Name, Vorname (Druckbuchstaben): _____

Legi Nummer: _____

252-0024-00L
Parallele Programmierung
ETH/CS: FS 2016
Basisprüfung
Montag, 30.1.2017
120 Minuten

Diese Prüfung enthält 24 Seiten (inklusive dieses Deckblatts) und 9 Aufgaben. Überprüfen Sie, dass keine Seiten fehlen. Füllen Sie alle oben verlangten Informationen aus. Schreiben Sie die Legi-Nummer oben auf jede einzelne Seite, für den Fall, dass Seiten verlorengehen oder abgetrennt werden.
Nehmen Sie sich am Anfang 5 Minuten Zeit, um alle Aufgaben durchzulesen. Während dieser Zeit ist es nicht erlaubt, Prüfungsfragen zu beantworten. Danach haben Sie 120 Minuten Zeit für die Lösung der Aufgaben.
Falls Sie sich durch irgendjemanden oder irgendetwas gestört fühlen, melden Sie dies sofort einer Aufsichtsperson. Wir sammeln die Prüfung zum Schluss ein. Wichtig: stellen Sie unbedingt selbst sicher, dass Ihre Prüfung von einem Assistenten eingezogen wird. Stecken Sie keine Prüfung ein und lassen Sie Ihre Prüfung nicht einfach am Platz liegen. Dasselbe gilt, wenn Sie früher abgeben wollen: bitte melden Sie sich lautlos und wir holen die Prüfung ab. Vorzeitige Abgaben sind nur bis 15 Minuten vor Prüfungsende möglich.
Wenn Sie zur Toilette müssen, melden Sie dies einer Aufsichtsperson durch Handzeichen. Es darf zur gleichen Zeit immer nur eine Studentin oder ein Student zur Toilette.
Es gelten die folgenden Regeln:

- **Lösungen müssen lesbar sein**. Verwenden Sie für Ihre Lösungen den verfügbaren Platz. Lösungen mit unklarer Reihenfolge oder anderweitig unverständlicher Präsentation können zu Punktabzügen führen.
- **Lösungen ohne Lösungsweg erhalten nicht die volle Punktzahl**. Eine korrekte Antwort ohne Lösungsweg, Erklärungen oder algebraischen Umformungen erhält keine Punkte; inkorrekte Antworten mit teilweise richtigen Formeln, Berechnungen und Umformungen können Teilpunkte erhalten.
- Falls mehr Platz benötigt wird, fordern Sie bei den Assistenten zusätzliche Blätter an. Versehen Sie die Aufgabe gegebenfalls mit einem klaren Hinweis. Richtlinie: **Die Aufgaben sollten sich alle in dem vorgegebenen Platz beantworten lassen.**
- Die Aufgaben können auf **Englisch oder Deutsch** beantwortet werden. Benutzen Sie keinen roten Stift!

Problem	Points	Score
1	10	
2	7	
3	8	
4	10	
5	10	
6	10	
7	12	
8	7	
9	7	
Total:	81	

Sequential Java (10 points)

Programmierung

Programming

1. (a) Vervollständigen Sie folgende Methode `greatestFive` so, dass sie die Position p einer zusammenhängende Teilfolge aus fünf Elementen des Arrays `int [] x` zurückgibt, welche eine maximale Summe $S_p = x_p + x_{p+1} + ... + x_{p+4}$ aufweist.
Beispiel: die größte Teilfolge aus 5 Zahlen der Folge $(1, 2, 3, 4, 5, 6, 7, 8, 9, 8, 7, 6, 5, 4, 3, 2, 1)$ ist die Folge $(7, 8, 9, 8, 7)$ an der Position 6.
Sie können davon ausgehen, dass das Array `x` mindestens 5 Elemente enthält.

Complement the code of the following method `greatestFive` such that it returns the position p of a contiguous five-element subsequence of the array `int x[]` such that $S_p = x_p + x_{p+1} + ... + x_{p+4}$ provides a maximal value.
For example the greatest five element subsequence of numbers in $(1, 2, 3, 4, 5, 6, 7, 8, 9, 8, 7, 6, 5, 4, 3, 2, 1)$ is provided as $(7, 8, 9, 8, 7)$ at position 6.
You can assume that the array `x` contains at least 5 elements.

