Parallel Programming Exercise Session 5

Spring 2025

Schedule

Post-Discussion Ex. 4

Theory and Exercises

Pre-Discussion Ex. 5

Quiz

Exam preparation (old exam tasks)

Post-Discussion
Exercise 4

Task 1: Pipelining

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

a) total time if strictly sequential order?

Task 1: Pipelining

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

a) total time if strictly sequential order?

50 + 90 + 15

155

Task 1: Pipelining

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

a) total time if strictly sequential order?

155 + 155

Task 1: Pipelining

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

a) total time if strictly sequential order?

155 + 155 + 155 + 155 = 620

Task 1: Pipelining

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

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

Task 1: Pipelining

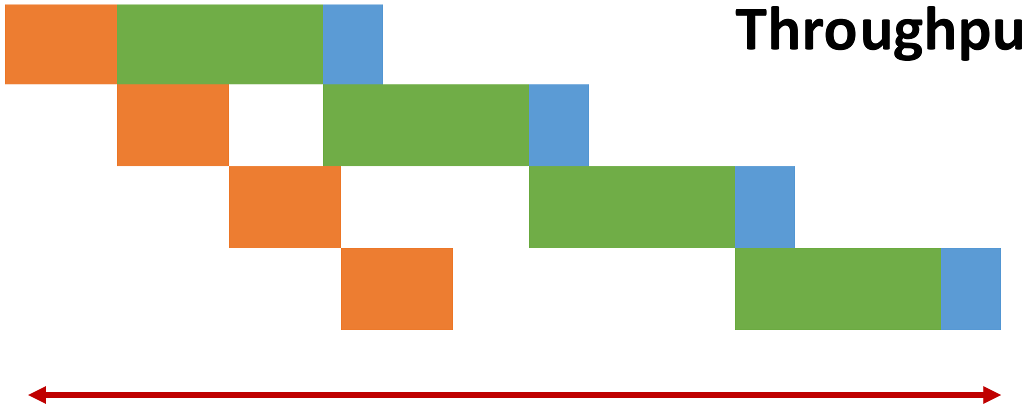
Washing - 50 min, **Dryer** - 90 min, **Iron** - 15 min

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

Task 1: Pipelining

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

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



Throughput?

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

1 person per 106.25 min (425 min / 4)

Task 1: Pipelining

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

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

Throughput?

1 person per 90 min

Full utilization
(indefinite number of students)

Task 1: Pipelining

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

b) what is the achieved speedup?

Total time (sequential): 620 min

Total time (optimized): 425 min

Speedup: $S = \frac{T_a}{T_b} = \frac{620}{425} \approx 1.46$

Pipelining

Time

Inputs	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9
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

Task 1: Pipelining

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

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

Task 1: Pipelining

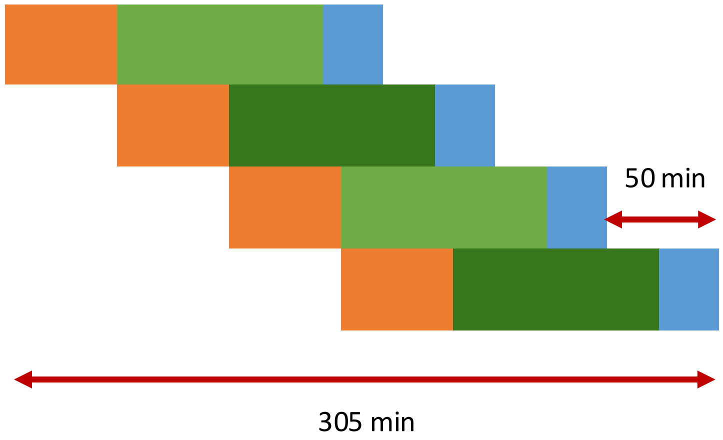
Washing - 50 min, **Dryer** - 90 min, **Iron** - 15 min

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

Task 1: Pipelining

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

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)

Task 1: Pipelining

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

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

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

Task 2: 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];
}
```

Task 2: 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];
}
```

Assumptions:

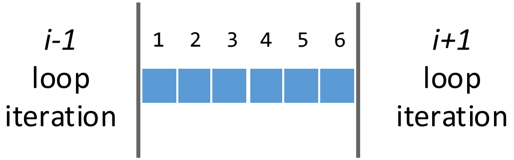
- Only one instruction can be issued per cycle.
- Loop body computation must be fully finished before next iteration starts
- Consider only arithmetic expressions in loop body and ignore all other operations (i.e., loads, stores, loop counter increment, etc.)
- Consider only *execute* step of an instruction (e.g., ignore fetch, decode, etc.)

How many cycles does the processor need to execute the following loops?

- Addition takes 3 cycles to execute
- Multiplication takes 6 cycles to execute

Task 2: Pipelining II

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

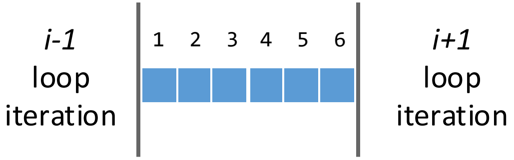



```
for (int i = 0; i < size; i += 2) {
  j = i + 1;
  data[i] = data[i] * data[i];
  data[j] = data[j] * data[j];
}
```

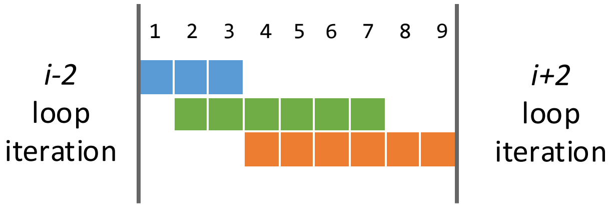
```
for (int i = 0; i < size; i += 4) {
  j = i + 1;
  k = i + 2;
  l = i + 3;
  data[i] = data[i] * data[i];
  data[j] = data[j] * data[j];
  data[k] = data[k] * data[k];
  data[l] = data[l] * data[l];
}
```

Task 2: Pipelining II

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



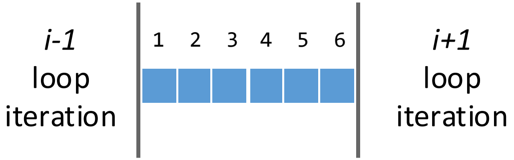

```
for (int i = 0; i < size; i += 2) {
  j = i + 1;
  data[i] = data[i] * data[i];
  data[j] = data[j] * data[j];
}
```



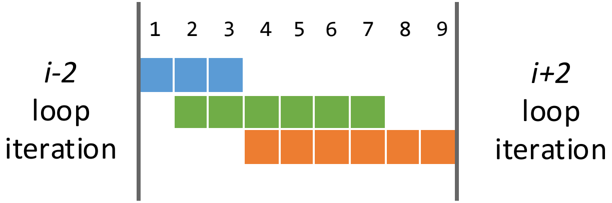

```
for (int i = 0; i < size; i += 4) {
  j = i + 1;
  k = i + 2;
  l = i + 3;
  data[i] = data[i] * data[i];
  data[j] = data[j] * data[j];
  data[k] = data[k] * data[k];
  data[l] = data[l] * data[l];
}
```

Task 2: Pipelining II

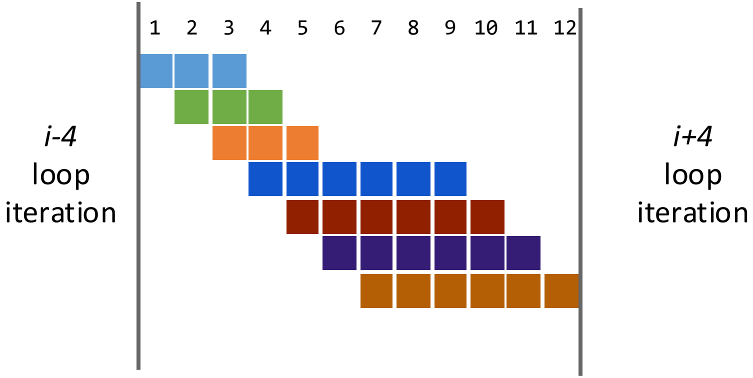
```
for (int i = 0; i < size; i++) {
  data[i] = data[i] * data[i];
}
```




```
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];
}
```



Task 3: Loop Parallelism

Can we parallelize the following loops and still obtain the correct result?

```
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
```

Task 3: Loop Parallelism

Can we parallelize the following loops and still obtain the correct result?

```
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
```

Theory and Exercises

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

```java
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

```java
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

```java
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

```java
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!

```java
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`

?

First attempt, part 1

```
class SumThread extends java.lang.Thread {

  int lo; // arguments
  int hi;
  int[] arr;

  int ans = 0; // result

  SumThread(int[] a, int l, int h) {
    lo=l; hi=h; arr=a;
  }


  public void run() { //override must have this type
    for(int i=lo; i < hi; i++)
      ans += arr[i];
  }
}
```

Because we must override a no-arguments/no-result `run`, we use fields to communicate across threads

First attempt, continued (wrong)

```
class SumThread extends java.lang.Thread {
  int lo, int hi, int[] arr; // arguments
  int ans = 0; // result
  SumThread(int[] a, int l, int h) { … }
  public void run(){ … } // override
}


int sum(int[] arr){ // can be a static method
  int len = arr.length;
  int ans = 0;
  SumThread[] ts = new SumThread[4];
  for(int i=0; i < 4; i++) // do parallel computations
    ts[i] = new SumThread(arr,i*len/4,(i+1)*len/4);

  for(int i=0; i < 4; i++) // combine results
    ans += ts[i].ans;
  return ans;
}
```

Second attempt (still wrong)

