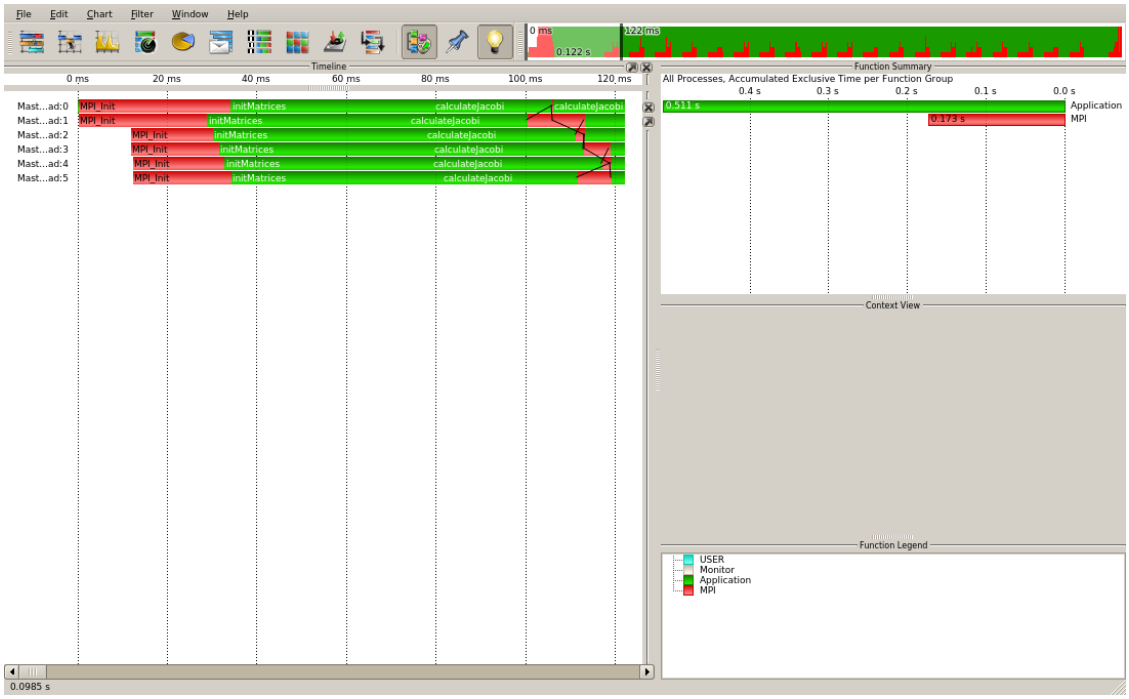
Hochleistungsrechnen

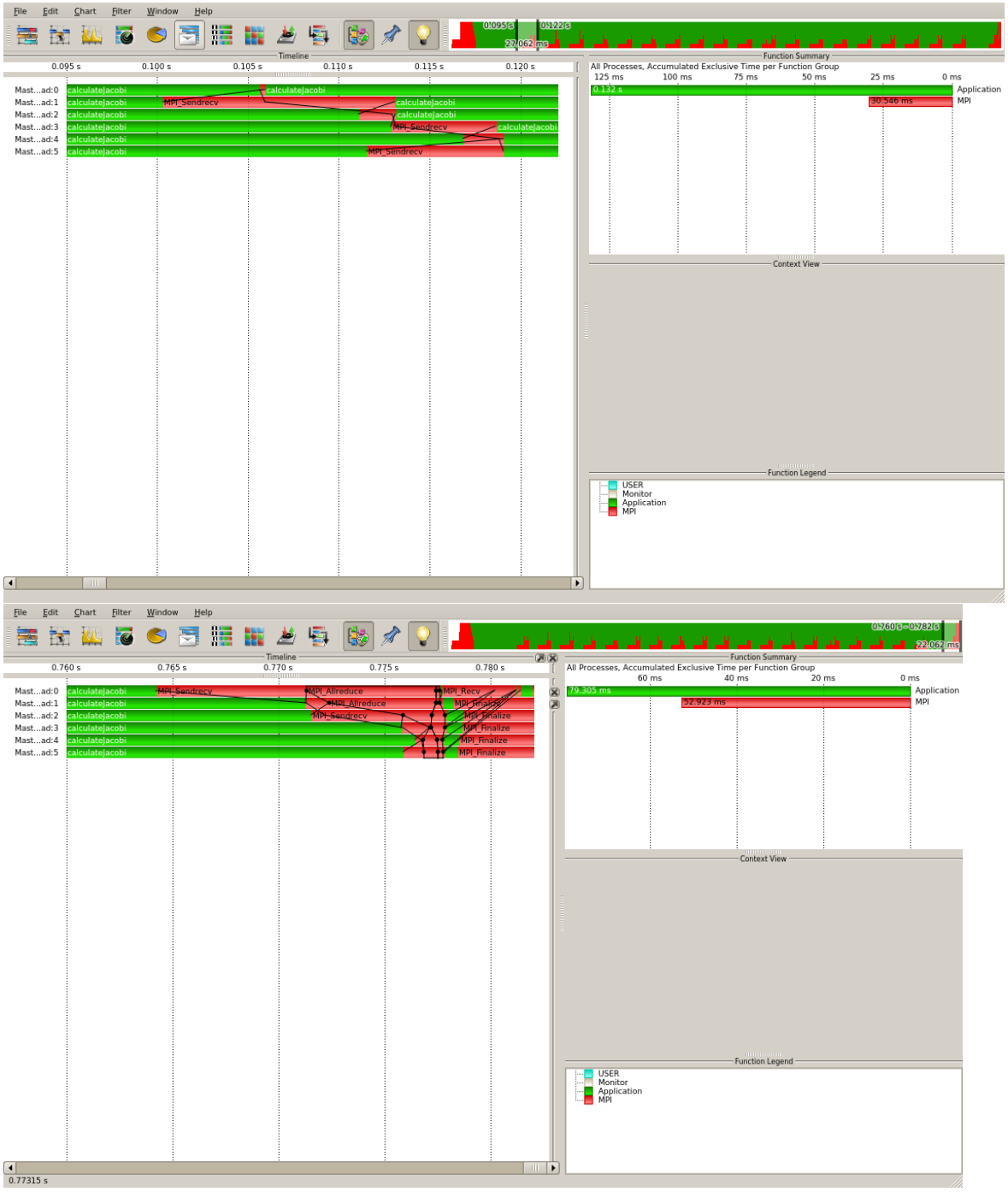
Tim Welge, Benjamin Wolf

21.01.2017

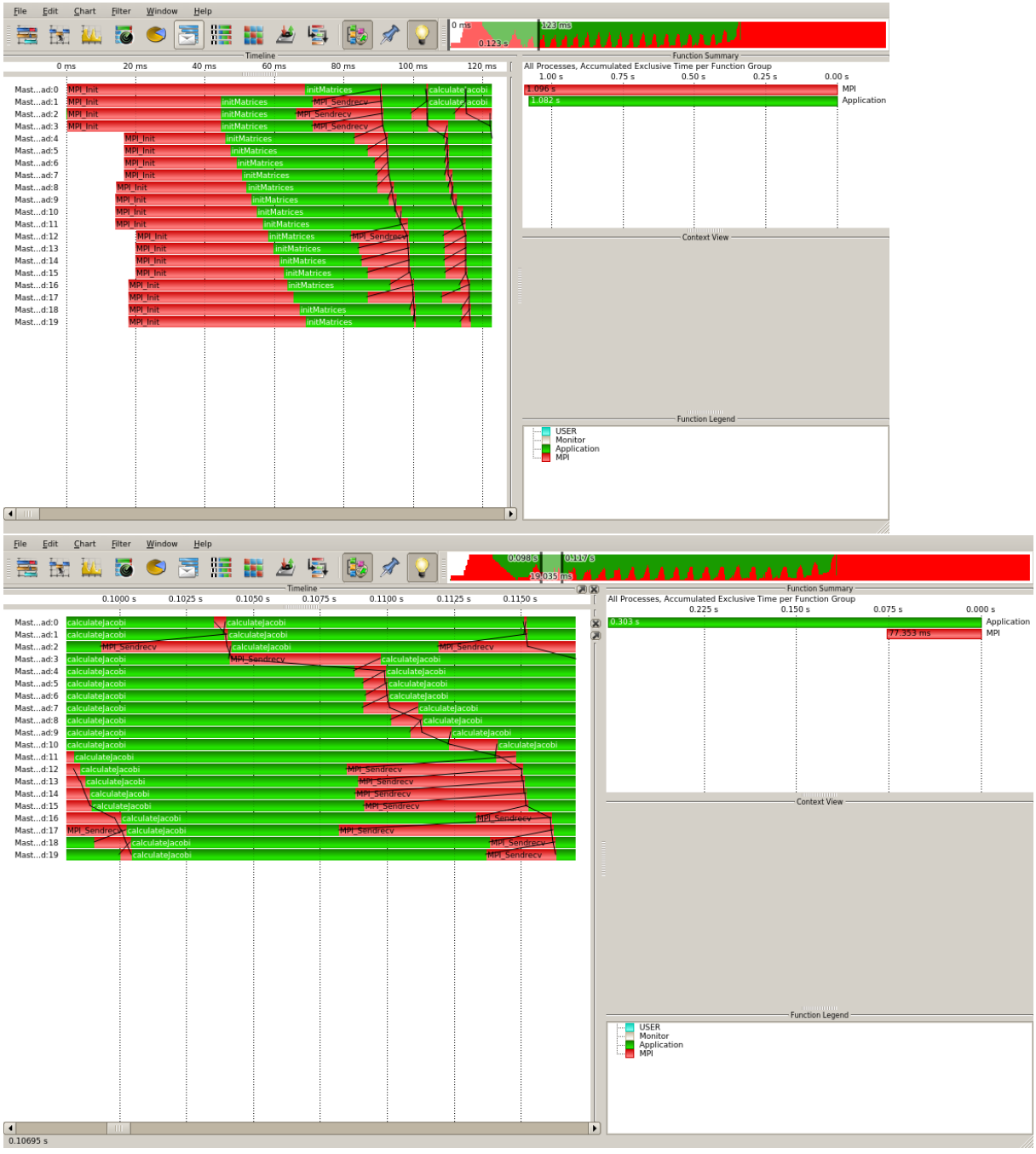
Jacobi

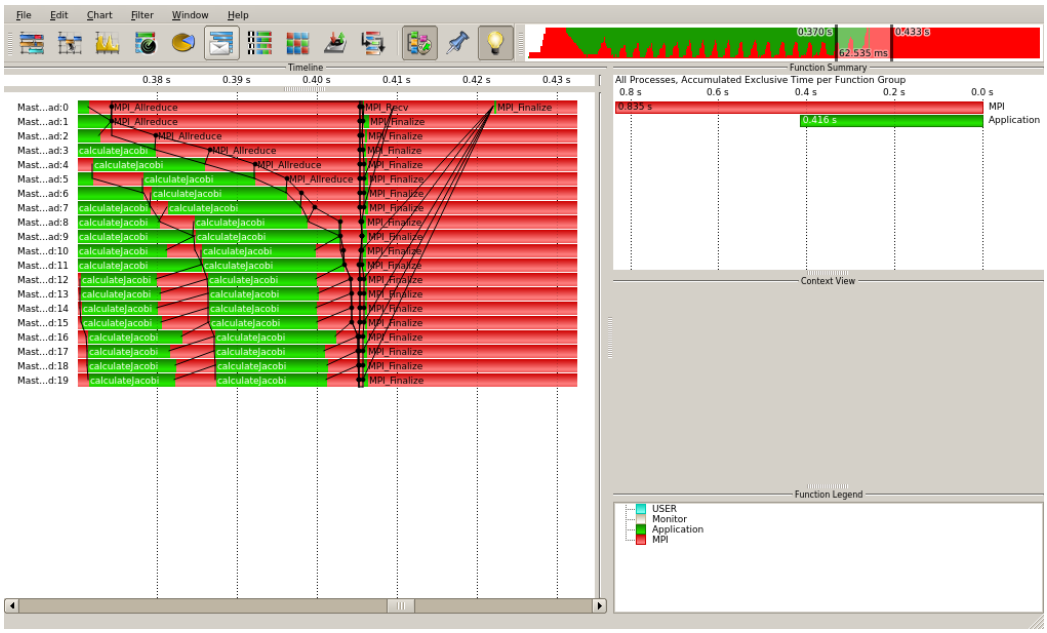
Knoten 2, 3 Prozesse




File Edit Chart Filter Window Help
0.095 s
0.122 s
22.062 ms
Timeline
0.095 s
0.100 s
0.105 s
0.110 s
0.115 s
0.120 s
Mast...ad:0
Mast...ad:1
Mast...ad:2
Mast...ad:3
Mast...ad:4
Mast...ad:5
calculateJacobi
MPI_Sendrecv
Function Summary
All Processes, Accumulated Exclusive Time per Function Group
125 ms
100 ms
75 ms
50 ms
25 ms
0 ms
0.132 s
30.546 ms
Application
MPI
Context View
Function Legend
USER
Monitor
Application
MPI
File Edit Chart Filter Window Help
0.76038 s
0.78213 s
22.062 ms
Timeline
0.760 s
0.765 s
0.770 s
0.775 s
0.780 s
Mast...ad:0
Mast...ad:1
Mast...ad:2
Mast...ad:3
Mast...ad:4
Mast...ad:5
calculateJacobi
MPI_Sendrecv
MPI_Allreduce
MPI_Recv
MPI_Finalize
Function Summary
All Processes, Accumulated Exclusive Time per Function Group
60 ms
40 ms
20 ms
0 ms
79.305 ms
52.923 ms
Application
MPI
Context View
Function Legend
USER
Monitor
Application
MPI
0.77315 s