(5)

```
// pre: x != null && x.length >= 5;
// post: return position p where x[p] + x[p+1] + ... +
   x[p+4] takes on a maximal value
static int greatestFive (int[] x){
    assert(x != null && x.length >= 5);
    Solution: (below)
}
```

Solution:

```
// pre: x != null && x.length >= 5;
// post: return position p where x[p] + x[p+1] + ... +
   x[p+4] takes on a maximal value
static int greatestFive (int[] x){
    assert(x != null && x.length >= 5);
    int fiveSum = 0;
    for (int i = 0; i < 5; ++i){
        fiveSum += x[i];
    }
    int greatest = fiveSum;
    int position = 0;
    for (int i=5; i < x.length-5; ++i)
    {
        fiveSum += x[i];
        fiveSum -= x[i-5];
        if (fiveSum > greatest){
            position = i-4;
            greatest = fiveSum;
        }
    }
    return position;
```

```
}

5 Points if everything correct

-1 Point for each substantial bug, such as a missing
   return, index bound violation, or semantic error
```

Theorie

Theory

(b) Seien a und b jeweils Objekte von einem beliebigen (nicht-primitivem) Typ. Im folgenden möchten wir a und b miteinander vergleichen. Erklären Sie den Unterschied zwischen den beiden Ausdrücken (a == b) und (a.equals(b)). Geben Sie ein Beispiel in welchem die beiden Ausdrücke verschiedene Werte zurückgeben könnten. (3)

Let a and b be objects of arbitrary (non-primitive) type. Imagine that you want to compare a and b. Explain the difference between the expressions (a == b) and (a.equals(b)). Give an example in which the two expressions return different values.

Solution: Objects of non-primitive type are represented as references in Java. A comparison of two objects provides a comparison of the two pointers and yield true if and only if the two objects point to the same instance. A (conterintuitive) example is String comparison:

```
String x1 = "ETH";
String x2 = "ETH";
assert(x1 == x2); // often goes "wrong".
```

+1 for mentioning references
+1 for comparison result
+1 for example

Multiple-choice

Multiple-choice

(c) Gegeben sei der folgende Code: (1)

Given the following Code:

```
int i = 0;
switch (i) {
    case 0:
        System.out.print("a");
        i = 2;
    case 1:
        System.out.print("b");
        break;
    case 2:
        System.out.print("c");
    default:
        System.out.print("d");
}
```

Was gibt der Code aus?

What does the code return?

- ☐ a — *a*
- ☐ ac — *ac*
- √ **ab** — *ab*
- ☐ abc — *abc*

(d) Gegeben sei der folgende Code: *Given the following Code:* (1)

```
String[] words = {"a", null, "c", "d"};
for (int i = 0; i <= words.length; i++) {
    if (!words[i].equals("c")) {
        System.out.print(words[i]);
    }
}
```

Was gibt der Code aus? *What does the code return?*

- ☐ `a` — `a`
- ☐ `ad` — `ad`
- ☐ `acd` — `acd`
- √ **Nach der Ausgabe von `a` wird eine Exception geworfen** — *After printing* `a` *an exception is thrown*
- ☐ Nach der Ausgabe von `acd` wird eine Exception geworfen — *After printing* `acd` *an exception is thrown*

Speedup, Amdahl, Gustafson (7 points)

2. Beantworten Sie die folgenden Fragen zum Thema Speedup sowie Amdahlsches und Gustafsonsches Gesetz:

Answer the following questions about speedup, Amdahl's and Gustafson's laws:

(a) Schreiben Sie die Formel für den Speedup nach dem **Amdahlschen** Gesetz auf und erläutern Sie kurz alle Terme. (2)

Write down the formula for speedup according to **Amdahl's** *law and briefly describe its terms*

Solution:

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

S_p - Speedup;
f - not paralelizable portion of any parallel process;
p - number of cores;

Grading: 0.5p for formula only, 0.5p for each explanation.

