Parallel Programming
Exercise Session 5

Spring 2024

Plan für heute

- Nachbesprechung Übung 4
- Demo: Loop unrolling
- Zusätzliche Aufgabe
- Theorie Recap: Divide & Conquer
- Vorbesprechung Übung 5
- Quiz

Nachbesprechung Übung 4

Pipelining

Washing - 50 min, **Dryer** - 90 min, **Iron** - 15 min

Task a) total time if strictly sequential order?

Pipelining

Washing - 50 min, **Dryer** - 90 min, **Iron** - 15 min

Task a) total time if strictly sequential order?

50 + 90 + 15

155

Pipelining

Washing - 50 min, **Dryer** - 90 min, **Iron** - 15 min

Task a) total time if strictly sequential order?

155 + 155

Pipelining

Washing - 50 min, **Dryer** - 90 min, **Iron** - 15 min

Task a) total time if strictly sequential order?

155 + 155 + 155 + 155 = 620

Pipelining

Washing - 50 min, **Dryer** - 90 min, **Iron** - 15 min

Task b) what would be a better (faster) strategy?

Pipelining

Washing - 50 min, **Dryer** - 90 min, **Iron** - 15 min

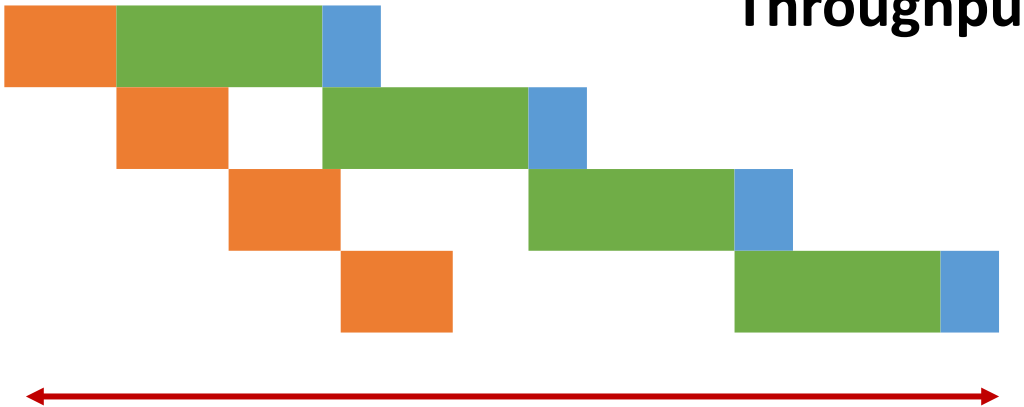
Task b) what would be a better (faster) strategy?

Pipelining

Washing - 50 min, **Dryer** - 90 min, **Iron** - 15 min

Task b) what would be a better (faster) strategy?

Throughput?



1 person per 106.25 min (425 min / 4)

With lead in & lead out
(fixed number of students)

Pipelining

Washing - 50 min, **Dryer** - 90 min, **Iron** - 15 min

Task b) what would be a better (faster) strategy?

Throughput?

1 person per 90 min

Full utilization
(indefinite number of students)

Pipelining

Washing - 50 min, **Dryer** - 90 min, **Iron** - 15 min

Task b) what would be a better (faster) strategy?

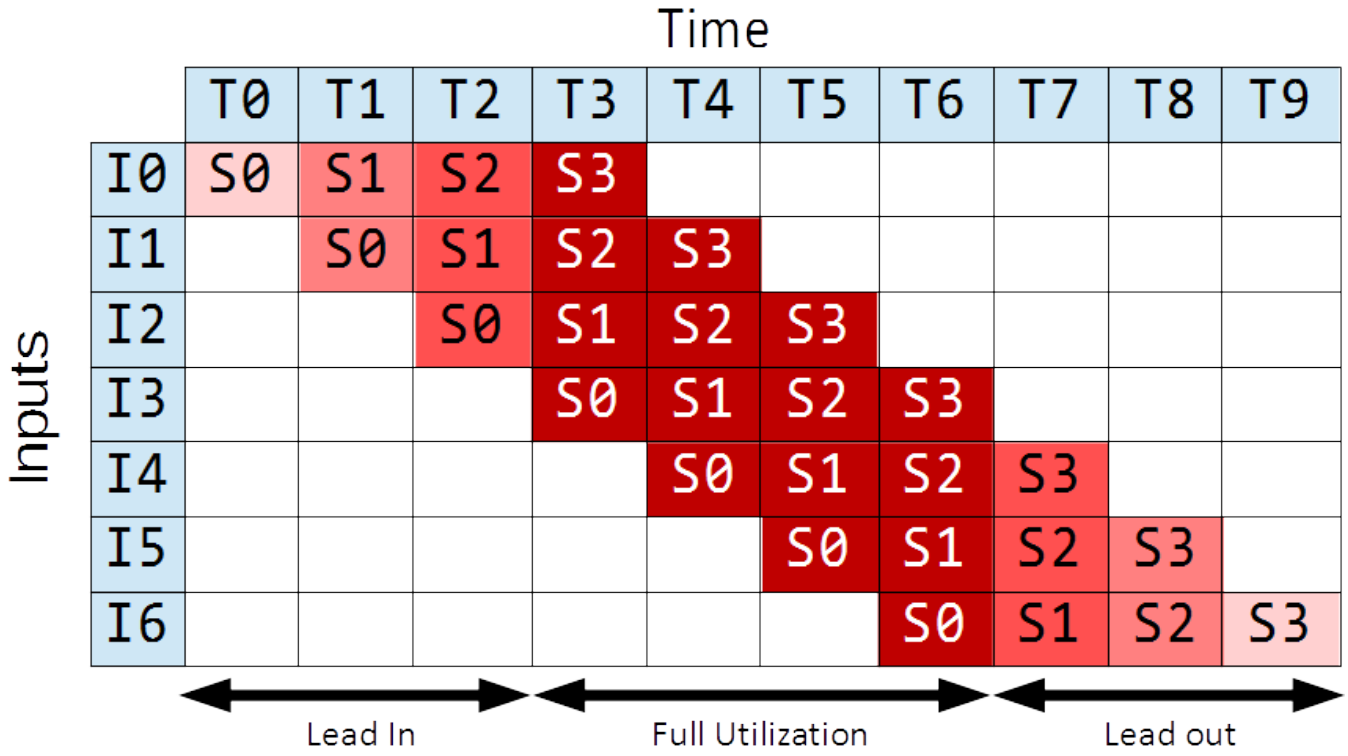
Throughput?

