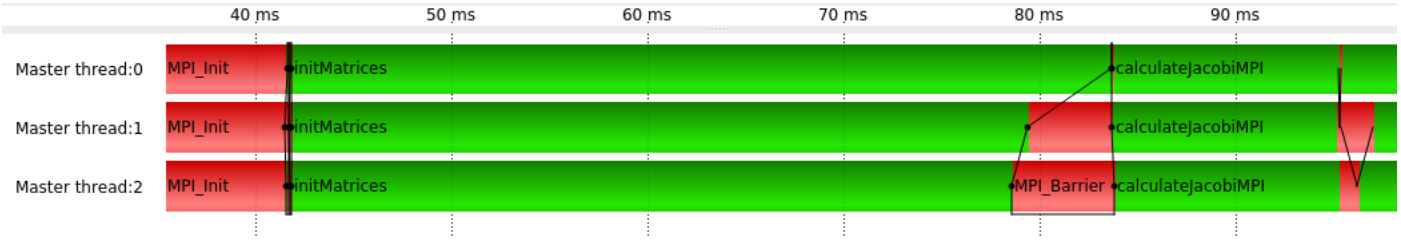
HLR Blatt 11 - Visualisierung

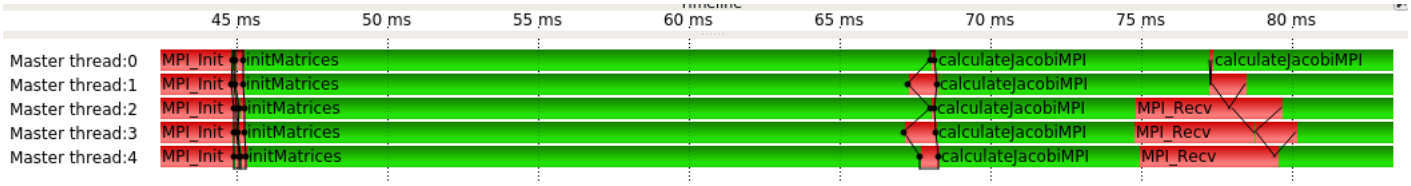
January 27, 2019

1 Jacobi-Verfahren

1.1 Startphase



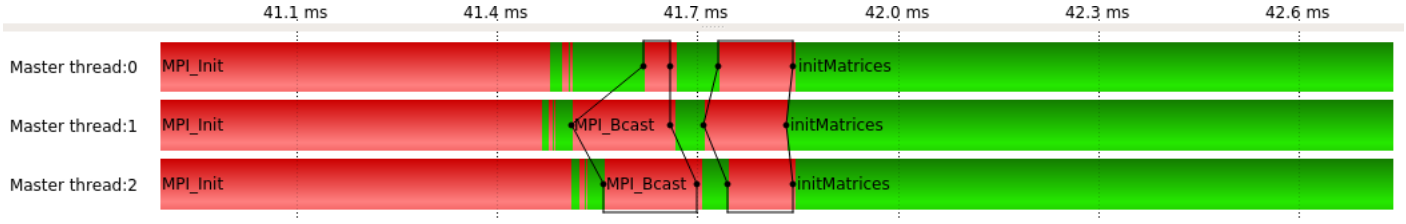
(a) 3 Prozesse, 2 Knoten



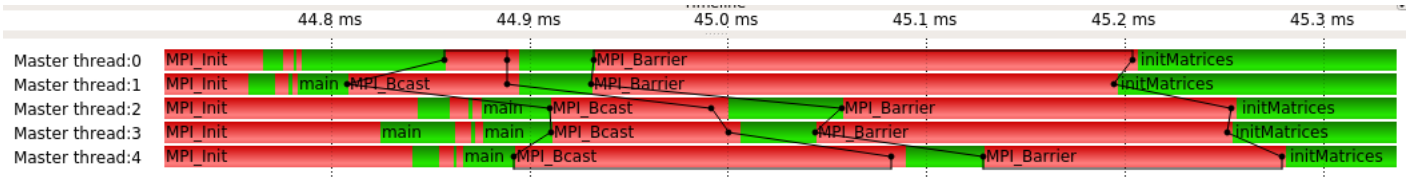
(b) 5 Prozesse, 4 Knoten

Figure 1: Startphase Jacobi-Verfahren

Die obigen Grafiken zeigen die Startphase des Jacobi-Programmes für die verschiedenen Prozess- und Knotenzahlen. Es sind mehrere Dinge gut zu erkennen: Das Programm beginnt nach dem MPI_Init eine Kommunikation und initialisiert dann die Matritzen. Anschließend wird mit einer weiteren Kommunikation der erste Kalkulationsschritt begonnen. Betrachten wir nun die beiden Kommunikationen zum Beginn und Ende von initMatrices genauer.