(b) Die Ausführungszeit eines parallelen Programmes betrage auf einer 16-core Maschine 1 Stunde. Es sei bekannt, dass 20 Prozent des Codes nicht parallelisiert werden kann. Was wäre dann die Ausführungsdauer desselben Programmes (nach dem **Gustavsonschen** Gesetz), wenn nur ein einziger Core zur Verfügung steht? Wie gross ist der Speedup gemäss dem Gustavsonschen Gesetz? (3)

The execution time of the parallel program on a 16 core machine is 1 hour. Knowing that 20 percent of the code cannot be parallelized, compute what is the execution time if only the single core is available according to **Gustafson's** *law? What is the speedup according to Gustafson's law?*

Solution:

$$p = 16, f = 0.2, T_p = 1$$
$$T_s = ?$$

$$\begin{aligned} T_s &= f \cdot T_p + p \cdot (1 - f) \cdot T_p \\ &= 0.2 \cdot 1 + 16 \cdot (1 - 0.2) \cdot 1 \\ &= 13 \\ S_p &= T_s / T_p = f + p \cdot (1 - f) = 13 \end{aligned}$$

Grading: 1p for formula only, 2p for result. The execution time on the single core machine is 13 hours. The speedup is 13.

(c) Was ist nach dem Gustavsonschen Gesetz der maximal erreichbare relative Speedup S_p/p, wenn 10% eines Algorithmus nicht parallelisiert werden können?

Using Gustavson's law, compute the maximally achievable relative speedup S_p/p when 10% of the algorithm cannot be parallelized

(2)

Solution:

$$S_p/p = \frac{T_1}{T_p \cdot p} = f/p + (1 - f)$$

With $p \to \infty$ we get $S/p = 1 - f = 0.9$

Grading: 1p for formula only, 2p for result.

Pipelining (8 points)

3. Abbildung 1 zeigt eine Pipeline, welche zur Reaktion auf Hindernisse für ein selbstfahrendes Auto entwickelt wurde. Gehen Sie davon aus dass für jede Stufe der Pipeline eine separate Ausführungseinheit zur Verfügung steht. Das gilt auch nach jeder Modifikation der Pipeline.

Figure 1 shows a pipeline that was designed for a self driving car to react to obstacles. We assume that there exists a separate execution unit for each stage of the Pipeline. This holds true after every modification of the pipeline.

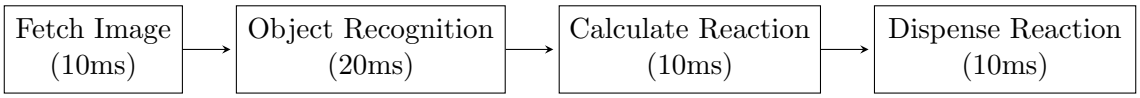


Abbildung 1: Pipeline

(a) Approximieren Sie den Durchsatz dieser Pipeline. *Approximate the throughput of the pipeline.* (2)

Solution: $1/(20 \cdot 10^{-3}) = 50[Elements/s]$

(b) Das Forschungsteam hat einen Weg gefunden die Stufe "Calculate Reaction" in die zwei Stufen "Wheel Reaction" (5ms) und "Speed Reaction" (5ms) zu teilen. Wie verändert sich der Durchsatz? *The research team found a way to divide the stage "Calculate Reaction" into two stages: "Wheel Reaction" (5ms) and "Speed Reaction" (5ms). How does this change the throughput?* (2)

Solution: The throughput does not change.

(c) Die Stufe "Object Recognition" konnte in zwei Stufen der Länge 10ms unterteilt werden, die Modifikation aus (b) wurde rückgängig gemacht. (Es gibt nun 5 Pipeline Stufen). Welche der folgenden Aussagen trifft auf die neue Version zu? *The stage "Object Recognition" could also be divided in two equal stages of time 10ms, the modification from (b) is reversed. (The pipeline has 5 stages now). Which of the following statements apply to the new version?* (2)

a) Die Version hat eine geringere Latenz in der ersten Iteration als eine sequentielle Version. *a) The version has a lower latency in the first iteration than the sequential version.*

b) Sie hat einen höheren Durchsatz als die anfängliche Version. *b) It has a higher throughput than the initial version.*

c) Sie hat eine geringere Latenz in der ersten Iteration als die anfängliche Version. *c) It has a lower latency in the first iteration than the initial version.*

d) Sie hat eine geringere Latenz als die anfängliche Version in der zweiten Iteration. *d) It has a lower latency in the second iteration than the initial version.*

Solution: a) False, b) True, c) False, d) True

(d) Die Pipeline wurde komplett neu entwickelt und in 40 Pipeline-Stufen zerlegt. Nach dieser Modifikation erhöhte sich der Durchsatz kaum merkbar während die Latenz merklich stieg. Was könnte die Ursache dafür sein?

