Aufgabe 4:

In Aufgabe 4 haben wir eine Linked List mit Hand-in-Hand Locking-Strategie implementiert.

Unser struct node sieht folgendermaßen aus:

```
typedef struct __node__
{
    struct __node__ *prev;
    struct __node__ *next;
    pthread_mutex_t lock;
    int *value;
} node;
```

Jede node hat ihr eigenes lock und dieses soll in kritischen sektionen gesperrt werden.

Diese treten auf, wenn neue int-Werte (value) hinzugefügt werden sollen.

```
node *list_add(node *head, int *val)
{
    pthread_mutex_lock(&head->lock);
    if (head->value == NULL)
    {
        head->value = val;
    }
    else if (head->next == NULL)
    {
        head->next = node_create(val);
    }
    else if (*head->value == *val)
    {
        pthread_mutex_unlock(&head->lock);
        return head;
    }
    else if (*head->value < *val)
    {
        head->next = list_add(head->next, val);
    }
    else if (*head->value > *val)
    {
        node *new_head = node_create(val);
        new_head->next = head;
        pthread_mutex_unlock(&head->lock);
        return new_head;
    }
    pthread_mutex_unlock(&head->lock);
    return head;
}
```

Unsere main-funktion nimmt mit getopt() als Argument entgegen:

-n Anzahl adds

-p Anzahl Threads

-F Schreibt die Zeitmessung in eine Datei

Aufgabe 5/6:

In Aufgabe 5 und 6 haben wir uns für einen BinärenSuchBaum entschieden und nutzten folgende Datastruktur: https://www.scaler.com/topics/binary-tree-in-c/.

Unser Baum ist simpel gehalten da wir uns auf die Locks konzentrieren wollen, somit enthalten die Nodes keine Tupel wie es oft üblich ist sondern nur einen Integer Wert, außerdem sind keine Duplikaten erlaubt, sollte es also zu einem Insert eines bereits existierenden Nodes kommen, so bleibt der Baum einfach gleich.

Als Parameter für unsere Programme haben wir:

-p = Anzahl Threads (Standard = 4)

-n = Anzahl Inserts (Standard=1000)

Wir haben uns entschieden die Performance Messung anhand der Inserts zu berechnen, somit haben wir eine **Worker-Funktion** erstellt, welcher von jedem Thread aufgerufen wird und n/threads - Inserts ausführt.

Es wird mithilfe der Funktion : **getTimeOfDay()** die Zeit gemessen bis alle Threads die **Worker**() funktion ausgeführt haben und somit alle Inserts getätigt wurden.

Der unterschied der beiden Implementierungen liegt bei der angewandten Locking-Strategie:

In Aufgabe 5 nutzen wir einen Global-Lock, dieser wird jeweils für einen BS-Baum nur einmal implementiert und umschließt die ganze rekursive InsertR-Funktion, was bedeutet das jeweils nur ein Thread gleichzeitig ein Insert durchführen kann.

```
typedef struct {
    node *root;
    pthread_mutex_t lock;
} binarytree;
```

In Aufgabe 6 haben wir uns für einen Hand-in-Hand Implementation entschieden, bei dieser erhält jede Node jeweils ein LockR und ein LockL für dessen beiden Kinder.

```
struct node {
    int item;
    struct node* left;
    struct node* right;
    pthread_mutex_t lockL;
    pthread_mutex_t lockR;
};
```

Hier sieht auch die InsertR methode deutlich Interessanter aus. Hier umschließt das Lock die Prüfung ob die nächste Node == NULL ist, ist das nämlich der Fall, so wird eine neue Node erstellt und eingefügt. Genau diese Prüfung und Erstellung des Nodes ist in unserem Code die kritische Sektion, daher muss der Rest nicht in den Lock und es erlaubt den Threads gleichzeitig Inserts durchzuführen, sie müssen lediglich auf ein Lock warten falls sie auf die gleiche Node zugreifen wollen.

Messungen

Eine Wichtige Erkenntnis war, dass die Zeit-Messungen von unserem BinaryTreeAdvanced mit der Hand-in-Hand Lock-Strategie sehr variierte. Wir schließen das darauf zurück, dass je nach Reihenfolge in der die Threads ausgeführt werden, der Baum sehr degeneriert sein kann, damit ist die Tiefe der Rekursion beim Insert sehr hoch und außerdem muss dann deutlich öfter auf das Freigeben eines Locks gewartet da oft mehrere Threads auf ein Node zugreifen wollen.