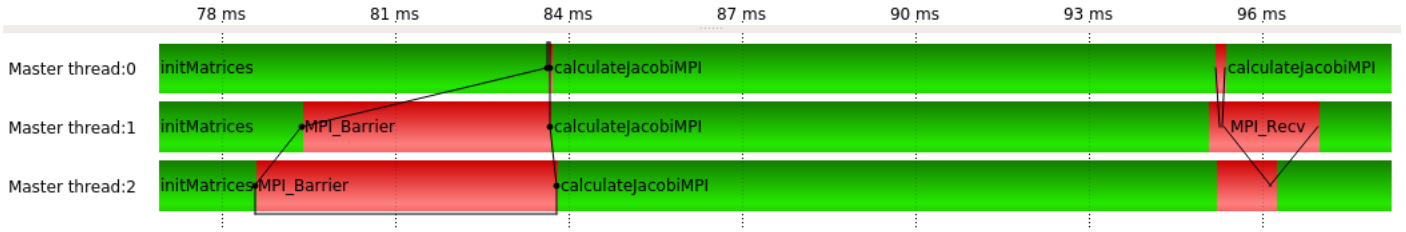
(a) 3 Prozesse, 2 Knoten



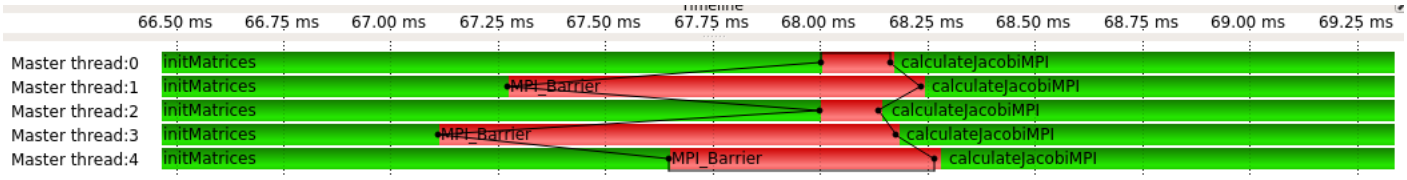
(b) 5 Prozesse, 4 Knoten

Figure 2: Jacobi-Verfahren, Beginnn von InitMatrices

Hier ist zu sehen, was zu Beginn des Programmes geschieht, so wird etwa der "options" struct gebroadcastet und mithilfe von Barriers dafür gesorgt dass die Matritzeninitialisierung nicht zu früh passiert.



(a) 3 Prozesse, 2 Knoten

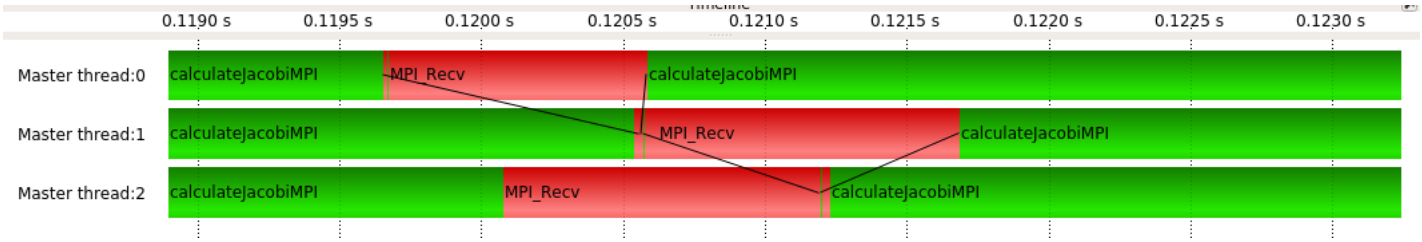


(b) 5 Prozesse, 4 Knoten

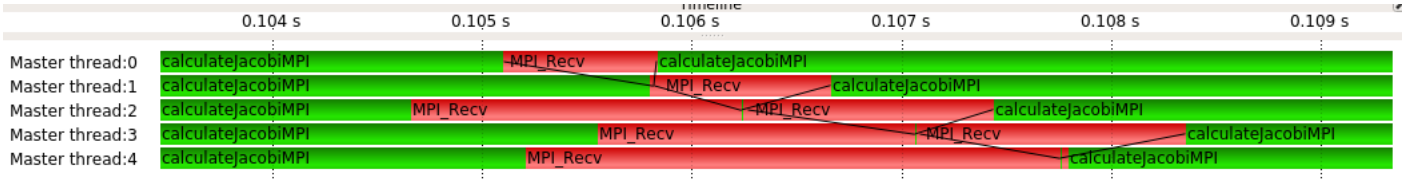
Figure 3: Jacobi-Verfahren, Ende von InitMatrices