Knoten 5, 4 Prozesse



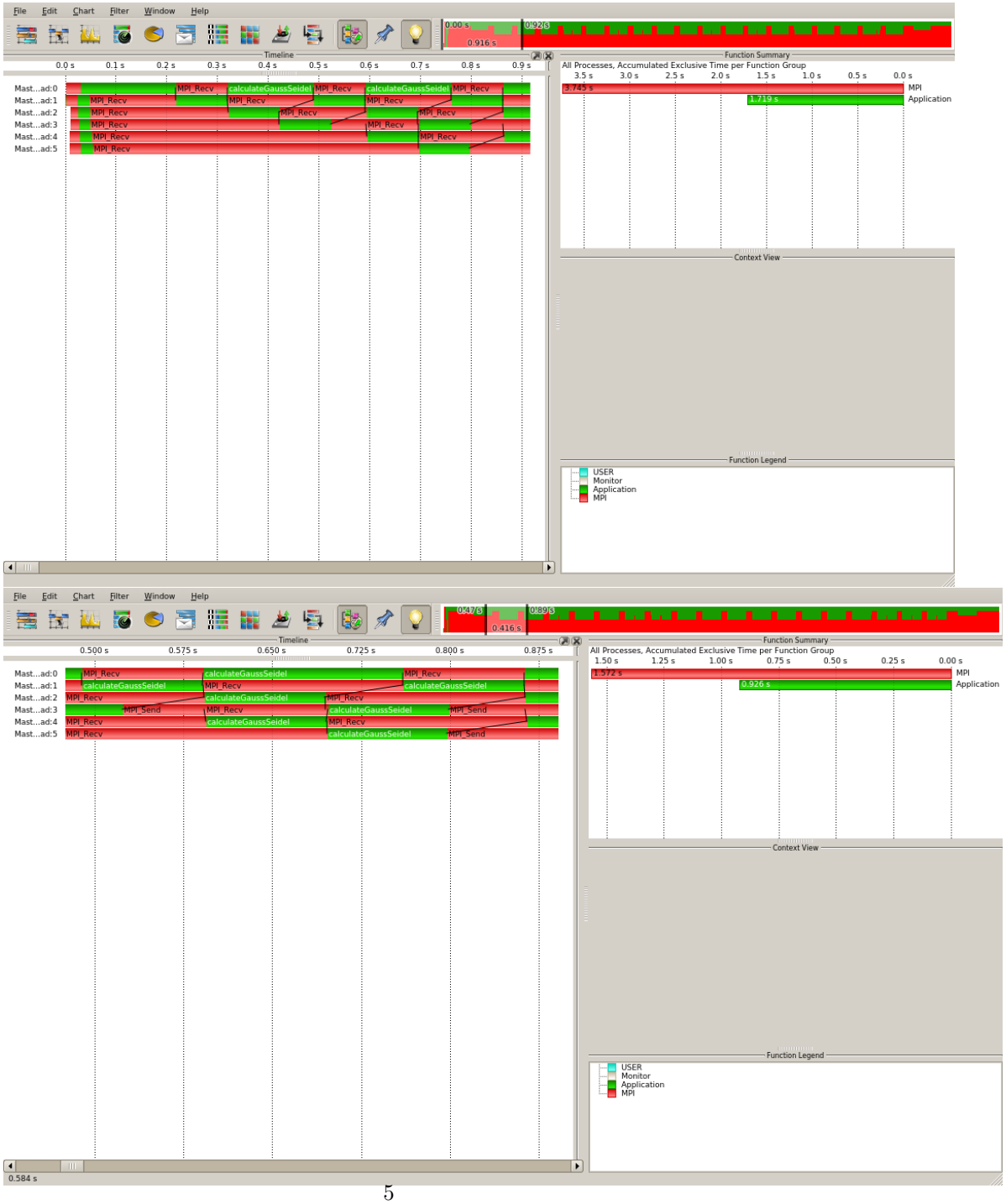


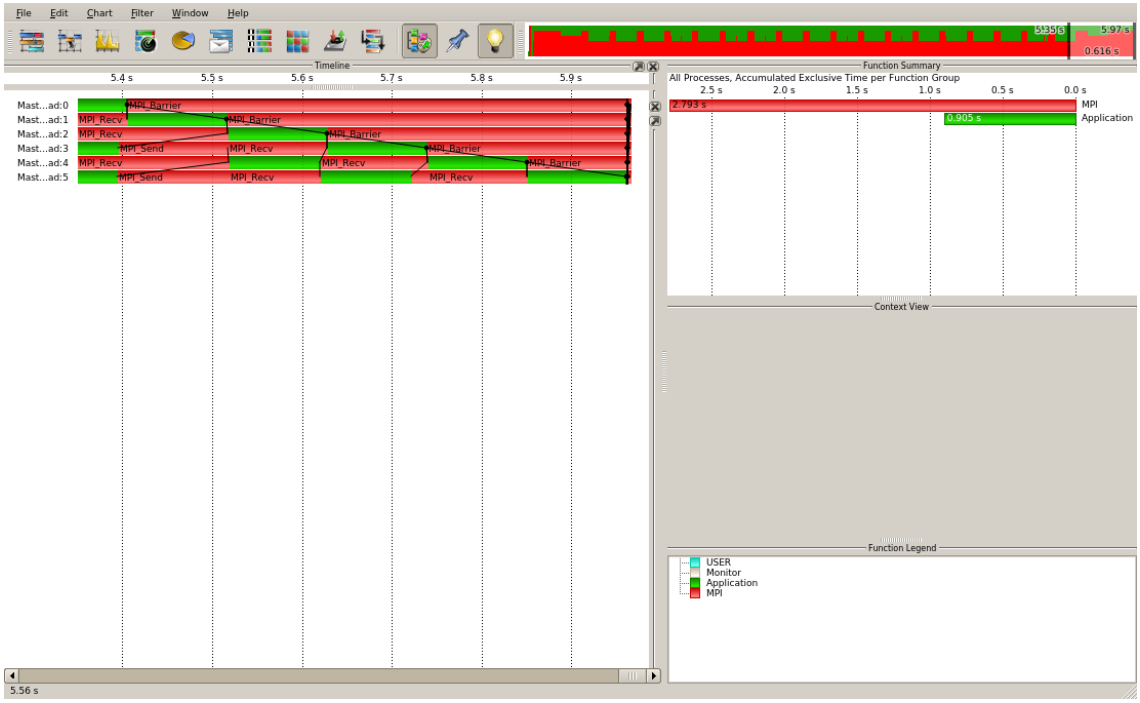
Interpretation

Start-Phase:
Das MPI Init werden nicht alle Prozesse gleichschnell initialisiert. Die liegt wahrscheinlich an MPI und dem Verfahren von MPI_INIT. Ansonsten sieht man hier wie alle Prozesse ungefähr gleichzeitig starten, da bei dem Jacobi-Verfahren die einzelnen Prozesse nicht erst auf die Werte aus den anderen Prozessen warten müssen.
Sync-Phase:
Hier tritt der Effekt auf, dass einzelne Prozesse schneller sind als andere und die benötigte Zeit für MPI_Send zum Teil auch relativ stark variiert, sonst verhält sich das Programm wie erwartet.
End-Phase:
Bei der 5,4-Variante fällt auf, das MPI_Finalize sehr lange braucht, nachdem alle Prozesse ihr Resultat an den root-Prozess gesendet haben. Man sieht ausserdem,dass das MPI_ALLREDUCE auch noch einiges an Zeit benötigt , um das Residuum zu bekommen. Des Weiteren ist auch hier wie bei jedem MPI Programm die sehr lange Start- und Endzeit zu sehen, die das Programm benötigt um MPI zu initialisieren und zu finalisieren.