Zum Beispiel haben wir folgende Werte für p = 6, n= 4000 erhalten :

```
6      11.034000 + 11.834000 + 17.888000 + 16.935000 + 20.003000 + 55.565000 + 54.951000 + 12.223000 + 20.466000 + 22.393000
Durchschnitt:   24.3292
```

Wie man sieht ist der Niedrigste Wert: 11,2870 und der höchste 55,565 was ein 5-facher unterschied ist .

Bei der Version mit dem Global Lock kam es zu minimalen Unterschieden in der Zeitmessung. Daher entschieden wir uns für Folgende Diagramme den Durchschnitt von 10 Messungen für die Hand-in-Hand und 5 Messungen für die Global Implementierung zu nehmen

Somit haben wir Folgende Ergebnisse erhalten:

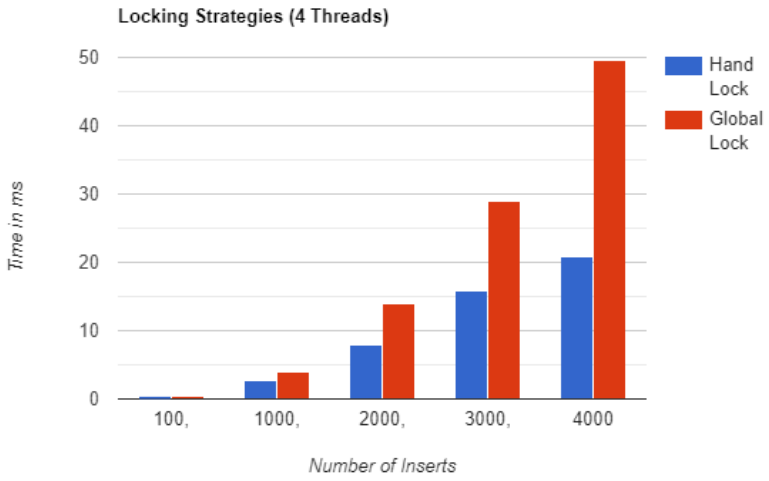

Abb1

Wie man in Abb1. sieht, bietet die Hand-in-Hand implementierung keinen bedeutenden Unterschied bei kleinen Mengen an Inserts, jedoch ist sie deutlich effizienter umso mehr Inserts es gibt.

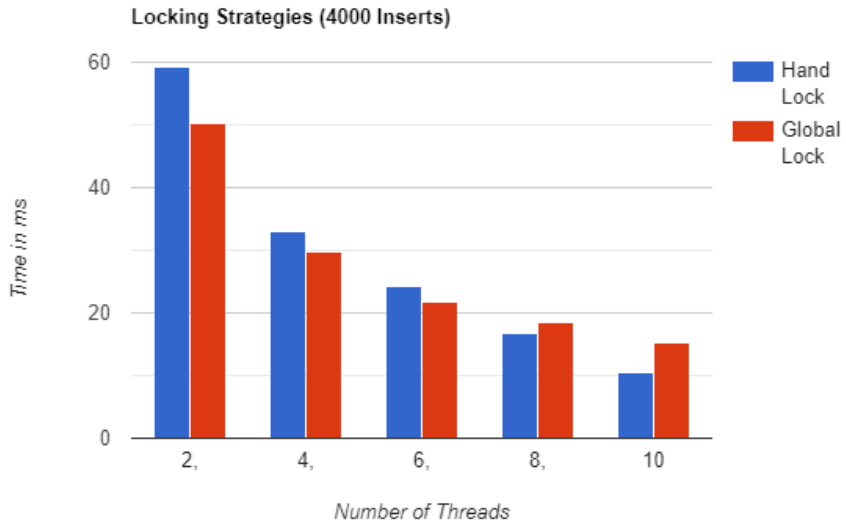

Abb2.

In Abb2 sieht man dass auch die Anzahl an Threads eine Rolle spielt, bei wenigen Threads fällt hier sogar die Hand-in-Hand Methode schlechter aus. Erst bei 8 Threads performt diese besser.

Fazit:

Die Hand-in-Hand Lock-Strategie bietet bei vielen Inserts und vielen Threads einen großen Performance Vorteil, jedoch schwankt dieser Stark. Je nach Anwendung müsste man Abwegen ob die inkonsistente Laufzeit den langfristigen Performance-Vorteil rechtfertigt.

Elias Keller, ele.keller@gmx.de

Robin Stockinger, robin.stockinger@gmx.de