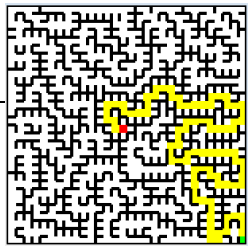

Übungsprojekt Parallele Programmierung: Parallelisierung der Suche in einem Labyrinth

Ausgabedatum: 03.05.2016

Diese Aufgabe muss spätestens bis zum 18.06.2016 in Teams zu je zwei Personen gelöst werden. Die Lösung ist eine Pflichtleistung für das Bestehen des Moduls. Die verbleibenden Übungsstunden behandeln nur noch zur Hälfte die Aufgaben auf den Übungsblättern; in der anderen Hälfte können Sie Fragen stellen zu Ihrer entstehenden Lösung und zu den Parallelisierungstechniken und -mechanismen, die Sie verwenden wollen. Da im weiteren Verlauf der Vorlesung noch neue Mechanismen vorgestellt werden, wird sich auch Ihre Lösung über die folgenden Wochen entwickeln.

Ihre Aufgabe ist die Parallelisierung eines gegebenen Java-Programms, das einen Weg zwischen zwei gegebenen Punkten in einem Labyrinth findet. Falls das Labyrinth nicht größer als 1000x1000 Pixel ist, wird es samt der Lösung grafisch und textuell ausgegeben; falls es größer ist, wird nur die Länge der Lösung und die Laufzeit ausgegeben. Das Programm erzeugt zunächst ein zufälliges Labyrinth vorgegebener Größe (die Größen können als Programmaufrufargumente übergeben werden) und sucht dann einen Weg vom Start zum Ziel; der Startpunkt liegt immer in der Mitte des Labyrinths, und das Ziel ist immer eine zufällig gewählte der vier Ecken. Alternativ zur Erzeugung eines zufälligen Labyrinths kann das Programm auch ein in einem früheren Programmlauf erzeugtes und in einer Datei gespeichertes Labyrinth verwenden, wenn dessen Dateiname als Programmaufrufargument übergeben wird. In beiden Fällen speichert das Programm das verwendete Labyrinth in der Datei `grid.ser`, die Sie z.B. geeignet umbenennen können, um verschiedene Labyrinthe wiederholt zu testen. Das erzeugte Labyrinth hat immer mindestens eine Lösung, in der Regel sogar mehrere, da es Schleifen enthalten kann (je nach Wert der Konstante `CYCLE_CREATION_PROBABILITY`). Das Programm prüft die gefundene Lösung selbsttätig und gibt aus, ob sie tatsächlich korrekt ist. Eine sequentielle Implementierung des Programms finden Sie auf dem Server im Quelltext-Paket `uebung_parallelisierung.sequentiell`.

Sie dürfen die Suchfunktion des Programms beliebig ändern, um es zu parallelisieren, solange eine korrekte Lösung gefunden wird. Es ist nicht nötig, alle Lösungen oder die kürzeste Lösung zu berechnen. Sie dürfen nicht die Erzeugung des Labyrinths und auch nicht das Datenformat von `grid.ser` ändern; um letzteres zu verhindern, testen Sie unbedingt Ihre Parallelisierung mit einem von der sequentiellen Version produzierten `grid.ser`. Es ist nicht erlaubt, die Position des Ziels bei der Suche auszunutzen, etwa um eine "grobe Richtung" zu bestimmen. Jede Zelle des Labyrinths darf **nur ein einziges Mal betreten** werden, unabhängig davon, ob mehrfaches Betreten eine Auswirkung auf das Ergebnis hat oder nicht. Sie dürfen nichts an den Teilen ändern, die auch in der sequentiellen Implementierung benutzt werden (z.B. an den Klassen `Point` oder `Grid`): Ein **Laufzeitgewinn** soll **ausschließlich durch die Parallelisierung** eintreten und nicht durch Optimierungen, die auch im sequentiellen Fall möglich wären. Die Initialisierung und das Herunterfahren der Parallelitäts-Infrastruktur (z.B. Threads, Thread Pools, Actor-Systeme, MPI) soll nicht Teil der Zeitmessung sein.

