Einführung in die Parallelverarbeitung
Prof. Dr. Claudia Fohry
M.Sc. Jonas Posner

Universität Kassel
Sommersemester 2019
Abgabe am 08.07.2019 um 10 Uhr

Aufgabenblatt APGAS

`groupX.BuddyKGV.java`:

Das *"kleinste gemeinsame Vielfache"* (kgV) von zwei ganzen Zahlen ist in der Mathematik definiert als die kleinste positive natürliche Zahl, die sowohl ein Vielfaches der ersten Zahl als auch ein Vielfaches der zweiten Zahl ist. Beispiel:

Die positiven Vielfachen von 5 sind: $5, 10, 15, 20, 25, 30...$
Die positiven Vielfachen von 3 sind: $3, 6, 9, 12, 15, 18, 21, 24, 27, 30...$
Die gemeinsamen positiven Vielfachen von 5 und 3 sind also $15, 30...$
$kgV(5,3) = 15$

In der nachfolgenden Aufgabe soll das kgV zweier Zahlen mit Hilfe des *"größten gemeinsamen Teiler"* (ggT) berechnet werden. Der ggT soll mit dem euklidischen Algorithmus berechnet werden. Siehe dazu: https://de.wikipedia.org/wiki/Kleinstes_gemeinsames_Vielfaches

Wir nennen zwei Zahlen *Buddies*, wenn der kgV dieser beiden Zahlen eine gewisse Schwelle (nachfolgend $minKgV$ genannt) erreicht bzw. überschreitet. Beispiel:

$minKgV = 10$, dann sind 5 und 3 Buddies, da $kgV(5,3) = 15 >= 10$
$minKgV = 50$, dann sind 5 und 3 **keine** Buddies, da $kgV(5,3) = 15 < 50$

Gegeben sind zwei $n \times n$-Arrays `a` und `b`. Beide enthalten Zufallszahlen aus dem Intervall $[1, m)$. Zwecks Vergleichbarkeit der Ergebnisse müssen die Elemente wie folgt initialisiert werden:

```
Random random = new Random();
for (int i = 0; i < n; i++) {
  for (int j = 0; j < n; j++) {
    random.setSeed(seedA + Integer.parseInt(i + "" + j));
    a[i][j] = random.nextInt(m) + 1;
    random.setSeed(seedB + Integer.parseInt(i + "" + j));
    b[i][j] = random.nextInt(m) + 1;
  }
}
```

Es ist Ihnen überlassen, ob Sie die Initialisierung parallelisieren. Wenn ja, müssen Sie sicherstellen, dass jedem `a[i][j]` und `b[i][j]` der gleiche Wert wie oben zugewiesen wird.

Es soll zu jedem `a[i][j]` genau ein Buddy aus `b` gesucht werden. Nach der Berechnung sollen alle Einträge aus `a`, für die es **keinen** Buddy in `b` gibt, ausgegeben werden. Die Ausgabe soll aus den Koordinaten der Einträgen bestehen und aufsteigend sortiert sein, siehe Beispiel 1. Diese Informationen sollen während der Ausgabe auf Place 0 liegen.

Das Programm muss sich mittels

```
java groupX.BuddyKGV n m seedA seedB minKgv verbose
```

aufrufen lassen. Dabei bezeichnen:

`n`	Arraylänge $n \times n$
`m`	Obergrenze für Zufallszahlen (exklusive m)
`seedA`	Initialwert des Zufallszahlengenerators für Array `a`
`seedB`	Initialwert des Zufallszahlengenerators für Array`b`
`minKgv`	Schwellwert für zwei Buddies
`verbose`	Wenn `true`, dann werden zusätzlich noch `a`, `b`, alle gefundenen Buddy Positionen in `b` und die zugehörigen kgVs ausgegeben werden, siehe Beispiel 2. Diese Informationen müssen auch berechnet werden wenn `verbose = false` ist. Der Speicherort dieser Informationen kann frei gewählt werden. "`-1:-1`" bzw. "`-1`" in der Ausgabe bedeutet, dass kein passender Buddy gefunden werden konnte.

Wenn `minKgv = -1` gesetzt ist, dann soll für jeden Eintrag aus `a` der Eintrag aus `b` mit dem **größten** kgV gefunden werden, siehe Beispiel 3.

