Aufgabe 3: Parallelisierung mit MPI - Schema

Es soll das Partdiff-Programm parallelisiert werden. Generell gilt, dass für die Berechnung eines Feldes der Matrix das Feld selber, sowie sein oberer, unterer, linker und rechter Nachbar benötigt werden. Für einen Prozess müssen diese Werte also immer bekannt sein, wenn ein Feld berechnet werden soll.

Jacobi-Verfahren

Eine geeignete Aufteilung der Matrix lässt sich durch eine Verteilung an die Prozesse in Zeilenblöcken erreichen. Damit die Bedingung, immer alle Nachbarn zu kennen, für die Prozesse erfüllt werden kann, muss jeder Prozess zusätzlich zu dem von ihm zu berechnenden Matrix-Zeilen-Block auch alle Außenränder dieses Blocks kennen, also die vorige sowie die nächste Zeile der Matrix, und den relevanten Abschnitt der linken und rechten Außenspalten.

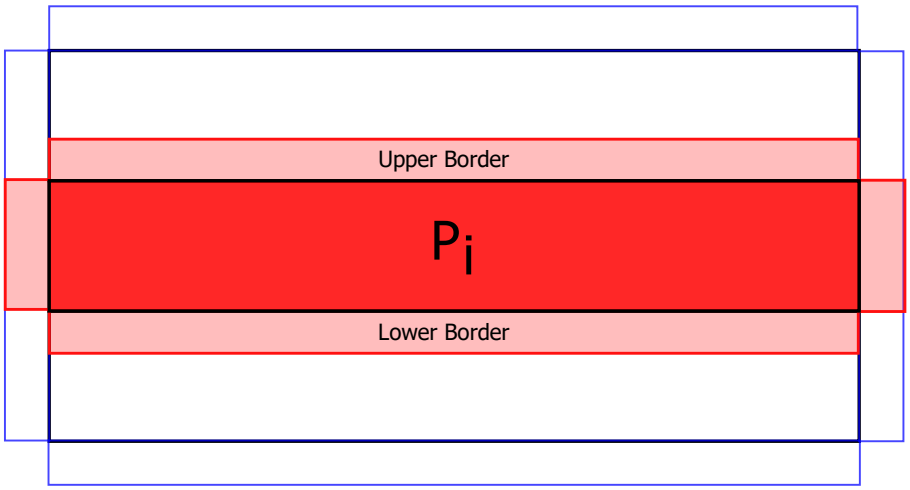

Figure 1: Die schwarzen Blöcke zeigen die Speicherblöcke mehrerer Prozesse, in blau eingezeichnet sind die konstanten Randfelder. Rot markiert ist der Speicherbereich eines Prozesses. Er muss alle benachbarten Zeilen kennen, also die Zeilen ober- und unterhalb sowie die passenden Randabschnitte.

Ein Prozess beim Jacobi-Verfahren kennt die Matrixeinträge in seinem eigenen Speicherbereich sowie die erwähnten umliegenden Zeilen (rot markiert). Der Prozess kann nun alle Einträge eines Iterationsschritts für seinen eigenen Matrixabschnitt berechnen (dunkelrot markiert). An dieser Stelle wird die Abbruchbedingung überprüft, für die Iterationszahl individuell (da sie ohnehin zu diesem Zeitpunkt für alle Prozesse gleich ist) und für die Genaugigkeit durch finden des Matrixweiten maximalen Residuums (mittels Reduce). Wenn keine Abbruchbedingung erreicht ist, wird fortgesetzt:

Der Prozess erhält von seinem Vorgängerprozess (rank-1, der Speicherbereich ist der oberhalb unseres Prozesses) dessen letzte Zeile (für unseren Prozess die Daten in “Upper Border”. Im Gegenzug gibt der betrachtete Prozess seine oberste Zeile an den Vorgänger weiter. Anschließend passiert ein gleichartier Tausch mit dem Nachfolgerprozess (rank+1, weiter unten) um hier ebenfalls “Lower Border” neu zu besetzen und dem Nachfolger die benötigten Daten zu geben. Nun kann ein neuer Iterationsschritt begonnen werden, der Prozess beginnt von vorne.