```
class SumThread extends java.lang.Thread {
  int lo, int hi, int[] arr; // arguments
  int ans = 0; // result
  SumThread(int[] a, int l, int h) { … }
  public void run(){ … } // override
}


int sum(int[] arr){// can be a static method
  int len = arr.length;
  int ans = 0;
  SumThread[] ts = new SumThread[4];
  for(int i=0; i < 4; i++){// do parallel computations
    ts[i] = new SumThread(arr,i*len/4,(i+1)*len/4);
    ts[i].start(); // start actually runs the thread in parallel
  }
  for(int i=0; i < 4; i++) // combine results
    ans += ts[i].ans;
  return ans;
}
```

Third attempt (correct in spirit)

```
class SumThread extends java.lang.Thread {
  int lo, int hi, int[] arr; // arguments
  int ans = 0; // result
  SumThread(int[] a, int l, int h) { … }
  public void run(){ … } // override
}


int sum(int[] arr){// can be a static method
  int len = arr.length;
  int ans = 0;
  SumThread[] ts = new SumThread[4];
  for(int i=0; i < 4; i++){// do parallel computations
    ts[i] = new SumThread(arr,i*len/4,(i+1)*len/4);
    ts[i].start();
  }
  for(int i=0; i < 4; i++) { // combine results
    ts[i].join(); // wait for helper to finish!
    ans += ts[i].ans;
  }
  return ans;
}
```

Issues with this approach (and some workarounds)

Several reasons why this is a poor parallel algorithm

Reason 1: want code to be reusable and efficient across platforms

```
int sum(int[] arr){// can be a static method
  int len = arr.length;
  int ans = 0;
  SumThread[] ts = new SumThread[4];
  for(int i=0; i < 4; i++){// do parallel computations
    ts[i] = new SumThread(arr,i*len/4,(i+1)*len/4);
    ts[i].start();
  }
  for(int i=0; i < 4; i++) { // combine results
    ts[i].join(); // wait for helper to finish
    ans += ts[i].ans;
  }
  return ans;
}
```

Issues with this approach (and some workarounds)

Reason 2: want to use (only) processors "available to you *now*"

- Not used by other programs or threads in your program
  - Maybe caller is also using parallelism
  - Available cores can change even while your threads run

Reason 3: Though unlikely for `sum`, in general subproblems may take significantly different amounts of time

Divide and Conquer

Examples of divide-and-conquer algorithms are quicksort, mergesort, Strassen matrix multiplication, and many others.

Basic structure of a divide-and-conquer algorithm:

1. If problem is small enough, solve it directly
2. Otherwise
   a. Break problem into subproblems
   b. Solve subproblems recursively
   c. Assemble solutions of subproblems into overall solution

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

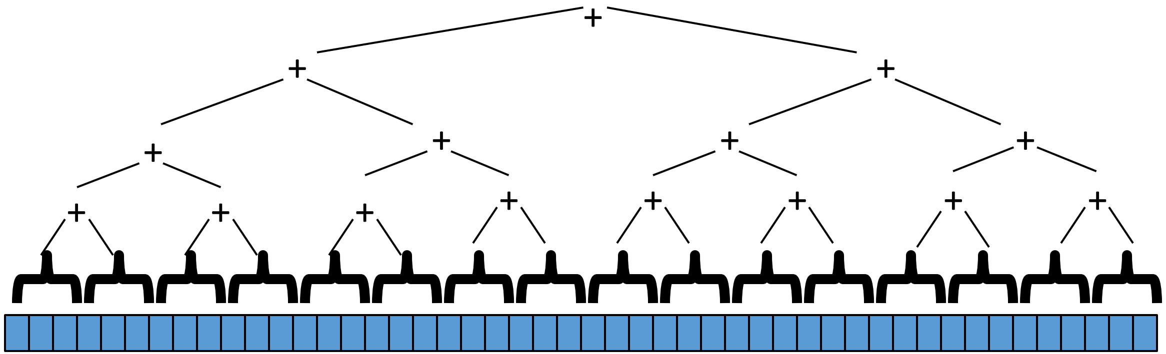
Basic flow of operations

- Create parallel tasks
- Wait for parallel tasks to complete

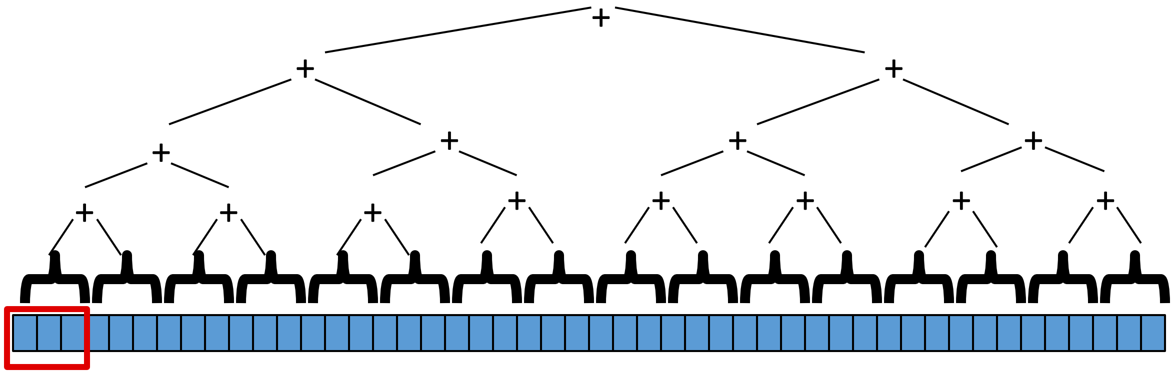
Parallel versions of divide-and-conquer on the task model are trivial:

- Create a task for the first and second part
- Wait for tasks to complete
- Combine their results

Divide and Conquer



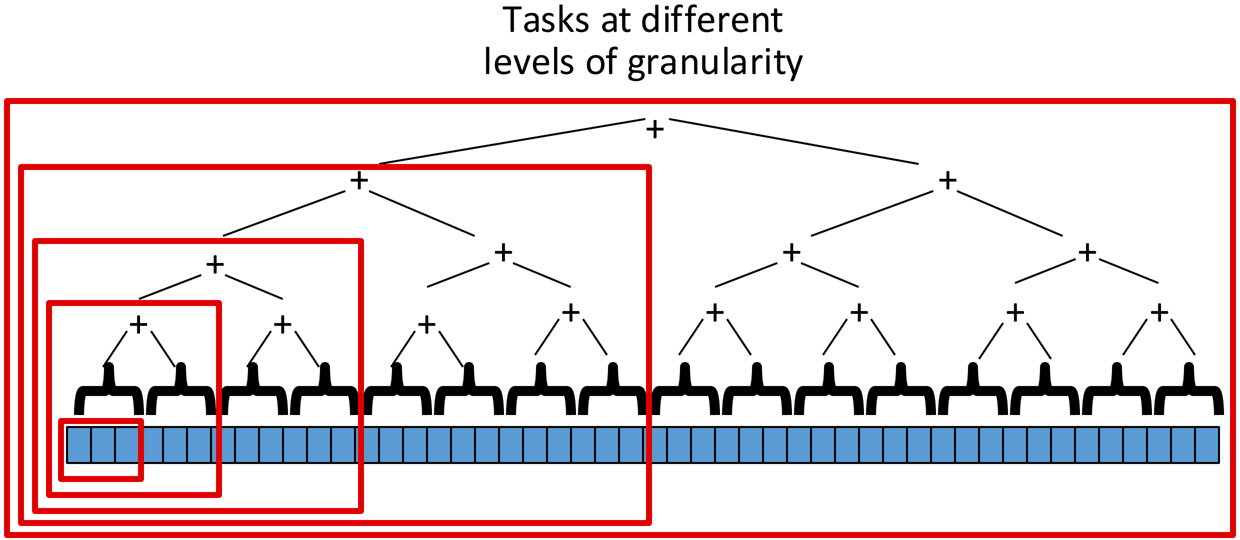
Divide and Conquer



base case

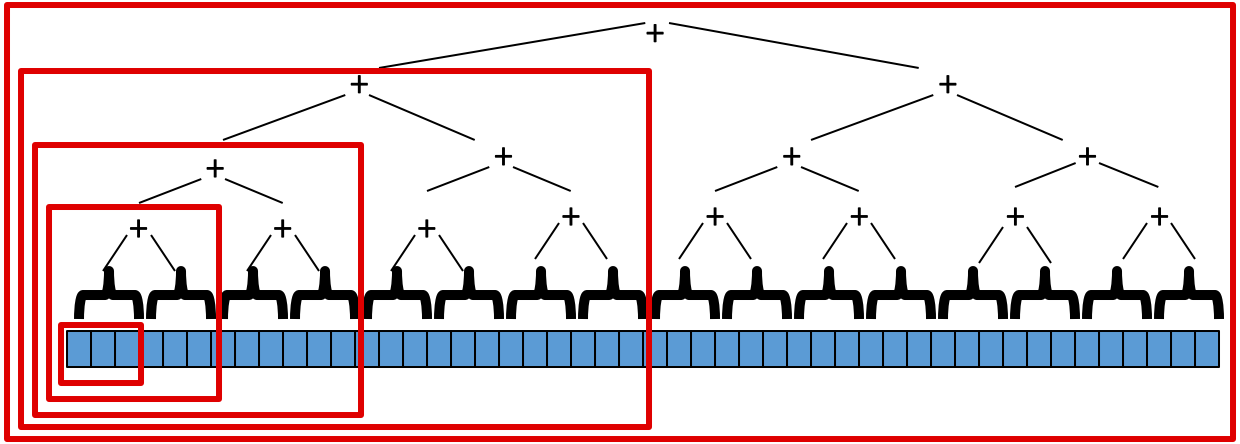
no further split

Divide and Conquer



Divide and Conquer

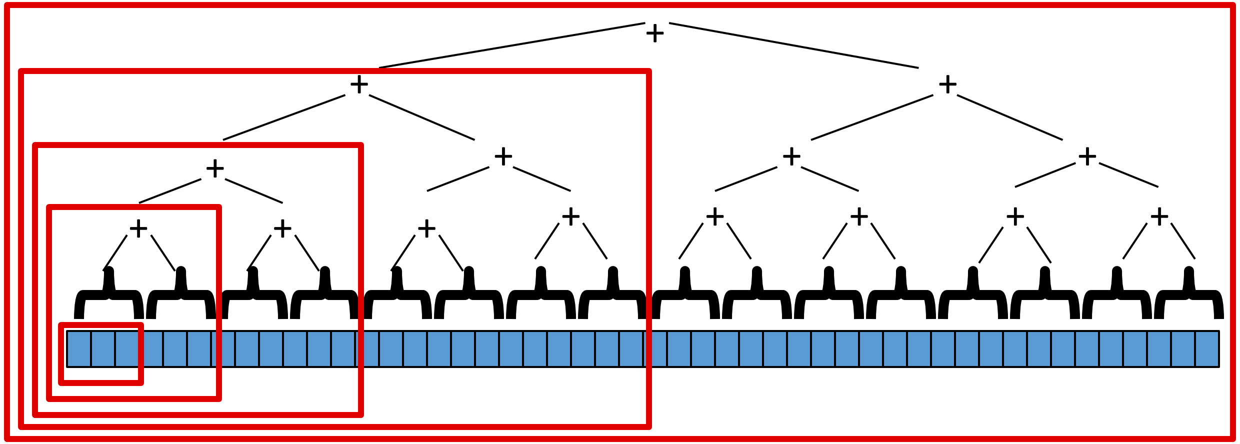
Tasks at different
levels of granularity



What determines a task?

Divide and Conquer

Tasks at different
levels of granularity



What determines a task?

i) input array

ii) start index
index

iii) length/end

These are fields we want to store in the task

Parallel Recursive Sum (with Threads)

