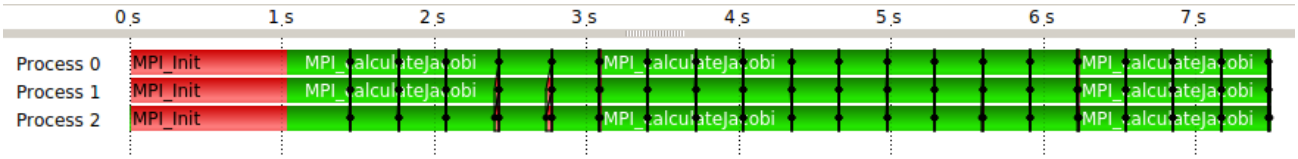
Visualisierung mit Vampir

Vorab ist zu sagen, dass wir diese Aufgabe ***nur*** für das Jacobi-Verfahren bearbeiten, da Gauss-Seidel keine korrekten Ergebnisse liefert bzw. nicht immer terminiert.

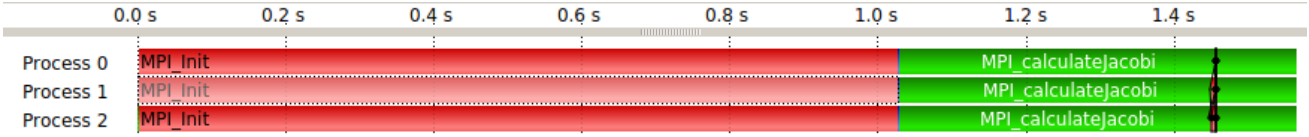
Im Folgenden wird das Programm mit **512 Interlines** und **20 Iterationen** aufgerufen.

Fall 1: 3 Prozesse auf 2 Knoten



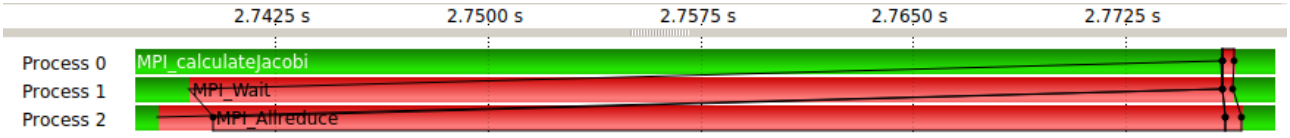
Lässt man drei Prozesse auf zwei Knoten laufen, so hat Knoten 1 zwei Prozesse und Knoten 2 einen.

Anfang:



Am Anfang wird MPI initialisiert, was immer ungefähr eine Sekunde dauert.
Ist dieser Schritt getan, so fängt das Programm mit der Berechnung an, welche bis 1,45 Sekunden dauert.

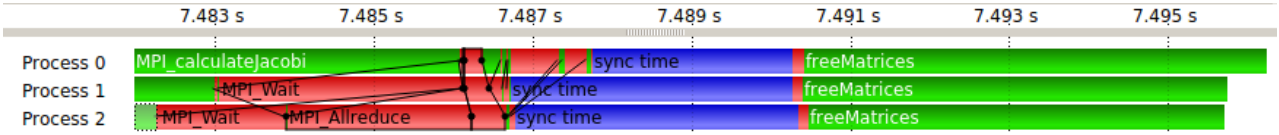
Phase der Synchronisation:



Rank 1 und Rank 0 warten am Anfang auf das Senden und Empfangen an den jeweils nächsten Rank, während Rank 2 und Rank 3 jeweils auf dasselbe vom vorherigen Rank warten.
Per MPI_Allreduce wird dann die Genauigkeit des einzelnen Ranks auf den Maximalwert reduziert (wir haben nicht abgefangen, dass er dies nur machen soll, wenn nach Genauigkeit gerechnet wird).
Aufgrund von Isend und Irecv kommt es zu unterschiedlichen Startpunkten, da er nicht auf den Erfolg wartet (erst mit MPI_Wait), jedoch rechnet jeder Rank zur ungefähr gleichen Zeit weiter, da auch hier mit MPI_Wait auf die einzelnen Send- und Receive-Operationen gewartet wird.

Außerdem sieht man, dass die Berechnung, welche der zweite Prozess ausführt, viel schneller geht als die beiden anderen Berechnungen, da Prozess 2 einen Knoten für sich alleine hat und so die Rechenpower voll ausnutzen kann.