Hier ist gut zu erkennen, was der Vorteil des Jacobi-Verfahren ist. Es erfolgt eine Kommunikation nach Abschluss einer kompletten Iteration, welche an sich ohne Kommunikation abläuft. Es müssen pro Prozess zwei Matrixzeilen übergeben werden. Eine Beschleunigung könnte durch das Verwenden von nicht sperrendem Senden erreicht werden: Ein Prozess versendet seine fertigen Randfelder und beginnt, ohne seine neuen Ränder von den Nachbarprozessen erhalten zu haben,

bereits mit der Berechnung aller Einträge, die er ohne “Fremddaten” berechnen kann. Erst wenn er damit fertig ist, wartet er auf die neuen “Randdaten” und berechnet die letzten fehlenden Felder. Hierdurch ist auch eine Parallelisierung über Prozessgrenzen hinweg möglich. Dies muss aber bei den Abbruchbedingungen beachtet werden, mehr dazu siehe unten.

Gauß-Seidel-Verfahren

Für das Gauß-Seidel-Verfahren lässt sich keine derart einfache Methode finden. Unter der Annahme der gleichen Speicheraufteilung lässt sich ein Problem erkennen. Um seine Matrixeinträge berechnen zu können, benötigt ein Prozess die fertig berechneten Werte des Prozesses oberhalb (da der linke und obere Wert nicht, wie im Jacobi-Verfahren, “alte” Werte sind, sondern die im gleichen Iterationsschritt berechneten “neuen” Werte). Aus diesem Grund kann ein Prozess erst dann beginnen zu rechnen, wenn der vorige seine Rechnung abgeschlossen hat. Dies ist jedoch identisch mit einem sequentiellen Programm und ermöglicht so keine Geschwindigkeitssteigerung. Was ist also möglich?

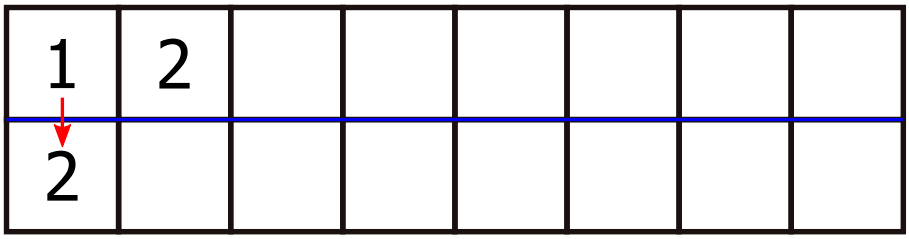

Figure 2: Betrachtet wird die Zeilengrenze zwischen zwei Speicherabschnitten (durch blaune Linie getrennt). Die Zahlen bezeichnen Momente auf einer arbiträren Zeitskala.

Es wird eine Grenze zwischen zwei Speicherabschnitten betrachtet. Wenn der obere Prozess die erste Stelle seiner Zeile zum Zeitschritt 1 berechnet hat, kann er diese Zahl an den unteren Prozess schicken. Dieser hat nun alle Informationen die er für die Berechnung seines ersten Eintrages benötigt. Er beginnt diesen zu rechnen, während der obere Prozess bereits im zweiten Feld ist. Es ist also möglich, zwei aufeinanderfolgende Zeilen zu parallelisieren. Es zeigt sich nun, dass eine andere Form der Speicheraufteilung sinnvoll wäre: Jeder Prozess erhält nur eine Zeile! Hierdurch können mehrere aufeinander folgende Zeilen parallel kaskadenartig durchlaufen werden und eine Parallelisierung ist möglich. Diese ist allerdings begrenzt, wenn die Anzahl Prozesse größer ist als die Anzahl an Matrixeinträgen pro Zeile kann ein zusätzlicher Prozess nichts mehr beitragen. Wenn eine solche Kaskade durchlaufen ist, kann die nächste gestartet werden. So wäre dann die ganze Matrix in Blöcke eingeteilt, in der jede Zeile von einem einzelnen Prozess verwaltet wird.

Zur Laufzeit würde ein Prozess die erforderlichen Daten für sein aktuelles Feld von oberhalb erhalten. Er berechnet das entsprechende Feld, gibt den Wert nach unten hin weiter und geht wieder in den Wartezustand, bis er von oben den nächsten Wert erhält. Falls das Ende einer Zeile erreiche ist, wird das ganze analog weiter unten in der Matrix fortgeführt. Erst nach Abschluss der letzten Zeile durch den letzten Prozess wird die Abbruchbedingung überprüft und gegebenenfalls eine neue Iteration gestartet.

