Visualisierung der Abläufe des Parallelisierten PDE-Solvers

Johannes Timm Guannan Hu

24. Januar 2015

1 Visualisierung der Implementierung des Gauss-Seidel und Jacobi Iterationsverfahrens

1.1 Konfiguration

Für alle Tests wurden 20 Iterationen und ein Abbruch nach Iterationen gewählt. Damit einzelne Iterationsschritte erkennbar werden wurden 1000 Interlines gewählt. Um die Konfiuration der Knoten odrungsgemäß abzuwickeln wurden die Jobscripte aus de vorherigen Aufgabe angepasst.

1.2 Jacobi

1.2.1 2 Knoten 3 Prozesse

1.2.2 4 Knoten 5 Prozesse

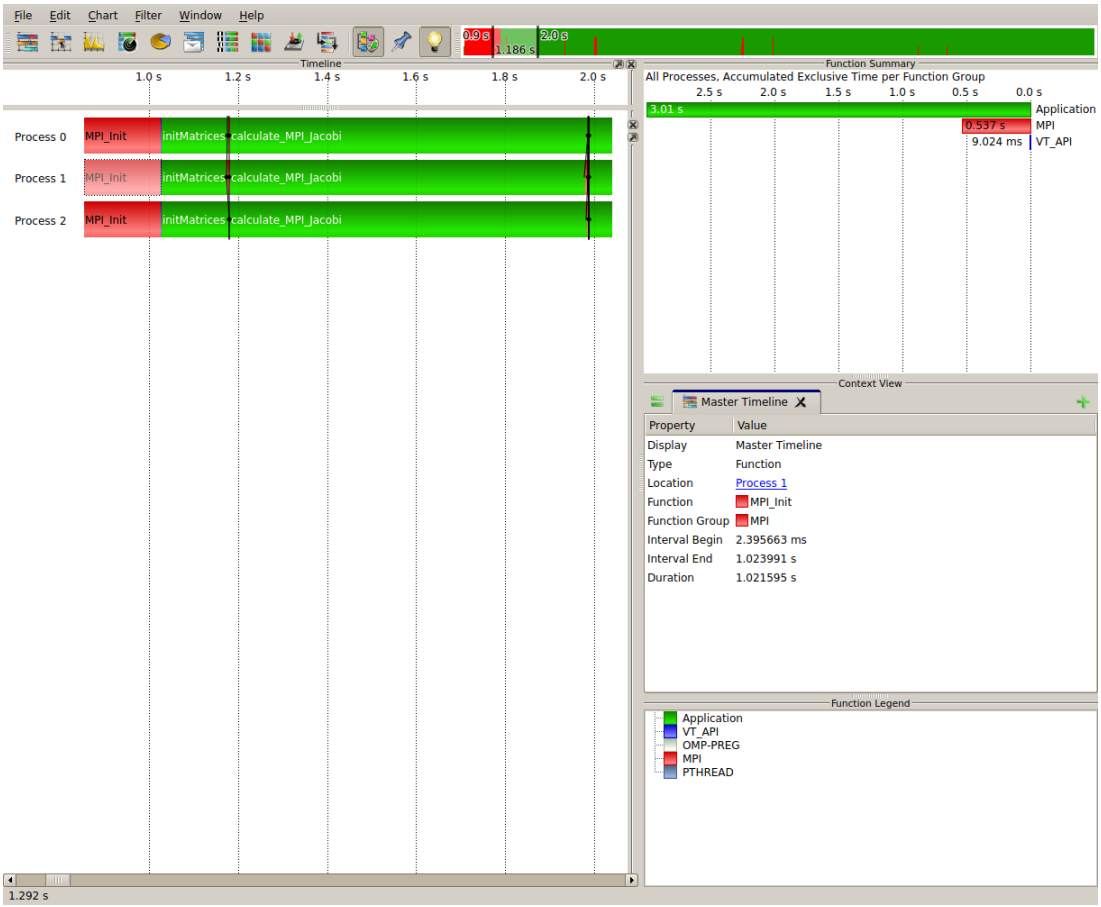

Abbildung 1: Der Beginn des Jacobi-Verfahrens bei 3 Prozessen

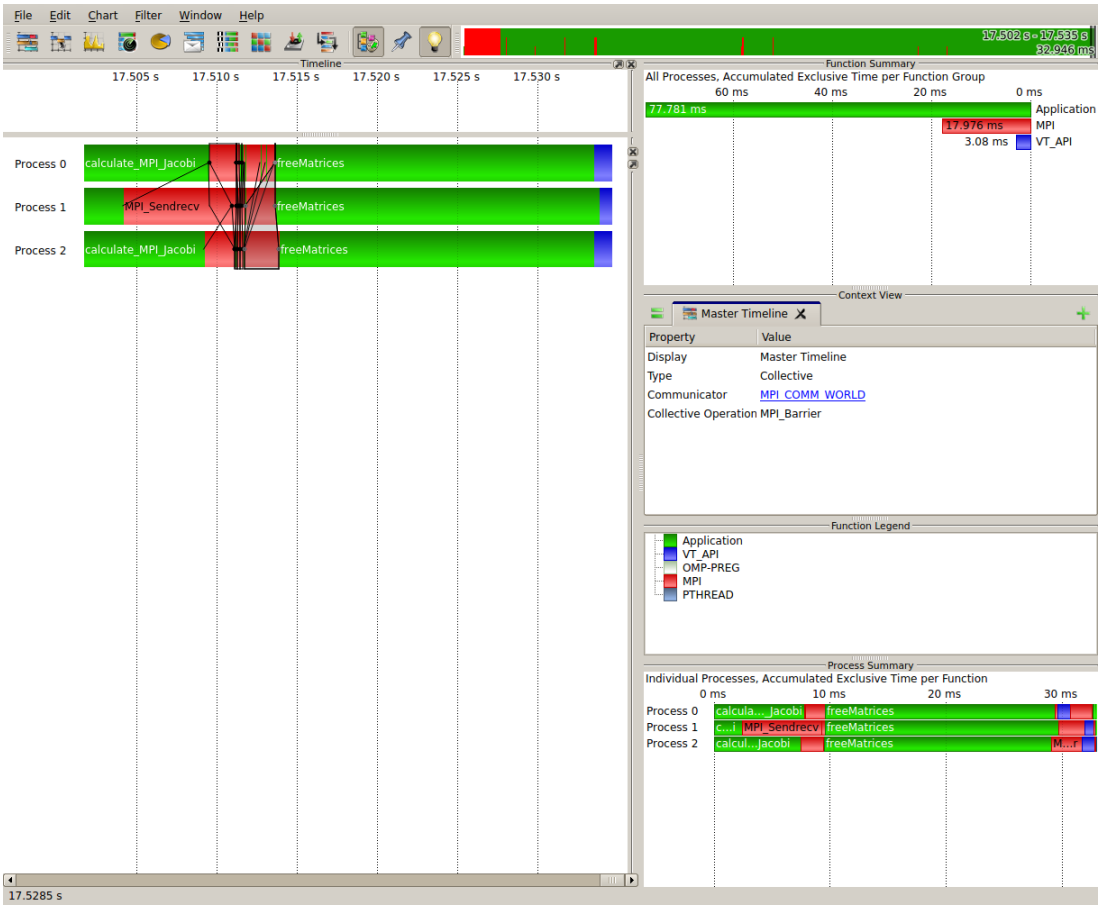

Abbildung 2: Das Ende des Jacobi-Verfahrens bei 3 Prozessen

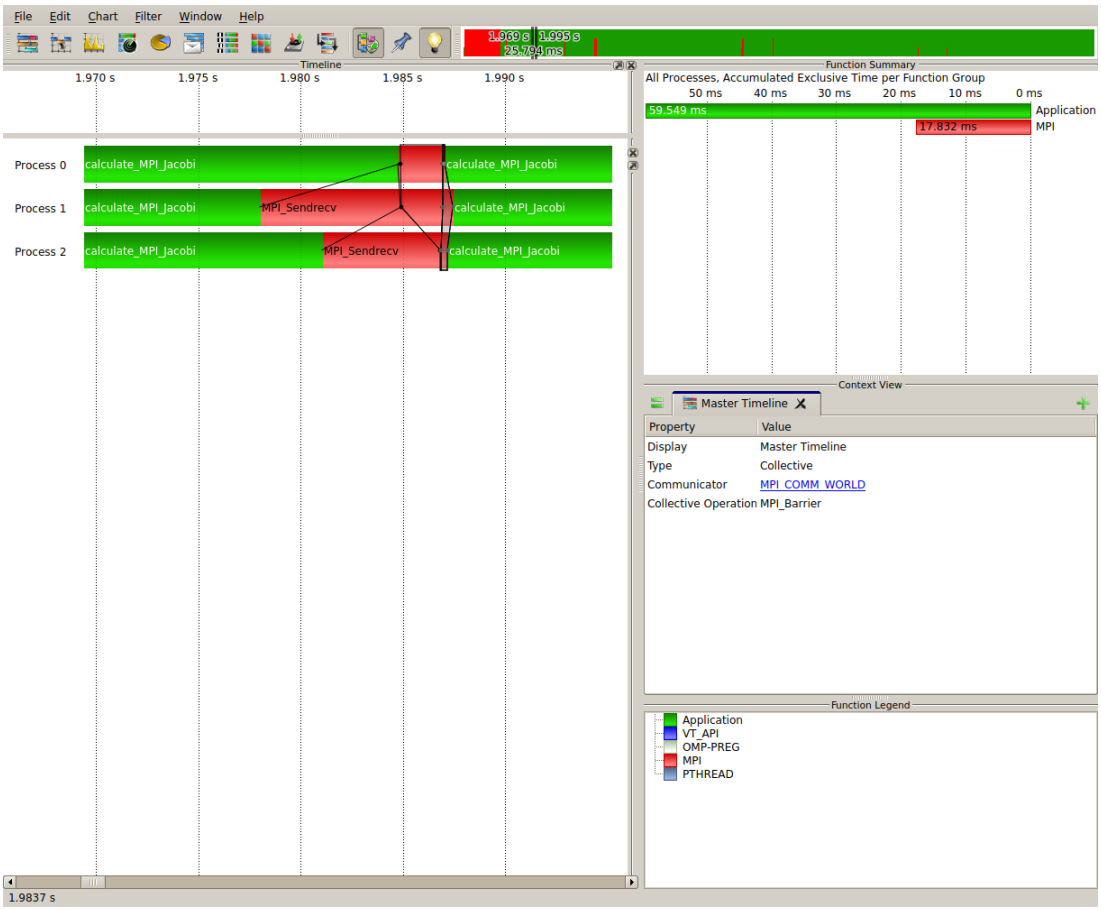

Abbildung 3: Die Syncronisation am Ende eines Iterationsschrittes des Jacobi-Verfahrens bei 3 Prozessen