```
public void run(){
    int size = h-l;
    if (size == 1) {
           result = xs[l];
           return;
    }
    int mid = size / 2;
    SumThread t1 = new SumThread(xs, l, l + mid);
    SumThread t2 = new SumThread(xs, l + mid, h);

    t1.start();
    t1.join();

    t2.start();
    t2.join();

    result = t1.result + t2.result;
    return;
}
```

Is this OK?

Parallel Recursive Sum (with Threads)

```
public void run(){
    int size = h-l;
    if (size == 1) {
        result = xs[l];
        return;
    }
    int mid = size / 2;
    SumThread t1 = new SumThread(xs, l, l + mid);
    SumThread t2 = new SumThread(xs, l + mid, h);

    t1.start();
    t2.start();

    t1.join();
    t2.join();

    result = t1.result + t2.result;
    return;
}
```

Remark: This doesn't compile because join() can throw exceptions. In reality we need a try-catch block here.

Result

Java.lang.OutOfMemoryError: unable to create new native thread

Divide-and-conquer – with manual fixes (Pt. I)

```
public void run(){
        int size = h-l;
        if (size < SEQ_CUTOFF)
                for (int i=l; i<h; i++)
                        result += xs[i];
        else {
           int mid = size / 2;
           SumThread t1 = new SumThread(xs, l, l + mid);
           SumThread t2 = new SumThread(xs, l + mid, h);
           t1.start();
           t2.start();
           t1.join();
           t2.join();
           result = t1.result + t2.result;
        }
}
```

Half the threads

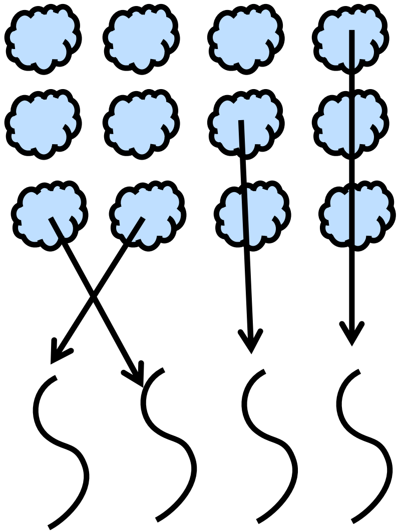
```
// wasteful: don't
SumThread t1 = …
SumThread t2 = …
t1.start();
t2.start();
t1.join();
t2.join();
result=t1.result+t2.result;
```

```
// better: do
// order of next 4 lines
// essential - why?
t1.start();
t2.run();
t1.join();
result=t1.result+t2.result;
```

Java Threads are too heavyweight. What should we do?

ExecutorService!

Alternative approach: schedule tasks on threads



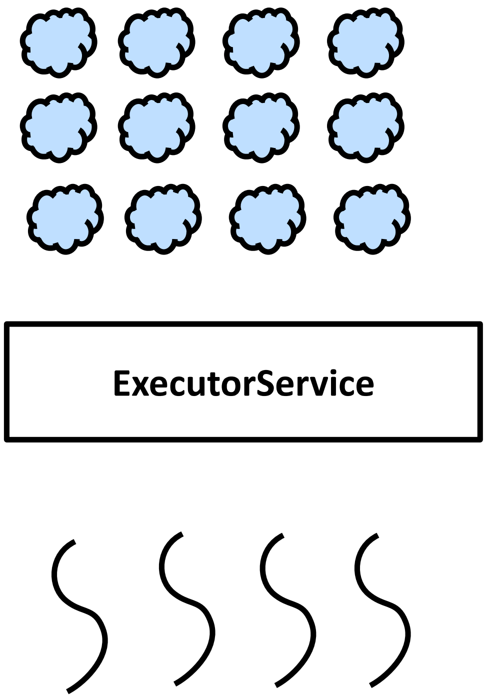
Tasks

(Thread pool)

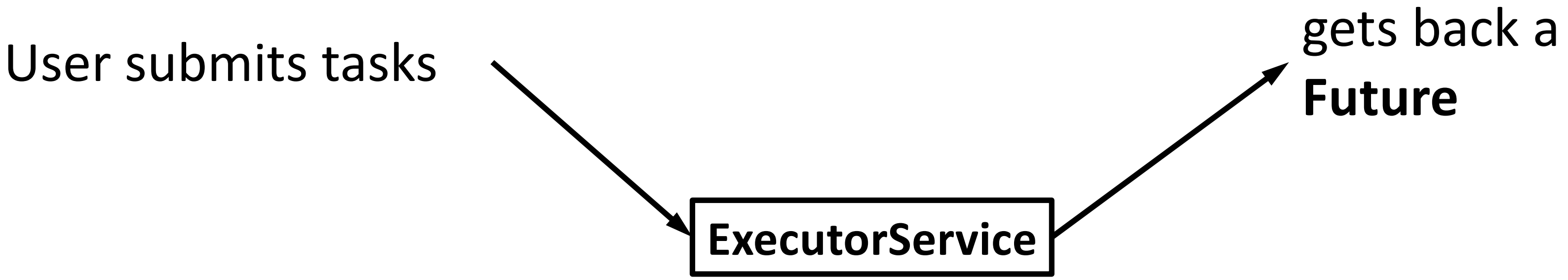
Threads

How many threads would you use?

Java's executor service: managing asynchronous tasks



Java's executor service:managing asynchronous tasks



.submit(Callable<T> task) → Future<T>
.submit(Runnable task) → Future<?>

Recursive Sum with ExecutorService

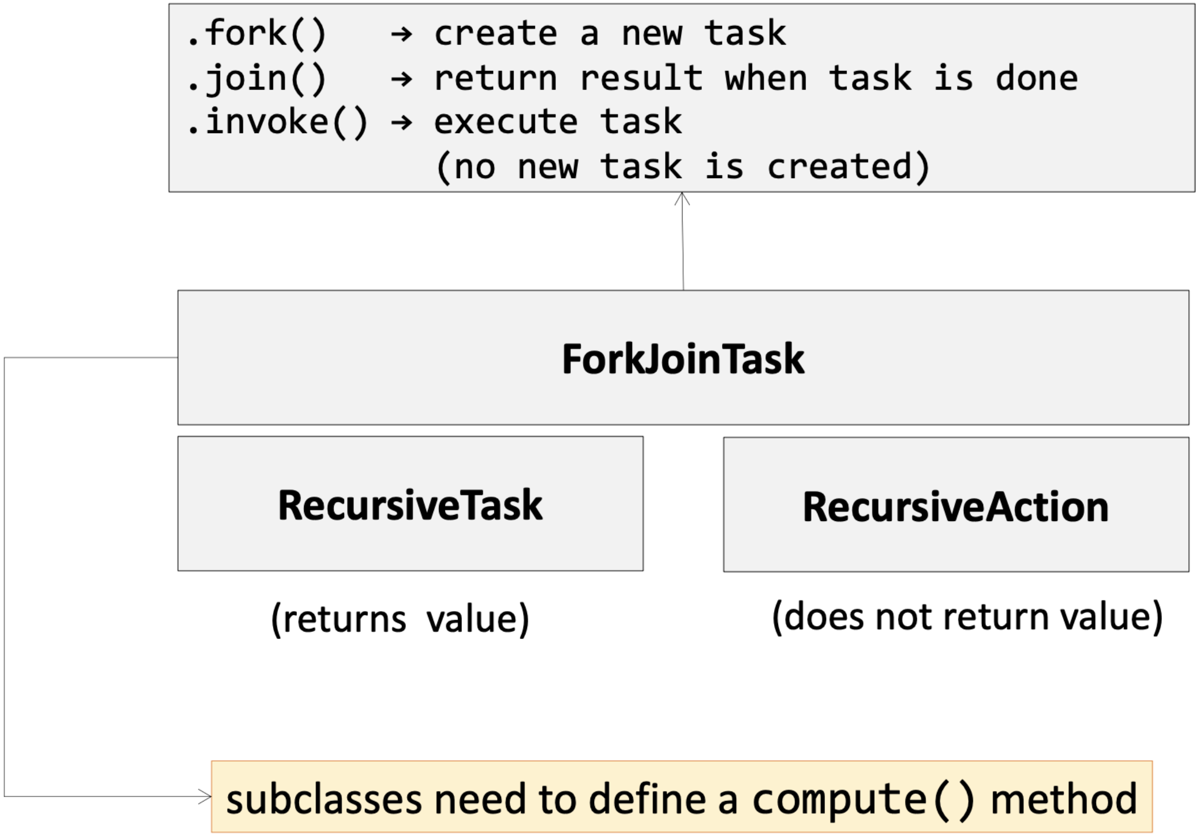
```java
public Integer call() throws Exception {
  int size = h – l;
  if (size == 1)
    return xs[l];