In dem Bild 3 ist das Ende von InitMatrices gezeigt. Man sieht wieder die Kommunikation einer Barrier, um den Start von calculateJacobiMPI zu synchronisieren.

1.2 Iteration



(a) 3 Prozesse, 2 Knoten



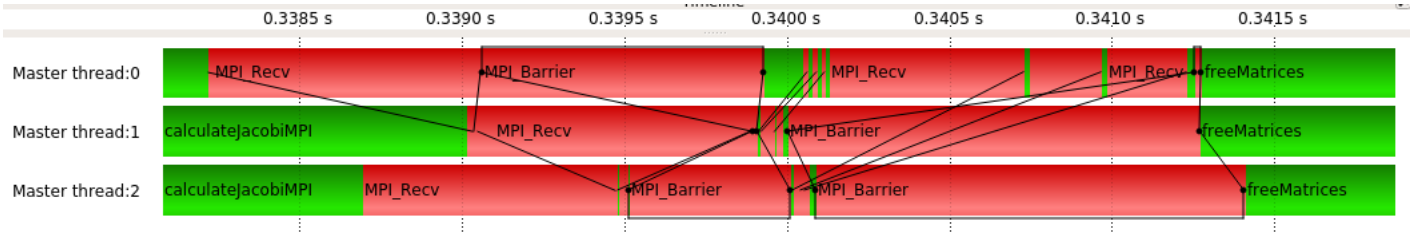
(b) 5 Prozesse, 4 Knoten

Figure 4: Jacobi-Verfahren, Kommunikation zwischen zwei Iterationen

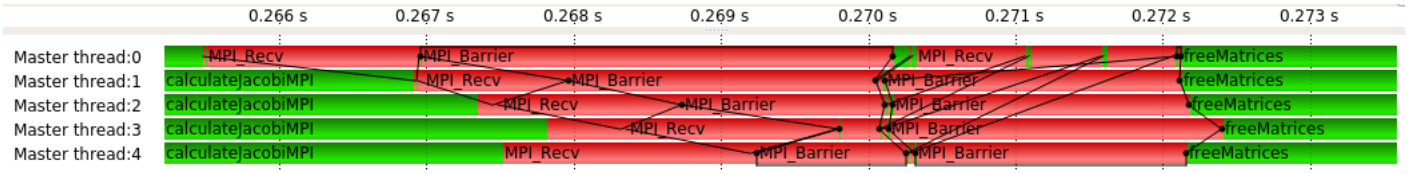
In der nächsten Grafik ist die Kommunikation zwischen zwei Iterationsschritten zu sehen. Man erkennt die Stufenartige Kommunikation von Thread 0 bis zum letzten Thread, an jeder "Stufe" nach unten schließt sich eine Kommunikation nach oben an. Dies entspricht genau dem implementierten Verhalten.

Auch ist zu sehen, dass die einzelnen Threads etwas unterschiedlich schnell an dem Ende der Iteration ankommen (etwa in der unteren Grafik Thread 2 als erstes) und diese Abweichungen durch die erzwungene Synchronisation durch den Nachrichtenaustausch wieder entfernt wird.