1 person per 90 min

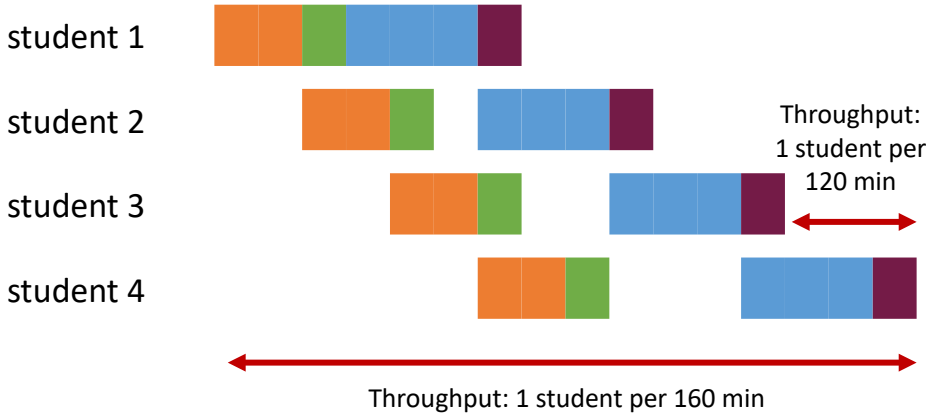
How to compute throughput fast for pipelines with no duplicated stages?

$$\frac{1}{\max(computationtime(stages))}$$

Pipelining


Time
T0 T1 T2 T3 T4 T5 T6 T7 T8 T9
Inputs
I0 S0 S1 S2 S3
I1 S0 S1 S2 S3
I2 S0 S1 S2 S3
I3 S0 S1 S2 S3
I4 S0 S1 S2 S3
I5 S0 S1 S2 S3
I6 S0 S1 S2 S3
Lead In
Full Utilization
Lead out

Last Week



Question 2: The library introduces a ”one book at a time” policy, i.e., the students have to return a book before they can start on the next one. How long will it now take for 4 students until all of them have started writing their essays?

Every student takes the exact same amount of time to read a book, concretely:

1) Reading book **A** takes 80 minutes

2) Reading book **B** takes 40 minutes

3) Reading book **C** takes 120 minutes

4) Reading book **D** takes 40 minutes

Pipelining

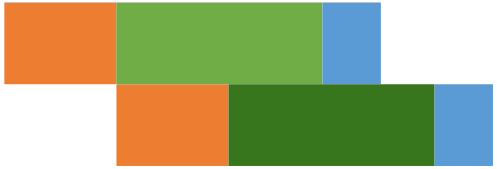
Washing - 50 min, **Dryer** - 90 min, **Iron** - 15 min

Task c) what if they bought another **dryer**?

Pipelining

Washing - 50 min, **Dryer** - 90 min, **Iron** - 15 min

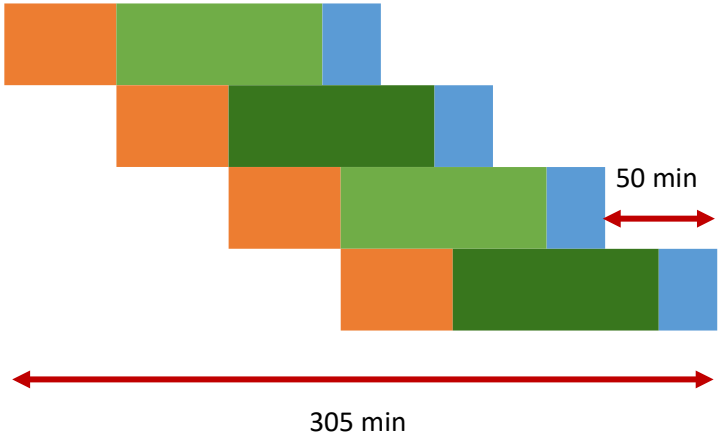
Task c) what if they bought another **dryer**?



Pipelining

Washing - 50 min, **Dryer** - 90 min, **Iron** - 15 min

Task c) what if they bought another **dryer**?



Latency?

155 min

Throughput?

1 person per 50 min
(full utilization)

1 person per 76.25 min
(with lead in & lead out)

Pipelining

Washing - 50 min, **Dryer** - 90 min, **Iron** - 15 min

Task c) what if they bought another **dryer**?

Pipeline is not balanced as the stages do not take the same time.

