Paralellisierungsschema partdiff

30. November 2019

1 Datenaufteilung

Hier biete sich die Aufteilung der Daten in Zeilen an, da der zeilenweise Zugriff auf Matrizen, sofern die Art der Speicherung gleichbehalten wird, schneller ist als die spaltenweise oder elementweiser Zugriff. Dabei sollten die Prozesse jeweils $\frac{N}{n}$ Zeilen bekommen, wobei n die Anzahl der Prozesse beschreibt. Der letzte Prozess kann dabei bis zu n - 1 zusätzlichen Zeilen mehr an Arbeit bekommen, oder man verteilt die jeweils noch auf die andere Prozesse, sodass der Unterschied in den Anzahl der Zeilen bei höstens 1 liegt. Die Datenaufteilung kann schon bei der Generierung der Daten in der Methode initMatrices geschehen, da die Elemente unabhängig voneinander berechnet werden können.

2 Parallelisierungschema Jacobi

Da beim Jacobi-Verfahren von einer Matrix gelesen und in der anderen geschrieben wird, kann jede Zeile eines Prozesses (sofern sie nicht an der Grenze zwischen zwei Prozessen liegen) beliebig verarbeitet werden. In Figur 1 ist gut zu sehen, das im Grenzbereich der Nachfolger für die erste Zeile auf die letzte Zeile des Vorgängers warten muss. Da die Bearbeitung der Zeilen meist linear erfolgt, würde die letzte Zeile auch zuletzt berechnet werden und am Schluss jeder Iteration an den Nachfolger geschickt werden. Dies kann beschleunigt werden, indem die Zeilen die verschickt werden müssen, zuerst verarbeitet und die Resultate asynchron verschickt werden. Darauffolgend werden die Zeilen verarbeitet, die unabhängig von anderen Prozessblöcken sind, gefolgt von den Zeilen, für die auf die Zeilen von den Vorgänger gewartet werden müssen. Der Nachrichtenaustausch sollte für eine Zeile geschehen und nicht für die einzelnen Elemente, höchstens Blöcke, um den Overhead der IPC so gering wie möglich zu halten.

3 Parallelisierungsschema Gauß-Seidel

Die Parallelisierung beim Gauß-Seidel-Verfahren gestaltet sich schwieriger als das Jacobi-Verfahren, da in dieselbe Matrix geschrieben und gelesen wird. Daher wird für jede Berechnung eines Elementes, außer der ersten in der ersten Iteration, immer Werte der vorherigen Iteration benötigt. Wenn die Matrix von oben nach unten und von links nach rechts verarbeitet wird, so werden nach Figur 2 Werte der vorherigen Zeile oder Spalte benötigt. Daraus kann man ableiten, das ein Element nur berechnet werden kann, wenn alle vorigen Elemente der Spalte und Zeile in dieser Iteration verarbeitet wurden. Man kann anhand der Figuren 4 und 3 erkennen, das sich eine spaltenweise Verarbeitungsweise hier eher anbietet, als eine zeilenweise Verarbeitung. Dies kann jedoch die Zugriffszeiten innerhalb eines Prozesses auf die Daten, durch nicht-optimale Nutzung des Caches, erhöhen. So wird in Prozess 1 in Spalte j von Zeile 1 bis zum Ende des Blocks die Werte berechnet, das Resultat an Prozess 2 geschickt, welches dann Spalte j selbst berechnen kann, während Prozess 1 dann die Spalte j + 1 verarbeitet.

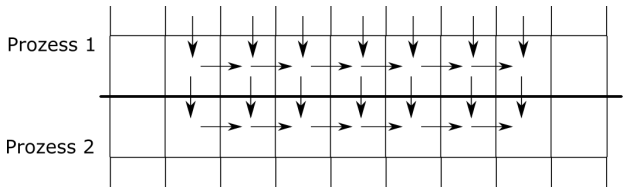

Abbildung 1: Abhängigkeiten für Berechnungen einzelner Zellen von Elementen an der Grenze der Blöcke einzelner Prozesse. Ein Block kann erst berechnet werden, wenn die Blöcke von den eingehenden Pfeile berechnet wurden, bzw. die Daten der vorherigen Iteration schon vorhanden sind. Die erste Spalte und Zeile wird nie direkt berechnet, da nicht über die Matrixgrenzen gerechnet wird/werden kann.

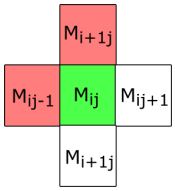

Abbildung 2: Berechnungsstern für das Gauß-Seidel-Verfahren für das Element der Matrix M in der i-ten Zeile und j-ten Spalte. Das grüne Feld soll berechnet werden. Die roten Felder müssen schon in der aktuellen Iteration berechnet sein, sofern von oben nach unten und von links nach rechts die Matrix M verarbeitet wird.

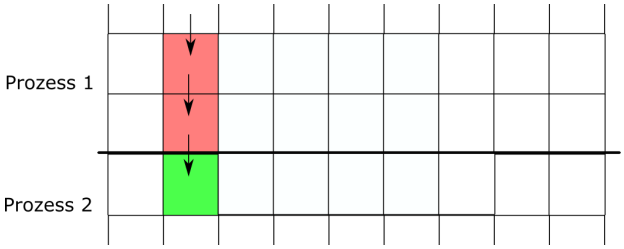

Abbildung 3: Felder von dem ein Element am linken Rand der zu berechnenden Elemente der Matrix M abhängt. Das zu berechnende Feld ist in grün, die Felder von dem es abhängt in rot gefärbt.

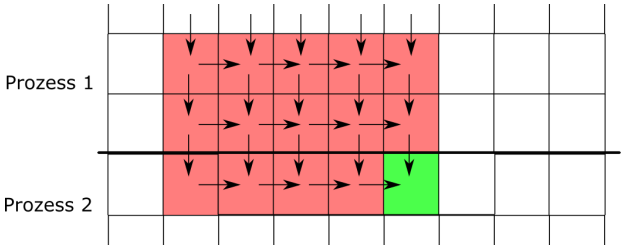

Abbildung 4: Felder von dem ein zufälliges Element am Rand des Prozessblocks abhängt. Das zu berechnende Feld ist in grün, die Felder von dem es abhängt in rot gefärbt.

4 Abbruchproblematik

Ein Prozess sollte höchstens in der Iteration um eins langsamer als der Vorgänger und eins weiter als der Nachfolger sein.

4.1 Iteration

Jeder Prozess kann sich unabhängig vom Verfahren merken, in welcher Iteration er sich gerade befindet und dementsprechend abbrechen.

4.2 Genauigkeit

Jeder Prozess sollte für sein Bereich ein Maxresiduum für seine Iteration berechnen (und die letzte dazu auch speichern, da er eventuell schon um eins weiter ist als gefordert). Diese Werte können dann an alle Prozesse verschickt und von allen ausgewertet werden oder ein spezieller Prozess, z.B. der letzte oder erste bekommt die Werte zugeschickt, wertet sie aus und signalisiert den anderen Prozessen über das Ergebnis mit z.B. einem Broadcast. Dies kann asynchron geschehen um nicht zu blockieren, was aber dann zu mehr Rechnungen führt als nötig, oder synchron, was durch die blockende Natur das Programm verlangsamen kann.