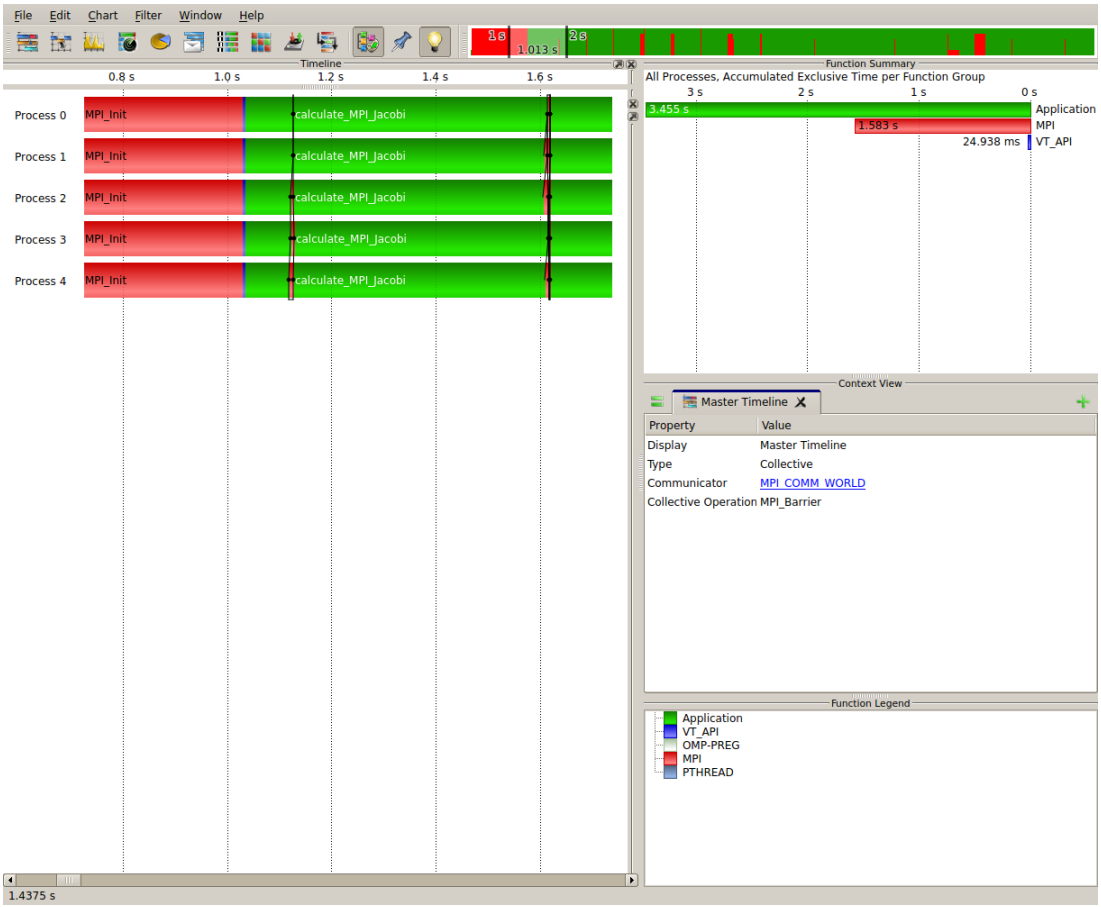

Abbildung 4: Der Beginn des Jacobi-Verfahrensbei 5 Prozessen

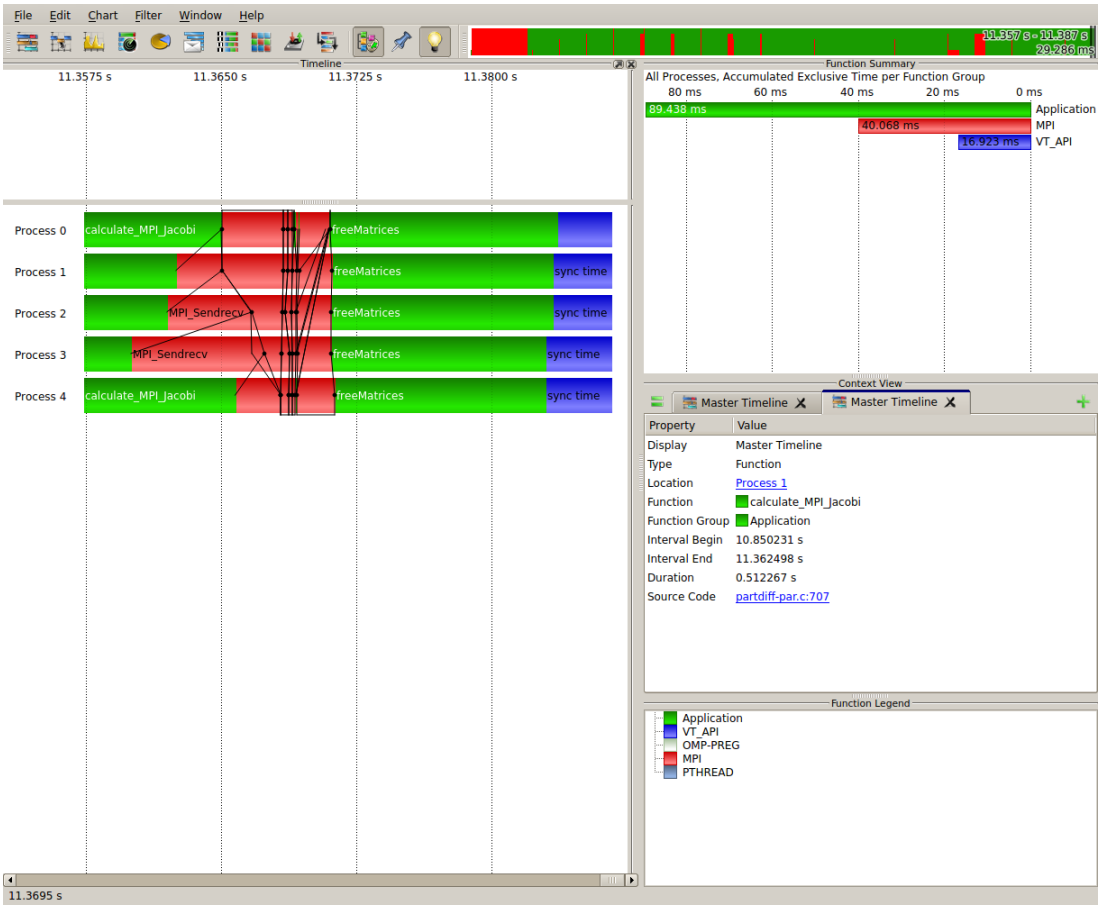

Abbildung 5: Das Ende des