Der Schlüssel für eine effiziente parallele Suche liegt wahrscheinlich in der richtigen Wahl der Datenstrukturen für den Suchzustand (schon besuchte Zellen und Pfade dorthin). Achten Sie darauf, dass der Aufwand für die Erzeugung nebenläufiger Aktivitäten nicht so groß wird, dass dadurch das parallelisierte Programm wieder langsamer wird. Machen Sie sich auch klar, wo synchronisiert werden muss und wo nicht.

Ihr Programm **muss** mindestens Labyrinthe der Größe 1000x1000 Zellen lösen können, ohne dass bei Standard-JVM-Einstellungen auf den Pool-PCs der Stack- oder Heap-Platz ausgeht. Ihr Programm **muss** tatsächlich nebenläufig sein. Ihr Programm **sollte** bei Verwendung von vier Prozessen bzw. Kernen (z.B. Pool-PCs) schneller als die sequentielle Version sein, dies ist aber keine zwingende Anforderung.

Die besten Teams (etwa zwei bis vier – wie viele genau, entscheide ich nach der Abgabe), d.h. im wesentlich die Teams mit der schnellsten Parallelisierung – falls das Programm keine groben Verstöße gegen die Prinzipien sauberer Programmierung (z.B. Zugriffskonflikte, die nur durch Glück die Korrektheit erhalten) oder gegen die obigen Vorgaben enthält – werden mit einem Notenbonus von 0,3 auf die Modulnote belohnt. Die letzte Entscheidung über den Notenbonus liegt beim Dozenten.

Zur Entscheidung werden die Pool-PCs und die mittlere Laufzeit auf einigen vom Dozenten vorgegebenen Labyrinthen benutzt; manche dieser Labyrinthe enthalten nur eine Lösung, andere enthalten viele. Wenn Sie Ihr Programm vorführen, sollte es im default package bzw. im aktuellen Verzeichnis liegen, weil die vorgegebenen Labyrinthe ohne package-Pfad erzeugt wurden. Es sollte einfach möglich sein, in Ihrem Programm eine gewisse Anzahl von Programmläufen entweder auf dem gleichen Labyrinth (das ist für die Abnahme nötig) oder auf jedesmal neu erzeugten Labyrinthen (das ist für die Entwicklung hilfreich und für die Dokumentation nötig) durchzuführen und die Median-Laufzeit auszugeben.

Falls Sie Ihr Programm auf mehrere Pool-PCs verteilen wollen, können Sie folgendes Namensschema für die PCs verwenden: `pool<Raum>pc<nn>` mit `Raum` ∈ {314, 339, 346, 348} und 01 ≤ `nn` ≤ 15, z.B. `pool339pc09`.

Sie müssen in einer der Übungen spätestens bis zum 16.6. (bitte nicht alle Teams an diesem letzten Tag!) Ihre Lösung vorstellen und den Code samt einer **schriftlichen Entwurfs-Dokumentation** (1 - 2 S.) abgeben. Beides ist Voraussetzung zum Bestehen des Moduls. Die Entwurfsdokumentation soll erklären, wie Sie das Programm parallelisiert haben, z.B. welche parallele Aktivitäten (Threads, Tasks etc.) Sie wofür eingesetzt haben, wann parallele Aktivitäten erzeugt werden und wann sie beendet werden, welche Daten lokal bleiben und welche gemeinsam genutzt werden, welche Daten zwischen Aktivitäten kopiert werden, wie synchronisiert wird, was passiert, wenn eine Lösung gefunden wird, wie Sie ggf. Ihr Programm auf verschiedene Rechner verteilt haben, welche Möglichkeiten zur Parametrisierung es ggf. enthält und was Sie sonst noch für erwähnenswert halten. Auch zunächst untersuchte aber dann verworfene Alternativen und ihre Bewertung können Sie beschreiben. Nennen Sie schließlich auch die Laufzeit Ihrer Lösung für ein 1000x1000- und für ein 5000x5000-Zellen-Labyrinth, jeweils als Median aus jeweils 10 unterschiedlichen Labyrinthen jeweils im Vergleich zur Laufzeit der sequentiellen Lösung (jeweils am selben Labyrinth gemessen) auf demselben Rechner, mit Angabe, wie viele Kerne der Rechner hatte, falls es kein Pool-PC ist (die Pool-PCs haben vier Kerne).