1 Lernziele

Allgemein Es soll eine Problemstellung aus dem Bereich des parallelen Rechnens eigenständig in einer Gruppe bearbeitet werden. Dazu ist eine Einarbeitung in das Themengebiet und das Erstellen eines Projektplans notwendig. Die entwickelten Lösungen sollen implementiert und in einem konkreten Anwendungsfall analysiert werden. Dabei ist das Problem mit Fachwissen unter Anwendung geeigneter Methoden und Verfahren entsprechend des aktuellen Stands der Technik zu lösen, das Projekt zu planen, durchzuführen, abzuschließen und zu dokumentieren.

Dieses Projekt sollte von drei bis fünf Studierenden bearbeitet werden. Eine Kombination mit dem Projekt „Routenplanung auf großem Kartenmaterial“ ist möglich.

Speziell Ziel des Projekts ist das Kennenlernen sowie die Implementierung und Analyse paralleler Algorithmen zur Lösung des Single-Source-Shortest-Path-Problems (SSSP) in C++. Zunächst sind verschiedene parallele SSSP-Algorithmen (wie Delta-Stepping[1], Crauser-Kainer-Ansatz[2], Radius-Stepping[3]) zu recherchieren. Dann wählen Sie mindestens zwei unterschiedliche Ansätze aus und implementieren diese in C++ unter Verwendung geeigneter Bibliotheken wie OpenMP[4], MPI[5], TBB[6] und OpenCL[7]. Schließlich erstellen Sie eine Testumgebung mit zufällig generierten und realen Graphen und vergleichen die Algorithmen anhand definierter Metriken wie Laufzeit, Skalierung und Ressourcenverbrauch.

2 Projekt

Zunächst ist der sequentielle Dijkstra-Algorithmus zu implementieren, der als Grundlage für die Analyse der Effizienz und des Speedups dient. Anschließend sind verschiedene parallele SSSP-Algorithmen zu recherchieren, bevor mindestens zwei der Algorithmen implementiert werden. Je nach Projektgröße können mehrere Bibliotheken wie OpenMP oder MPI genutzt werden, um bspw. die Skalierbarkeit und Komplexität der Programmierung zu untersuchen.

Dieses Projekt sollte von drei bis fünf Studierenden bearbeitet werden.

[1] Δ-Stepping: A Parallel Single Source Shortest Path Algorithm. U. Meyer, P. Sanders. `https://ae.iti.kit.edu/documents/people/sanders/papers/wmain.pdf`

[2] More Parallelism in Dijkstra's Single-Source Shortest Path Algorithm. M. Kainer, J.L. Träff. `https://arxiv.org/pdf/1903.12085`

[3] Parallel Shortest-Paths Using Radius Stepping. G.E. Blelloch, Y. Gu, Y. Sun, K. Tangwongsan. `https://arxiv.org/pdf/1602.03881`

[4] `https://de.wikipedia.org/wiki/OpenMP`

[5] `https://de.wikipedia.org/wiki/Message_Passing_Interface`

[6] `https://de.wikipedia.org/wiki/Threading_Building_Blocks`

[7] `https://de.wikipedia.org/wiki/OpenCL`

2.1 Delta-Stepping-Algorithmus

Im Gegensatz zu Dijkstras Algorithmus gruppiert der Delta-Stepping-Algorithmus[8] die Knoten mit ähnlichen Distanzen in „Buckets“ (Behältern) mit einer Breite Δ. Dies ermöglicht eine Parallelisierung, da Knoten innerhalb desselben Buckets gleichzeitig bearbeitet werden können, was zu einer signifikanten Beschleunigung gegenüber rein sequentiellen Ansätzen führen kann. Der Parameter Δ ist dabei entscheidend für die Leistung, da er die Balance zwischen Parallelität und Arbeitseffizienz bestimmt.

Der Algorithmus verwendet eine Reihe von Buckets, die Knoten mit Distanzen innerhalb eines bestimmten Intervalls $[i\Delta, (i+1)\Delta)$ speichern. Kanten mit Gewichten kleiner oder gleich Δ (leichte Kanten) werden bevorzugt behandelt, da sie oft zu kürzeren Pfaden führen und eine engere Korrektur der Distanzen erlauben. Wie Dijkstra korrigiert der Delta-Stepping-Algorithmus die Distanzen von Knoten. Er ist ein „Label-korrigierender“ Algorithmus, was bedeutet, dass die vorläufigen Distanzen mehrmals „relaxiert“ (verbessert) werden können, bevor sie endgültig sind.