Pipelining II

```
for (int i = 0; i < size; i++) {
  data[i] = data[i] * data[i];
}
```

```
for (int i = 0; i < size; i += 2) {
  j = i + 1;
  data[i] = data[i] * data[i];
  data[j] = data[j] * data[j];
}
```

```
for (int i = 0; i < size; i += 4) {
  j = i + 1;
  k = i + 2;
  l = i + 3;
  data[i] = data[i] * data[i];
  data[j] = data[j] * data[j];
  data[k] = data[k] * data[k];
  data[l] = data[l] * data[l];
}
```

Pipelining II

```
for (int i = 0; i < size; i++) {
  data[i] = data[i] * data[i];
}
```

```
for (int i = 0; i < size; i += 2) {
  j = i + 1;
  data[i] = data[i] * data[i];
  data[j] = data[j] * data[j];
}
```

```
for (int i = 0; i < size; i += 4) {
  j = i + 1;
  k = i + 2;
  l = i + 3;
  data[i] = data[i] * data[i];
  data[j] = data[j] * data[j];
  data[k] = data[k] * data[k];
  data[l] = data[l] * data[l];
}
```

Annahmen:

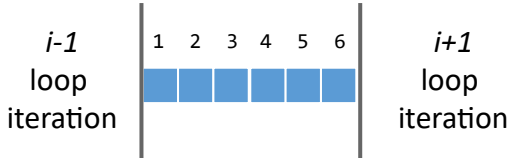
- Nur eine Instruktion kann pro CPU-Zyklus gestartet werden
- Loop-Body muss fertig sein, bevor die nächste Loop Iteration startet
- Nur arithmetische Operationen im Loop-Body werden gezählt

Wie viele Zyklen braucht der Prozessor, um fertig zu werden?

- Latenz Addition: 3 Zyklen
- Latenz Multiplikation: 6 Zyklen

Pipelining II

```
for (int i = 0; i < size; i++) {
  data[i] = data[i] * data[i];
}
```

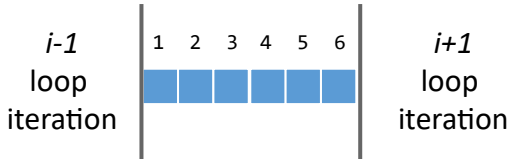



```
for (int i = 0; i < size; i += 2) {
  j = i + 1;
  data[i] = data[i] * data[i];
  data[j] = data[j] * data[j];
}
```

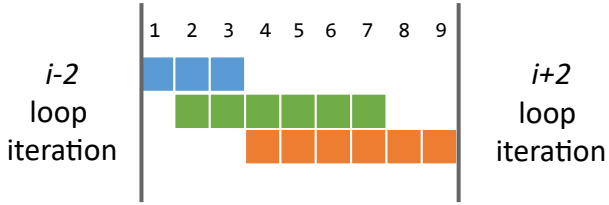
```
for (int i = 0; i < size; i += 4) {
  j = i + 1;
  k = i + 2;
  l = i + 3;
  data[i] = data[i] * data[i];
  data[j] = data[j] * data[j];
  data[k] = data[k] * data[k];
  data[l] = data[l] * data[l];
}
```

Pipelining II

```
for (int i = 0; i < size; i++) {
  data[i] = data[i] * data[i];
}
```



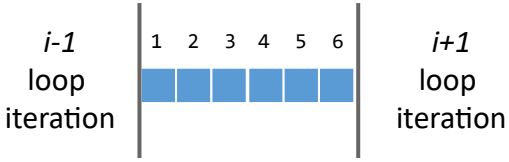

```
for (int i = 0; i < size; i += 2) {
  j = i + 1;
  data[i] = data[i] * data[i];
  data[j] = data[j] * data[j];
}
```



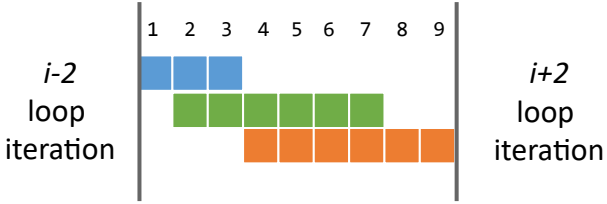

```
for (int i = 0; i < size; i += 4) {
  j = i + 1;
  k = i + 2;
  l = i + 3;
  data[i] = data[i] * data[i];
  data[j] = data[j] * data[j];
  data[k] = data[k] * data[k];
  data[l] = data[l] * data[l];
}
```

Pipelining II

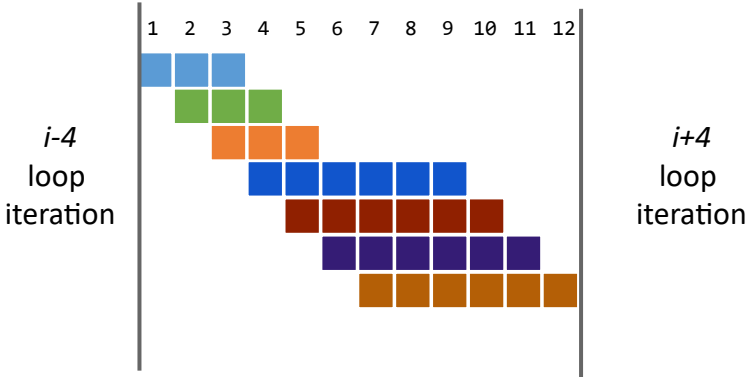
```
for (int i = 0; i < size; i++) {
  data[i] = data[i] * data[i];
}
```




```
for (int i = 0; i < size; i += 2) {
  j = i + 1;
  data[i] = data[i] * data[i];
  data[j] = data[j] * data[j];
}
```




```
for (int i = 0; i < size; i += 4) {
  j = i + 1;
  k = i + 2;
  l = i + 3;
  data[i] = data[i] * data[i];
  data[j] = data[j] * data[j];
  data[k] = data[k] * data[k];
  data[l] = data[l] * data[l];
}
```