1.3 Endphase



(a) 3 Prozesse, 2 Knoten



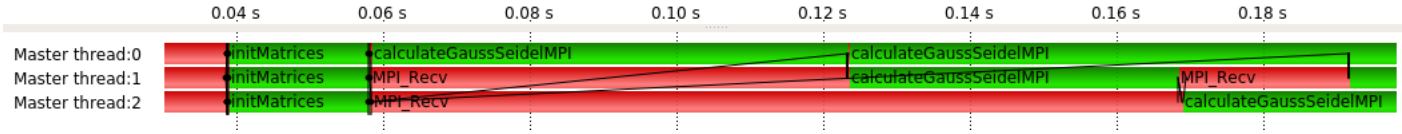
(b) 5 Prozesse, 4 Knoten

Figure 5: Jacobi-Verfahren, Ende der Berechnung

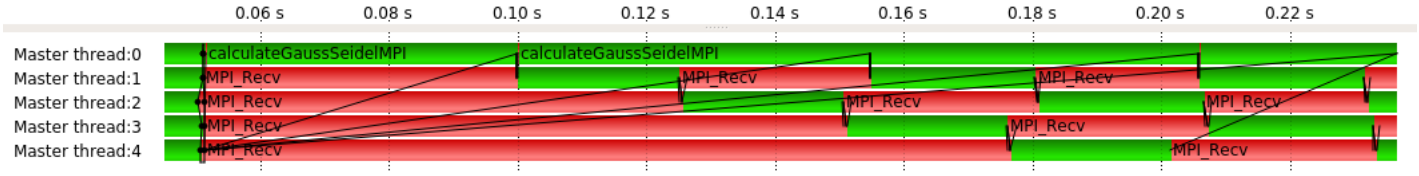
In Grafik 5 ist das Ende der Berechnung zu sehen. Es passiert eine Abschließende Kommunikation in der Zeilen ausgetauscht werden (MPI_Recv), anschließend gibt es eine Barrier, damit alle Prozesse sicher auf dem gleichen Stand sind. Im Anschluss an die Barrier erkennt man gut das Verhalten von der DisplayMatrix Funktion: alle Prozesse schicken ihre Daten an Prozess 0, welcher sie jeweils auf die Konsole druckt. Abschließend wird dann mit freeMatrices das Programm beendet.

2 Gauß-Seidel-Verfahren

2.1 Startphase



(a) 3 Prozesse, 2 Knoten

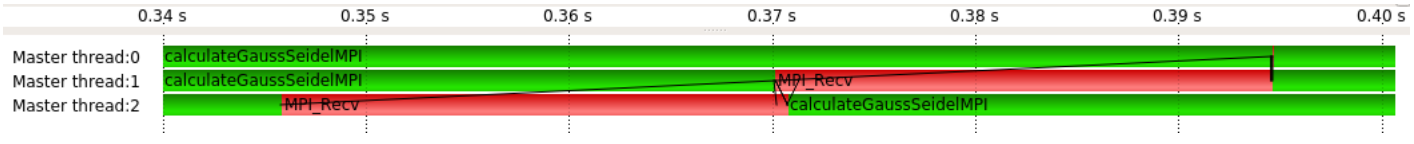


(b) 5 Prozesse, 4 Knoten

Figure 6: Startphase Gauß-Seidel-Verfahren

Grafik 6 zeigt die Startphase des Gauß-Seidel-Verfahrens. Wieder ist die initiale Kommunikation durch Broadcast und Barrier zu sehen, bevor dann initMatrices beginnt. Nach Init Matrices sehen wir wieder eine Barrier, woraufhin dann die Rechnung beginnt. Anders als bei dem Jacobi-Verfahren beginnen hier nicht alle Prozesse direkt zu rechnen (da sie ja die Zahlen der Prozesse über ihnen benötigen). So gibt es hier eine Art "Ramp" bis alle Threads schließlich rechnen.

2.2 Iteration



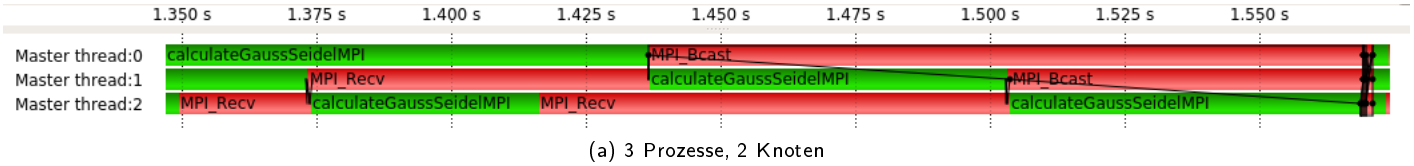
(a) 3 Prozesse, 2 Knoten

(b) 5 Prozesse, 4 Knoten

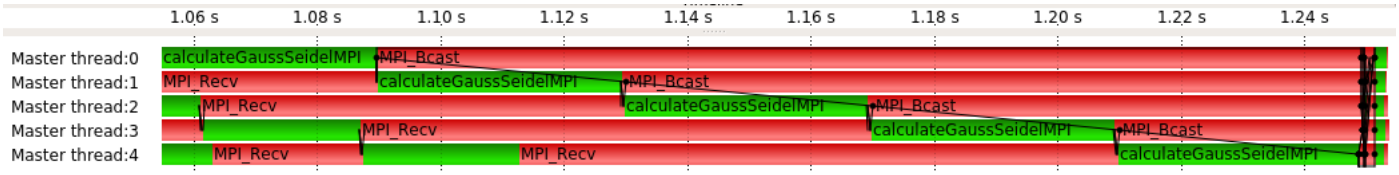
Figure 7: Iterationsphase Gauß-Seidel-Verfahren