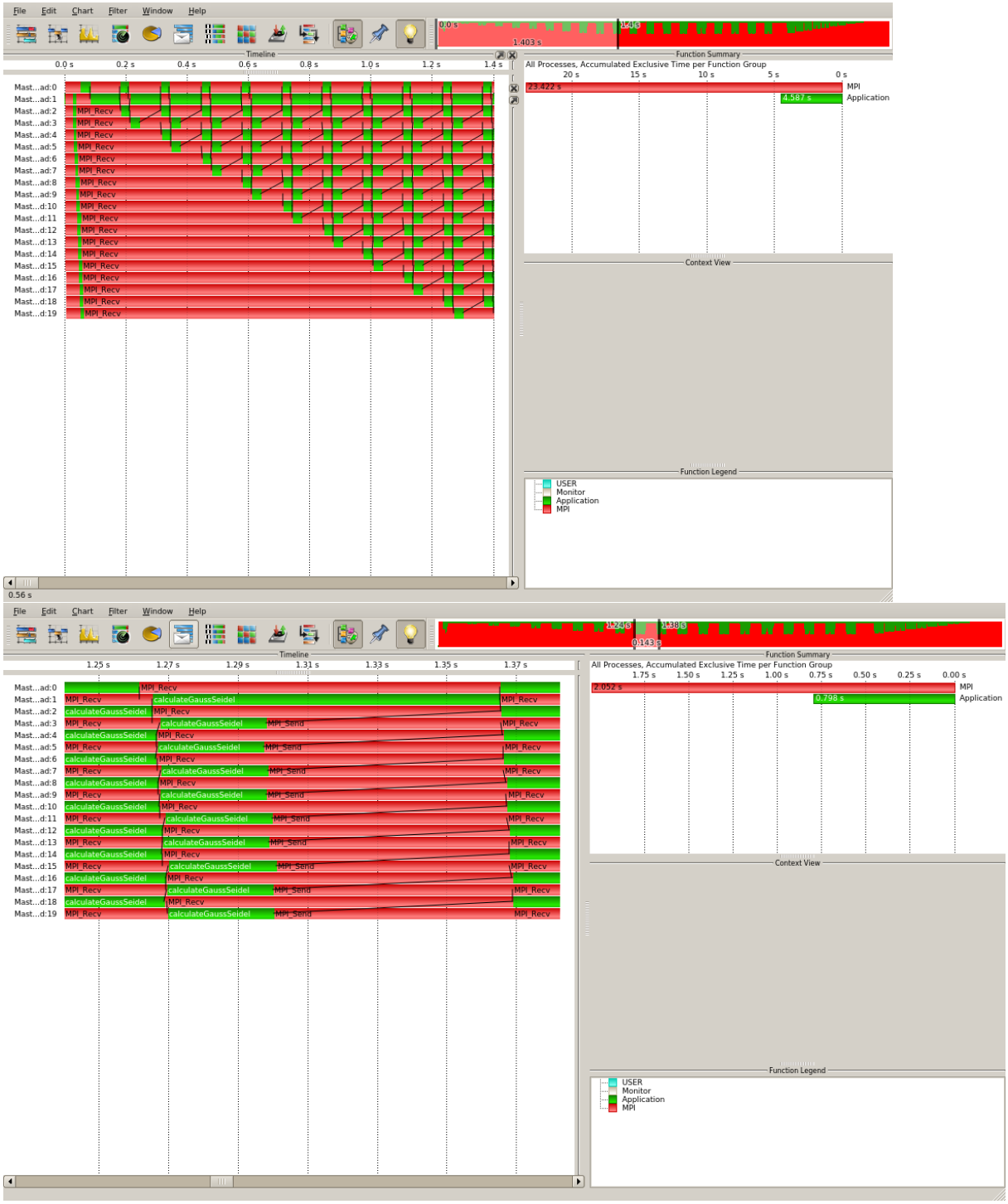
Gauss-Seidel

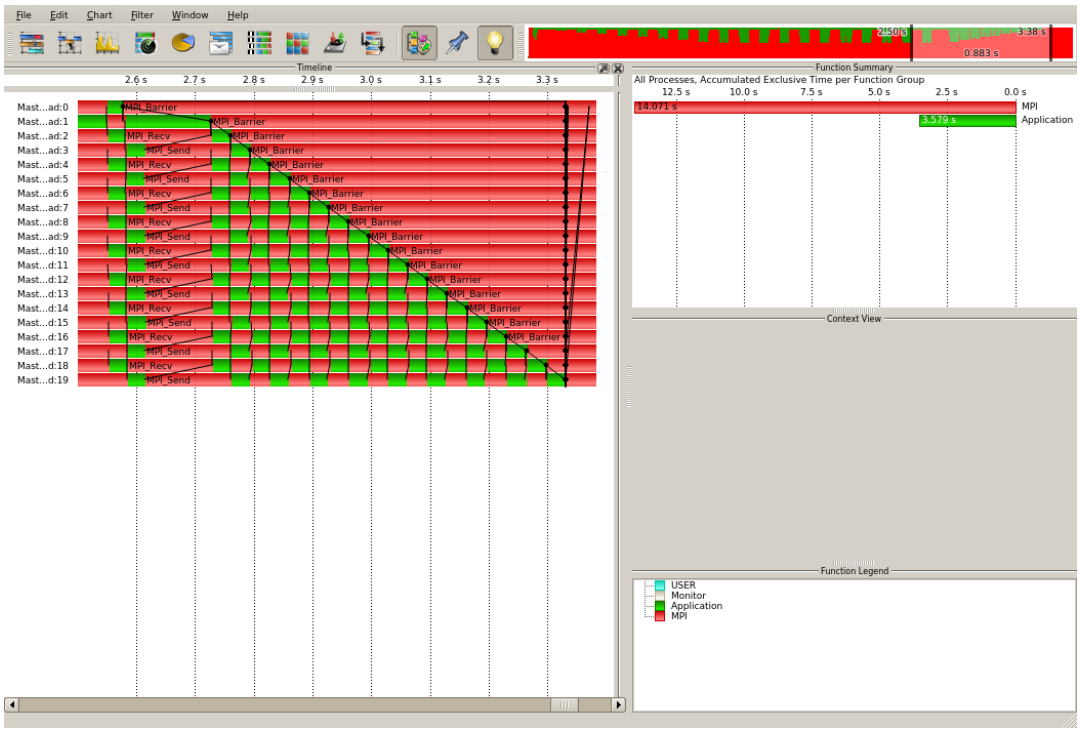
Knoten 2, 3 Prozesse




File
Edit
Chart
Filter
Window
Help
5.35 s
5.97 s
0.616 s
Timeline
5.4 s
5.5 s
5.6 s
5.7 s
5.8 s
5.9 s
Mast...ad:0
Mast...ad:1
Mast...ad:2
Mast...ad:3
Mast...ad:4
Mast...ad:5
MPI_Barrier
MPI_Recv
MPI_Send
Function Summary
All Processes, Accumulated Exclusive Time per Function Group
2.5 s
2.0 s
1.5 s
1.0 s
0.5 s
0.0 s
2.793 s
0.905 s
MPI
Application
Context View
Function Legend
USER
Monitor
Application
MPI
5.56 s