Demo: Loop unrolling

Extra: ILP & Loop unrolling

- Eine Form von ILP
- Ziel: Mehr Instruktionen zu pipelinen
- Space-Time Tradeoff

- Andere Formen von ILP: Superscalar, Out-of-order execution, branch prediction, etc. (kommt alles in DDCA)
- Links: ILP, Loop unrolling, Space-Time Tradeoff

Loop Parallelism

Can we parallelize the following loop?

```
for (int i=1; i<size; i++) { // for loop: i from 1 to (size-1)
  if (data[i-1] > 0)         // If the previous value is positive
    data[i] = (-1)*data[i];  // change the sign of this value
}                            // end for loop
```

Loop Parallelism

Can we parallelize the following loop?

```
for (int i=1; i<size; i++) { // for loop: i from 1 to (size-1)
  if (data[i-1] > 0)         // If the previous value is positive
    data[i] = (-1)*data[i];  // change the sign of this value
}                            // end for loop
```

```
for (int i = 0; i < size; i++) {    // for loop: i from 0 to (size-1)
  data[i] = Math.sin(data[i]);      // calculate sin() of the value
}                                   // end for loop
```

Zusätzliche Aufgabe

SOLA Stafette

- 14 runners (numbered from 0 to 13)
- each runner can start after the previous runner finished (except the first one).

Under which assumption does the following code work?

Initial value of y?

0

Reference to object y?

needs to be the same object shared across all the threads

```
public class RunnerThread extends Thread {

    private AtomicInteger y;
    private int id;

    public RunnerThread(int id, AtomicInteger y) {
      this.id = id;
      this.y = y;
    }

    public void run(){
      while (y.get() != this.id) {
        // warten bis ich an der Reihe bin / wait until it is my turn
      }

      // laufen / do running
      y.incrementAndGet();
    }
}
```

SOLA Stafette

- 14 runners (numbered from 0 to 13)
- each runner can start after the previous runner finished (except the first one).

Under which assumption does the following code work?

Initial value of y?

0

Reference to object y?

needs to be the same object shared across all the threads

```
public class RunnerThread extends Thread {

    private AtomicInteger y;
    private int id;

    public RunnerThread(int id, AtomicInteger y) {
      this.id = id;
      this.y = y;
    }

    public void run(){
      while (y.get() != this.id) {
        // warten bis ich an der Reihe bin / wait until it is my turn
      }

      // laufen / do running
      y.incrementAndGet();
    }
}
```

Ensures that `y.get()` returns the updated value after calling `y.incrementAndGet();`

SOLA Stafette

Complete the implementation using wait/notify

```
public class WaitNotifyRunnerThread extends Thread {

    private ..................  x;
    private int id;

    public WaitNotifyRunnerThread(int id, ..................  x) {
        this.id = id;
        this.x = x;
    }

    public ..................  void run(){
        ..................................................
        ..................................................

        ..................................................

        ..................................................

        ..................................................

        ..................................................
```

SOLA Stafette

Complete the implementation using wait/notify

1. We'll definitely need synchronization
2. What to synchronize on?

```
public class WaitNotifyRunnerThread extends Thread {

    private ..................... x;
    private int id;

    public WaitNotifyRunnerThread(int id, ..................... x) {
        this.id = id;
        this.x = x;
    }

    public ..................... void run(){
        synchronized (?) {
        ..................................................

        ..................................................

        ..................................................

        ..................................................

        ..................................................
```

SOLA Stafette

Complete the implementation using wait/notify

1. We'll definitely need synchronization
2. What to synchronize on?
3. Check that it's our turn, wait otherwise

```
public class WaitNotifyRunnerThread extends Thread {

    private ..................... x;
    private int id;

    public WaitNotifyRunnerThread(int id, ..................... x) {
        this.id = id;
        this.x = x;
    }

    public ..................... void run(){
        synchronized (?) {
            while (condition) {
                ?.wait();
            }
        ..................................................

        ..................................................

        ..................................................

        ..................................................
```

SOLA Stafette

Complete the implementation using wait/notify

1. We'll definitely need synchronization
2. What to synchronize on?
3. Check that it's our turn, wait otherwise
4. Do work (run)

```
public class WaitNotifyRunnerThread extends Thread {

    private ..................  x;
    private int id;

    public WaitNotifyRunnerThread(int id, ..................  x) {
        this.id = id;
        this.x = x;
    }

    public ..................  void run(){
        synchronized (?) {
            while (condition) {
                 ?.wait();
            }
            // laufen / do running
        ..................................................

        ..................................................

        ..................................................
```

SOLA Stafette

Complete the implementation using wait/notify

1. We'll definitely need synchronization
2. What to synchronize on?
3. Check that it's our turn, wait otherwise
4. Do work (run)
5. Update the condition

```
public class WaitNotifyRunnerThread extends Thread {

    private ..................  x;
    private int id;

    public WaitNotifyRunnerThread(int id, ..................  x) {
        this.id = id;
        this.x = x;
    }

    public ..................  void run(){
        synchronized (?) {
            while (condition) {
                ?.wait();
            }
            // laufen / do running
            // update the condition
        ..................................................

        ..................................................

    }
```