  int mid = size / 2;
  sumRecCall c1 = new sumRecCall(ex, xs, l, l + mid);
  sumRecCall c2 = new sumRecCall(ex, xs, l + mid, h);

  Future<Integer> f1 = ex.submit(c1);
  Future<Integer> f2 = ex.submit(c2);

  return f1.get() + f2.get();
}
```

Lesson:

ExecutorService is good for “flat” problems.

But for recursive problems (like divide-and-conquer) we need a more complex system: Work stealing

Tasks in Fork/Join Framework



Different terms, same basic idea

To use the ForkJoin Framework:

A little standard set-up code (e.g., create a **`ForkJoinPool`**)

Don't subclass **`Thread`**

Don't override **`run`**

Do not use an **`ans`** field

Don't call **`start`**

Don't just call **`join`**

Don't call **`run`** to hand-optimize

Don't have a topmost call to **`run`**

Do subclass **`RecursiveTask<V>`**

Do override **`compute`**

Do return a **`V`** from **`compute`**

Do call **`fork`**

Do call **`join`** which returns answer

Do call **`compute`** to hand-optimize

Do create a pool and call **`invoke`**

Recursive sum with ForkJoin (compute)

```
protected Long compute() {
        if(high - low <= 1)
            return array[high];
        else {
            int mid = low + (high - low) / 2;
            SumForkJoin left  = new SumForkJoin(array, low, mid);
            SumForkJoin right = new SumForkJoin(array, mid, high);
            left.fork();
            right.fork();
            return left.join() + right.join();
        }
    }
```

Fixes/Work-Around – cont’d