Schreiben Sie ein effizientes APGAS-Programm zur Lösung der Aufgabe. Der sequentielle Algorithmus soll nicht optimiert werden. D.h., jeder $kgV(x,y)$ (soweit benötigt) soll mit dem Vorgehen von oben berechnet werden

Das APGAS-Programm soll sowohl place-intern als auch place-übergreifend parallelisiert werden. Es dürfen nur die in der Vorlesung vorgestellten APGAS-Konstrukte verwendet werden (bei Unsicherheit bitte im Forum nachfragen).

Es darf vorausgesetzt werden, dass n ein Vielfaches von $Workerthreadanzahl \cdot Placeanzahl$ ist.

Das Programm sollte es, zumindest weitgehend, vermeiden,

- für eine Zahl `a[i][j]` mehr als einen passenden Buddy zu finden,
- einen $kgV(x,y)$ und einen $ggT(x,y)$ zu berechnen, der für die weitere Berechnung nicht gebraucht wird und
- den gleichen $kgV(x,y)$ mehr als einmal pro Place zu berechnen.

Darüber hinaus gehende Optimierungen des Basisalgorithmus sind erlaubt, aber nicht gefordert. Alle in der Vorlesung behandelten Effizienzaspekte sollten beachtet werden.

Ihr Programm muss die Ausgaben exakt so formulieren wie in den nachfolgenden Beispielen dargestellt.

Beispiel 1:

```
java BuddyKGV 128 1000 123 345 1100 false
there are no buddies for the following elements
[19:59, 22:76, 26:94, 38:106, 45:39, 52:45, 52:74, 56:29, 56:58, 59:47,
114:102, 122:24, 126:113]
```

Beispiel 2:

```
java BuddyKGV 2 10 123 456 20 true
there are no buddies for the following elements
[0:1, 1:1]
a =
[3, 1]
[8, 1]
b =
[9, 4]
[5, 10]
Found buddy positions in b=
[1:1, -1:-1]
[0:0, -1:-1]
Found kgVs=
[30, -1]
[72, -1]
```

Beispiel 3:

```
java BuddyKGV 32 1000 123 345 -1 false
all elements have found buddies
```

Ermitteln Sie den Speedup mit

- 1 Place und $1, 2, 4, 8, 16$ Workerthreads und
- $1, 2, 4$ Places mit jeweils 16 Workerthreads

und den Parametern $n = 2048$, $m = 10000$, $seedA = 123$, $seedB = 456$, $minKgv = 50000$, $verbose = false$. Die Zeitmessung soll nach der APGAS Initialisierung beginnen und dann das ganze restliche Programm umfassen. Geben Sie das Programm inklusive der gemessenen Laufzeiten und errechneten Speedups ab. Die Laufzeiten sollen wie folgt gemessen und ausgegeben werden:

```
long start = System.nanoTime();
....
long end = System.nanoTime();
System.out.println("Process time = " + ((end - start) / 1E9D) + " sec");
```

Abschlussbemerkungen:

Eventuelle Fragen zu den Aufgaben stellen Sie bitte im Diskussionsforum auf der Moodle-Seite der Veranstaltung.

Cluster: Das Programm soll auf dem Cluster der Uni Kassel ausgeführt werden. Auf dem Cluster soll die Partition “exec” benutzt werden, wie in den Beispielskripten in Moodle.

Wettbewerb: Falls es einen klaren Gewinner gibt, so verbessert die Gruppe mit dem schnellsten Programm ihre Abschlussnote um 0.3. Allerdings kann eine Gruppe nur maximal einen Wettbewerbsbonus in die Gesamtveranstaltung (inkl. OpenMP, CUDA) einbringen. Am Wettbewerb nehmen nur Gruppen teil, deren abgegebenes Programm exakt der Spezifikation entspricht und für alle getesteten Eingaben korrekt ist. Die Messungen werden mit 4 Places mit jeweils 16 Workerthreads für mehrere exemplarische Parameterwerte durchgeführt, so dass die gemessenen Laufzeiten mindestens 30 Sekunden betragen.

Abgabe: Die Lösungen von der APGAS und CUDA Aufgabe müssen zusammen bis spätestens 08.07.2019 um 10 Uhr per Email an jonas.posner@uni-kassel.de mit dem Betreff “IntroPV-Abgabe APGAS+CUDA Gruppe X” gesendet werden. Später abgegebene Lösungen werden als nicht abgegeben gewertet. Bitte geben Sie zu jedem Programm die vollständigen Kommandozeilen zur Kompilierung und Ausführung des Programms, sowie die Slurm Skripts zum Starten auf dem Cluster mit ab.