Jacobi-Verfahrens bei 5 Prozessen

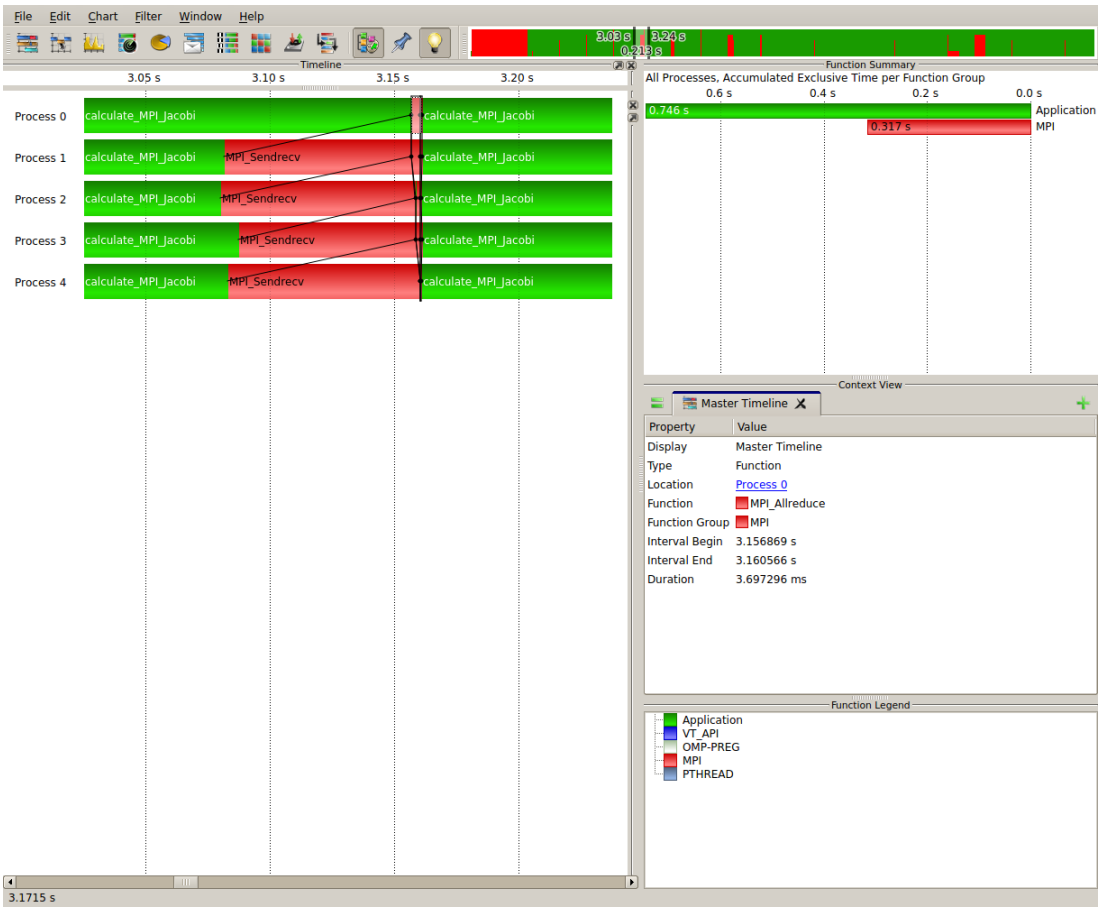

Abbildung 6: Die Syncronisation am Ende eines Iterationsschrittes des Jacobi-Verfahrens bei 5 Prozessen