Zusammengefasst unterscheidet sich dieses Vorgehen vom Jacobi-Verfahren dadurch, dass nicht nur einmal am Ende einer Iteration Nachrichtenaustausch passiert, sondern permanent während jeder Iteration.

Es gibt noch eine andere Möglichkeit der Parallelisierung des Gauß-Seidel-Verfahrens. Ein einzelner Prozess beginnt sich, wie beim sequentiellen Programm, durch die Matrix zu arbeiten. Nachdem dieser die zweite Zeile erreicht hat (und die von ihm berechneten Werte der ersten Zeile verwendet hat), übergibt er diese an den nächsten Prozess. Dieser beginnt wieder in der ersten Zeile und rastert hinter dem ersten Prozess her durch die Matrix. Die Parllelisierung erfolgt

hier nicht innerhalb einer Iteration, sondern stattdessen dadurch, dass mehrere Iterationsschritte gleichzeitig berechnet werden können. Hierbei muss jeder Prozess einen Bereuch von zwei drei Zeilen im Speicher haben (die eigene sowie die darüber und darunter. Wenn ein Prozess das Ende der Matrix erreicht hat, springt er wieder an das obere Ende und rechnet weiter. In der folgenden Grafik ist dieses Prinzip verdeutlicht:

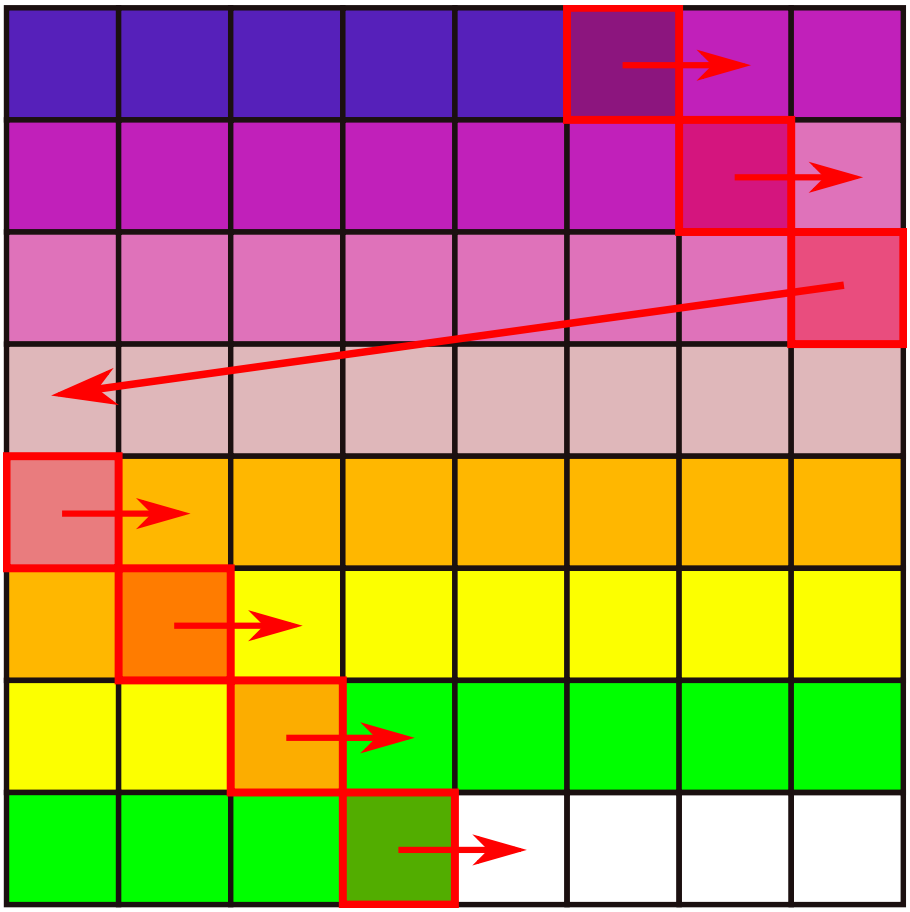

Figure 3: Betrachtet werden einzelne Matrixeinträge. Rot markiert sind die Prozesse, welche durch die Matrix rastern (die Pfeile markieren das nächste Feld). Die Farben der Felder von Blau zu Grün zeigen an, in welchem Iterationsschritt sich die jeweiligen Matrixeinträge befinden. Man erkennt, dass die Matrix sich hierbei je nach Ort in sehr unterschiedlichen Iterationsschritten befindet!