Knoten 5, 4 Prozesse





Interpretation

Startphase:
Zusätzlich zu den Anmerkungen die wir bei Jacobi schon zu dem generellen MPI-Overhead gemacht haben sieht man hier noch das zu erwartende Muster des Programmes, nämlich das die Prozesse alle nachenander starten, da immer auf das Ergebnis des vorangegangenen Prozesses gewartet werden muss bevor die eigene Berechnung beginnen kann.
Sync-Phase:
Bei dem Gauß-Seidel-Verfahren sieht man sehr deutlich wie einzelne Prozesse das ganze Programm verlangsamen können, gerade bei unserer 4,5-Version erkennt man das der Prozess 1 länger braucht und damit alle nachfolgenden Prozesse länger warten müssen, wodurch viel down-time innerhalb der einzelnen Prozesse entsteht.
End-Phase:
Bei der Enphase entsteht noch eine Menge idle-Zeit für die ersten Prozesse, da hier wieder gewartet werden muss, bis der letzte Prozess abgeschlossen hat und dann erst die Ergebnisse an den ROOT-Prozess übergeben werden. Durch die ganzen Wartezeiten die am Ende und bei der Sync-Phase entstehen sieht man sehr deutlich, das es sich für kleinere Matrizen kaum lohnt viele Prozesse zu benutzen und erst ab einer gewissen Größe die Parallelisierung Sinn macht.

Zusammenfassung:
Die Berechnung der Matrizen mittels des Gauss-Seidel Verfahrens läuft eigentlich wie erwartet, die einzige Anomalie, welche aber zu erwarten war, tritt bei der Synchronisation der Teilmatrizen nach einer Iteration auf. Hierbei berechnen einige Prozesse die Teilmatrix immer deutlich schneller als die Restlichen, dies folgt aber aus der Berechnung der halben Matrix, da hierbei einige Prozesse deutlich weniger Last haben können als andere.
Wie bei Jacobi schon bemerkt ist auch hier wie bei jedem MPI Programm der Overhead durch Start- und Endzeit zu beachten.
Dies entspricht im wesentlichen wieder der Erwartungshaltung, dass sich die Prozesse in der Iteration abwechseln und sonst sauber starten sowie korrekt beenden. Auch hier tritt aber die Auffälligkeit auf, dass einige Prozesse immer schneller rechnen als andere, dies lässt sich aber mit der bereits beschrieben Ursache erklären.

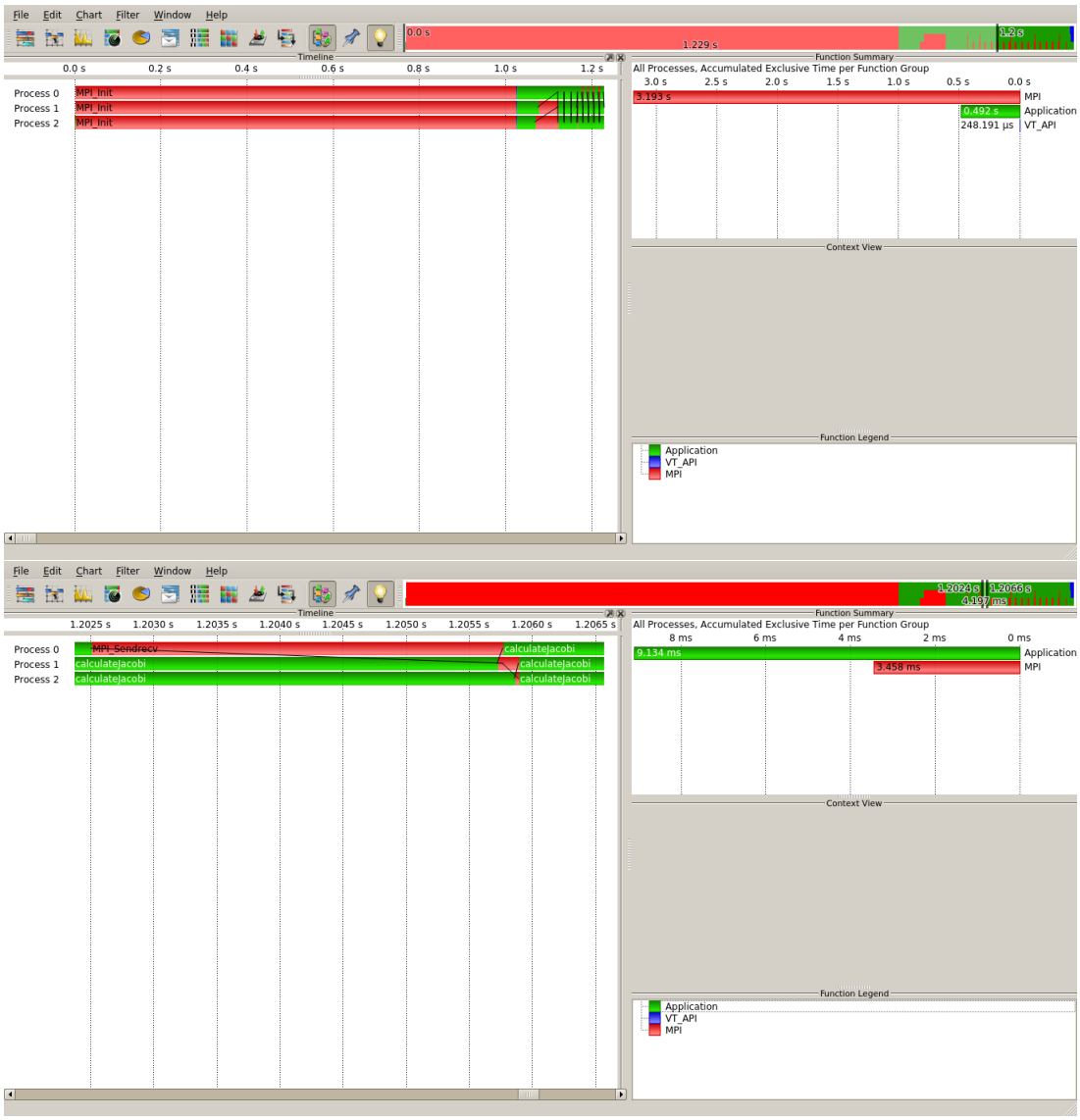
Anmerkung:
Wir waren uns bei der Aufgabenstellung nicht sicher, ob es sich um Knoten mit jeweils den gegebenen Prozessen handelt, oder ob die Prozesse insgesamt auf die Knoten aufgeteilt werden sollten. Deswegen haben wir noch die anderen Screnshots angehängt.

