Vor und Nachname (Druckbuchstaben): _____

Legi Nummer: _____

Unterschrift: _____

252-0024-00L
Parallele Programmierung
ETH/CS: FS 2014
Basisprüfung
Samstag, 09.08.14
120 Minuten

Diese Prüfung enthält 21 Seiten (inklusive diesem Deckblatt) und 8 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 das Seiten verlorengehen oder abgetrennt werden. Vergessen Sie nicht die Prüfung zu unterschreiben!

Als Hilfsmittel sind nur 4 Seiten (2 A4-Blätter) handgeschriebene Notizen erlaubt. Es sind keine zusätzliche Hilfsmittel (Unterlagen, Bücher, Notizen, Taschenrechner) für diese Prüfung zugelassen.

Nehmen Sie sich am Anfang 5 Minuten Zeit, um alle Aufgaben durchzulesen. Während dieser Zeit ist es nicht erlaubt Prüfungsfragen zu beantworten.

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 Algebraische Umformungen erhält keine Punkte; inkorrekte Antworten, mit teilweise richtigen Formeln, Berechnungen und Umformungen können Teilpunkte erhalten.
- Falls mehr Platz benötigt wird, schreiben Sie auf die leeren Seiten der Prüfung oder fordern Sie bei den Assistenten extra Blätter an. Versehen Sie die Aufgabe mit einem klaren Hinweis, falls das der Fall ist. Als 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	17	
2	8	
3	10	
4	8	
5	10	
6	14	
7	22	
8	17	
Total:	106	

1. **Kurzfragen**

   (a) (4 points) Ein single-threaded Programm läuft nicht schnell genug. Wir nehmen an, dass dieses Programm aus einem sequentiellen Teil (60% der Ausführungszeit) und einem parellelisierbaren Teil besteht (40% der Ausführungszeit). Sie haben zwei Optimierungsmöglichkeiten:

   1. Optimierung des sequentiellen Teiles auf die Hälfte der ursprünglichen Ausführungszeit
   2. Weitere Parallelisierung des parallelisierbaren Teiles (unter der Annahme, dass dieser perfekt skaliert)

   Beide Ansätze sind gleich aufwändig. Wann würden Sie die zweite Option bevorzugen? Begründen Sie Ihre Antwort.

   .....

   .....

   .....

   .....

   .....

   .....

   .....

   .....

   .....

   .....

   (b) (2 points) Nehmen Sie an, dass sie eine Anwendung entwickeln müssen, welche alle Anfragen eines Webservers verarbeitet. Dafür können sie entweder Threads oder Tasks verwenden. Welche Methode würden sie wählen und wieso?

   .....

   .....

   .....

   .....

   .....

   .....

(c) (2 points) Nennen sie zwei Ansätze um *Lock Contention* zu reduzieren und erklären Sie knapp die Auswirkungen beider Ansätze.

(d) (3 points) Beschreiben Sie in Ihren eigenen Worten was eine *Barriere (eng.: barrier)* ist und wann Sie dieses Konstrukt benutzen würden. (Verwenden Sie ggf. ein Beispiel).

(e) (3 points) In Java gibt es zwei verschiedene Typen von *thread-safe collections*: *synchronized (Collections.synchronizedMap)* und *concurrent* (z.B. *ConcurrentHashMap*). Erklären sie den Unterschied und geben Sie einen Vorteil von *concurrent* gegenüber *synchronized* an.

(f) (3 points) Vergleichen Sie *shared* und *isolated mutability* miteinander. Welches ist das Hauptunterscheidungskriterium?

.....

.....

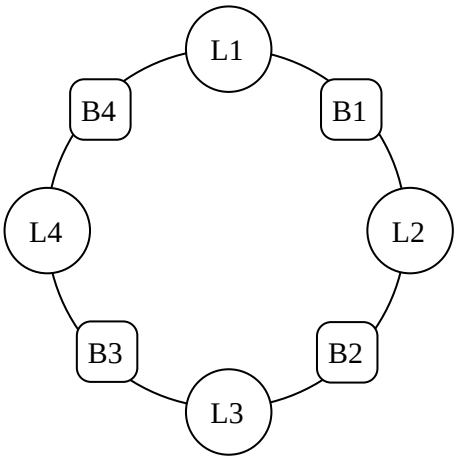
.....

.....

.....