The pipeline was completely redesigned and divided into 40 stages. After this modification the throughput did hardly improve, while the latency went up significantly. What could be a reason? (2)

Solution: Overhead (Communication Overhead). The throughput is dominated by the maximim of communication overhead eventually while the latency is provided by the sum of all communication overheads.

Task Parallelism (10 points)

4. Fork/Join (10)

Ihre Aufgabe ist es, den untenstehenden Code so zu vervollständigen, dass der Maximalwert einer Zahlereihe zurück gegeben wird. Der Input ist dabei in einem möglicherweise sehr grossen Array gespeichert. Benutzten Sie Javas Fork/Join Framework, um den Parallelismus bestmöglich auszunutzen. Die `compute` Methode soll dabei den maximalen Wert aus dem Array zurückgeben. Geben Sie darüberhinaus einen sinnvollen Wert für den SEQUENTIAL_THRESHOLD an und begründen Sie warum dieser notwendig ist und welcher Wertebereich in etwa dafür in Frage kommt.

Fork/Join

You are tasked with implementing a method to find the maximum element in a potentially very large input array. Complete the below code to make use of Java's Fork/Join framework in order to parallelize this task. The `compute` method is to return the maximum of the array. Choose a sensible SEQUENTIAL_THRESHOLD and briefly provide a rationale why this is necessary and which approximate value to choose.

```
public class MaximumFinder extends RecursiveTask<Integer> {

    private static int SEQUENTIAL_THRESHOLD = _______;
    private int[] data;
    private int start;
    private int end;

    public MaximumFinder(int[] data, int start, int end) {
          this.data = data;
          this.start = start;
          this.end = end;
    }

    public MaximumFinder(int[] data) {
          this(data, 0, data.length);
    }



    protected Integer compute() {
          int length = end - start;
          if (length < SEQUENTIAL_THRESHOLD) {
             return computeSequential();
          }

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

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

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

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

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

                .....................................................
    }

    private Integer computeSequential() {
        System.out.println(Thread.currentThread() + "
            computing: " + start  + " to " + end);

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

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

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

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

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

                .....................................................
    }

}
```

Solution:

Consider that at some point a sequential computation will be faster than creating further (recursive) parallel tasks. (1 pt)

SEQ CUTOFF should be in the range of 1000-10000 this depends on architecture number of cores. But if it's too small or omitted overheads will dominate if it's too large we do not exploit parallelism. Extra points for mentioning parallel slackness... (1 pt)

Solution:

```
public class MaximumFinder extends RecursiveTask<Integer> {

    private static int SEQUENTIAL_THRESHOLD = 500;
    private int[] data;
    private int start;
```

```
    private int end;

    public MaximumFinder(int[] data, int start, int end) {
          this.data = data;
          this.start = start;
          this.end = end;
    }

    public MaximumFinder(int[] data) {
          this(data, 0, data.length);
    }


    protected Integer compute() {
          int length = end - start;
          if (length < SEQUENTIAL_THRESHOLD) {
             return computeSequential();
          }
         int split = length / 2;
         MaximumFinder left = new MaximumFinder(data, start,
            start + split);
         left.fork();
         MaximumFinder right = new MaximumFinder(data, start
            + split, end);
         return Math.max(right.compute(), left.join());
     }


     private Integer computeSequential() {
          System.out.println(Thread.currentThread() + "
             computing: " + start  + " to " + end);

          int max = Integer.MIN_VALUE;
          for (int i = start; i < end; i++) {
              if (data[i] > max) {
              max = data[i];
              }
           }
           return max;
     }


}
```

Synchronization (10 points)

5. Die Klasse CyclicBarrier von Java ist eine Synchronisationshilfe, welche einer Menge von Threads erlaubt, an einem gemeinsamen Punkt aufeinander zu warten. Die Barriere wird zyklisch genannt, weil sie, nachdem die wartenden Threads los gelassen wurden, wiederverwendet werden kann. Die Klasse CyclicBarrier stellt die Methode await zur Verfügung. Ein Thread, welcher diese Methode aufruft, wartet solange bis alle anderen Threads auch await auf dieser Barriere aufgerufen haben. Vervollständigen Sie die Implementation der Klasse CyclicBarrier auf der nächsten Seite. Der Konstruktor der Klasse hat zwei Parameter, "parties" ist die Anzahl an Threads, welche await() aufrufen müssen, bevor die Barriere passiert wird, und "barrierAction" ist der Befehl, der von einem einzigen Thread ausgeführt wird, wenn die Barriere passiert wird. (6)