2.2 Radius-Stepping-Algorithmus

Der Radius-Stepping-Algorithmus[9] ist besonders für große Graphen nützlich. Er verbessert den einfacheren Delta-Stepping-Algorithmus, indem er in jeder Stufe einen dynamischen statt eines festen Radius (Schrittweite) auswählt, um die Anzahl der erforderlichen Teilschritte zu begrenzen, und er ordnet Knoten innerhalb eines „Radius“ in zunehmender Entfernung von der Quelle an. Dieser Ansatz ermöglicht bessere theoretische Grenzen für die Anzahl der Schritte und die parallele Ausführung, indem die Knoten in Stapeln angeordnet werden.

Der Radius wird auf der Grundlage des minimalen vorläufigen Abstands zwischen den Knotenpunkten in der „Grenze“ (der Menge der kürzlich festgelegten Knotenpunkte) ausgewählt. Nach der Auswahl des neuen Radius führt der Algorithmus Bellman-Ford-ähnliche Teilschritte aus, um alle Knoten innerhalb des neuen Radius zu setzen. Diese Teilschritte werden parallel ausgeführt. Um gute Leistungsgrenzen zu erreichen, kann der Graph vorverarbeitet werden, um eine bestimmte Eigenschaft zu erhalten, z.B. einen (k, ρ)-Graphen[10], der dabei hilft, die Anzahl der Teilschritte pro Runde zu begrenzen.

2.3 Testumgebung

Um die parallelen Algorithmen bewerten zu können, muss eine Testumgebung erstellt werden, mit der gezielt verschiedene Einflussgrößen auf die Laufzeit und Effizienz analysiert werden können.

- **Zufallsgraphen** Erzeugen Sie Graphen mit variabler Knotenzahl und zufällig verteilten Kanten, um Skalierungsverhalten zu prüfen. Hier kann man sich an dem Graph500-Benchmark[11] orientieren.
- **Sparse Graphs und Dense Graphs** Testen Sie dünn und dicht besetzte Graphen, weil sich die Algorithmen ggf. unterschiedlich verhalten.

[8] `https://en.wikipedia.org/wiki/Parallel_single-source_shortest_path_algorithm`

[9] `https://en.wikipedia.org/wiki/Parallel_single-source_shortest_path_algorithm`

[10] Insights into (k, ρ)-shortcutting algorithms. A. Leonhardt, U. Meyer, M. Penschuck. `https://arxiv.org/pdf/2402.07771`

[11] `https://graph500.org`

- **Realwelt-Graphen** Falls Sie der Meinung sind, dass das Projekt mehr Anwendungsbezug benötigt, dann verwenden Sie öffentlich verfügbare Netzwerke wie die Open-Street-Map-Daten. Dieses Projekt kann mit dem Projekt „Routenplanung auf großem Kartenmaterial“ kombiniert werden.
- **Worst-Case-Graphen** Graphen mit gezielt schwierigen Strukturen (lange Ketten, große Cluster usw.), um Grenzfälle und Problemstellungen zu identifizieren.

2.4 Zentrale Metriken zur Bewertung

Die wesentliche Metrik ist natürlich die Laufzeit, da wir parallele Algorithmen vor allem dann einsetzen, wenn die sequentiellen Algorithmen zu langsam sind. Aber wir wollen auch andere Metriken betrachten.

- **Speedup und Skalierung** Der Speedup $S = {}^{T_{\text{seriell}}}/_{T_{\text{parallel}}}$ zeigt, wie effektiv die Parallelisierung ist und zeigt das Skalierungsverhalten bei wachsender Thread-/Prozessanzahl.
- **Ressourcenauslastung (CPU, RAM)** Messen Sie die Auslastung von Prozessor und Hauptspeicher während der Tests.

Dieses Projekt sollte von drei bis fünf Studierenden bearbeitet werden.