1.3 Gauss-Seidel

1.3.1 2 Knoten 3 Prozesse

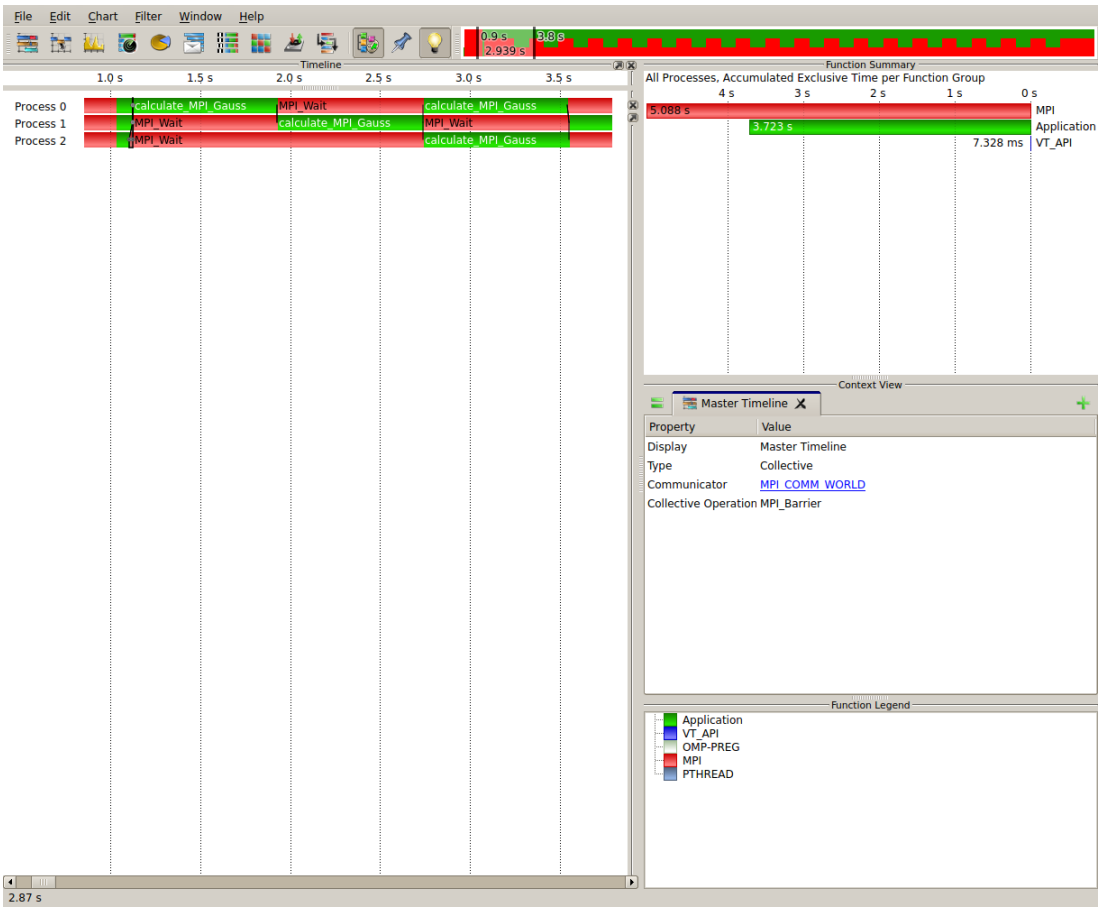

Abbildung 7: Der Beginn des Gauss-Seidel-Verfahren bei 3 Prozessen

1.3.2 4 Knoten 5 Prozesse

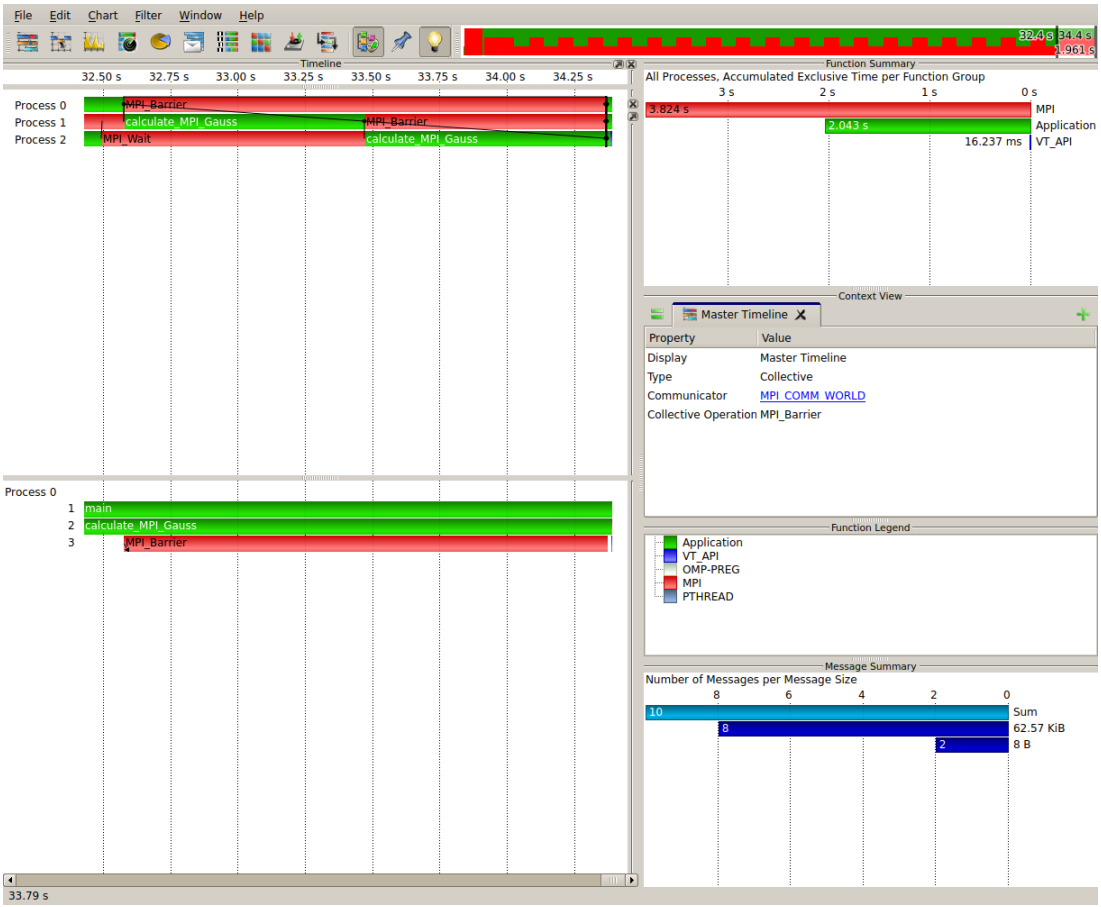