2. (8 points) **Synchronisierung**

Nehmen Sie an Sie haben 4 Lampen (`L`) und 4 Lichtschalter (Buttons) (`B`), die in einem Kreis gemäss der folgenden Abbildung angeordnet sind. Die Lampen sind in der Klasse `Light` implementiert, deren Eigenschaften von einem *intrinsic lock* geschützt sind (wie zum Beispiel für `setColor()`).



Erstellen Sie eine Funktion `tryPressButton()`, der die Funktion nur ausführt wenn **beide** Nachbarn grün (GREEN) sind. Achten Sie auch auf die Hinweise in den Kommentaren im folgenden Quelltext. Andere Threads können auf den Objekten der Klasse `Light` arbeiten (z.B. durch Aufruf von `setColor()`).

```
enum Color { // Different colors
    GREEN,
    RED,
    ...
}

public class Light {
    // for simplicity, fields are public
    public int id;      // light number (1, 2, 3, 4);
    public Color color; // current color;

    // class is protected by the intrinsic lock
    public void synchronized setColor(Color c) {this.color  = c;}

    ...
}

// You can use the following functions:
// get the i light. Assume this function is thread-safe
public static Light getLight(int i)  { ... }
// get the i button. Assume this function is thread-safe
public static Button getButton(int i) { ... }

// You need to complete this function:
public void tryPressButton(i) {

    // Get the ith Button
    // You can use b.press() to press it
    Button b = getButton(i);

    // get the ith Light
    Light li = getLight(i);
    ..........................................................
    ..........................................................
    ..........................................................
    ..........................................................
    ..........................................................
    ..........................................................
    ..........................................................
    ..........................................................
    ..........................................................
    ..........................................................
    ..........................................................
    ..........................................................
    ..........................................................
    ..........................................................
    ..........................................................
    ..........................................................
```

```
        . . . . . .
        . . . . . .
        . . . . . .
        . . . . . .
        . . . . . .
        . . . . . .
        . . . . . .
        . . . . . .
        . . . . . .
        . . . . . .
        . . . . . .
        . . . . . .
        . . . . . .
        . . . . . .
        . . . . . .
}
```

Falls Sie ein Synchronisierungsschema verwendet haben, erklären Sie Ihren Ansatz:

. . . . . .

. . . . . .

. . . . . .

3. **Iterative Berechnung**

(a) (7 points) Nehmen Sie an, dass Sie ein *Array* der Größe **N** sowie einen Algorithmus mit **R** Wiederholungen gegeben haben. Bei jeder dieser Wiederholungen produziert der Algorithmus ein *Array*, welches auf der alten Version des *Arrays* basiert. Jedes Element in dem neuen *Array* basiert **ausschließlich (eng: exclusively)** auf dem alten *Array* und der Position des Elements (Beachten Sie die Funktion `fn` im Pseudocode im folgenden Beispiel).

Pseudocode:

```
for (r=0; r<R; r++) {   // r: repetitions
  for (i=0; i<N; i++) { // i: array element
    A_{r+1}[i] = fn(A_r,i) // compute ith element
  }
}
```

Dieser Algorithmus kann wie folgt parallelisiert werden:

```
for (r=0; r<R; r++) {
  create T threads
  give each thread a part of the output array to compute
  wait untill all threads are finished
}
```

Wobei jeder Thread folgendermaßen implementiert ist:

```
// perform the computation for the given part of the array
for (/* thread's part of the array */) {
  A_{r+1}[i] = fn(A_r,i)
}
```

Die Erzeugung der *Threads* in jedem Schritt ist mit einen Mehraufwand behaftet. Beschreiben Sie die Änderungen, die notwendig wären um eine, weiterhin **korrekte**, Version des Codes zu erhalten, bei der jeder *Thread* nur einmal erzeugt wird. Vervollständigen Sie die folgende Vorlage.

Algorithmus:

```
allocate all arrays
.....................................................
.....................................................
create T threads
.....................................................
.....................................................
give each thread a part of the output array to compute
.....................................................
.....................................................
wait until all threads finish
.....................................................
.....................................................
```

Zuletzt sollen Sie den Code, der von den individuellen Threads ausgeführt wird, implementieren:

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

(b) (3 points) Nehmen Sie an, dass f(A_r, x) Zeit k braucht für $x \in \{1, 2, \ldots, n\}$. Außerdem braucht f($A_r, x = 0$) $20 \times k$ Zeiteinheiten. Würde dies ein Problem verursachen? Begründen Sie Ihre Antwort und beschreiben Sie kurz eine Lösung des Problems wenn es eines gibt.