Wie bei dem vorigen Gauß-Seidel-Verfahren muss auch hier konstant mit dem Vorgänger- und Nachfolgeprozess kommuniziert werden. Außerdem gibt es auch hier einen sehr verzahnten Durchlauf, kein Prozess kann voranschreiten ohne Daten vom Prozess unter ihm zu empfangen (und ohne seinerseits Daten an den Prozess über ihm zu geben).

Abbruchbedingungen

Abbruch nach Iterationszahl

Diese Abbruchbedingung stellt kein großes Problem dar. Im Jacobi-Verfahren genügt es, wenn alle Prozesse mitzählen, wie viele Iterationen bereits geschehen sind. Nach Abschluss der letzten Iteration kann die Ausgabe der Ergebnisse erfolgen (die dann von einem Prozess gesammelt, sortiert und ausgegeben wird). Wenn ein System verwendet wird, in dem auch über Iterationsgrenzen

hinweg parallelisiert wird, kann es hier passieren, dass ein Prozess fertig ist, während die ihn Umgebenden noch laufen. Diese befinden sich dann aber auch schon in der letzten Iteration (da sie die Randdaten der vorletzten weitergegeben haben müssen damit der mittlere Prozess fertig werden kann).

Ähnlich ist es beim Gauß-Seidel-Verfahren, auch hier kann jeder Prozess wissen, in welcher Iteration er sich befindet und nach Abschluss seiner Berechnung die Ergebnisse verschicken. Da die Verzahnung zwischen den Prozessen hier so eng ist, werden die Prozesse nahezu zeitgleich fertig werden (der Reihe nach von oben nach unten).

Für die andere vorgeschlagene Parallelisierung des Gauß-Seidel-Verfahrens muss ebenfalls jeder Prozess wissen, mit welchem Iterationsschritt er beginnt, wenn er am oberen Ende der Matrix neu startet. Wenn dies der letzte ist, gibt er die Daten nicht an einen nachfolgenden Prozess ab, sondern gibt sie zur Ausgabe weiter, nachdem er sie fertig berechnet hat. Außerdem muss er allen anderen Prozessen signalisieren, nicht wieder durch die Matrix zu rastern. Wenn ein Prozess fertig ist, gibt es also unter Umständen noch andere, die die letzten paar Iterationen abschließen.

Abbruch nach Genauigkeit

Der Abbruch nach Genauigkeit ist problematischer. Im Jacobi-Verfahren wäre dies kein Problem, wenn alle Prozesse nach Ende einer Iteration warten würden, um dann mittels Reduce aus den individuellen MaxResiduums ein gemeinsames zu errechnen, dieses zu prüfen und anschließend entweder abzubrechen oder weiter zu rechnen. Wenn jedoch eine iterationsübergreifende Parallelisierung gewünscht ist, muss dies berücksichtigt werden. Das kombinierte MaxResiduum kann vom letzten Prozess berechnet werden, der mit einer Iteration fertig wird (da erst dann alle einzelnen MaxResiduums da sind). Er müsste dann die anderen Prozesse "zurückpfeiffen", die schon weitergerechnet haben. Es ist also wichtig, dass die anderen Prozesss in der Lage sind, zu dem "Speicherpunkt" zwischen den Iterationen zurückzukehren.

Beim simplen Gauß-Seidel-Verfahren kann es keine Verschiebung geben, da die Prozesse so eng verzahnt sind. Wenn die zweite Methode verwendet wird, muss ein jeder Prozess am Ende der Matrix die Abbruchbedingung prüfen. Wenn sie erreicht ist, muss, analog zum Jacobi-Verfahren, die bereits begonnen Arbeit der ihm Nachfolgenden Prozesse rückgängig gemacht werden. Dies verlangt, dass die "alten" Daten eines jeden Prozesses irgendwo gespeichert bleiben, bis dieser das Ende der Matrix erreicht.