SOLA Stafette

Complete the implementation using wait/notify

1. We'll definitely need synchronization
2. What to synchronize on?
3. Check that it's our turn, wait otherwise
4. Do work (run)
5. Update the condition
6. Notify waiting threads

This is a basic pattern that you will see in many tasks. Now, let's fill in the details.

```
public class WaitNotifyRunnerThread extends Thread {

    private ..................  x;
    private int id;

    public WaitNotifyRunnerThread(int id, ..................  x) {
        this.id = id;
        this.x = x;
    }

    public ..................  void run(){
        synchronized (?) {
            while (condition) {
                ?.wait();
            }
            // laufen / do running
            // update the condition
            ?.notify(); // or notifyAll()
        }
    }
}
```

SOLA Stafette

Complete the implementation using wait/notify

1. We'll definitely need synchronization
2. **What to synchronize on?**
3. Check that it's our turn, wait otherwise
4. Do work (run)
5. Update the condition
6. Notify waiting threads

```
public class WaitNotifyRunnerThread extends Thread {

    private ..................  x;
    private int id;

    public WaitNotifyRunnerThread(int id, ..................  x) {
        this.id = id;
        this.x = x;
    }

    public ..................  void run(){
        synchronized (?) {
            while (condition) {
                 ?.wait();
            }
            // laufen / do running
            // update the condition
            ?.notify(); // or notifyAll()
        }
    }
}
```

SOLA Stafette

Complete the implementation using wait/notify

1. We'll definitely need synchronization
2. **What to synchronize on?**
3. Check that it's our turn, wait otherwise
4. Do work (run)
5. Update the condition
6. Notify waiting threads

```
public class WaitNotifyRunnerThread extends Thread {

    private ..................  x;
    private int id;

    public WaitNotifyRunnerThread(int id, ..................  x) {
        this.id = id;
        this.x = x;
    }

    public ..................  void run(){
        synchronized (x) {
            while (condition) {
                 x.wait();
            }
            // laufen / do running
            // update the condition
           x.notify(); // or notifyAll()
        }
    }
}
```

SOLA Stafette

Complete the implementation using wait/notify

1. We'll definitely need synchronization
2. What to synchronize on?
3. **Check that it's our turn, wait otherwise**
4. Do work (run)
5. Update the condition
6. Notify waiting threads

```
public class WaitNotifyRunnerThread extends Thread {

    private ..................  x;
    private int id;

    public WaitNotifyRunnerThread(int id, ..................  x) {
        this.id = id;
        this.x = x;
    }

    public ..................  void run(){
        synchronized (x) {
            while (condition) {
                x.wait();
            }
            // laufen / do running
            // update the condition
            x.notify(); // or notifyAll()
        }
    }
}
```

```
while (y.get() != this.id) {
```

original condition

can we reuse it?

Yes!

SOLA Stafette

Complete the implementation using wait/notify

1. We'll definitely need synchronization
2. What to synchronize on?
3. **Check that it's our turn, wait otherwise**
4. Do work (run)
5. Update the condition
6. Notify waiting threads

```
public class WaitNotifyRunnerThread extends Thread {

    private AtomicInteger x;
    private int id;

    public WaitNotifyRunnerThread(int id, AtomicInteger x) {
        this.id = id;
        this.x = x;
    }

    public ................. void run(){
        synchronized (x) {
            while (x.get() != this.id) {
                x.wait();
            }
            // laufen / do running
            // update the condition
            x.notify(); // or notifyAll()
        }
    }
}
```

SOLA Stafette

Complete the implementation using wait/notify

1. We'll definitely need synchronization
2. What to synchronize on?
3. Check that it's our turn, wait otherwise
4. Do work (run)
5. **Update the condition**
6. Notify waiting threads

```
public class WaitNotifyRunnerThread extends Thread {

    private AtomicInteger x;
    private int id;

    public WaitNotifyRunnerThread(int id, AtomicInteger x) {
        this.id = id;
        this.x = x;
    }

    public ..... void run(){
        synchronized (x) {
            while (x.get() != this.id) {
                x.wait();
            }
            // laufen / do running
            x.incrementAndGet(); // update the condition
            x.notify(); // or notifyAll()
        }
    }
}
```

SOLA Stafette

Complete the implementation using wait/notify

1. We'll definitely need synchronization
2. What to synchronize on?
3. Check that it's our turn, wait otherwise
4. Do work (run)
5. Update the condition
6. **Notify waiting threads**

Should we use notify or notifyAll?

```
public class WaitNotifyRunnerThread extends Thread {

    private AtomicInteger x;
    private int id;

    public WaitNotifyRunnerThread(int id, AtomicInteger x) {
        this.id = id;
        this.x = x;
    }

    public ................. void run(){
        synchronized (x) {
            while (x.get() != this.id) {
                x.wait();
            }
            // laufen / do running
            x.incrementAndGet(); // update the condition
            x.notify(); // or notifyAll()
        }
    }
}
```

SOLA Stafette

Complete the implementation using wait/notify

1. We'll definitely need synchronization
2. What to synchronize on?
3. Check that it's our turn, wait otherwise
4. Do work (run)
5. Update the condition
6. **Notify waiting threads**

Should we use notify or notifyAll?