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

4. **Data und Pipeline Parallelismus** Nehmen Sie an sie haben folgende Funktionen: `f()`, `g()`, `h()`. Diese erwarten ein `Integer`-Argument und geben einen `Integer`-Wert zurück.

   Zum Beispiel:

   ```
   public Integer f(Integer x) {
           // only access x, and no other data
   }
   ```

   Sie möchten nun `f(g(h(x)))` für jedes Element `x` eines großen Arrays berechnen.

   (a) (2 points) Beschreiben Sie in einigen Sätzen wie Sie eine parallele Version dieser Berechnung unter Verwendung eines data parallelen Modells (eng.: *data parallel model*) programmieren würden.

   (b) (2 points) Beschreiben Sie in einigen Sätzen wie Sie diese Berechnung unter Verwendung einer *Pipeline* programmieren würden.

(c) (2 points) Was ist die Maximale Beschleunigung die sie mit der *Pipeline* Version erzielen können.

(d) (2 points) Ist es möglich die *pipeline* und die *data parallele* Versionen zu kombinieren? Konstruieren Sie ein Beispiel.

5. (10 points) **Klammern Matching**

   Diese Aufgabe behandelt das Problem Klammerpaare einander zuzuordnen (eng.: *Matching Parentheses*). Wir wollen die Anzahl der isolierten (eng: unmatched) öffnenden (opening) oder schliessenden (closing) Klammern (parantheses) in einem *String* zählen. Als Vereinfachung repräsentieren wir einen *String* als *Array* mit den Elementen 0 als öffnende und *1* als schliessende Klammer. Die folgenden Beispiele dienen der Veranschaulichung:

String	**Array**	**Result**
(	[0]	open = 1, closed = 0
)(	[1,0]	open = 1, closed = 1
()(	[0,1,0]	open = 1, closed = 0
()()	[0,1,0,1]	open = 0, closed = 0
((())	[0,0,0,0,1,1]	open = 2, closed = 0
))()()(	[1,1,0,1,0,1,0]	open = 1, closed = 2

   Zeigen sie auf wie Sie die Methode `matchParenParallel(int[] array)` implementieren würden. Die Methode nimmt als Argument das Array und führt eine *parallele* Berechnung des Ergebnisses aus. Sie können die sequentielle Implementation des Algorithmus als gegeben annehmen (siehe weiter unten).

```
public static Pair<Integer, Integer> matchParentheses(
    int[] array, int start, int end) {
  int open = 0, closed = 0;
  for (int i = start ; i < end; i++) {
    // (Details of how this is implemented don't matter.)
  }
  return Pair.of(open, closed);
}
```

   Eine Implementation in pseudo-code und/oder schematische Veranschaulichung genügen. Falls Sie Frameworks (z.B. Threads, Java standard library, MapReduce) verwenden, sollten Sie sicher stellen, dass erklärt wird wie Sie diese benutzen würden.

6. Java 8 führt ein neues Framework zum Transformieren von Collections aus Elementen ein. Als Erinnerung sei hier ein Teil der API dazu wiedergegeben:

```
interface Stream<T> {
  /* Returns the count of elements in this stream. */
  long count();

  /* Returns only elements that match the given predicate. */
  Stream<T> filter(Predicate<? super T> predicate);

  /* Transforms a stream by applying the given function to every
     element. */
  <R> Stream<R> map(Function<? super T,? extends R> mapper);
  ...
}
```

(a) (2 points) Erklären Sie einen Vor- und einen Nachteil dieses Design (im Vergleich zu for-loops und Threads.)

.....

.....

.....

.....

.....

.....

(b) (4 points) Eine Bank hat fehlerhafte Bankomaten die manchmal eine Transaktion mehrfach in das Transaction-Log schreiben. Darum hat die Bank eine Routine hinzugefügt, die ein Element löscht, wenn dieses identisch mit dem Nachbarn ist. Dazu wurde die neue Streams API verwendet. Die verwendete Implementation funktioniert gut solange sie sequentiell ausgeführt wird, schlägt aber fehl wenn sie parallel ausgeführt wird. Erklären Sie das Problem bei der parallelen Ausführung in 2-3 Sätzen.

```
List<String> accounts = Arrays.asList(new String[]{ "C3",
   "B2", "B2", "A1", "C3", "D4", "D4", "D4" });
// Expected output: ["C3", "B2", "A1", "C3", "D4"]

String last = null;
List<String> result =
  accounts.parallelStream()
          .filter(x -> {
              if (x.equals(last)) {
                return false; // drop
              } else {
                last = x;
                return true; // keep
              }
          })
          .collect(); // create a collection from the string
```

.....