```
protected Long compute() {
        if(high - low <= SEQUENTIAL_THRESHOLD) {
            long sum = 0;
            for(int i=low; i < high; ++i)
                sum += array[i];
            return sum;
        } else {
            int mid = low + (high - low) / 2;
            SumForkJoin left  = new SumForkJoin(array, low, mid);
            SumForkJoin right = new SumForkJoin(array, mid, high);
            left.fork();
            long rightAns = right.compute();
            long leftAns  = left.join();
            return leftAns + rightAns;
        }
    }
```

Make sure each task has ‘enough’ to do!

Pre-Discussion
Exercise 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.

2. *Amdahl's and Gustafson's Law*

3. *Amdahl's and Gustafson's Law II*

4. *Task Graph*

Amdahl's & Gustafson's Law

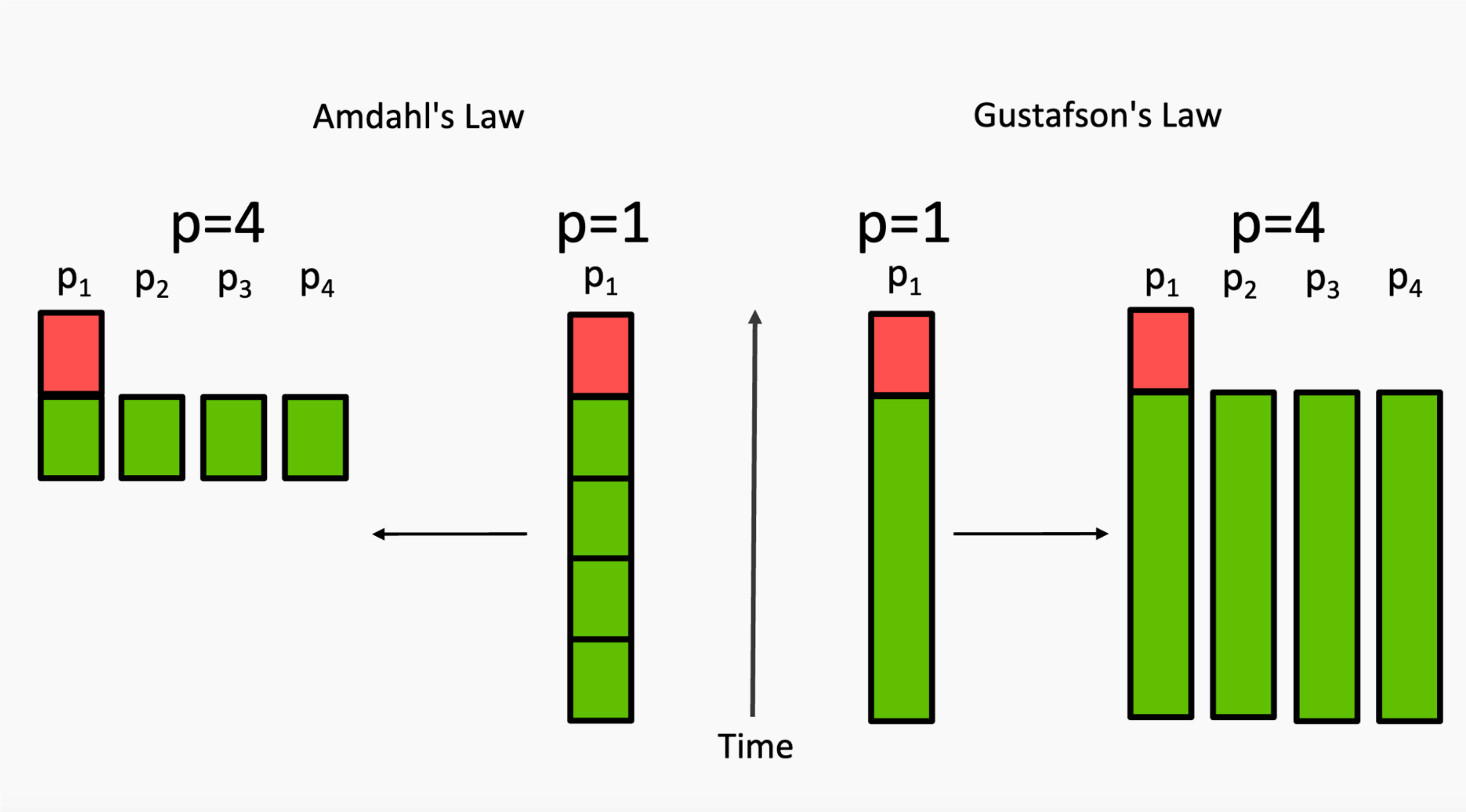
Let f be the non-parralelizable serial fractions of the total work and p be the number of workers.

Amdahl's Law

$$S_p \leq \frac{1}{f + \frac{1-f}{p}}$$

Gustafson's Law

$$S_p = p \; - f(p-1)$$

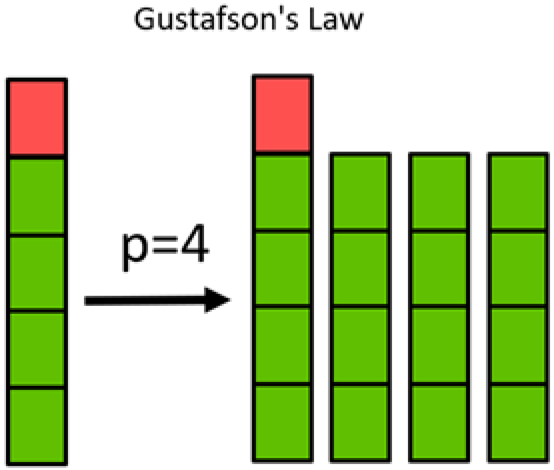

Amdahl's Law
Gustafson's Law
p=4
p=1
p=1
p=4
p_1 p_2 p_3 p_4
p_1
p_1
p_1 p_2 p_3 p_4
Time

Amdahl’s & Gustafson’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.



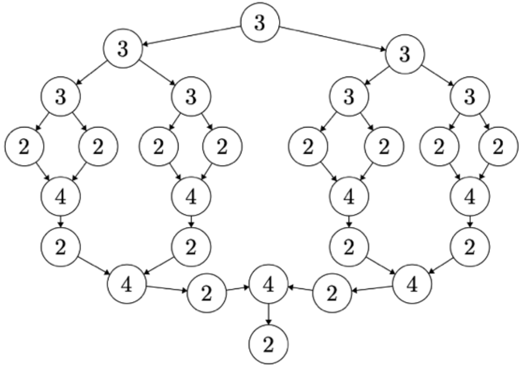
Time is constant

→ More work overall


QUIZ

Exam preparation (old exam tasks)

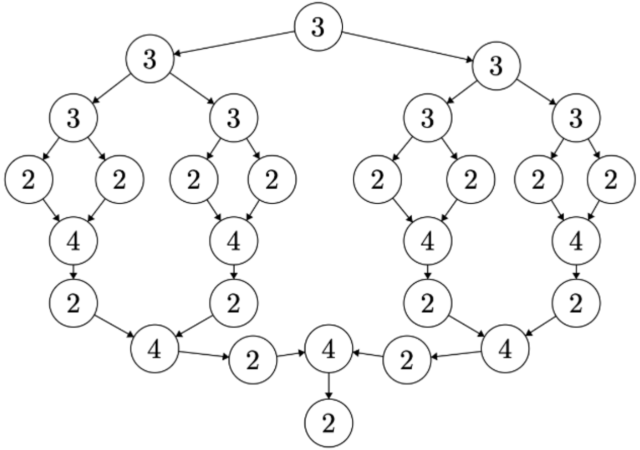
Task Graph



- Way to model program execution
- DAG: directed acyclic graph
- Nodes are tasks / work laballed with execution time
- Edges show dependencies → data dependency along the edge

Task Graph

- Used to calculate speed up
- T_1 = sum of all nodes
- T_p = sum of all nodes on critical (= longest) path

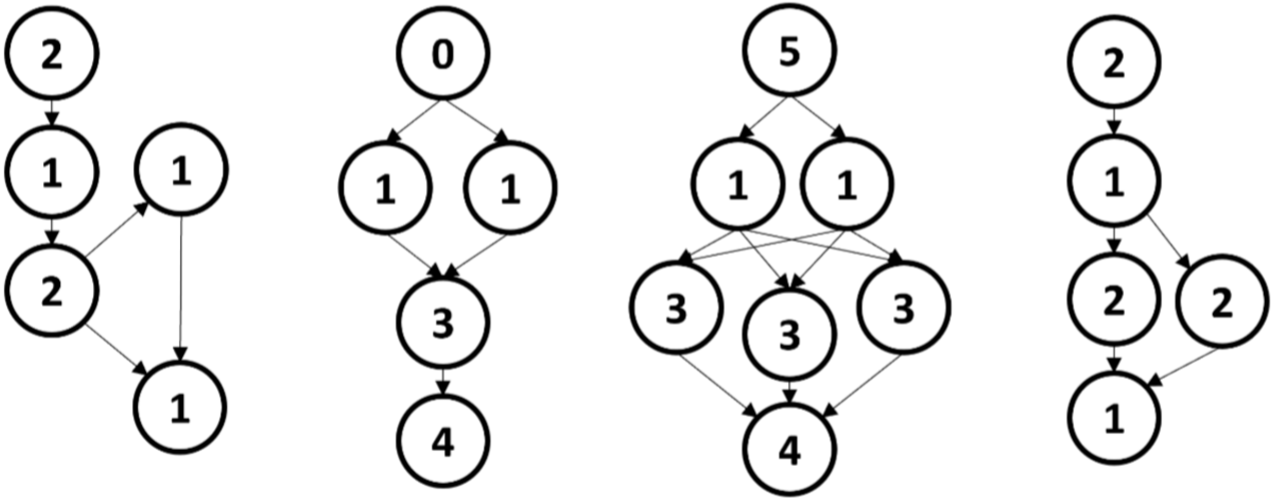


(e) Kreisen Sie in den folgenden Fragen die richtige Antwort ein.

In the questions below, circle the correct answer.

i. Welcher der folgenden Task Graphen ist **kein** gültiger Task Graph?

*Which of the following task graphs is **not** a valid task graph?* (1)



A B C D

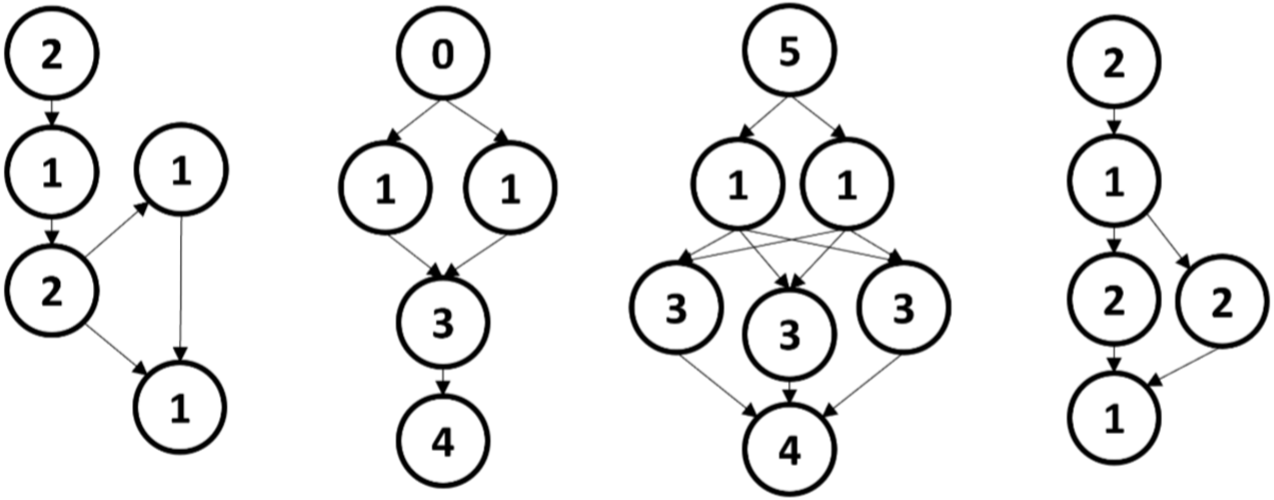
(A) (B) (C) (D) (E) None of the above

(e) Kreisen Sie in den folgenden Fragen die richtige Antwort ein.

In the questions below, circle the correct answer.

i. Welcher der folgenden Task Graphen ist **kein** gültiger Task Graph?

Which of the following task graphs is ***not*** *a valid task graph?* (1)

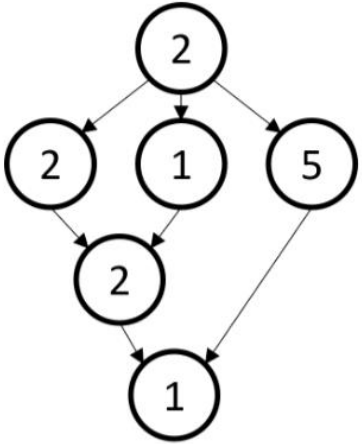


A B C D

(A) (B) (C) (D) (E) None of the above

ii. Was ist die Länge des kritischen Pfades des folgenden Task Graphen?

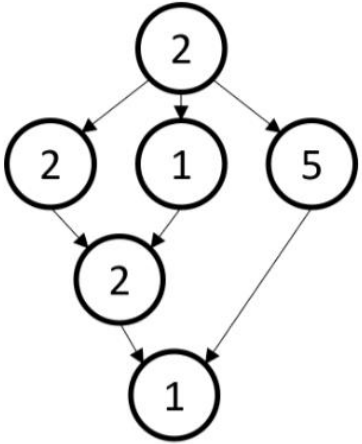
What is the length of the critical path of the following task graph?



(A) 3 (B) 6 (C) 8 (D) 10 (E) 13

ii. Was ist die Länge des kritischen Pfades des folgenden Task Graphen?

What is the length of the critical path of the following task graph?



(A) 3 (B) 6 $\boxed{\text{(C) 8}}$ (D) 10 (E) 13

iii. Welcher Ausdruck gibt eine Untergrenze für T_p an, die Zeit, die zum Ausführen eines Task Graphen auf p Prozessoren benötigt wird?

Which expression gives an lower bound on T_p, the time it takes to execute a task graph on p processors?

(A) $\frac{T_1}{p}$ (B) $T_1 \cdot p$ (C) $\frac{T_1 \cdot (p-1)}{p}$ (D) $\frac{p-1}{T_1}$ (E) $2 \cdot T_1$

iii. Welcher Ausdruck gibt eine Untergrenze für T_p an, die Zeit, die zum Ausführen eines Task Graphen auf p Prozessoren benötigt wird?

Which expression gives an lower bound on T_p, the time it takes to execute a task graph on p processors?

(A) $\boxed{\frac{T_1}{p}}$ (B) $T_1 \cdot p$ (C) $\frac{T_1 \cdot (p-1)}{p}$ (D) $\frac{p-1}{T_1}$ (E) $2 \cdot T_1$

Execution time $\mathbf{T_p}$ on $\mathbf{p}$ CPUs

- $\mathbf{T_p = T_1 / p}$ (perfection)
- $\mathbf{T_p > T_1 / p}$ (performance loss, what normally happens)
- $\mathbf{T_p < T_1 / p}$ (sorcery!)

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)

Gustafson: $S_p = p - f(p-1)$.

From the text: $S_p = 4$ and $f = 0.2$

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)

Gustafson: $S_p = p - f(p-1)$.

From the text: $S_p = 4$ and $f = 0.2$

$$
\begin{aligned}
& 4 = p - 0.2(p-1) \\
\Leftrightarrow\ & 4 = 0.8p + 0.2 \\
\Leftrightarrow\ & 3.8 = 0.8p \\
\Leftrightarrow\ & p = 4.75
\end{aligned}
$$

→ 5 processors are required!

Pipelines and Speedup (15 points)

2. (a) i. Welcher Speedup kann bei einem Task mit einem sequentiellen Anteil von 0.16 erreicht werden, wenn der Task gemäss dem Amdahl-Gesetz auf $p = 126$ Prozessoren ausgeführt wird? *Hinweis:* $126 \times 16 = 2016$

Given a task with a sequential fraction of 0.16, *what speedup can be achieved when running the task on* $p = 126$ *processors according to Amdahl's law? Hint:* $126 \times 16 = 2016$

(3)

Pipelines and Speedup (15 points)

2. (a) i. Welcher Speedup kann bei einem Task mit einem sequentiellen Anteil von 0.16 erreicht werden, wenn der Task gemäss dem Amdahl-Gesetz auf $p = 126$ Prozessoren ausgeführt wird? *Hinweis:* $126 \times 16 = 2016$

Given a task with a sequential fraction of 0.16, what speedup can be achieved when running the task on $p = 126$ processors according to Amdahl's law? Hint: $126 \times 16 = 2016$ (3)