Abbildung 8: Das Ende der Pipeline des Gauss-Seidel-Verfahren bei 3 Prozessen

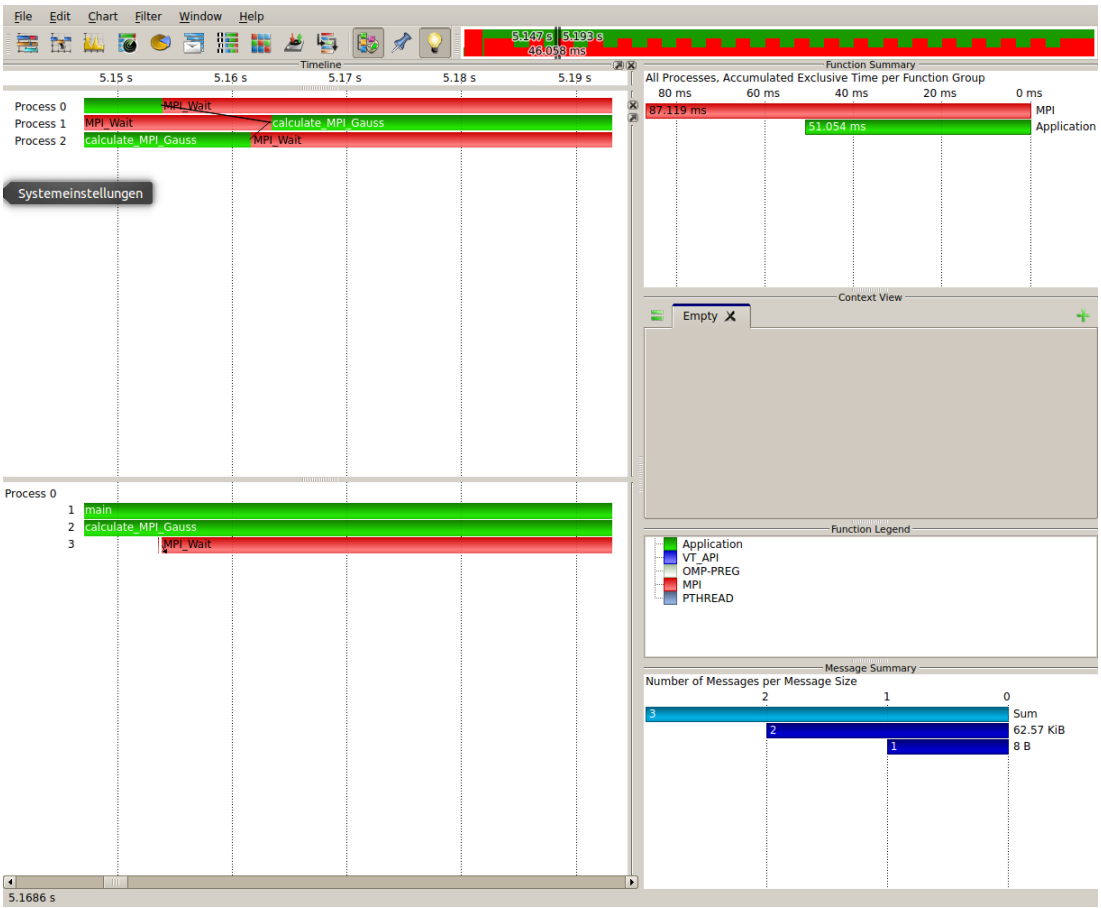

Abbildung 9: Die Syncronisation am Ende eines Iterationsschrittes des Gauss-Seidel-Verfahren bei 3 Prozessen