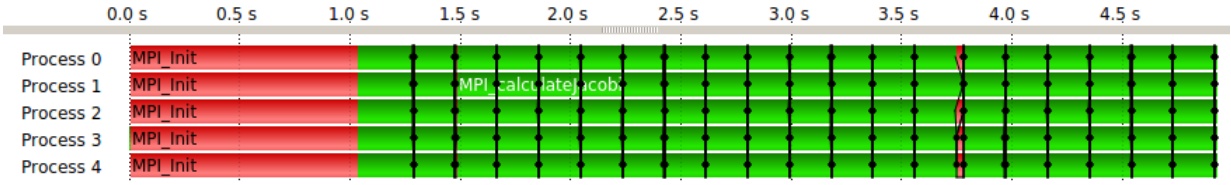
Ende:



Ist das Jacobi-Verfahren bis zum Ende gekommen, so werden alle Ranks auf die gleiche Zeit gebracht, woraufhin mit MPI_Finalize wieder zum sequentiellen Programm zurückgekehrt wird – durch die hier blau markierte sync time und einer MPI_Barrier.

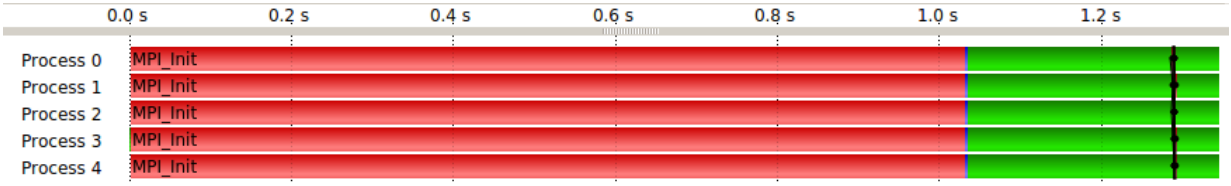
Auch hier ist der zweite Prozess wieder am schnellsten, da er alleine auf einem Knoten rechnet.

Fall 2: 5 Prozesse auf 4 Knoten



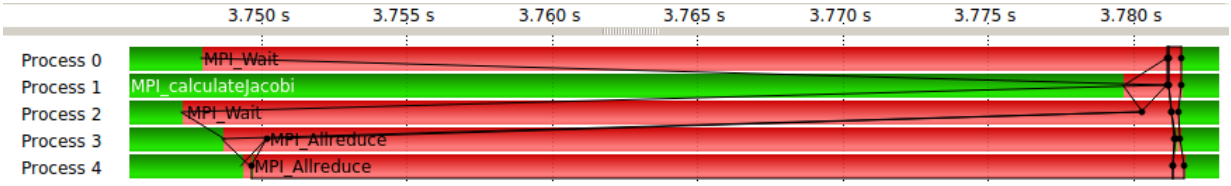
Lässt man fünf Prozesse auf vier Knoten laufen, so hat Knoten 1 zwei Prozesse und die restlichen Knoten haben jeweils einen Prozess.

Anfang:



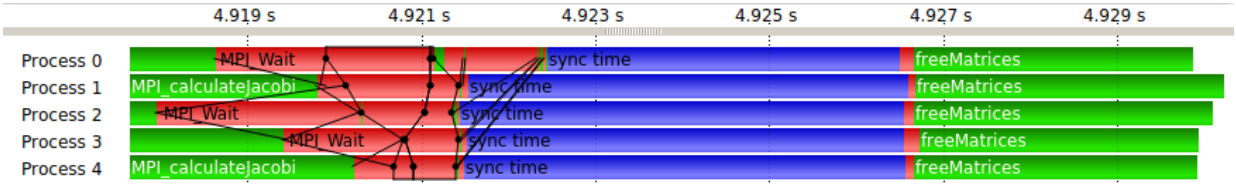
Wie immer dauert auch hier das Initialisieren eine Sekunde, wohingegen die Berechnung der ersten Iteration bereits nach 1,3 Sekunden erfolgt ist, was daran liegt, dass das Problem nun von mehreren Prozessen berechnet wird.

Phase der Synchronisation:



Hier ist besonders auffällig, dass Prozess 1, welcher auf dem ersten Knoten liegt, am längsten rechnet. Das dürfte daran liegen, dass der Knoten in diesem Fall komplett von Prozess 0 ausgelastet wurde und Prozess 1 daher einen verzögerten Start hatte. Wie man in der obigen Gesamtgrafik jedoch sieht, bildet dies eher die Ausnahme, kommt aber dennoch vor.
Idealerweise rechnen Prozess 2 - 4 schneller als Prozess 0 und 1.

Ende:



Zum Ende hin werden wieder alle Prozesse auf ungefähr die gleiche Zeit gebracht (sync time und MPI_Barrier), woraufhin MPI_Finalize ausgeführt und der Speicher der Matrizen freigegeben wird.

Zeitaufwand: ca. 2h