Tobias Steinbrecher @tsteinbreche · 8 months ago

11

We have

$$f = 0.16 := \text{sequential part of the program}$$

With Amdahl's law we obtain:

$$S_{126} = \frac{1}{0.16 + 0.84/126} = \frac{126}{20.16 + 0.84} = \frac{126}{21} = 6$$

ii. Kann eine solcher Speedup in der Praxis auf einem realen System erreicht werden? Wenn nicht, was verhindert den optimalen Speedup und unter welchen Umständen kann ein Speedup erreicht werden, der nahe am theoretischen Wert liegt?

Can such a speedup be obtained in practice on a real system? If not, what hinders the optimal speedup and under what circumstances can a speedup be achieved that is close to the theoretical value? (3)

ii. Kann eine solcher Speedup in der Praxis auf einem realen System erreicht werden? Wenn nicht, was verhindert den optimalen Speedup und unter welchen Umständen kann ein Speedup erreicht werden, der nahe am theoretischen Wert liegt?

Can such a speedup be obtained in practice on a real system? If not, what hinders the optimal speedup and under what circumstances can a speedup be achieved that is close to the theoretical value? (3)

Sven Glinz @sglinz · 8 months ago

12

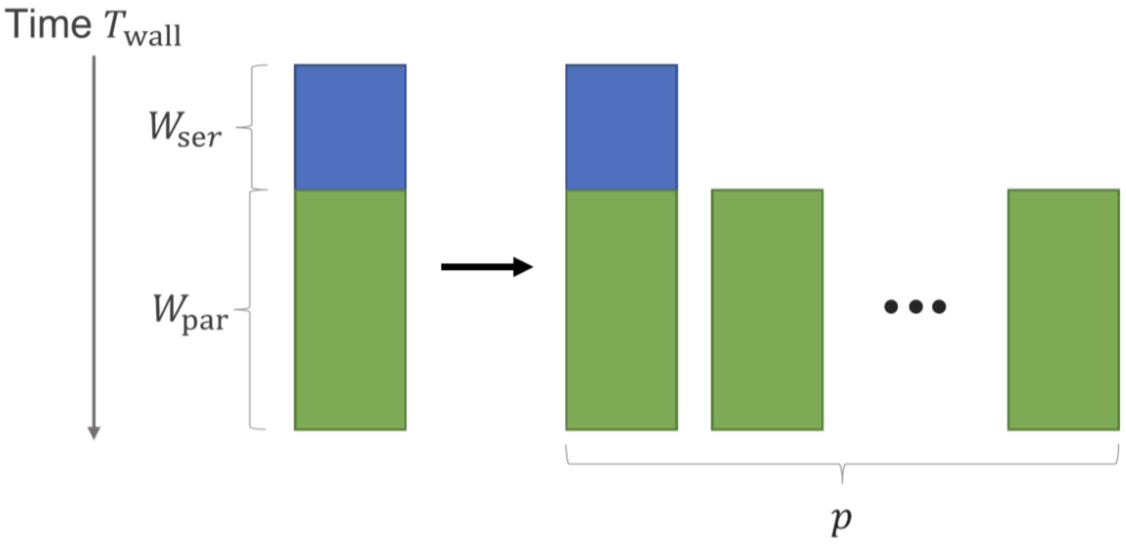
such a speedup could be reached if the needed communication between the individual threads is very low (eg. no locks, no need for cache coherency), each thread can optimally use the CPU and the number of threads is close to the number of logical cores such that only few context switches take place.

In practice, synchronization between threads, memory contention (multiple threads trying to access memory at the same time), load imbalance between different threads etc. could prevent an optimal speedup.

+ Add Comment ··· More

iii. Ist Amdahl's Gesetz oder Gustafson's Gesetz in der folgenden Abbildung dargestellt? Benennen Sie das Gesetz, definieren Sie den seriellen Anteil f anhand der seriellen und parallelen Anteil der Arbeit, W_{ser} und W_{par}, und leiten Sie daraus die Formel für den Speedup S_p ab.

Is Amdahl's Law or Gustafson's Law demonstrated in the illustration below? Name the law, define the serial fraction f in terms of the serial and parallelizable fractions of the work, W_{ser} and W_{par}, and use it to derive the formula for speed up S_p. (4)



Tobias Steinbrecher @tsteinbreche · 8 months ago · edited 8 months ago

16

Gustafson's Law

We have

$$f = \frac{W_{ser}}{W_{ser} + W_{par}} := \text{sequential part}$$

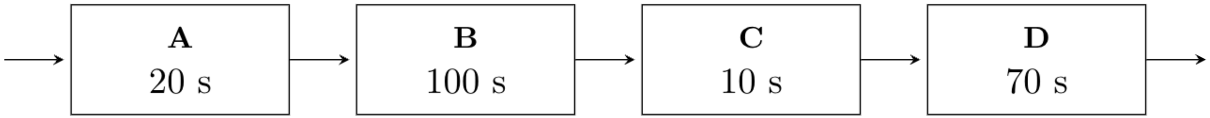
Thus, for the speedup we obtain:

$$S_p = \frac{W_{ser} + pW_{par}}{W_{ser} + W_{par}} = \frac{W_{ser}}{W_{ser} + W_{par}} + \frac{pW_{par}}{W_{ser} + W_{par}} = f + p \cdot \frac{W_{par} + W_{ser} - W_{ser}}{W_{ser} + W_{par}} = f + p(1 - f)$$

+ Add Comment ··· More

(b) Betrachten Sie die unten gezeigte Pipeline. Es gibt mehrere parallele Eingänge und A, B, C und D sind unterschiedliche Funktionseinheiten.

Consider the pipeline shown below. There are multiple parallel inputs and A, B, C, and D are different functional units.

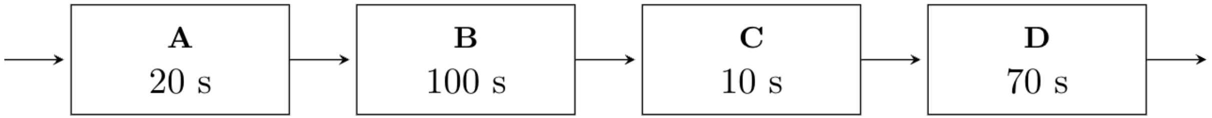


i. Ist die Pipeline balanced? Begründen Sie Ihre Antwort. Sie müssen keine Zahl oder Berechnung angeben.

Is the pipeline balanced? Justify your answer. You do not need to provide a number or calculation. (2)

(b) Betrachten Sie die unten gezeigte Pipeline. Es gibt mehrere parallele Eingänge und A, B, C und D sind unterschiedliche Funktionseinheiten.

Consider the pipeline shown below. There are multiple parallel inputs and A, B, C, and D are different functional units.



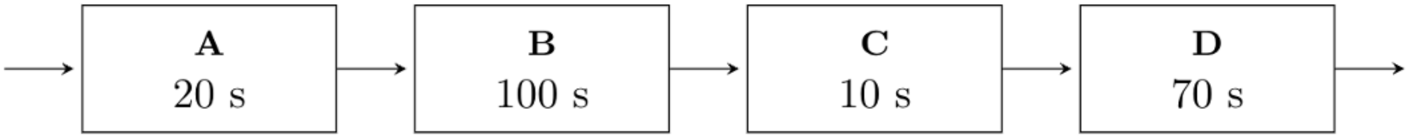
i. Ist die Pipeline balanced? Begründen Sie Ihre Antwort. Sie müssen keine Zahl oder Berechnung angeben.

Is the pipeline balanced? Justify your answer. You do not need to provide a number or calculation. (2)

No, since the stages don‘t take the same amount of time

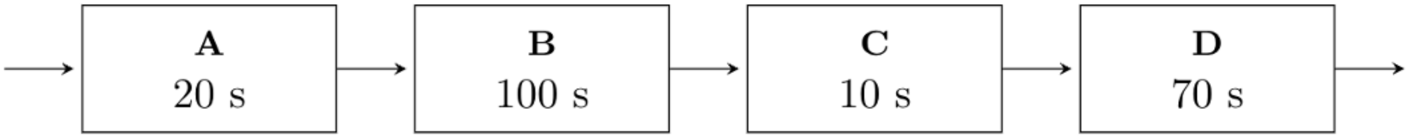
ii. Wie lange wird es dauern, die gesamte Pipeline (d.h. die Stufen A-B-C-D) *15* Mal auszuführen? Gehen Sie davon aus, dass wir die Pipeline so parallelisieren, dass jede Stufe parallel zu den anderen Stufen laufen kann, jedoch nicht zu sich selbst. Achten Sie darauf, Ihre Antwort in den entsprechenden Einheiten anzugeben.

How long will it take to run the whole pipeline (i.e., stages A-B-C-D) 15 times? Assume that we parallelize the pipeline such that each stage can run in parallel with the other stages, but not with itself. Make sure to leave your answer in the appropriate units. (3)



ii. Wie lange wird es dauern, die gesamte Pipeline (d.h. die Stufen A-B-C-D) *15* Mal auszuführen? Gehen Sie davon aus, dass wir die Pipeline so parallelisieren, dass jede Stufe parallel zu den anderen Stufen laufen kann, jedoch nicht zu sich selbst. Achten Sie darauf, Ihre Antwort in den entsprechenden Einheiten anzugeben.

How long will it take to run the whole pipeline (i.e., stages A-B-C-D) 15 times? Assume that we parallelize the pipeline such that each stage can run in parallel with the other stages, but not with itself. Make sure to leave your answer in the appropriate units. (3)



t = 15*(100s) + 20s + 10s + 70s = 1600s