.....

.....

.....

.....

(c) (8 points) Als Vereinfachung des vorhergehenden Problems nehmen wir nun an, dass wir nur Duplikate erkennen müssen und diese nicht zusätzlich entfernen. Indem Sie das fork-join Paradigma verwenden, schreiben Sie eine Funktion die true zurückgibt, falls Elemente paarweise doppelt vorhanden sind und false, falls keine Elemente paarweise doppelt auftreten. (Antwort in pseudo-code ist erlaubt, Sie dürfen das `spawn` und `sync` keywords aus der Vorlesung verwenden.)

7. **Implementing Locks**

(a) (4 points) Wenn in einer Synchronisierungs-Primitive gewartet wird, gibt es entweder die Möglichkeit zu "spinnen" oder die Ausführung zu unterbrechen. Erklären Sie beide Möglichkeiten und den Performance trade-off zwischen den beiden Möglichkeiten.

(b) (8 points) Sie sind daran beteiligt, einen neuen Computer zu bauen, welcher nur eine einzige Thread-Synchronisations-Primitive in Hardware unterstützt: *compare-and-swap* (CAS).
[NOTE: Java use different naming ... add a note that compare-and-swap and compare-and-set are one and the same]

```
class AtomicBoolean {
  /**
   * Atomically sets the value to the given updated value
   * only if both the current value and the expected values
   * are equal.
   *
   * @param expect the expected value
   * @param update the new value
   * @return True if successful; False return indicates that
   * the actual value was not equal to the expected value.
   */
  boolean compareAndSet(boolean expect, boolean update);

  /* Returns the current value. */
  public final boolean get();

  /* Unconditionally sets to the given value. */
  public final void set(boolean newValue);

  ...
}
```

Ihre Aufgabe ist es, den Code für ein einfaches Spinlock, welches nur diese atomare Operation und nichts anderes braucht, zu implementieren. Sie können annehmen, dass Benutzer Ihres Spinlocks das Lock korrekt verwenden (z.B. keine doppelten Releases oder Releases durch einen Thread, welcher das Lock nicht hält).

```
class SpinLock {
  // Define any new members you need here.
  private final AtomicBoolean locked  = new AtomicBoolean();
  ..........................................................
  ..........................................................

  public void acquire() {
    ........................................................
    ........................................................
    ........................................................
    ........................................................
  }

  public void release() {
    ........................................................
    ........................................................
    ........................................................
```

```
    ..................................................
  }
}
```

(c) (10 points) Nachdem Sie Ihre Implementation gebenchmarkt haben, realisieren sie, dass spinning ineffizient ist, wenn sie lange warten müssen. Die Hardware-Designer teilen Ihnen mit, dass sie zwei weitere Synchronisierungs-Funktionen in Hardware implementieren können um Locks effizienter zu machen:

- `atomicSleep` ist eine atomare Operation, welche folgedem Pseudo-Code entspricht:

```
// Note: SuspendLock is discussed below
void atomicSleep(SuspendLock l, AtomicBoolean val,
   boolean x) {
  atomic { // atomically:
    val.set(x); // set value to x
    suspend(l); // stop (suspend) the thread.
                // the thread is now sleeping on lock l.
  }
}
```

- `wakeUp` ist nicht atomar und entspricht folgendem Pseudo-Code:

```
void wakeUp(SuspendLock l) {
  wakeup_thread(l); // wake up a thread sleeping on lock l
}
```

Ihre zweite Aufgabe ist es nun, das Lock effizienter zu implementieren, indem Sie diese Operationen verwenden. Diese Variante ist ähnlich zu dem Design eines *Mutexes*. Für diese Teilaufgabe gelten die selben Annahmen wie bisher.

```
class SuspendLock {
  // Define any new members you need here.
  ..................................................
  ..................................................
  ..................................................
  ..................................................
  ..................................................
  ..................................................

  public void acquire() {
    ..................................................
    ..................................................
    ..................................................
    ..................................................
    ..................................................
    ..................................................
    ..................................................
    ..................................................
    ..................................................
    ..................................................
  }

  public void release() {
    ..................................................
    ..................................................
    ..................................................
    ..................................................
    ..................................................
    ..................................................
  }
}
```

8. **GPU Programmierung / OpenCL**

(a) (2 points) Vergegenwärtigen Sie sich Flynn's Taxonomie. In welche Kategorie würden Sie (i) sequentielle CPUs und (ii) GPUs einordnen?

