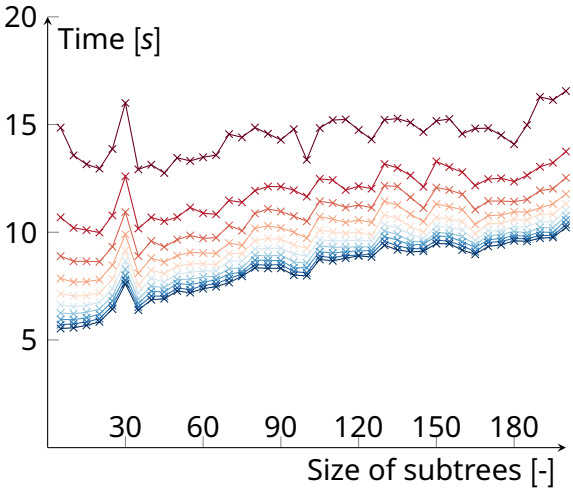
Times, Gründe, weshalb MPI schneller ist

- Programmcode mit OpenMP-Tasks kompiliert miserabel mit gnu-Kompilern. Intel-Kompiler sind besser
- Tasks werden nicht nach Priorität sortiert. Der Baum ist kein Binärbaum und unbalanciert. (Ansatz: Nutze neuere OpenMP Version, lese derzeit Literatur dazu)

UFZ

OpenMP

Times, mit Intel kompiliert

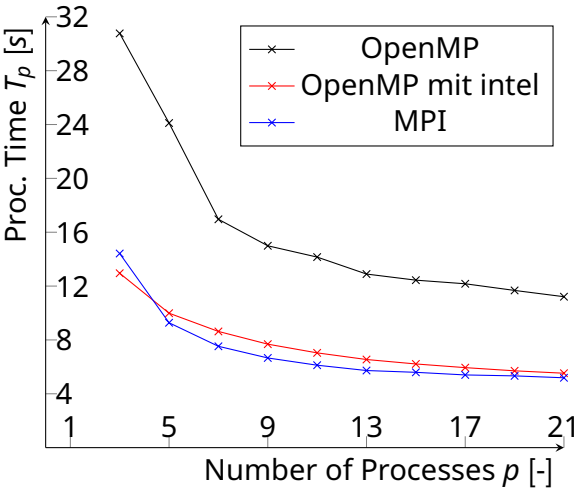


- Das Basin ist wieder die Donau mit $>$ 22000 Gitterzellen
- Wir sammeln Arrays der Größe 1000, bevor wir sie ins shared arrays schreiben
- Die Zeit wird alle 1000 Schritte gemessen (sodass wieder 1000×1000 mal pro Zeitschritt geroutet wird)

UFZ

OpenMP

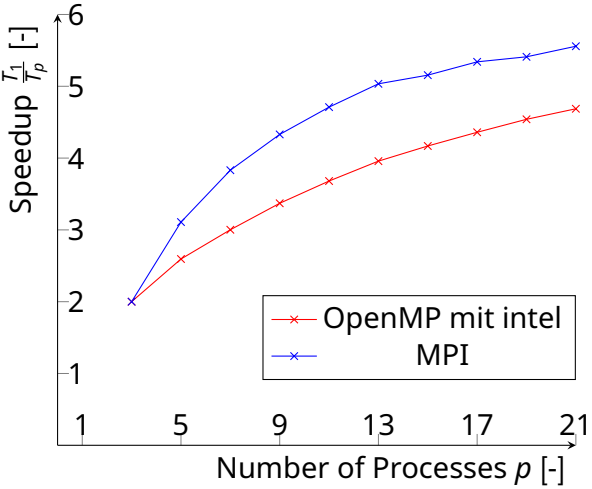
Times



- für jede Anzahl an Prozessen ist der y-Wert wieder das Minimum über alle Werte zu einem `lowBound`-Wert

OpenMP

Times



- Speedup ist die sequentielle Laufzeit geteilt durch die Laufzeit mit der entsprechenden Prozessanzahl p: $\frac{T_1}{T_p}$.
- Das bestmögliche Ergebnis in der Parallelisierung ist $\frac{T_1}{T_p} = p$
- Das werden wir nie erreichen, da im Fall der Baumparallelisierung eine Schranke durch die Baumtiefe gegeben ist

UFZ

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