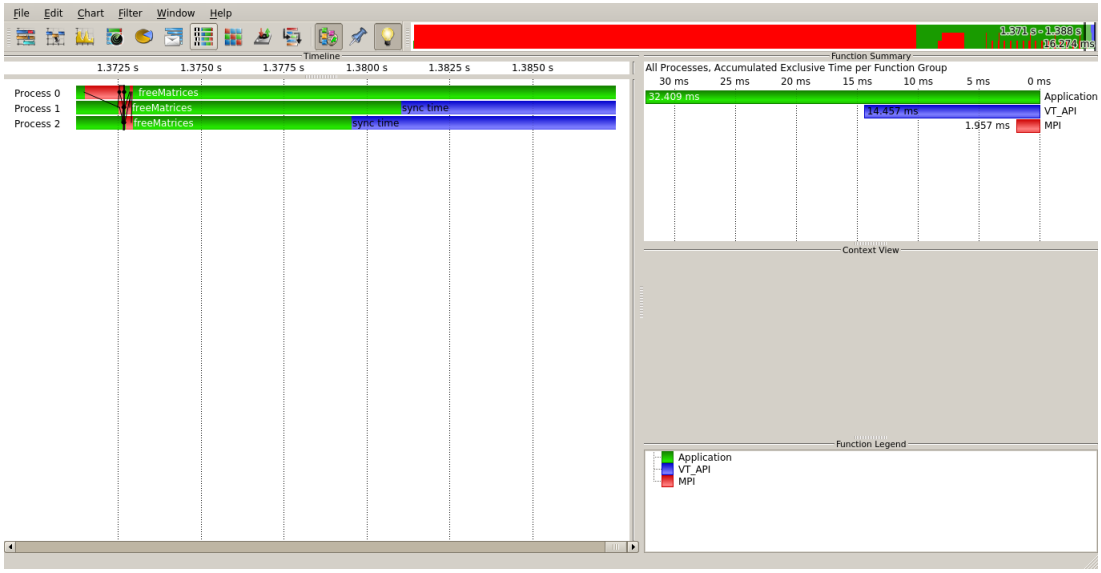
benötigte Zeit:
Messungen und Screenshots, 2h30m
Zusammentragen und Interpretation, 2h

Alternative Bilder

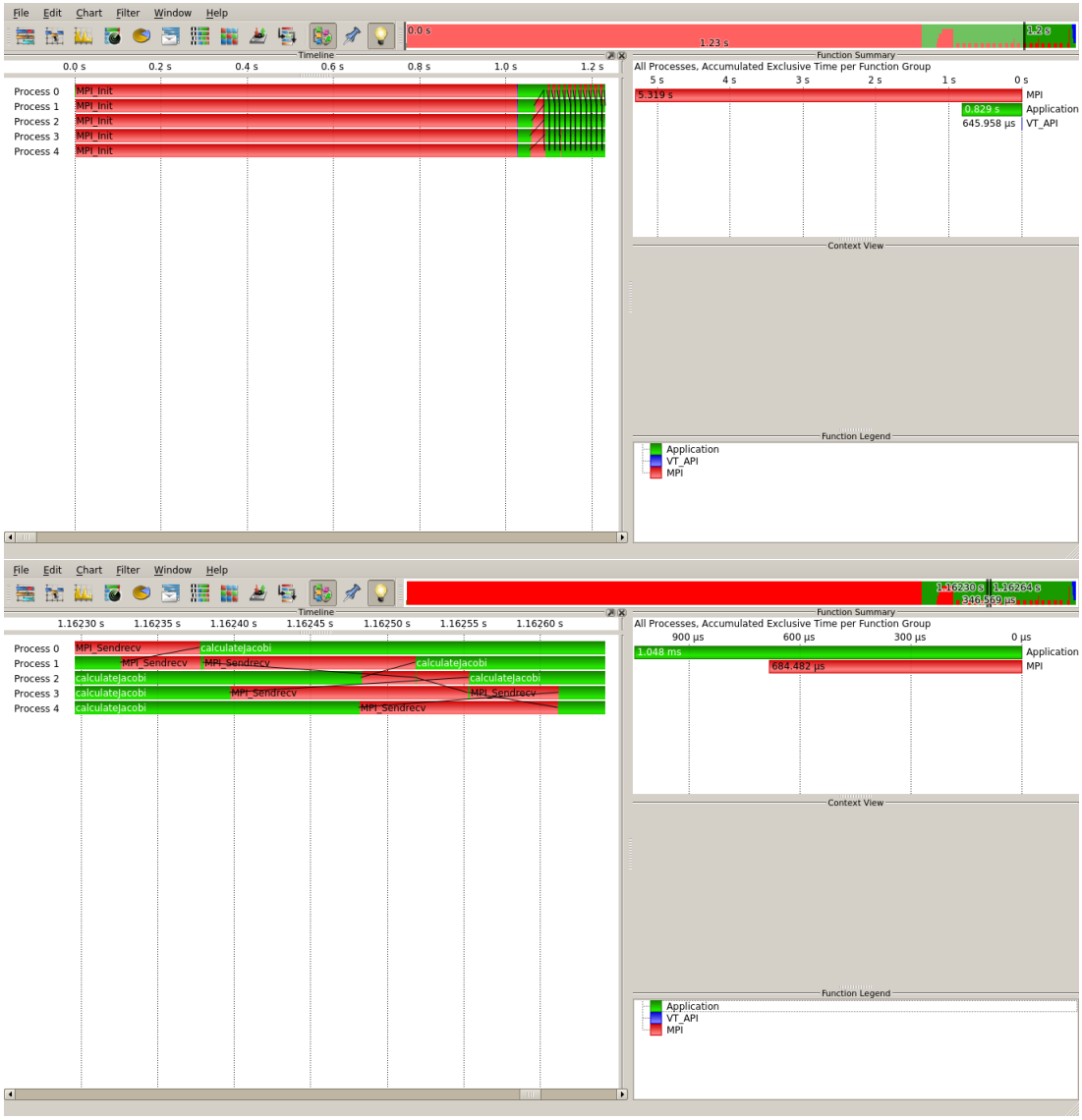
Jacobi

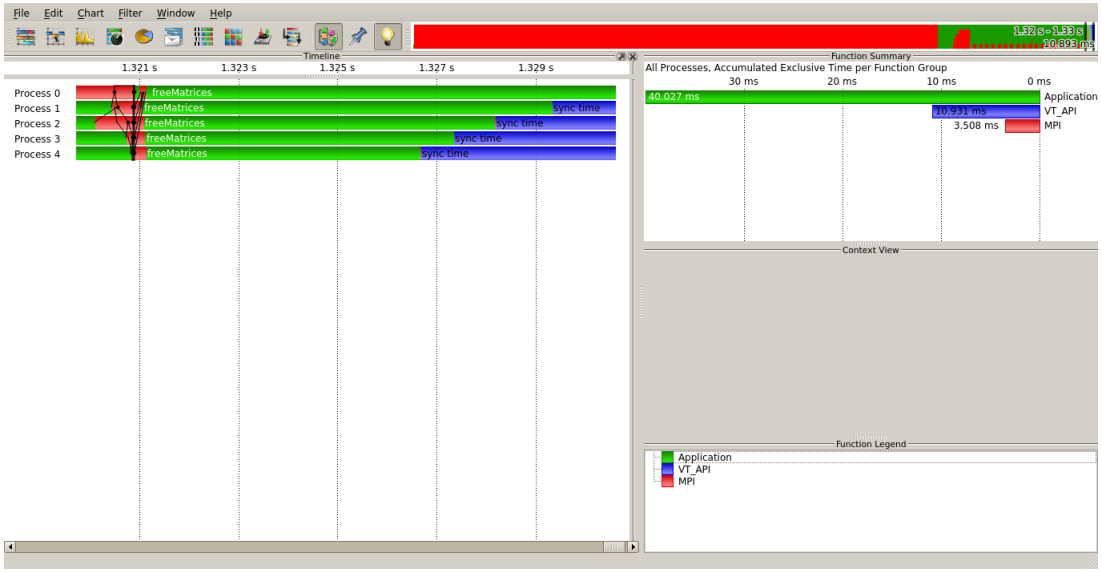
Knoten 2, 3 Prozesse




File Edit Chart Filter Window Help
1.371 s - 1.388 s
16.274 ms
Timeline
1.3725 s
1.3750 s
1.3775 s
1.3800 s
1.3825 s
1.3850 s
Process 0
Process 1
Process 2
freeMatrices
freeMatrices
freeMatrices
sync time
sync time
Function Summary
All Processes, Accumulated Exclusive Time per Function Group
30 ms
25 ms
20 ms
15 ms
10 ms
5 ms
0 ms
32.409 ms
14.457 ms
1.957 ms
Application
VT_API
MPI
Context View
Function Legend
Application
VT_API
MPI