   The CyclicBarrier class of Java is a synchronization aid that allows a set of threads to all wait for each other to reach a common barrier point. The barrier is called cyclic since it can be re-used after the waiting threads are released. The CyclicBarrier class provides the method await. A thread that invokes this method waits until all threads have invoked await on this barrier. Please complement the implementation of the CyclicBarrier class on the opositet page. The constructor of the class has two parameters, "parties" is the number of threads that must invoke await() before the barrier is tripped, and "barrierAction" is the command to execute by a single thread when the barrier is tripped.

   (a) Schreiben Sie die benötigten Felder für die Klasse MyCyclicBarrier auf und erklären deren Zweck mittels eines kurzen Kommentars.

   Write the fields required for the MyCyclicBarrier class and explain what they serve with a short comment.

   (b) Schreiben Sie den Rumpf des Konstruktors.

   Write the body of the constructor.

   (c) Schreiben Sie den Rumpf der await Methode

   Write the body of the await method.

   (d) Stellen Sie sicher, dass die Implementation als zyklische Barriere funktioniert. D.h.: die Barriere kann wiederverwendet werden, nachdem alle Threads losgelassen wurden.

   Make sure that your implementation works as a cyclic barrier, i.e., the barrier can be re-used after the waiting threads are released.

   (e) Stellen Sie sicher, dass "barrierAction" nur genau einmal pro Barriere-Punkt ausgeführt wird, nachdem der letzte Thread in der Gruppe ankommt, aber bevor irgend ein anderer Thread losgelassen wird.

   Make sure that the "barrierAction" executes once per barrier point, after the last thread in the party arrives, but before any threads are released.

```
class MyCyclicBarrier {
   Solution: (below)

  public MyCyclicBarrier(int parties, Runnable
     barrierAction) {
       Solution: (below)
  }

  public synchronized void await() throws
     InterruptedException {
       Solution: (below)
  }
}
```

Solution:

```
class MyCyclicBarrier {
    private final Runnable barrierAction;
    private final int totalParties; // total number of
       threads
    private int awaitParties;   // number of threads still
       needed to reach the barrier

    public MyCyclicBarrier(int parties, Runnable
       barrierAction) {
        this.totalParties = parties;
        this.awaitParties = parties;
        this.barrierAction = barrierAction;
    }

    public synchronized void await() throws
       InterruptedException {
        this.awaitParties--;
        if (this.awaitParties > 0) {
            this.wait();
        } else {
            this.awaitParties = this.totalParties;
            this.barrierAction.run();
            this.notifyAll();
        }
    }
```

(f) Der folgende Code ist ein vereinfachter Ausschnitt der Buchhaltung eines Spiels mit mehreren Spielern. Wenn ein Spieler einen anderen Spieler trifft, so kann er von diesem Credits bekommen. Spieler können zur gleichen Zeit treffen und getroffen werden.

The following code provides a simplified part of the credit book-keeping of a game with several players. A player hitting another player can win parts of his credits. Players can hit and be hit concurrently. (4)

```
class Player {
    private int credits = 100;

    // Player is hit and has to give away 7 credits
    public synchronized int loseCredits() {
        credits -= 7;
        return 7;
    }

    // player hits another player and can win up to 7 credits
    public synchronized void hits(Player opponent) {
        int gain = opponent.loseCredits();
        credits += gain;
    }

    ...
}
```

Jemand simuliert das Spiel, wobei die verschiedenen Spieler mit Threads implementiert werden. Unter Verwendung des obigen Codes für die Buchhaltung der Spielercredits bleibt das Spiel ab und zu einfach stehen. Erklären Sie warum und skizzieren Sie sehr kurz eine Lösung.

Someone runs a simulation of the game, where the behavior of several players are simulated by threads. Using the code above for the players book-keeping the game gets stuck from time to time. Explain why and sketch a solution very briefly.

Solution: The game deadlocks because there might be a circular dependency of players being hit and hitting other players resulting in a deadlock in the bookkeeping (bank account example from the lecture). Solution is to keep an id on the players and use it for a global lock order.

Behind Locks (10 points)

6. (a) Beschreiben Sie kurz, was eine atomare Read-Modify-Write(RMW)-Operation ist und skizzieren Sie die Funktionsweise einer RMW-Operation Ihrer Wahl mit Pseudo-Code.