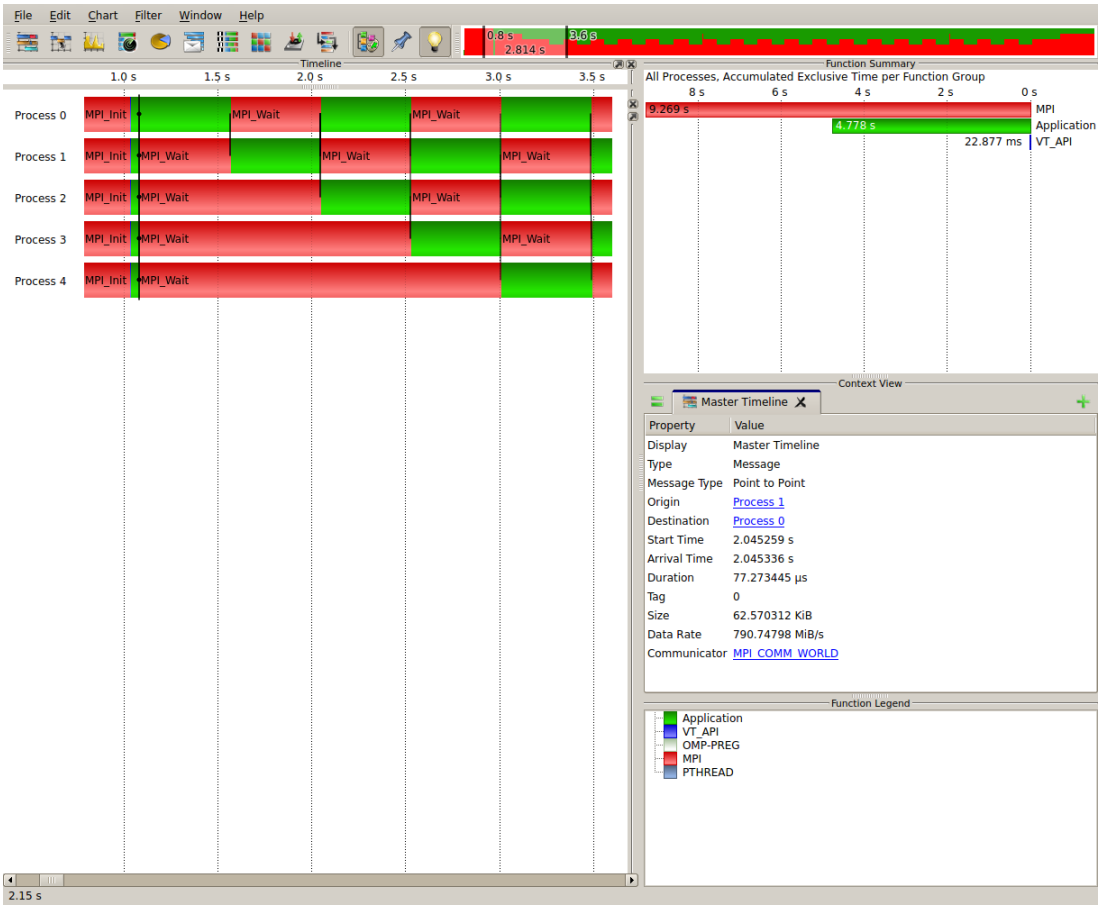

Abbildung 10: Der Beginn des Gauss-Seidel-Verfahren bei 5 Prozessen

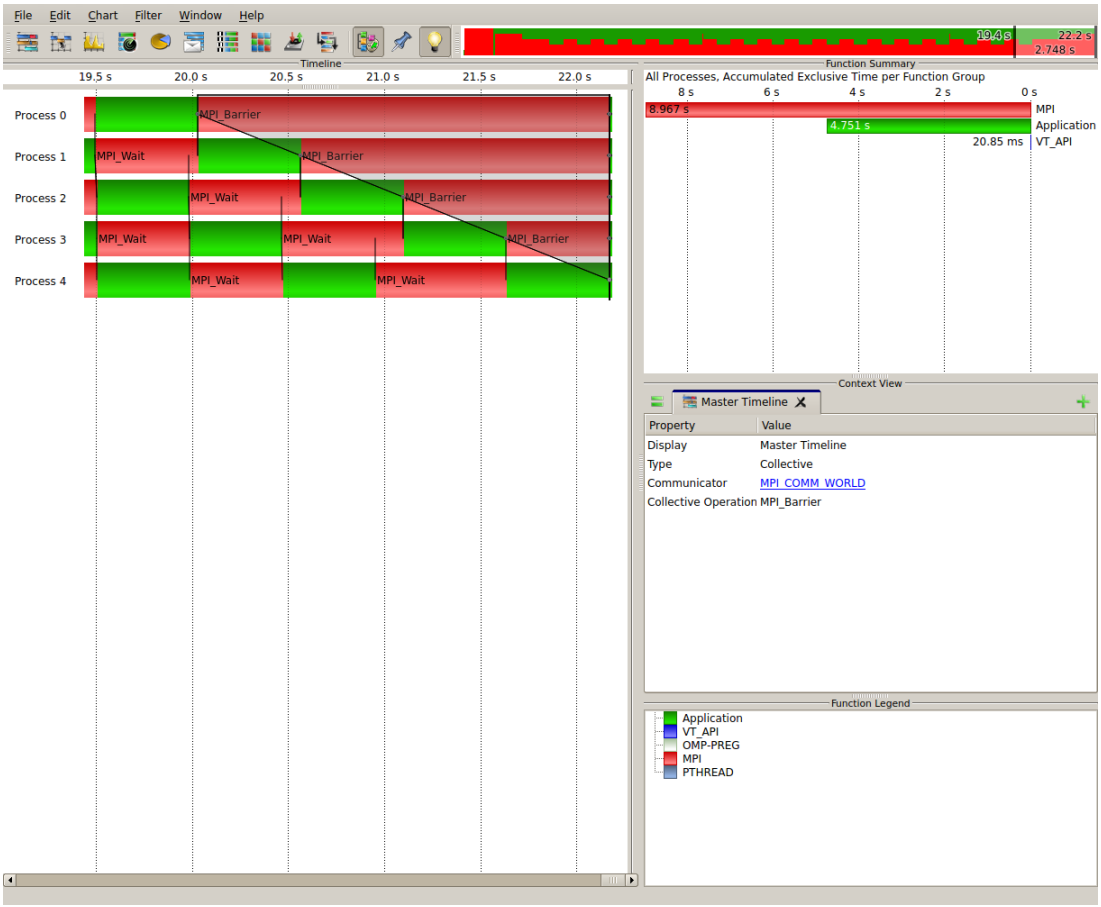

Abbildung 11: Das Ende der Pipeline des Gauss-Seidel-Verfahren bei 5 Prozessen

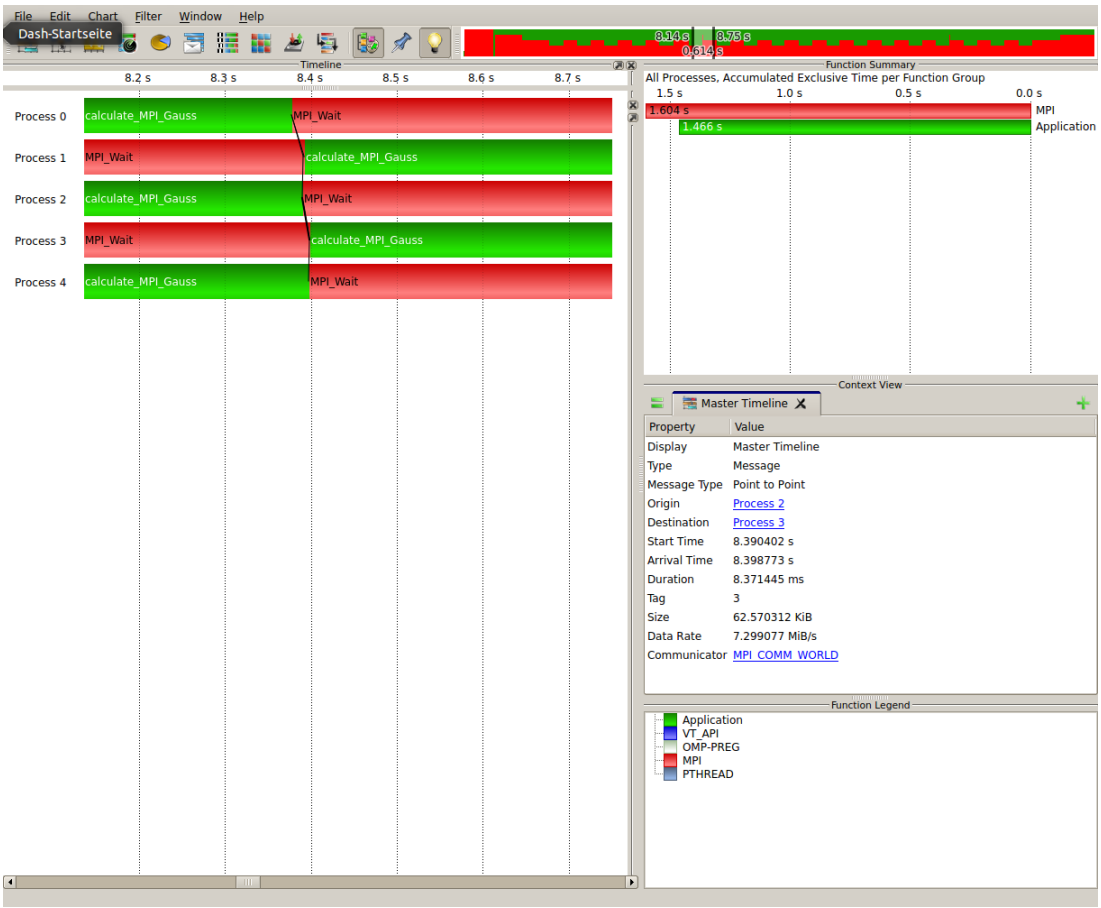

Abbildung 12: Die Syncronisation am Ende eines Iterationsschrittes des Gauss-Seidel-Verfahren bei 5 Prozessen

1.4 Diskussion

Das Jacobi Verfahren hat in diesen Konfigurationen, die hier Visualisiert wurden kaum Probleme performant zu arbeiten Es kommt beim Jacobi-Verfahren nie vor, dass Desysncronisationen länger als einen Iterationsschritt andauern. Die ist eine Folge aus der Benutzung der Blockierenden Kommunikation von Allreduce und SendRecv, sodass alle Prozesse am Abschluss eines Iterationsschrittes aufeinander warten. Beim Gauss-Seidel-Verfahren ist durch das Pipelining anfälliger für Desyncronistation, leider ist sie auch wesentlich schwerer zu erkennen. Die Prozesse des Gauss-Seidel-Verfahrens scheinen sich ledoch größtenteils sehr gut dynamisch miteinander zu sysnchronisieren. Ein Fehler in der Kommunikationsstruktur des Gauss-Seidel-Verfahrens wurde jedoch offenbar, da sich die Berechnnung jeweils mit einem Wait gleicher Länge abwechselt. Leider wird aus der Visualisierung nicht offenbar, in welcher Zeile des Programmcodes dieses spezifische Wait aufgerufen wird. Dies ergibt ein Schachbrettmuster. Dieses führt dazu, dass im Durchschnitt 50% der Rechenzeit für Warten vergeudet wird. Eine Änderung des Verhaltens der spezifischen Wait funktion in ein Wait mittels Interrupt würde die Energieeffizienz der Berechnung mit dem Gauss-Seidel-Verfahren erhöhen, ohne dass Änderungen an der Kommunikationsstruktur und ggf. Algorithmus vorzunehmen sind.