.....

.....

.....

(b) (3 points) Vergleichen Sie CPUs und GPUs bezüglich Datendurchsatz (eng.: throughput) und Latenzzeit (eng.: latency). Erklären Sie diese Performanceunterschiede anhand der unterschiedlichen Hardware Architekturen.

.....

.....

.....

.....

(c) (5 points) In OpenCL gibt es mehrere verschiedene Typen von Speicher (eng.: memory types). Bezugnehmend auf untenstehendes Diagramm, erklären Sie wie OpenCL Zugriffsrechte regelt. Ausserdem erklären Sie, welche Auswirkungen der Speicherort auf Zugriffszeit und Latenz von Lese und Schreiboperationen hat.

Hinweis: Gehen Sie in Ihrer Antwort auf folgende Stichworte ein: private, local, global, constant und host memory.

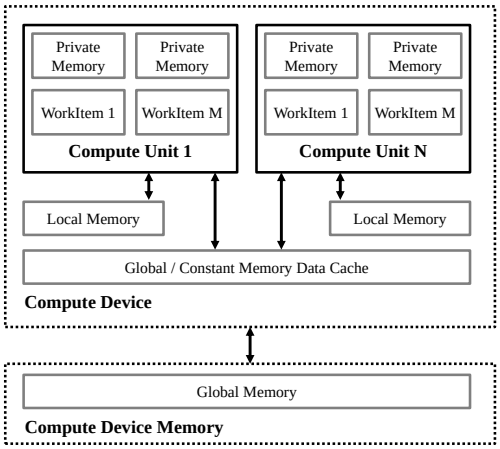


Abbildung 1: GPU Memory Hierarchy

(d) (3 points) Nehmen Sie an, dass Sie einen Algorithmus implementieren, der eine *tree reduction* benötigt. Das input Array ist zu gross für eine einzelne *work-group*. Dabei soll das finale Ergebnis der Reduktion in den globalen Speicher (eng.: *global memory*) geschrieben werden. Wie stellen Sie sicher, dass das korrekte Ergebnis berechnet wird. Sie können annehmen, dass die verwendete Operation Assoziativ und Kommutative ist.

(e) (4 points) Untenstehend finden Sie einen OpenCL Kernel der ein 2D grid von work-groups verwendet um ein 2D Problem zu berechnen (die genaue Natur des Problems ist irrelevant). Der Code liest Daten aus zwei Puffern (eng.: *Buffer*) im globalen Speicher. Desweiteren werden kleinere, lokale 2D Speicherbereiche (eng.: tiles) mit diesen Daten gefüllt. Das Ergebnis wird dann berechnet, indem die Werte aus zwei unterschiedlichen lokalen Speicherbereichen gelesen werden und miteinander addiert werden. Das Ergebnis wird dann in den globalen Speicher zurück geschrieben. Erklären Sie was die Funktion barrier(...) bewirkt und warum dieser Funktionsaufruf nötig ist.

```
__kernel void fill_tiles(__global float* a,
                         __global float* b,
                         __global float* c)
{
  // find our coordinates in the grid
  int row = get_global_id(1);
  int col = get_global_id(0);

  // allocate local memory shared among the workgroup
  __local float aTile[TILE_DIM_Y][TILE_DIM_X];
  __local float bTile[TILE_DIM_Y][TILE_DIM_X];

  // define the coordinates of this workitem thread
  // in the 2D tile
  int y = get_local_id(1);
  int x = get_local_id(0);

  aTile[y][x] = a[row*N + col];
  bTile[y][x] = b[row*N + col];
  barrier(CLK_LOCAL_MEM_FENCE);

  // Note the change in tile location in bTile!
  c[row*N + col] = aTile[x][y] + bTile[y][x];
}
```

.....

.....

.....

.....

.....