Describe shortly what an atomic Read-Modify-Write (RMW) operations is and sketch the functionality of one RMW operation of your choice with pseudo-code. (3)

Solution: An atomic RMW Operation provides a means to read and conditionally update a variable atomically. Only one process at a time can win and execute the operation. Examples:
CAS(ref x, old, new): if res = [x]; if [x] = old then x = new; return res.
TAS(ref x, new): if [x] != new then [x] = new return true else return false.

(b) Beschreiben Sie kurz, was ein Spinlock ist und skizzieren Sie eine Implementation mit einer Read-Modify-Write Operation Ihrer Wahl mit Pseudocode.

Explain shortly what a spinlock is. Sketch an implementation with a Read-Modify-Write operation of your choice using Pseudocode. (3)

Solution: A spinlock provides a means to implement mutual exclusion. When several processes want to enter the critical region, they test a variable (memory location) for a certain value. Only one process at a time can see and change the value, which is why RMW operations are used.
Example implementation:

```
acquire: while (cas(lock, 0, 1) = 1) do {};
release: cas(lock, 1, 0);

acquire: while (!tas(lock, true)) do {};\\
release: lock = false;
```

An atomic RMW Operation provides a means to read and conditionally update a variable atomically. Only one process at a time can win and execute the operation. They can be used as a means of synchronization in lock-free programming and to implement a spinlock.

(c) Jemand schlägt den folgenden Algorithmus zur Implementation des gegenseitigen Ausschlusses zweier Processe vor. Prozesse haben IDs 0 oder 1.

Someone proposes the following algorithm to implement mutual exclusion for two processes. Processes provide ids 0 or 1. (4)

```
class NearlyBakery
{
    AtomicInteger ticket[] = new AtomicInteger[2];

    public void Acquire(int id)
    {
        ticket[id].set(ticket[1-id].get()+1);
        while (ticket[1-id].get() != 0 && ticket[1-id].get()
           < ticket[id].get())
    }

    public void Release(int id)
    {
        ticket[id].set(0);
    }
}
```

Was ist das Problem mit dem Algorithmus? Skiziieren Sie ein Szenario, in dem es fehlschlägt und geben Sie einen kurzen Hinweis darauf, wie man den Algorithmus richtigstellen könnte.

What is the problem with this algorithm? Sketch a scenario where it fails. Give a short hint on how to correct this algorithm.

Solution: This algorithm is nearly the Bakery lock, so it is almost correct: problem is that both processes can have the same ticket number, in which case both processes have access. So it does not provide mutual exclusion. Replace the smaller than sign by : smaller than or equal and one of the processes can continue:

```
ticket[1-id].get() < ticket[id].get() ||
ticket[1-id].get() == ticket[id].get() && id==0)
```

Transactional Memory (12 points)

7. Beantworten Sie die folgenden Fragen zum Thema Transactional Memory:

Answer the following questions about Transactional Memory:

(a) Beschreiben Sie zwei Probleme welche unter Verwendung von Locks auftreten können, nicht jedoch mit Transactional Memory. (2)

Describe two problems that might occur using locks but do not occur with Transactional Memory

Solution: Convoying: thread holding a resource R is descheduled while other threads queue up waiting for R.
Priority Inversion: lower priority thread holds a resource R that a high priority thread is waiting on.
Deadlocks: At least not with transactions itself, but still exists on the application level.

Grading: 1 point for each correct description.

(b) Was muss der Programmierer beachten (in Referenz basierten STMs, z.B. scala-stm), wenn er veränderbare geteilte Variablen benutzt ? (2)

What does the programmer have to pay attention to when using shared mutable variables with Reference-based STMs (e.g. scala-stm)?

Solution: Enclosing in special variables: the shared variables has to be wrapped in special variables.
Only access from transactions: the shared variables should only be accessed from within transactions.

Grading: 1 point for each correct description.

(c) Nennen Sie einen Vorteil und einen Nachteil von Hardware Transactional Memory gegenüber Software Transactional Memory. (2)

Name one advantage and one disadvantage of Hardware Transactional Memory compared to Software Transactional Memory.

Solution: Advantage: It is fast, can have lower power consumption.
Disadvantage: Cannot handle big transactions, bounded resources, less flexible, .