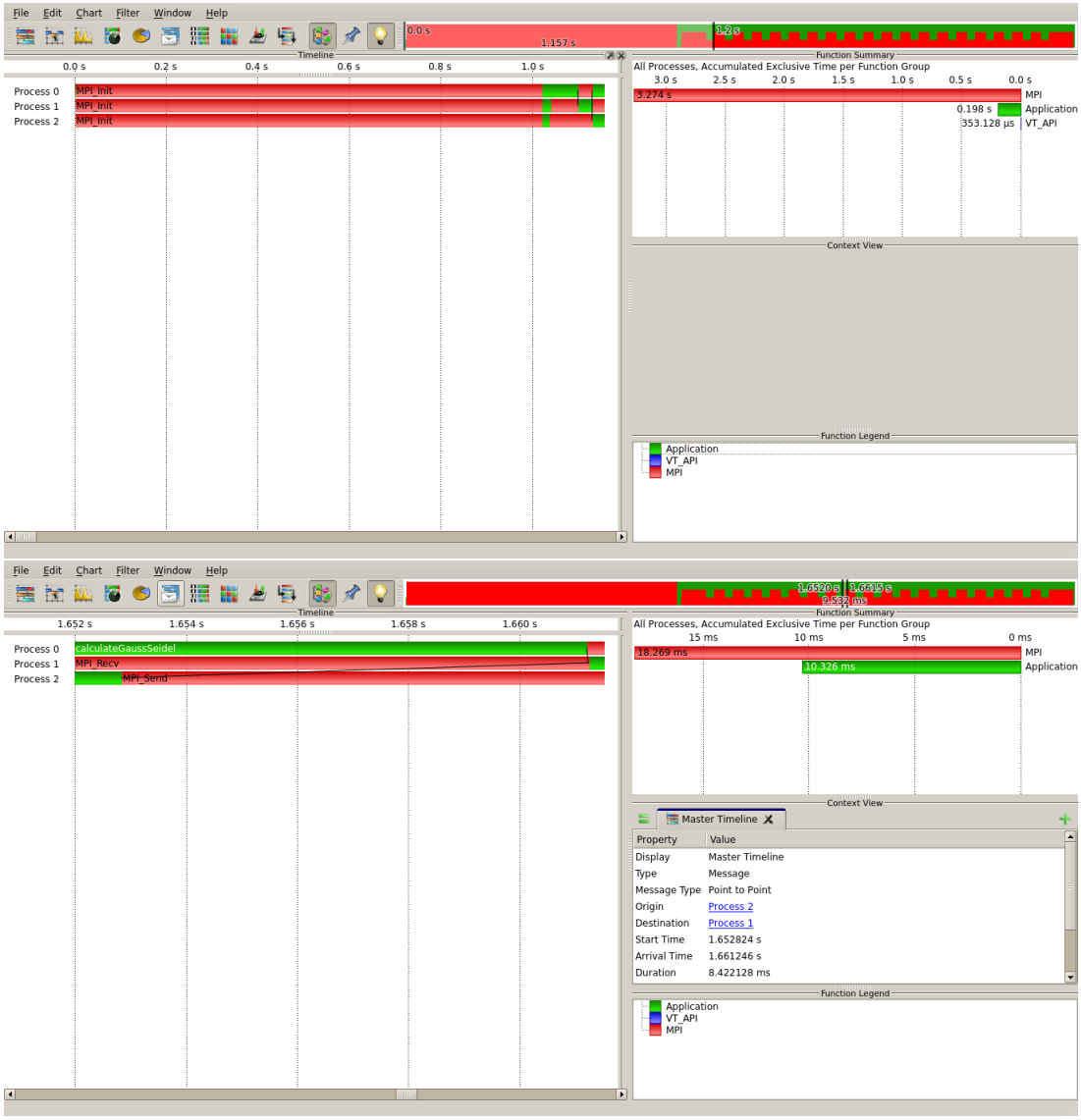
Knoten 5, 4 Prozesse

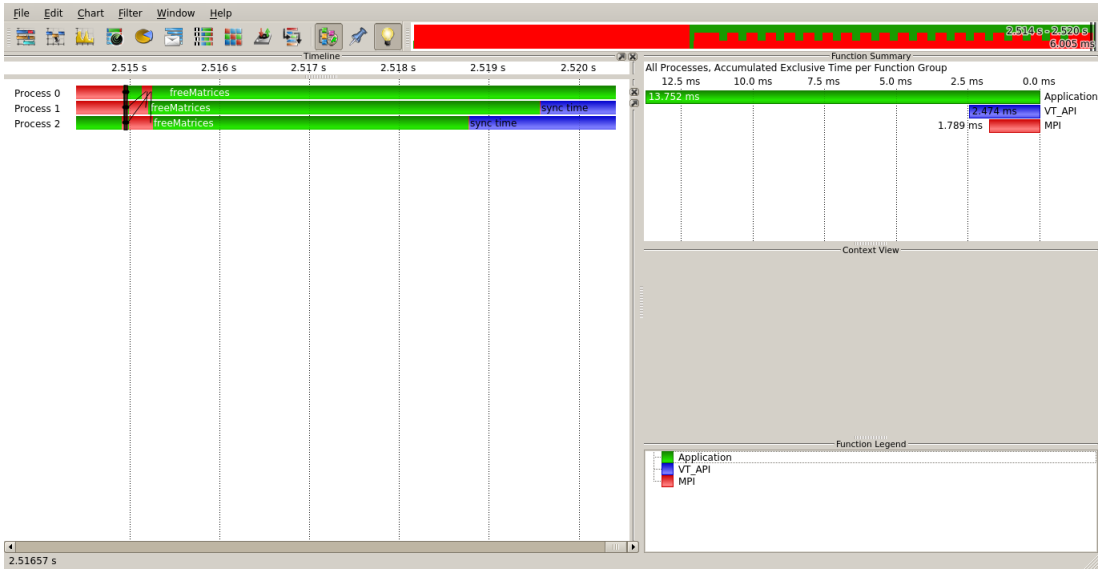



File Edit Chart Filter Window Help
1.32 s - 1.33 s
10.893 ms
Timeline
1.321 s
1.323 s
1.325 s
1.327 s
1.329 s
Process 0
Process 1
Process 2
Process 3
Process 4
freeMatrices
sync time
Function Summary
All Processes, Accumulated Exclusive Time per Function Group
30 ms
20 ms
10 ms
0 ms
40.027 ms
Application
10.931 ms
VT_API
3.508 ms
MPI
Context View
Function Legend
Application
VT_API
MPI