notifyAll because multiple threads can be waiting

```
public class WaitNotifyRunnerThread extends Thread {

    private AtomicInteger x;
    private int id;

    public WaitNotifyRunnerThread(int id, AtomicInteger x) {
        this.id = id;
        this.x = x;
    }

    public ................. void run(){
        synchronized (x) {
            while (x.get() != this.id) {
                x.wait();
            }
            // laufen / do running
            x.incrementAndGet(); // update the condition
            x.notifyAll();
        }
    }
}
```

SOLA Stafette

Complete the implementation using wait/notify

1. We'll definitely need synchronization
2. What to synchronize on?
3. Check that it's our turn, wait otherwise
4. Do work (run)
5. Update the condition
6. Notify waiting threads

```
public class WaitNotifyRunnerThread extends Thread {

    private AtomicInteger x;
    private int id;

    public WaitNotifyRunnerThread(int id, AtomicInteger x) {
        this.id = id;
        this.x = x;
    }

    public ..................  void run(){
        synchronized (x) {
            while (x.get() != this.id) {
                x.wait();
            }
            // laufen / do running
            x.incrementAndGet(); // update the condition
            x.notifyAll();
        }
    }
}
```

Can we replace

```
synchronized (x)
```

with synchronizing the method?

No

SOLA Stafette

Complete the implementation using wait/notify

1. We'll definitely need synchronization
2. What to synchronize on?
3. Check that it's our turn, wait otherwise
4. Do work (run)
5. Update the condition
6. Notify waiting threads

```
public class WaitNotifyRunnerThread extends Thread {

    private AtomicInteger x;
    private int id;

    public WaitNotifyRunnerThread(int id, AtomicInteger x) {
        this.id = id;
        this.x = x;
    }

    public void run(){
        synchronized (x) {
            while (x.get() != this.id) {
                x.wait();
            }
            // laufen / do running
            x.incrementAndGet(); // update the condition
            x.notifyAll();
        }
    }
}
```

Can we replace

`AtomicInteger x`

With

`Integer x`

?

SOLA Stafette

Complete the implementation using wait/notify

1. We'll definitely need synchronization
2. What to synchronize on?
3. Check that it's our turn, wait otherwise
4. Do work (run)
5. Update the condition
6. Notify waiting threads

```
public class WaitNotifyRunnerThread extends Thread {

    private Integer x;
    private int id;

    public WaitNotifyRunnerThread(int id, Integer x) {
        this.id = id;
        this.x = x;
    }

    public void run(){
        synchronized (x) {
            while (x.get() != this.id) {
                x.wait();
            }
            // laufen / do running
            x += 1; // update the condition
            x.notifyAll();
        }
    }
}
```

Can we replace

`AtomicInteger x`

With

`Integer x`

?

SOLA Stafette

Complete the implementation using wait/notify

Do these assumptions still hold?

Initial value of x?

0

Reference to object x?

needs to be the same object shared across all the threads

No, because x += 1 in Java creates a new Object!

```
public class WaitNotifyRunnerThread extends Thread {

    private Integer x;
    private int id;

    public WaitNotifyRunnerThread(int id, Integer x) {
        this.id = id;
        this.x = x;
    }

    public void run(){
        synchronized (x) {
            while (x.get() != this.id) {
                x.wait();
            }
            // laufen / do running
            x += 1; // update the condition
            x.notifyAll();
        }
    }
}
```

Can we replace

`AtomicInteger x`

With

`Integer x`

?

Theorie Recap: Divide & Conquer

Divide and Conquer

Beispiele für Divide-and-conquer Algorithmen: Quicksort, Mergesort, Strassen matrix multiplication, und viele weitere.

Aufbau eines Divide-and-conquer Algorithmus:

1. Falls das Problem klein genug ist -> Direkt lösen
2. Sonst
   a. Problem in Teilprobleme zerlegen
   b. Teilprobleme rekursiv lösen
   c. Lösung der Teilprobleme zusammenführen

Adding Numbers from Vector: (Recursive Version)

```
public static int do_sum_rec(int[] xs, int l, int h) {
        int size = h – l;
        if (size == 1)
                return xs[l];

        int mid  = size / 2;
        int sum1 = do_sum_rec(xs, l, l + mid);
        int sum2 = do_sum_rec(xs, l + mid, h);

        return sum1 + sum2;
}
```