Grading: 1 point for each correct point mentioned above.

(d) Implementieren Sie Double-Compare-And-Swap mit Transactional Memory. Sie dürfen atomic{} benutzen um atomare Blöcke zu definieren und return innerhalb des Blockes ist erlaubt. Um Ref.View Referenzen zu lesen oder schreiben können Sie set und get benutzen (z.B. val1.get(), val1.set(10)) (6)

Implement double Compare And Swap using Transactional Memory. You can use atomic{} to specify an atomic block and returns within the block are allowed. For Ref.View references you can use set and get to read or modify the values (i.e. val1.get(), val1.set(10)).

```
public class doubleAtomicInteger {
```

```
    Ref.View<Integer> val1;
    Ref.View<Integer> val2;

   /**
    * \brief CompareAndSwap2 atomically sets val1 and val2
       to up1 and up2
    * If both val1 and val2 match exp1 and exp2.
    *
    * @param exp1 expected value for val1
    * @param exp2 expected value for val2
    * @param up1 the new value for val1
    * @param up2 the new value for val2
    *
    * @return true if operation succeeded otherwise false.
    */
    boolean CompareAndSwap2(int exp1, int exp2, int up1, int
       up2) {
     Solution: (below)
    }
}
```

Solution:

```
public class doubleAtomicInteger {

    Ref.View<Integer> val1;
    Ref.View<Integer> val2;

   /**
    * \brief CompareAndSwap2 atomically sets val1 and val2
       to up1 and up2
    * If both val1 and val2 match exp1 and exp2.
    *
    * @param exp1 expected value for val1
    * @param exp2 expected value for val2
    * @param up1 the new value for val1
    * @param up2 the new value for val2
    *
    * @return true if operation succeeded otherwise false.
    */
    boolean CompareAndSwap2(int exp1, int exp2, int up1,
       int up2) {
        atomic {
            int tmp1, tmp2;
            tmp1 = val1.get();
```

```
            tmp2 = val2.get();
            if (exp1 == tmp1 && exp2 == tmp2) {
                val1.set(up1);
                val2.set(up2);
                return true;
            } else {
                return false;
            }
        }
    }
}
```

2 points for correct atomic block, 2 points for correct comparison and reading out of values, 2 points for correct return values and correctly setting val1/val2

Linearizability (7 points)

8. In den folgenden Aufgaben seien A, B und C Threads, welche mit einem gemeinsam benutzten Stack-Objekt arbeiten. Die Spezifikation ist in der folgenden Interface-Definition ersichtlich:

In the following questions let A, B and C denote threads operating on a shared stack object as specified in the listing below.

```
public interface Stack {
    /* pushes element v onto the stack */
    public void push(int v);

    /* removes the top element and returns it */
    public int pop();

    /* returns the top element */
    public int top();
}
```

(a) Welches der folgenden Szenarien ist linear konsistent? Markieren Sie entweder den Linearisationspunkt oder erklären Sie warum es nicht linear konsistent ist. (4)

Which of the following scenarios are linearly consistent? Either mark the point of linearization or explain why it is not linearly consistent.

1.
```
A s.push(1):  *--------------*
B s.push(2):         *-------------*
A s.pop()->2:                              *-------------------*
B s.pop()->1:                           *----------------------*
```

2.
```
C s.push(1):  *--------------*
B s.push(2):         *-------------*
A s.pop()->1:        *----------*
C s.pop()->2:    *-----------------*
B s.push(3):                  *-----*
C s.push(4):                        *-----------*
A s.pop()->4:                     *--------*
```

3.
```
A s.push(1):  *----------------------*
B s.push(2):                   *----------------------*
A s.top()->2:       *----------------------*
B s.push(3):                  *------*
C s.pop():->2                               *-----*
C s.pop():->3                                     *----------*
A s.pop():->1                                  *-------------*
```

.....

.....

4.
```
A s.push(1):  *----------------*
B s.push(2):     *-------*
C s.pop()->1:       *------------------*
B s.push(3):                  *----*
B s.pop():->2                              *--------*
```

.....

.....