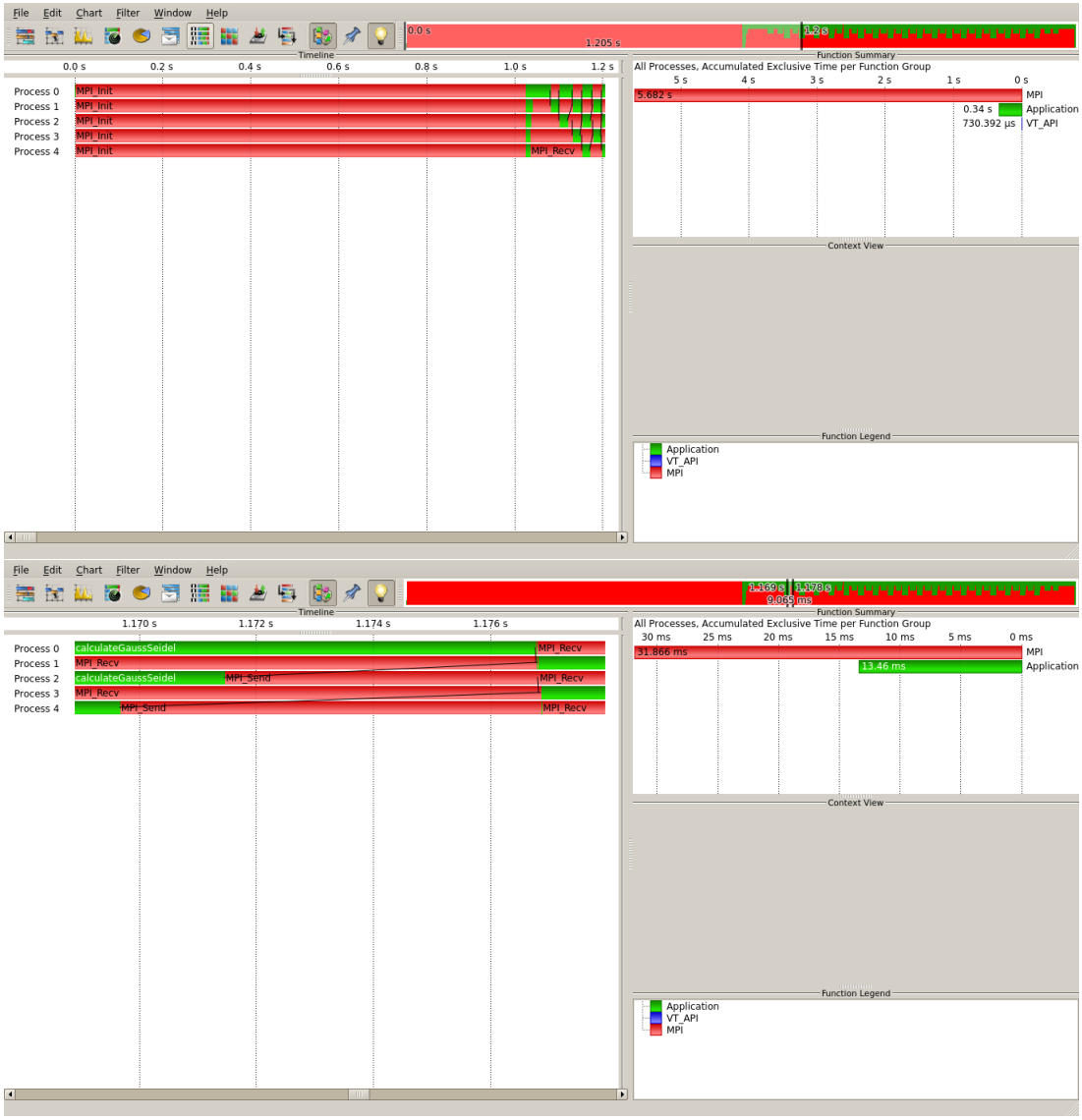
Gauss-Seidel

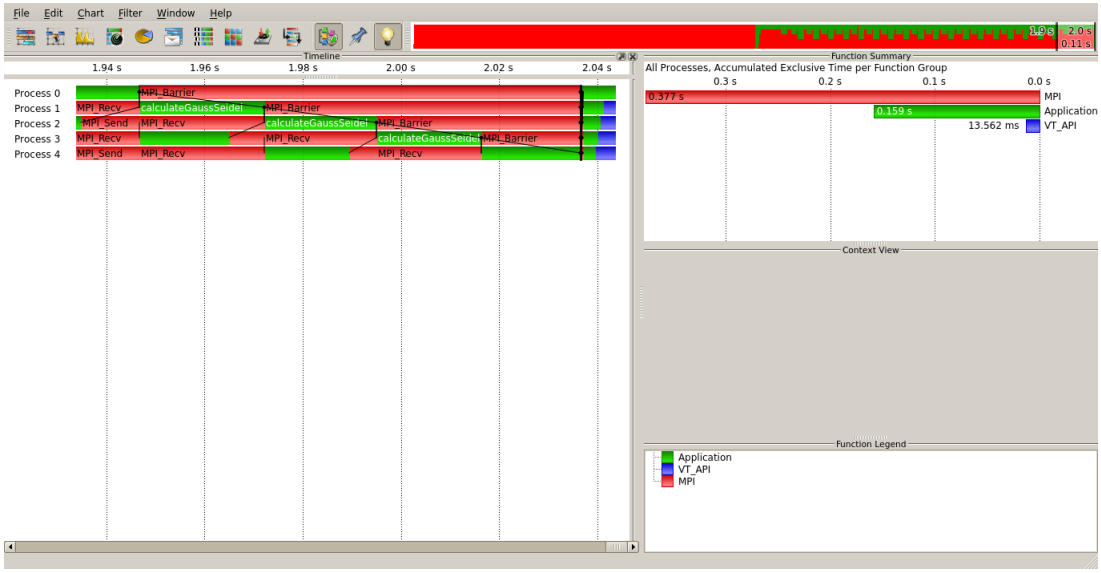
Knoten 2, 3 Prozesse




File Edit Chart Filter Window Help
2.514 s - 2.520 s
6.005 ms
Timeline
2.515 s
2.516 s
2.517 s
2.518 s
2.519 s
2.520 s
Process 0
Process 1
Process 2
freeMatrices
freeMatrices
Sync time
freeMatrices
Sync time
2.51657 s
Function Summary
All Processes, Accumulated Exclusive Time per Function Group
12.5 ms
10.0 ms
7.5 ms
5.0 ms
2.5 ms
0.0 ms
13.752 ms
Application
2.474 ms
VT_API
1.789 ms
MPI
Context View
Function Legend
Application
VT_API
MPI

Knoten 5, 4 Prozesse




File Edit Chart Filter Window Help
Timeline
1.94 s
1.96 s
1.98 s
2.00 s
2.02 s
2.04 s
Process 0
Process 1
Process 2
Process 3
Process 4
MPI_Barrier
MPI_Recv
calculateGaussSeidei
MPI_Send
Function Summary
All Processes, Accumulated Exclusive Time per Function Group
0.3 s
0.2 s
0.1 s
0.0 s
0.377 s
MPI
0.159 s
Application
13.562 ms
VT_API
Context View
Function Legend
Application
VT_API
MPI