```java
public class Main {
  public static Thread CreateThread(int start) {
    return new Thread(new Runnable() {

      @Override
      public void run() {
        for (int i = start; i < 7; i+=2) {
            System.out.println("Number " + i);
        }
      }
    });
  }

  public static void main(String[] args) throws InterruptedException {
    CreateThread(1).start();
    CreateThread(2).start();
  }
}
```

Markieren Sie alle Ausgaben, welche durch den Codeausschnitt ausgegeben werden können.

Mark all the print sequences that can be produced by running the program shown above.

☐

```
Number 1
Number 2
Number 3
Number 4
Number 5
Number 6
```

☐

```
Number 1
Number 6
Number 3
Number 4
Number 5
Number 2
```

☐

```
Number 6
Number 5
Number 4
Number 3
Number 2
Number 1
```

☐

```
Number 2
Number 4
Number 6
Number 1
Number 3
Number 5
```

```java
public class Main {
  public static Thread CreateThread(int start) {
    return new Thread(new Runnable() {

      @Override
      public void run() {
        for (int i = start; i < 7; i+=2) {
            System.out.println("Number " + i);
        }
      }
    });
  }

  public static void main(String[] args) throws InterruptedException {
    CreateThread(1).start();
    CreateThread(2).start();
  }
}
```

Markieren Sie alle Ausgaben, welche durch den Codeausschnitt ausgegeben werden können.

Mark all the print sequences that can be produced by running the program shown above.

☐

```
Number 1
Number 2
Number 3
Number 4
Number 5
Number 6
```

☐

```
Number 1
Number 6
Number 3
Number 4
Number 5
Number 2
```

☐

```
Number 6
Number 5
Number 4
Number 3
Number 2
Number 1
```

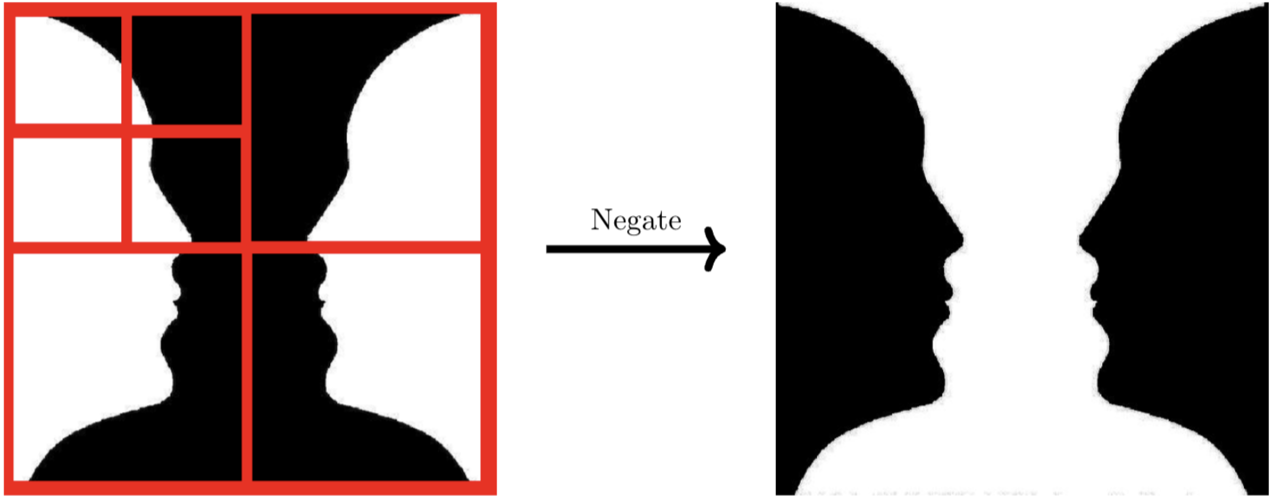
☐

```
Number 2
Number 4
Number 6
Number 1
Number 3
Number 5
```

Fork/Join Framework (16 points)

3. Der folgende Code zielt darauf ab, ein Bild zu negieren, indem es mithilfe des Fork/Join-Frameworks rekursiv in mehrere Unterfenster (vier pro Rekursionsschritt) unterteilt wird. Die Unterfenster können dann parallel negiert werden. Das folgende Beispiel verdeutlicht die Unterteilung des Bildes und die Negierung der einzelnen Unterfenster.

The following code aims to negate an image by recursively subdividing it into multiple subwindows (four per recursion step) using the Fork/Join framework. The subwindows can then be negated in parallel. The example below illustrates the subdivision of the image and negation of the individual subwindows.



Bitte lesen Sie den Code sorgfältig durch und beantworten Sie dann die Fragen zum Code:

Please read the code carefully and then answer the questions regarding the code:

```java
public class ImageNegationFJ extends RecursiveAction {
    final static int CUTOFF = 32;
    double[][] image, invertedImage;
    int startx, starty;
    int length;

    public ImageNegationFJ(double[][] image, double[][] invertedImage,
            int startx, int starty, int length) {
        this.image = image;
        this.invertedImage = invertedImage;
        this.startx = startx;
        this.starty = starty;
        this.length = length;
    }

    @Override
    protected void compute() {
```

```java
@Override
protected void compute() {
    if (this.length <= CUTOFF) {
        for (int offsetX = 0; offsetX < this.length; offsetX++) {
            for (int offsetY = 0; offsetY < this.length; offsetY++) {
                this.invertedImage[this.startx + offsetX][this.starty + offsetY] = 1
                        - this.image[this.startx + offsetX][this.starty + offsetY];
            }
        }
    } else {
        int halfSize = (this.length) / 2;
        ImageNegationFJ upperLeft = new ImageNegationFJ(this.image,
                this.invertedImage, this.startx, this.starty, halfSize);
        ImageNegationFJ upperRight = new ImageNegationFJ(this.image,
                this.invertedImage, this.startx + halfSize, this.starty,
                halfSize);
        ImageNegationFJ lowerLeft = new ImageNegationFJ(this.image,
                this.invertedImage, this.startx, this.starty + halfSize,
                halfSize);
        ImageNegationFJ lowerRight = new ImageNegationFJ(this.image,
                this.invertedImage, this.startx + halfSize,
                this.starty + halfSize, halfSize);
        upperLeft.fork();
        upperLeft.join();
        upperRight.fork();
        upperRight.join();
        lowerLeft.fork();
        lowerLeft.join();
        lowerRight.compute();
    }
}
```

```java
final static int CUTOFF = 32;
```

```java
double[][] image, invertedImage;
```

```java
@Override
protected void compute() {
    if (this.length <= CUTOFF) {
        for (int offsetX = 0; offsetX < this.length; offsetX++) {
            for (int offsetY = 0; offsetY < this.length; offsetY++) {
                this.invertedImage[this.startx + offsetX][this.starty + offsetY] = 1
                        - this.image[this.startx + offsetX][this.starty + offsetY];
            }
        }
    } else {
        int halfSize = (this.length) / 2;
        ImageNegationFJ upperLeft = new ImageNegationFJ(this.image,
                this.invertedImage, this.startx, this.starty, halfSize);
        ImageNegationFJ upperRight = new ImageNegationFJ(this.image,
                this.invertedImage, this.startx + halfSize, this.starty,
                halfSize);
        ImageNegationFJ lowerLeft = new ImageNegationFJ(this.image,
                this.invertedImage, this.startx, this.starty + halfSize,
                halfSize);
        ImageNegationFJ lowerRight = new ImageNegationFJ(this.image,
                this.invertedImage, this.startx + halfSize,
                this.starty + halfSize, halfSize);
        upperLeft.fork();
        upperLeft.join();
        upperRight.fork();
        upperRight.join();
        lowerLeft.fork();
        lowerLeft.join();
        lowerRight.compute();
    }
}
```

(a) Welche Annahme trifft der Code bezüglich der Abmessungen des Arrays, das das Eingabebild darstellt?

What assumption does the code make concerning the dimensions of the array representing the input image? (2)

Tobias Steinbrecher @tsteinbreche · 8 months ago · edited 8 months ago

13

The image should be square $s \times s$ and we should have $s = d2^k$, where $d \leq 32$. This is necessary, because we want `length` to be divisible by 2 in the case `length > 32`. If this would not be the case, we would do floor division and leave pixels unprocessed.

+ Add Comment ··· More

```java
@Override
protected void compute() {
    if (this.length <= CUTOFF) {
        for (int offsetX = 0; offsetX < this.length; offsetX++) {
            for (int offsetY = 0; offsetY < this.length; offsetY++) {
                this.invertedImage[this.startx + offsetX][this.starty + offsetY] = 1
                        - this.image[this.startx + offsetX][this.starty + offsetY];
            }
        }
    } else {
        int halfSize = (this.length) / 2;
        ImageNegationFJ upperLeft = new ImageNegationFJ(this.image,
                this.invertedImage, this.startx, this.starty, halfSize);
        ImageNegationFJ upperRight = new ImageNegationFJ(this.image,
                this.invertedImage, this.startx + halfSize, this.starty,
                halfSize);
        ImageNegationFJ lowerLeft = new ImageNegationFJ(this.image,
                this.invertedImage, this.startx, this.starty + halfSize,
                halfSize);
        ImageNegationFJ lowerRight = new ImageNegationFJ(this.image,
                this.invertedImage, this.startx + halfSize,
                this.starty + halfSize, halfSize);
        upperLeft.fork();
        upperLeft.join();
        upperRight.fork();
        upperRight.join();
        lowerLeft.fork();
        lowerLeft.join();
        lowerRight.compute();
    }
}
```

(b) Parallelisiert der Code die beabsichtigte Aufgabe korrekt oder gibt es weitere Optimierungsmöglichkeiten? Wenn ja, welche Optimierung würden Sie vorschlagen und warum?

Does the code correctly parallelize the intended task or is there further optimization that could be done? If so, which optimization would you propose and why? (4)

Not good:

```
upperLeft.fork();
upperLeft.join();
upperRight.fork();
upperRight.join();
lowerLeft.fork();
lowerLeft.join();
lowerRight.compute();
```

Tobias Steinbrecher @tsteinbreche · 8 months ago · edited 7 months ago

8

No, the parallelization is incorrect, as we have subsequent `fork()` and `join()` calls, which means that we wait for the corresponding subproblem to be finished, before calling `fork()` on the next one. To fix this, we should do the following:

```
upperLeft.fork();
upperRight.fork();
lowerLeft.fork();
lowerRight.compute();
upperLeft.join();
upperRight.join();
lowerLeft.join();
```

```java
public class ImageNegationFJ extends RecursiveAction {
    final static int CUTOFF = 32;
    double[][] image, invertedImage;
    int startx, starty;
    int length;

    public ImageNegationFJ(double[][] image, double[][] invertedImage,
            int startx, int starty, int length) {
        this.image = image;
        this.invertedImage = invertedImage;
        this.startx = startx;
        this.starty = starty;
        this.length = length;
    }

    @Override
    protected void compute() {
```

(c) Vervollständigen Sie das folgende Codegerüst, indem Sie die oben implementierte `ImageNegationFJ` Klasse und die `ForkJoinPool` Klasse verwenden, um die Variable `negatedImage` mit den negierten Werten zu füllen.

Complete the following code skeleton by using the above implemented `ImageNegationFJ` class and the `ForkJoinPool` class to fill the variable `negatedImage` with the negated values. (4)

```java
double[][] image = {{0, 1}, {1, 0}};
int imageSize = image.length;
double[][] negatedImage = new double[imageSize][imageSize];
```

. . . .

. . . .

. . . .

. . . .

. . . .

. . . .

Tobias Steinbrecher @tsteinbreche · 8 months ago

10

```
double[][] image = {{0,1}, {1,0}};
int imageSize = image.length;
double[][] negatedImage = new double[imageSize][imageSize];
ForkJoinPool fjp = new ForkJoinPool();
ForkJoinTask t = new ImageNegationFJ(image, negatedImage, 0, 0 ,imageSize);
fjp.invoke(t);
```

+ Add Comment ··· More

(d) Unter der Annahme, dass die Klasse `ImageNegationFJ` korrekt parallelisiert ist, wie viele Threads verwendet der ForkJoin-Pool effektiv, um das 2×2 `negatedImage` Array aus Aufgabe 3c) zu füllen?

Assuming that the `ImageNegationFJ` *class is correctly parallelized, how many threads does the ForkJoinPool effectively use to fill the* 2×2 `negatedImage` *array from task 3c)?* (2)

```java
double[][] image = {{0,1}, {1,0}};
```

```java
@Override
protected void compute() {
    if (this.length <= CUTOFF) {
        for (int offsetX = 0; offsetX < this.length; offsetX++) {
            for (int offsetY = 0; offsetY < this.length; offsetY++) {
                this.invertedImage[this.startx + offsetX][this.starty + offsetY] = 1
                        - this.image[this.startx + offsetX][this.starty + offsetY];
            }
        }
    } else {
        int halfSize = (this.length) / 2;
        ImageNegationFJ upperLeft = new ImageNegationFJ(this.image,
                this.invertedImage, this.startx, this.starty, halfSize);
        ImageNegationFJ upperRight = new ImageNegationFJ(this.image,
                this.invertedImage, this.startx + halfSize, this.starty,
                halfSize);
        ImageNegationFJ lowerLeft = new ImageNegationFJ(this.image,
                this.invertedImage, this.startx, this.starty + halfSize,
                halfSize);
        ImageNegationFJ lowerRight = new ImageNegationFJ(this.image,
                this.invertedImage, this.startx + halfSize,
                this.starty + halfSize, halfSize);
        upperLeft.fork();
        upperLeft.join();
        upperRight.fork();
        upperRight.join();
        lowerLeft.fork();
        lowerLeft.join();
        lowerRight.compute();
    }
}
```

```java
final static int CUTOFF = 32;
```

(d) Unter der Annahme, dass die Klasse `ImageNegationFJ` korrekt parallelisiert ist, wie viele Threads verwendet der ForkJoin-Pool effektiv, um das 2×2 `negatedImage` Array aus Aufgabe 3c) zu füllen?

Assuming that the `ImageNegationFJ` class is correctly parallelized, how many threads does the ForkJoinPool effectively use to fill the 2×2 `negatedImage` array from task 3c)? (2)

```
double[][] image = {{0,1}, {1,0}};
```

```
@Override
protected void compute() {
    if (this.length <= CUTOFF) {
        for (int offsetX = 0; offsetX < this.length; offsetX++) {
            for (int offsetY = 0; offsetY < this.length; offsetY++) {
                this.invertedImage[this.startx + offsetX][this.starty + offsetY] = 1
                        - this.image[this.startx + offsetX][this.starty + offsetY];
            }
        }
    } else {
        int halfSize = (this.length) / 2;
        ImageNegationFJ upperLeft = new ImageNegationFJ(this.image,
                this.invertedImage, this.startx, this.starty, halfSize);
        ImageNegationFJ upperRight = new ImageNegationFJ(this.image,
                this.invertedImage, this.startx + halfSize, this.starty,
                halfSize);
        ImageNegationFJ lowerLeft = new ImageNegationFJ(this.image,
                this.invertedImage, this.startx, this.starty + halfSize,
                halfSize);
        ImageNegationFJ lowerRight = new ImageNegationFJ(this.image,
                this.invertedImage, this.startx + halfSize,
                this.starty + halfSize, halfSize);
        upperLeft.fork();
        upperLeft.join();
        upperRight.fork();
        upperRight.join();
        lowerLeft.fork();
        lowerLeft.join();
        lowerRight.compute();
    }
}
```

```
final static int CUTOFF = 32;
```

Tobias Steinbrecher @tsteinbreche · 8 months ago · edited 8 months ago

Because of the sequential cutoff, only **one** Thread would be used *effectively*.

(e) Gehen Sie von einem konstanten Overhead von 16 MB $= 2^4$ MB pro Thread aus und dass pro Split immer vier neue Threads erstellt werden. Dies bedeutet, dass die Anzahl der Threads nicht durch den ForkJoinPool festgelegt wird, sodass kein Thread wiederverwendet wird und es zu keinem Work Stealing zwischen den Threads kommt. Was ist der niedrigste Wert für `CUTOFF`, wenn Sie ein Bild der Größe 4000×4000 eingeben, bevor Ihnen bei einem RAM der Größe 10 GB der Speicher ausgeht? Hinweis: 1 GB $= 2^{10}$ MB.

Assume a fixed overhead of 16 MB $= 2^4$ MB per thread and that there are always four new threads created per split. This means that the number of threads is not fixed by the ForkJoinPool, so no thread is re-used and there is no work stealing among the threads. What is the lowest value for `CUTOFF` *if you input an image of size 4000×4000 before you run out of memory using a RAM of size 10 GB? Hint:* 1 GB $= 2^{10}$ MB. (4)

(e) Gehen Sie von einem konstanten Overhead von $16\,\text{MB} = 2^4\,\text{MB}$ pro Thread aus und dass pro Split immer vier neue Threads erstellt werden. Dies bedeutet, dass die Anzahl der Threads nicht durch den ForkJoinPool festgelegt wird, sodass kein Thread wiederverwendet wird und es zu keinem Work Stealing zwischen den Threads kommt. Was ist der niedrigste Wert für `CUTOFF`, wenn Sie ein Bild der Größe 4000×4000 eingeben, bevor Ihnen bei einem RAM der Größe $10\,\text{GB}$ der Speicher ausgeht? Hinweis: $1\,\text{GB} = 2^{10}\,\text{MB}$.

Assume a fixed overhead of $16\,\text{MB} = 2^4\,\text{MB}$ per thread and that there are always four new threads created per split. This means that the number of threads is not fixed by the ForkJoinPool, so no thread is re-used and there is no work stealing among the threads. What is the lowest value for `CUTOFF` *if you input an image of size 4000×4000 before you run out of memory using a RAM of size $10\,\text{GB}$? Hint: $1\,\text{GB} = 2^{10}\,\text{MB}$.* (4)

Tobias Steinbrecher @tsteinbreche · 8 months ago · edited 8 months ago

14

Number of threads, which we can use:

$$N = \frac{10 \cdot 2^{10}}{2^4} = 10 \cdot 2^6 = 10 \cdot 4^3$$

In each recursive call, we will use 4 new threads (under given assumptions). Thereby, we have the constraint ($i :=$ number of divisions)

$$4^i \leq 10 \cdot 4^3 \iff i \leq \log_4(10) + 3 \iff i \leq 4$$

and the smallest possible value is $\text{CUTOFF} = 4000/2^4 = \mathbf{250}$ to avoid a fifth division.

+ Add Comment ··· More