Solution:

(b)
```
A s.push(1):  *--x-----------*
B s.push(2):         *-----x-------*
A s.pop()->2:                                  *---x---------------*
B s.pop()->1:                               *--------------x-------*
```

(c)
```
C s.push(1):  *--x-----------*
B s.push(2):         *---x---------*
A s.pop()->1:        *x----------*
C s.pop()->2:    *--------x---------*
B s.push(3):                  *-x---*
C s.push(4):                           *x----------*
A s.pop()->4:                    *--------x*
```

(d)
```
A s.push(1):  *--------x--------------*
B s.push(2):                   *--x--------------------*
A s.top()->2:       *-------------x---------*
B s.push(3):                  *x-----*
C s.pop():->2                              *--x--*
C s.pop():->3                                        *--x-------*
A s.pop():->1                                     *--------x----*
```

(e)
```
A s.push(1):  *--x-------------*
B s.push(2):     *------x*
C s.pop()->1:       *x-----------------*
B s.push(3):                  *---x*
B s.pop():->2                           *#-------*
```

The operation with # cannot be satisfied.

(f) Was ist der Unterschied zwischen Linearisierbarkeit und sequenzieller Konsistenz? Konstruiere ein Beispiel welches sequenziell konsistens aber nicht linearisierbar ist. (3)

What are the differences between linearizability and sequentical consistency? Construct an example that is sequential consistent but not linearizable.

Solution:

Linearizability requires both the order between event across threads and the order between thread local events to be preserved. Sequential consistency only requires thread local event order to be preserved.

(g)

```
A s.push(x):  *------*
B s.push(y):                  *------*
A s.pop():->y          *----*
```

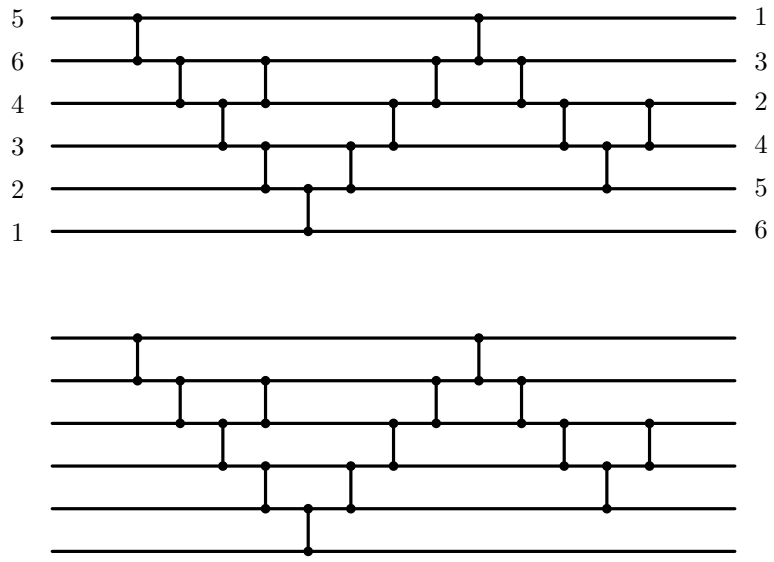
Parallel Algorithms (7 points)

9. Paralleles Sortieren und Sortiernetzwerke *Parallel Sorting and Sorting Networks*

(a) Was ist die beste asymptotische untere Schranke für die Anzahl Vergleichsoperationen, die ein vergleichsbasierter Sortieralgorithmus im schlechtesten Fall durchführen muss, um eine Sequenz von n Elementen zu sortieren? *What is the best asymptotic lower bound on the number of comparisons needed by a comparison sorting algorithm in the worst case for sorting an input sequence of n elements?* (3)

Solution: $\Omega(n \log n)$.

(b) Für den gegebenen Input liefert das folgende Komparatornetzwerk den, offenbar nicht sortierten, Output. Ersetzen Sie die Eingabewerte durch eine Folge von 0 und 1, so dass der resultierende Output auch nicht sortiert ist. Schreiben Sie auf alle Leitungen die entsprechenden Werte für den neuen Input.
Hinweis: Denken Sie an den besprochenen Beweis für das Null-Eins-Prinzip. *For the given input, the following comparator network produces the indicated output, which is obviously not sorted. Provide a sequence of 0s and 1s as an input to the comparator network that is not sorted as well. For the new input, write all values carried by the wires next to them. Hint: Recall the discussed proof for the zero-one-principle.* (4)



Solution: The following mapping delivers an unsorted output: $1 \mapsto 0, 2 \mapsto 0, 3 \mapsto 1, 4 \mapsto 1, 5 \mapsto 1, 6 \mapsto 1$