In dieser Grafik sieht man einen Ausschnitt aus dem Iterationszeitraum des Gauß-Seidel-Verfahrens. Auffällig sind die langen Wartezeiten. Analog zum Programmcode sieht man für jeden Prozess eine kurze Berechnung (einer Zeile), dann eine Kommunikation nach oben, die Berechnung des Rests und schließlich eine Kommunikation nach unten. Außerdem ist gut die Nachricht vom letzten Prozess an den ersten Prozess zu sehen. Es fällt hier bereits ein offensichtlicher Nachteil gegenüber dem Jacobi-Verfahren zu erkennen: es gibt hier sehr viele Wartezeiten.

2.3 Endphase



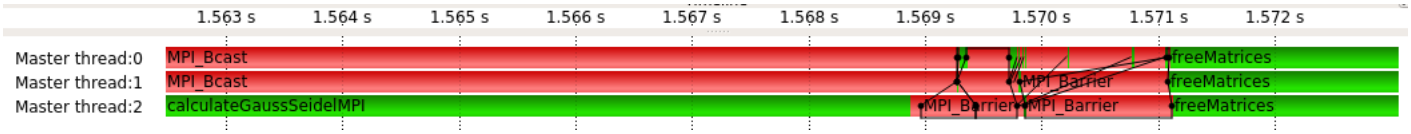
(a) 3 Prozesse, 2 Knoten



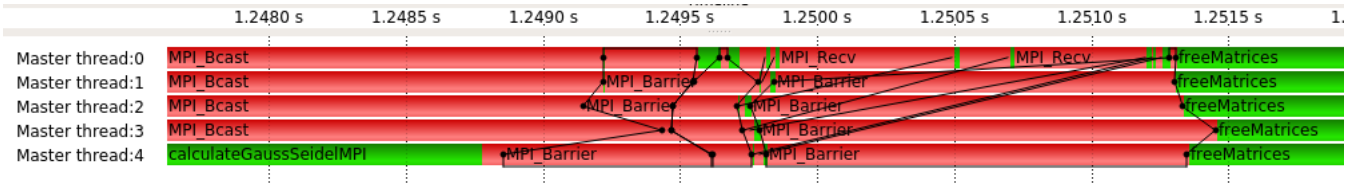
(b) 5 Prozesse, 4 Knoten

Figure 8: Gauß-Seidel-Verfahren, Ende der Berechnung

In Grafik 8 ist das Ende des Gauß-Seidel-Verfahren gezeigt. Man sieht, wie die Prozesse nach und nach fertig mit ihren Berechnungen werden. Es wird dann noch ein Broadcast ausgeführt (um das maxresiduum auszutauschen) und anschließend eine Barrier, um synchron in das Programmende hineinzugehen.



(a) 3 Prozesse, 2 Knoten



(b) 5 Prozesse, 4 Knoten

Figure 9: Gauß-Seidel-Verfahren, Zoom auf den Beginn von freeMatrices

In der letzten Grafik ist eine genauere Betrachtung des Prozessendes möglich. Man erkennt gut den letzten noch laufenden Thread (2 bzw. 4) sowie die Barrier. Anschließend ist die DisplayMatrices-Funktion zu sehen, welche gnz analog zur Jacobi-Methode alle Daten an Prozess 0 sendet, welcher sie dann druckt.

3 Fazit

Insgesamt ist zu erkennen, dass das Gauß-Seidel-Verfahren deutliche Nachteile gegenüber dem Jacobi-Verfahren zeigt (zumindest in unserer Implementation). Dies ist vor allem auf die vielen Wartezeiten während der Iterationszeit zurückzuführen. Auch ist das Jacobi-Verfahren im Start und Ende des Programmes schneller, da diese aber jeweils nur einmal passieren fällt das nicht so ins Gewicht wie die Leistungsunterschiede während der Iterationen.