Parallel Version: Task Parallelism Model

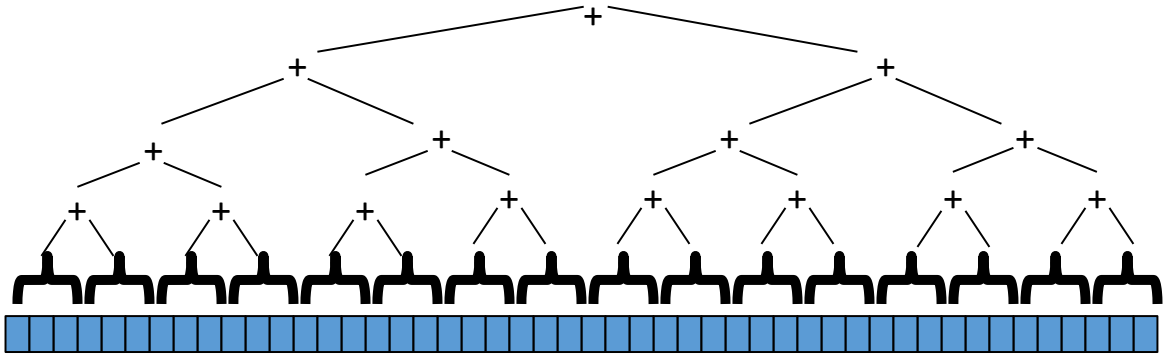
Normaler Kontrollfluss

- Erstelle parallele Tasks
- Warte bis die parallelen Tasks fertig sind

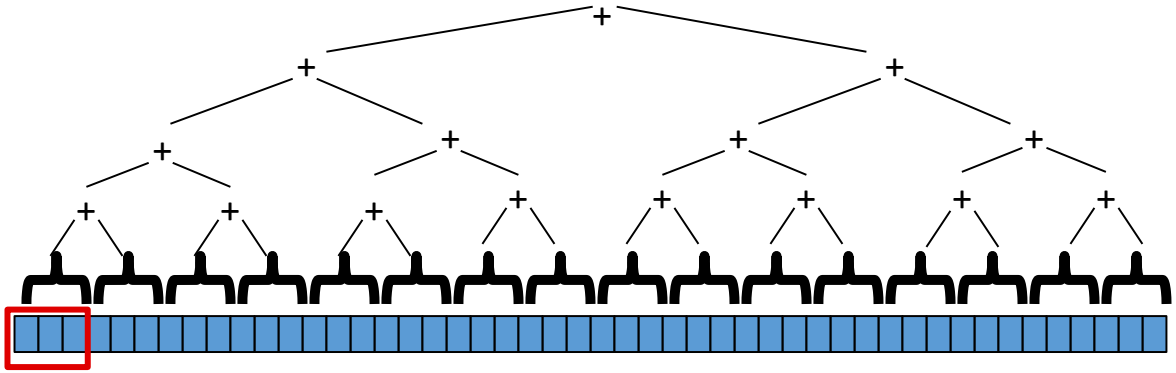
Parallele Version eines Divide-and-conquer Problems:

- Erstelle einen parallelen Task für jedes Teilproblem
- Warte bis die Teilprobleme gelöst sind
- Resultate zusammenführen

