Universität Augsburg
University

**Vertiefte Multicore-Programmierung**

Universität Augsburg
Institut für Informatik
Prof. Dr. Theo Ungerer
Dr. Sebastian Weis

27. April 2016

---

# Aufgabenblatt 2

## 1. Aufgabe (1 + 2 + 2 Punkte)

a) Implementieren Sie eine thread-sichere, doppelt verkettete Liste mit folgenden Eigenschaften:

- Die Liste soll beliebige Datentypen speichern können. Nutzen Sie dazu Templates.
- Ihre Implementierung soll objektorientiert und effizient sein.
- Methoden, die unterstützt werden müssen:
  - `void push_back(T e)`: Hängt ein Element am Ende der Liste an.
  - `T at(int index)`: Liefert das Element an der Stelle `index` zurück.
  - `T pop_back()`: Liefert das Element am Ende der Liste zurück und entfernt es.
  - `void print()`: Gibt die Liste auf der Kommandozeile aus.
- Verwenden Sie den selben Mutex für alle Zugriffe auf die Elemente der Liste.

b) Implementieren Sie einen *Readers-Writer-Lock* der den parallelen, lesenden Zugriff erlaubt, aber nur einen einzelnen schreibenden Zugriff. Lesezugriffe sollen dabei priorisiert werden. Erstellen Sie zwei Klassen nach dem RAII-Idiom, analog zur `std::lock_guard`-Klasse, um den RW-Lock entweder für Lese- oder für Schreibzugriff innerhalb eines *scopes* zu belegen. Erweitern Sie ihre verkettete Liste aus Teilaufgabe a) um diese Locks.

c) Modifizieren Sie Teilaufgabe b) zur Steigerung der Performance, indem Sie *fine-grained locking* verwenden. Zugriffe sollen individuell pro Element durch einzelne RW-Locks geschützt werden. Eine Einfüge-Operation benötigt dann die Schreib-Locks des linken und des rechten Elements, um ein neues Element einfügen zu können.

Testen Sie alle Teilaufgaben mit mehreren Threads, unterschiedlichen Zugriffsmustern und treffen Sie Aussagen zur Skalierbarkeit Ihrer Teillösungen.

## 2. Aufgabe (2 + 3 Punkte)

a) Implementieren Sie, analog zum Quicksort-Beispiel aus der Vorlesung, eine sequentiell-rekursive Version des *mergesort*-Sortieralgorithmus[1] mit der Funktionssignatur `template<typename T> std::list<T> mergesort(std::list<T>)`.

b) Parallelisieren Sie Ihre Version des *mergesort*-Sortieralgorithmus aus Teilaufgabe a) mit Hilfe der Klassen `std::future` und `std::async`, indem Sie rekursive Funktionsaufrufe asynchron ausführen lassen. Testen und untersuchen Sie die Skalierbarkeit ihrer Lösung und suchen Sie nach möglichen Optimierungen (z.B. durch die explizite Verwendung der Klassen `std::packaged_task und std::promise`), mit denen sich die Laufzeit verbessern lässt.

## Hinweis

Die Abnahme der Aufgaben soll bis 3. Mai erfolgen.

[1] `https://de.wikipedia.org/wiki/Mergesort`