Divide and Conquer



Divide and Conquer

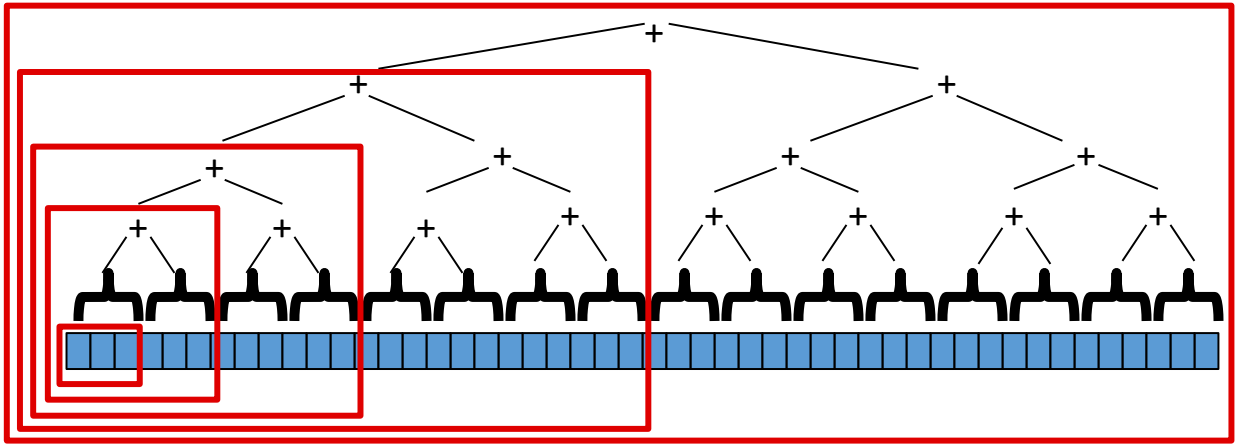


base case

no further split

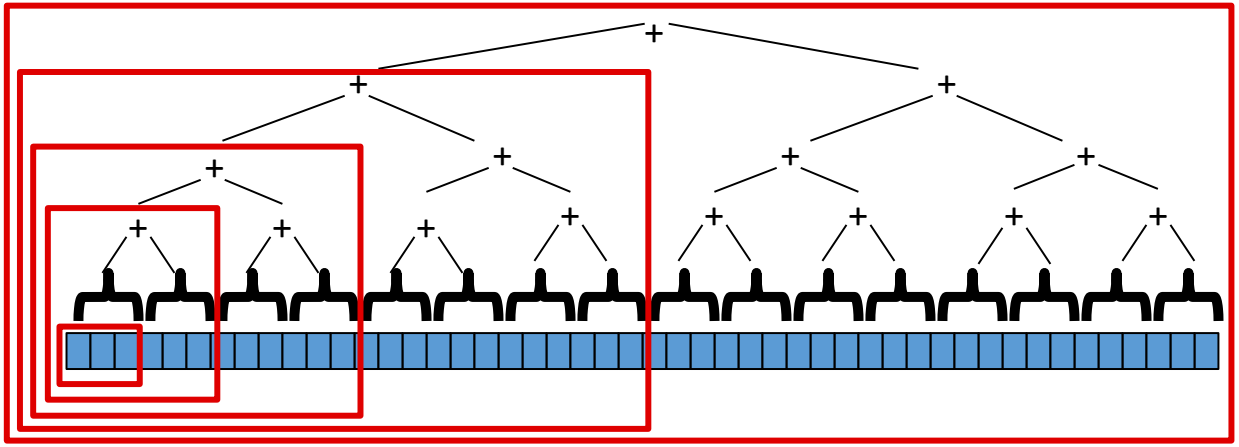
Divide and Conquer

Tasks mit unterschiedlicher Granularität



Divide and Conquer

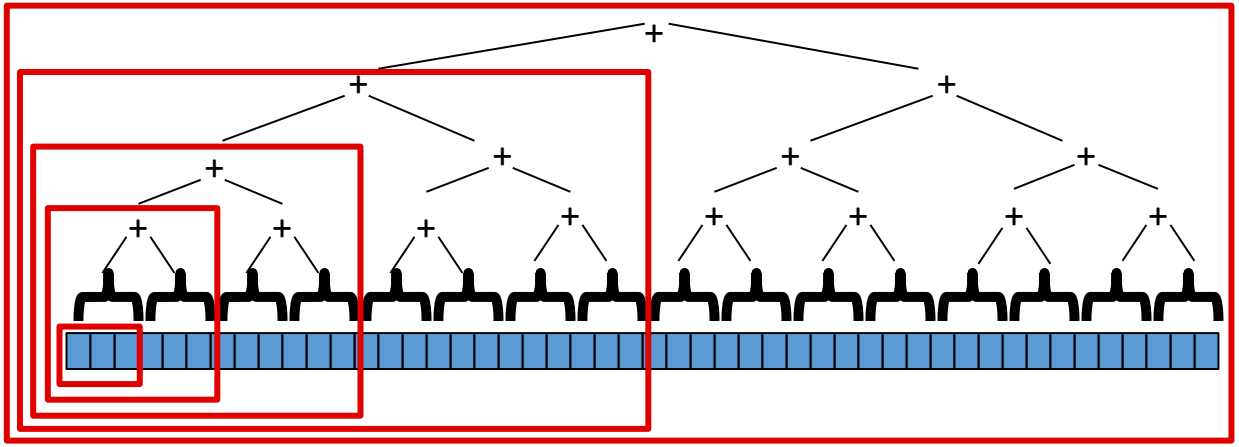
Tasks mit unterschiedlicher Granularität



Was definiert einen Task?

Divide and Conquer

Tasks mit unterschiedlicher Granularität



Was definiert einen Task?

i) Input array ii) Startindex iii)Länge/Endindex

Das sind die Attribute, die wir pro Task speichern

Vorbesprechung Übung 5

Assignment 5

1. *Parallel Search and Count*

   Search an array of integers for a certain feature and count integers that have this feature.

   - Light workload: count number of non-zero values.
   - Heavy workload: count how many integers are prime numbers.

   We will study single threaded and multi-threaded implementation of the problem.

   -> Code durchgehen

Assignment 5

1. *Parallel Search and Count*

   Search an array of integers for a certain feature and count integers that have this feature.

   - Light workload: count number of non-zero values.
   - Heavy workload: count how many integers are prime numbers.

   We will study single threaded and multi-threaded implementation of the problem.

3. *Amdahl's and Gustafson's Law II*
4. *Amdahl's and Gustafson's Law*
5. *Task Graph*

Amdahl’s Law

- Menge an Arbeit fix, d.h.
- wird durch mehr Threads beschleunigt.
- Speedup bezieht sich auf die Zeit
- D.h. (Wegen Overheads ist es und nicht)

- Beschränkter Speedup!

Gustafson’s Law

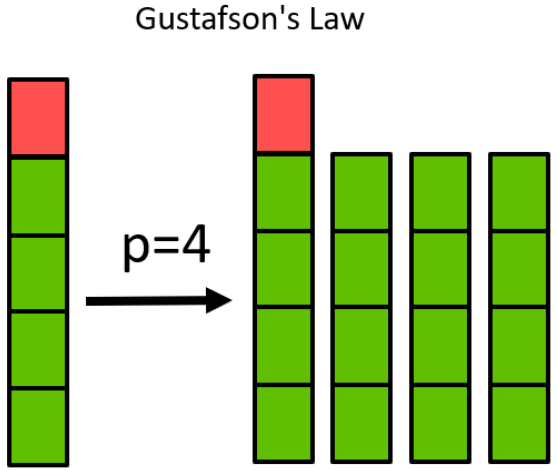
- Gegebene Zeit fix
- Mit mehr Threads kann mehr Arbeit erledigt werden.
- Speedup bezieht sich auf die Arbeit

Amdal's & Gustavson's Law

Assuming a program consists of 50% non-parallelizable code.

a) Compute the speed-up when using 2 and 4 processors according to Amdahl's law.

b) Now assume that the parallel work per processor is fixed. Compute the speed-up when using 2 and 4 processors according to Gustafson's law.



Time is constant

→ More work overall

Old Exam Task (HS20 – Task 1)

1. Nehmen Sie an ein Programm besteht zu 20% aus nicht-parallelisierbarer Arbeit. Wir wollen einen Speedup von 4 gemäss Gustafson's Gesetz erlangen. Wie viele Prozessoren sind notwendig? Geben Sie alle Rechenschritte an.

Assume a program consisting of 20% non-parallelizable work. We want to achieve a speed up of 4 according to Gustafson's law. How many processors are necessary? Show all calculation steps. (5)

https://quizizz.com/admin/quiz/65f7e8ede558fa3845d65bf3?source=quiz_share

Replace link with link to quiz